NASA Contractor Report 177593

P. 514

Army-NASA Aircrew/Aircraft Integration Program: Phase IV A^{3}I Man-Machine Integration Design and Analysis System (MIDAS) Software Detailed Design Document

Carolyn Banda, David Bushnell, Scott Chen, Alex Chiu, Betsy Constantine, Jerry Murray, Christian Neukom, Michael Prevost, Renuka Shankar, and Lowell Staveland

(NASA-CR-177593) ARMY-NASA AIRCREW/AIRCRAFT INTEGRATION PROGRAM: PHASE 4 A(3)I MAN-MACHINE INTEGRATION DESIGN AND ANALYSIS SYSTEM (MIDAS) SOFTWARE DETAILED DESIGN DOCUMENT (Sterling (Walter V.)) 514 p

N92-29413

Unclas

G3/54 0104022

CONTRACT NAS2-13210
December 1991

NASA

National Aeronautics and
Space Administration

NASA Contractor Report 177593

Army-NASA Aircrew/Aircraft Integration Program: Phase IV A^3I Man-Machine Integration Design and Analysis System (MIDAS) Software Detailed Design Document

Carolyn Banda, David Bushnell, Scott Chen, Alex Chiu, Betsy Constantine, Jerry Murray, Christian Neukom, Michael Prevost, Renuka Shankar, and Lowell Staveland

Sterling Federal Systems, Inc.
1121 San Antonio Road
Palo Alto, CA 94303-4380

Prepared for
Ames Research Center
CONTRACT NAS2-13210
December 1991

NASA
National Aeronautics and
Space Administration

Ames Research Center
Moffett Field, California 94035-1000

Table of Contents

Figures

MAN-MACHINE INTEGRATION DESIGN & ANALYSIS SYSTEM (MIDAS) SOFTWARE DETAILED DESIGN DOCUMENT PHASE IV:

OVERVIEW

1.0 INTRODUCTION

1.1 IDENTIFICATION OF DOCUMENT

This document is the Software Product Specification for the Man-Machine Integration Design and Analysis System (MIDAS). Introductory descriptions of the processing requirements, hardware/software environment, structure, I/O, and control are given in the main body of the document for the overall MIDAS system, with detailed discussion of the individual modules included in Annexes A-J.

1.2 SCOPE OF DOCUMENT

The A^3I Program is a phased development program in which software development takes place in well-defined cycles, beginning with an off-site planning meeting to establish requirements, progressing through system design and development and culminating in demonstration to interested persons in government, academia and industry. After the demonstrations, the software is documented and then a new phase begins. This document records the requirements and design of MIDAS as it existed at the end of Phase IV. A similar document has been produced during each phase of development, tracking the history of changes to the configuration, and more importantly, the motive for such changes. For historical information regarding Phases I, II and III, the reader is urged to consult Section 9, Historical Information. Documentation from previous phases is referenced in Section 2.2, Information Documents.

This document is intended for use by programmers and other technical specialists working with MIDAS. Sufficient high level information is provided to allow any reader to become familiar with the objectives of the A^3I Program, the current (as well as previous) overall architecture and development philosophies, while at the same time containing implementation detail useful to programmers involved with specific application software modules.

1.3 PURPOSE AND OBJECTIVE OF DOCUMENT

The purpose of this document is to meet three objectives: one, to record the status and philosophy of design of MIDAS as it existed at the end of Phase IV, two, to provide programmers with sufficient implementation detail to facilitate use or modification of MIDAS in particular application environments, and three, to provide technically oriented readers with a more detailed description of MIDAS than can be given in a typical 3-hour demonstration.

2.0 RELATED DOCUMENTS

2.1 APPLICABLE DOCUMENTS

Army-NASA Aircrew/Aircraft Integration Program, A^{3}I, Executive Summary, 1 Sept, 1990

2.2 INFORMATION DOCUMENTS

A^{3}I Phase II Human Factors/Computer-Aided Engineering Workstation Suite Architecture Description Document, Revision 1, 30 Nov 87

Cody, William J., *Recommendations for Supporting Helicopter Crew System Design*, US Army Human Engineering Laboratory Contract No. DAAD05-87-M-L584 to Search Technology, Inc., June 1988.

Human Performance Models for Computer-Aided Engineering, National Research Council Committee on Human Factors, National Academy Press, 1989

Product Specification Documentation Standard and Data Item Descriptions (DID) Volume of the Information System Life-Cycle and Documentation Standards, Release 4.3, NASA Office of Safety, Reliability, Maintainability, and Quality Assurance,Software Management and Assurance Program (SMAP),Washington, DC, February 28, 1989.

Army-NASA Aircrew/Aircraft Integration Program (A^{3}I) Software Detailed Design Document: Phase III, Contractor Report 177557, NASA Ames Research Center, Moffett Field, California 94035-1000, June 1990.

Smith, B., Six Years into the A3I Program: Progress & Problems. Paper presented at the AFHRL Workshop on Human-Centered Design Technology for Maintainability, 12-13 September, 1990

3.0 CONCEPT

3.1 DEFINITION OF SOFTWARE

MIDAS is an integrated suite of software components that constitutes a prototype workstation to aid designers in applying human factors principles to the design of complex human-machine systems. MIDAS is intended to be used at the very early stages of conceptual design to provide an environment wherein designers can use computational representations of the crew station and operator, instead of hardware simulators and man-in-the-loop studies, to discover problems and ask "what if" questions regarding the projected mission, equipment, and environment.

3.1.1 Purpose and Scope

The purpose of MIDAS is to provide design engineers/analysts with interactive symbolic, analytic, and graphical components which permit the early integration and visualization of human engineering principles guiding design. Currently hosted on a number of networked Symbolics and Silicon Graphics workstations, MIDAS serves as the framework in which research findings and models, developed by, or sponsored through the Computational Human Engineering Research Office, are incorporated.

MIDAS contains tools to describe the operating environment, equipment, and mission of manned systems, with models of human performance/behavior used in static and dynamic modes to evaluate aspects of the crewstation design and operator task performance. The results are presented graphically and visually to the design engineers, often as a computer simulation of "manned flight." In this sense, MIDAS is similar in concept to computational tools such as finite element analysis and computational fluid dynamics which are used to improve designs and reduce costs.

Seventy to eighty percent of the life-cycle cost of an aircraft is determined in the conceptual design phase. After hardware is built, mistakes are hard to correct and concepts are difficult to modify. Engineers responsible for developing crew training simulators and instructional systems currently begin work after the cockpit is built and too late to impact its design. MIDAS gives designers an opportunity to "see it before they build it", to ask "what if" questions about all aspects of crew performance, including training, and to correct problems early. The system is currently focused on helicopters, however its model and principle basis permits generalization to other vehicles.

3.1.2 Goals and Objectives

The A^{3}I Program has three major goals organized under a research and development umbrella covering a wide range of activities from the sponsoring and conduct of basic research in perception to the application of software engineering practices to provide useable human factors tools. These goals are:

1) Develop an integrated methodology for crew station prototyping based on models and principles of human factors and engineering psychology. This methodology shall be developed around a flexible facility for collecting and using tools/models which initially can answer specific critical questions required by the existing crew station design process.

2) Advance the capabilities and use of computational representations of human performance in the conceptual design, synthesis, and analysis of manned systems. These representations or models should be normative if possible, and focused for use in a simulation with explicit inputs and outputs.

3) Transfer the relevant technology/findings, amassed from the above, to interested research and practitioner organizations in industry, government, and academia. The A^{3}I Program (through consortia) must lay the foundations for participation by the larger community of researchers and designers who are contributing to the development of computational human factors or who might become users of the methods and models developed by researchers in the field.

These goals are to be accomplished by developing a Human-Factors / Computer-Aided-Engineering (HF/CAE) system, called MIDAS, which assists designers in considering human performance characteristics to be an integral part of the overall system operation from the earliest stages of the design process. To meet the overall Program goals, the software system, MIDAS, must meet the following objectives.

1) MIDAS must be maintained as a flexible integrated modeling environment. An enormous amount of modeling is anticipated throughout the life of the Program, with the operator, vehicle and world as the major categories of models. Many of these models, and the subsequent tools based on them, will exist in various formats and stages of development. Consequently, the architecture of MIDAS must be modular, allowing for the existence of a

collection of models rather than one monolithic model. The collection of tools will grow and change over time. MIDAS must be designed as a framework within which a heterogeneous collection of tools and models can be integrated and used effectively by a design team. It is also intended that the MIDAS workstation will serve a dual purpose by providing an integrated environment for inspection and testing of new models. As the workstation architecture evolves, it is expected that both deterministic and stochastic constructs will be required within the same simulation environment to make use of the widest possible range of extant models. The use of abstraction is also strongly encouraged as part of this integrated environment. The goal is to be able to simplify each model from precise computational representations to approximate, qualitative models, as determined by the interactions under study and the answers desired.

2) MIDAS must be supportive of a wide range of designer activities. As described in a study commissioned by the A^{3}I Program (Cody 1988), the crew station designer engages in a wide array of activities. Initially, the tools and models contained within the MIDAS workstations must support design specification, static analysis and dynamic analysis, growing into the more complex synthesis task once successful at these. To support the specification phase, MIDAS must contain tools to allow the user to input or "specify" the elements of the mission, crew, cockpit, vehicle performance, and environment that are given or known. Where possible, routine static analysis tasks such as assessing reach or visibility under fixed conditions must also be supported. However, the majority of the difficult and interesting crew station design problems will require a dynamic simulation and analysis capability. Only through this capability will users be able to discover the inherent complex interactions among the operator, task, equipment, and environment which drive aspects of workload, performance, and training.

3) MIDAS must support and facilitate interdisciplinary communication. Historically, a major impediment to rational cockpit design has been the lack of communication among the practitioners of the various design disciplines involved in the development process. Even when placed in proximity on a development team, there is frequently a lack of appreciation by team members for each others' positions, due in large part to differences in training, language (terminology), objectives, and points of view. Development team specialist disciplines may include design engineers, avionics engineers, mission specialists (operations and doctrine), pilots, engineering psychologists, training systems specialists, reliability/maintainability analysts, and cost and production experts. MIDAS must attempt to provide a basis for communication among these specialists, integrating the information generated by them, possibly arbitrating between them, alerting users to the need for specialists not currently engaged, and mediating between the design team and the end-product user or sponsor.

3.1.3 Description

MIDAS consists of a set of software components providing multiple perspectives or "windows" which contain information about the mission, operator, and environment in varying levels of modeling detail. Visualization is emphasized, with 3-D graphic and iconic representations used as the major means of communication to the wide range of potential users.

Figure 1 depicts what is notionally expected to be contained within MIDAS and shows how these various components might interact and overlap to address the various stages of crew station design and analysis.

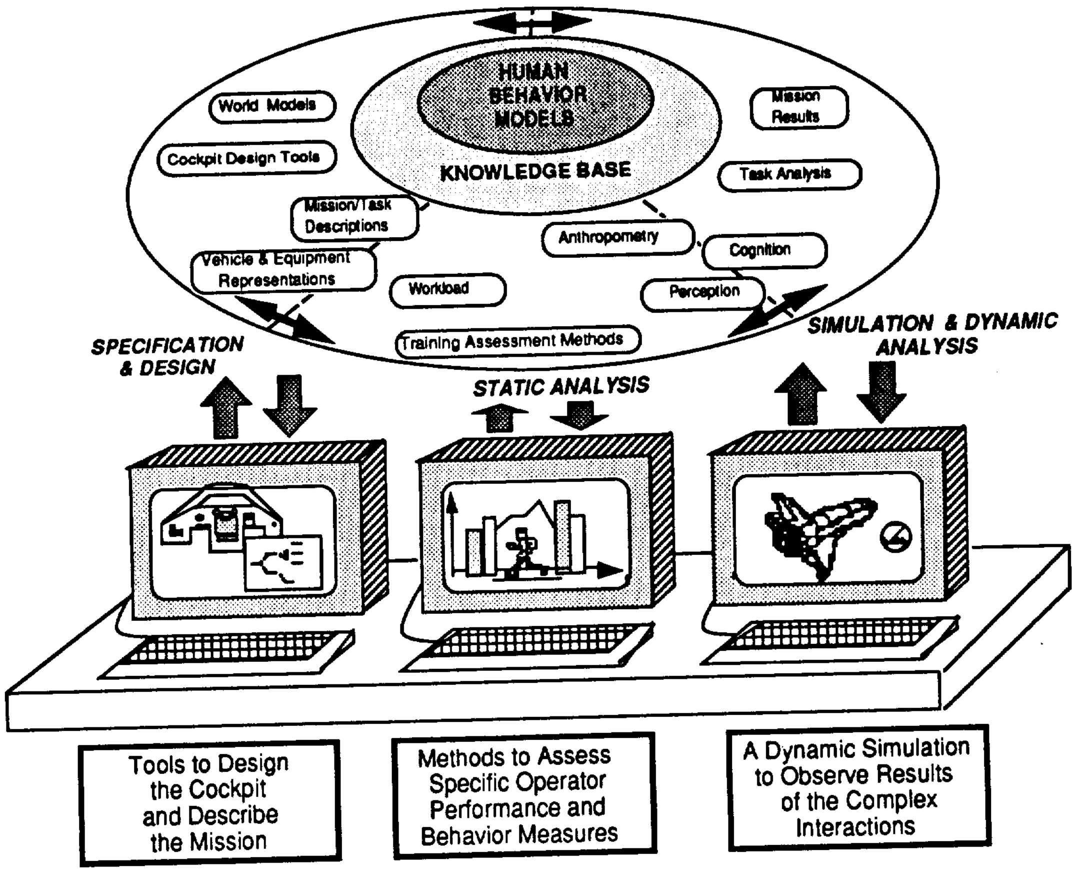

HUMAN BEHAVIOR MODELS
KNOWLEDGE BASE
World Models
Cockpit Design Tools
Mission/Task Descriptions
Vehicle & Equipment Representations
Workload
Training Assessment Methods
Anthropometry
Cognition
Perception
Mission Results
Task Analysis
SPECIFICATION & DESIGN
STATIC ANALYSIS
SIMULATION & DYNAMIC ANALYSIS
Tools to Design the Cockpit and Describe the Mission
Methods to Assess Specific Operator Performance and Behavior Measures
A Dynamic Simulation to Observe Results of the Complex Interactions

Figure 1. Functional Content of MIDAS

Forming the core of this system will be a number of human behavior and performance models, intended to address the critical areas of perception, cognition, workload, etc. as shown. Surrounding and augmenting this core will be heuristic methods or knowledge bases. These are needed to supplement what is explicitly known and capable of being modeled about the operator in order to complete the human representation and form a closed-loop system. Included within this overall functional context will be a number of tools and methods to design and describe elements of the cockpit, environment, and task, used as stimuli for a range of models. Finally, analytical methods are also included to address anticipated training requirements, summary mission results, and task analysis parameters.

These notional elements are shown distributed among the specification, static analysis, and dynamic analysis stages, indicating a high degree of overlap between these phases. Arrows between such phases are meant to depict that the process is not constrained to follow a linear progression from specification to static analysis and then dynamic analysis, but instead could be used to begin with results from a particular stage, and then progress backward to find out characteristics of the crew station or environment which are the "drivers" for such predictions.

3.2 USER DEFINITION

The study cited above (Cody 1988) found that crew system design involved manipulating the effects of six major system factors and their interactions. These factors are aircrew selection, training, equipment design, job design, aiding, and protection. Furthermore, they found that designers, as individual problem solvers, engage in seven major classes of activity: problem formulation, synthesis, consequence finding, consequence analysis, fabrication and prototyping, evaluation and judgement, and information control. Additionally, the study found that approximately 50 distinct disciplines, ranging from acoustics to probability theory, were involved in the crew station design process, representing 15 major fields of study. Finally, the report surveyed existing and emerging forms of design support and found that less than one third of the design problem space (characterized as a matrix of the aforementioned activities and factors), had some form of tool for support. More significantly, virtually none of these models or tools surveyed addressed the complex interactions between combinations of the design activities and system factors.

The substantial body of data and information generated during the simulation must be interpreted in a manner that is meaningful and relevant to various specialists evaluating a candidate cockpit design. Because the crew station design process involves professionals from a wide range of disciplines, the A^{3}I Program has chosen to emphasize visualization as the primary form of information output. Graphic and iconic representations are predominantly used as a means to foster interdisciplinary communication.

Furthermore, because the crew station design process includes extremely varied activities, the tools and models contained within the workstation are designed to support specification and static analysis as well as dynamic analysis and simulation. For example, analysis of reach and fit of cockpit controls can be performed on geometric representations of the cockpit without the need for elaborate vehicle dynamics/systems or human performance simulation models. On the other hand, modeling the complex interactions of competing tasks during the performance of a mission requires considerable dynamic modeling of both human behavior/performance and vehicle systems to capture the context-sensitive, time-varying nature of task sequencing and resultant loads.

3.3 CAPABILITIES AND CHARACTERISTICS

The components of MIDAS at the end of Phase IV include:

1) The Symbolic Operator Model, which contains methods to 1) represent and decompose decomposition of mission goals to their lowest level, 2) sort these matched goal-activity patterns by priority, 3) find matching equipment operation patterns or activities which will satisfy them, 4) interact with the scheduling and loading operator model components as appropriate and 5) execute these activities subject to physical resource (hand, eye, etc) requirements, Visual Auditory, Cognitive, and Psychomotor (VACP) load limits, and temporal/logical constraints.

2) The Scheduler (Z), which solves for a near-optimal sequence and schedule based on a strategy of either time minimizing or load balancing, intended to represent possible operator behaviors.

3) The Task Loading Model, which, in accordance with current research in multiple resource theory, classifies individual tasks in terms of their demands on the

Visual, Auditory, Cognitive and Motor processing dimensions based on attributes of the mission tasks, world state, operator, and crew station equipment.

4) The Symbolic Equipment Models, which allow characteristics of the crew station equipment to be represented in terms of both physical and functional attributes which are used to specialize the mission tasks prior to a simulation.

5) The Visual Editor and Simulation Tool (VEST), which provides an interactive 3-D tool used to create, control and observe, from several perspectives, the 3-D graphic environment of the mission simulation. VEST includes the The Cockpit Design Editor (CDE) component, containing 3-D CAD utilities for graphically prototyping the crew station geometry, instruments, controls, and displays using a built-in library of primitive cockpit objects.

6) The Display Layout Analysis (DLA), which is a prototypical tool intended to provide designers with assistance in determining the spatial configuration of cockpit displays.

7) The Anthropometric Model (Jack), which provides realistic and physically quantifiable human figure definition and motion within a 3-D space environment.

8) The Vision Models, which are fully integrated into the Jack environment, include the Volume Field-of-View Model, which provides computer graphic methods for representing the relationship between two-dimensional visual field maps and the three-dimensional visual space they serve, and the Cockpit Display Visibility Model, which provides an assessment of the visibility of cockpit objects imaged on the retina in terms of a visual system footprint.

9) The Aerodynamics & Guidance Model (AGM), which represents rather generic helicopter guidance and dynamics for uncoupled controls.

10) The Simulation Executive and Communications Module, which facilitates inter-machine communication and dynamic message sharing between all MIDAS components, using a "data pool" concept during the simulation to synchronize and distribute state variables among the simulation processes and objects.

11) The Training Assessment Module, which provides a means to estimate the training media, instructional techniques, and time necessary to qualify in the cockpit under development. This module was developed in Phase III and not modified during Phase IV; detailed documentation of this module is included in the Phase III design document.

3.4 SAMPLE OPERATIONAL SCENARIOS

One type of user of MIDAS, a crew station designer, might first approach MIDAS with a requirement to design a new crew station. He/she would have a mission that must be performed and from that derive a preliminary specification of the kinds of information that must be presented to the operator in the crew station. The initial point of contact with MIDAS could be through the Display Layout Analysis tool, through which the designer could plan and evaluate alternate configurations for displays in the crew station. With an initial configuration in mind, the designer could then use the Cockpit Design Editor to render a 3D graphic representation of the displays and controls in the cockpit. The designer could then put Jack in the cockpit and perform some static analyses of reach and fit using figures with different physical characteristics. Field of view and visibility analyses could be performed in the context of Jack using the Vision Models.

To perform a dynamic analysis of the interaction of a human model in the cockpit environment, a mission scenario, with goals and activities would be defined in the context of the Symbolic Operator Model. In addition, he/she would specify both the physical and functional characteristics of the cockpit equipment by selecting or building equipment models. With the mission and cockpit environment defined, the designer could run a simulation of the mission scenario and observe a 3D graphic rendition of operator behavior in the proposed cockpit and world environment. Data on satisfied and unsatisfied goals could be obtained from the Symbolic Operator model. Mission tasks could be evaluated by the Task Loading Model to obtain load traces in the four dimensions: visual, auditory, cognitive and motor.

4.0 REQUIREMENTS

A number of key developmental aspects must be mentioned prior to describing Phase IV specifics because they are central requirements for the overall A^{3}I Program. Perhaps most importantly, the Program attempts to use extant software and systems to the greatest extent possible. A considerable amount of research effort and money has been expended nationally, over more than two decades, to produce analytic methods, models and structures representing the behavior and functions of human operators, avionics systems, and missions, with varying degrees of success. The A^{3}I Program is not expressly chartered to develop new models and methods. Where possible, the staff will selectively employ those which have already been developed and will engage in research and development of new models/methods only when a critical void is encountered. The advice and guidance of the National Research Council's Committee on Human Factors (offered by the study group on Human Performance Models and the report *Human Performance Models for Computer-Aided Engineering*) has been solicited and followed on this matter.

An enormous amount of modeling is anticipated throughout the life of the Program, with the operator, vehicle and world as major categories of models. The A^{3}I Program has elected a simulation-based approach to system design and evaluation that proposes to make extensive use of graphic and iconic representations of the underlying model structures. Models will generally be prescriptive, providing results relevant to mission success in terms of such parameters as performance, errors, duration, and rates. Wherever practical, input-output models with 5-10 years of development and validation, supported by a substantial body of empirical data, will be utilized. However, many instances will require the use of models emerging from ongoing research, where final verification and validation has yet to be completed, as well as qualitative models emerging from the AI field. It is expected that the MIDAS workstation will serve a dual purpose in aiding this process by providing an integrated environment for model inspection and testing.

4.1 REQUIREMENTS APPROACH AND TRADEOFFS

On the first day of the A^{3}I Program, *what* needed to be done and *why*, was known, at least in general terms. *How* to do it was not known, nor could one predict very far into the future as to progress which might be expected. The inability to clearly specify the ultimate end product dictated the adoption of certain approaches to ensure that even in the face of uncertainty, this highly ambitious program could be directed toward its goals while managing risk and ensuring that intermediate products could be made available to interested users and collaborators.

4.1.1 Incremental Development

In the A^3I Program, a standard practice, incremental development, has evolved to take the following programmatic form. Development is accomplished in phases, with each phase consisting of planning, design, development, testing, demonstration and documentation of software to fulfill a specific set of requirements. A development phase begins with an off-site meeting, lasting three to four days and attended by the full in-house staff, extramural associates, and invited peer reviewers, which is held at a location conducive to uninterrupted deliberations. During the off-site, plans are made for the next phase of development based on what has been learned during the previous phase, advice from reviewers and associates, assessment of weaknesses in current methods, and comments/recommendations made by visitors and reviewers of the previous development phase. The products of the off-site are a formal statement of requirements for the next phase of development, an agreement as to the length of the development period (usually 8 to 14 months), and an understanding among the staff and extramural members of what is expected to be accomplished — a common vision.

Upon our return to NASA-Ames and the laboratory, the details of the work are finalized and development begins. Work breakdown structures and schedules are developed to track phased development in each major work area. Depending on the length of the development period, a number of in-progress reviews (IPRs) are held to monitor the development and discover/correct problems. Two weeks before the end of the development, the writing of any new code is terminated. Final checks of the code and models are made and any bugs corrected.

The end of the development phase is marked by a series of demonstrations highlighting the work produced during that phase or the integration of new work with products of the previous phase(s). These demonstrations are broadly advertised, and as the Program has grown so have the number of visitors, and therefore, the number of demonstrations. The demonstrations average about two and one half hours, but have been known to exceed five hours. Depending on the audience, the detail of the demonstrations may go down to the code level. Much is learned during these demonstrations from the visitors to the laboratory. Many are from academia in the fields of psychology, AI, and Computer Science and their suggestions and comments are extremely helpful, frequently being used as inputs for the next phase of development.

At the end of the demonstration period, effort is devoted to documentation (primarily this Software Detailed Design Document) and recording of lessons learned in preparation for the next off-site meeting. By this method, the program has been "boot-strapping" itself along very successfully since the first off-site meeting in the Fall of 1985.

Figure 2 below depicts major phase milestones since the Program's inception.

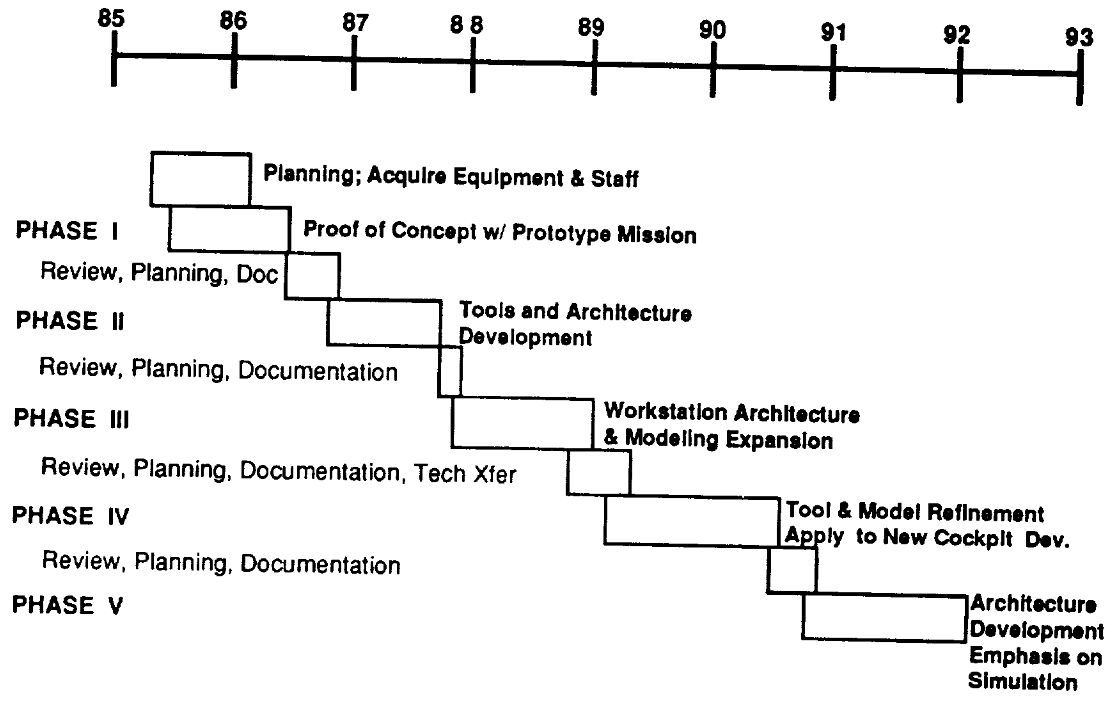

85
86
87
88
89
90
91
92
93
Planning; Acquire Equipment & Staff
PHASE I
Proof of Concept w/ Prototype Mission
Review, Planning, Doc
PHASE II
Tools and Architecture Development
Review, Planning, Documentation
PHASE III
Workstation Architecture & Modeling Expansion
Review, Planning, Documentation, Tech Xfer
PHASE IV
Tool & Model Refinement Apply to New Cockpit Dev.
Review, Planning, Documentation
PHASE V
Architecture Development Emphasis on Simulation

Figure 2. A^{3}I Program Timeline

4.1.2 Development Techniques

MIDAS is a networked collection of graphical, numeric, and symbolic processors which forms a flexible, integrated modeling environment. One of the central challenges is to design and implement a general environment that can easily accommodate models written in various languages by a variety of developers, while maintaining inspectability and modularity at various levels of modeling abstraction. A number of methodologies have been used to satisfy these requirements.

4.1.2.1 Rapid Prototyping

Since its inception, the A^{3}I Program has emphasized the evaluation of architectural issues and model requirements through rapid prototyping. Much of what is being attempted by the software development staff is radically new with no known, proven methods. Rapid prototyping as a means to develop the various tools and models has both guided and facilitated subsequent effort, providing the opportunity for scientific scrutiny and user-community feedback before an overwhelming development effort is expended.

4.1.2.2 Distributed, Tick-Based Simulation

The key characteristic of MIDAS is its support for a realistic dynamic analysis or simulation capability. At each step in the selected scenario, human performance models interact with the aircraft dynamics models, equipment models, and environment to generate the next event-step in the scenario as it unfolds. Thus if mission demands, display interpretation requirements, information handling demands, or multiple tasking, etc. exceed reasonable human capabilities, the operator performance and behavior models should indicate that reduced performance would be expected. In MIDAS, the various models and tools are distributed across a number of platforms and inter-machine communication is achieved through an internally developed communication package which uses standard EtherNet and TCP/IP protocol. The Communications module sends ticks across the network to each module, implementing a discrete, scaled time or tick-based simulation of the objects or models of interest. This computer graphic simulation of a man-in-the-loop simulation provides results relevant to mission success in terms of parameters such as performance, errors, duration, and rates, by maintaining world, operator, and equipment states and propagating their effects over time. Since computational models and principles of human performance replace the human operator found in typical man-in-the-loop simulations, MIDAS contains no requirement for a real-time simulation capability.

4.1.2.3 Graphic & Iconic Interface

Visualization methods were chosen as the preferred method for presenting complex computational results. MIDAS' emphasis on visualization carries several important connotations. First, it is hoped that visualization will facilitate the use of the system in a design/analysis session without requiring an undue amount of knowledge about the underlying implementation. Second, data and information selected as interesting can be presented in a form the designer/user finds easy to interpret. Alphanumeric tabular forms, though easiest to generate, are often ill-suited to designer needs. Hence, alternate forms such as graphic and iconic representations are provided to facilitate the designer/user's easy insight into the overall progress of the simulation or analysis session and a global understanding of complex and interrelated man-machine factors. Finally, visualization is perhaps the best way for MIDAS to facilitate communication between designers from different technical disciplines. Commonly-understood pictures can be substituted for words, which may have different meanings to each. The result is a depiction of the impact of design decisions in a form which is meaningful to a wider range of potential users.

4.2 HARDWARE ENVIRONMENT

The A^{3}I Program has adopted the requirement to use existing and proven hardware, namely networked Silicon Graphics Workstations and Symbolics Lisp machines. The hardware architecture in place at the end of Phase IV is depicted in Figure 3 below. These components, together with their resident software and peripherals are described in further detail in the subsections which follow. The specific computational hardware environment for each module is described in detail within Annexes A-J. The history of changes in the hardware environment throughout the development phases is described in Section 9, Historical Information.

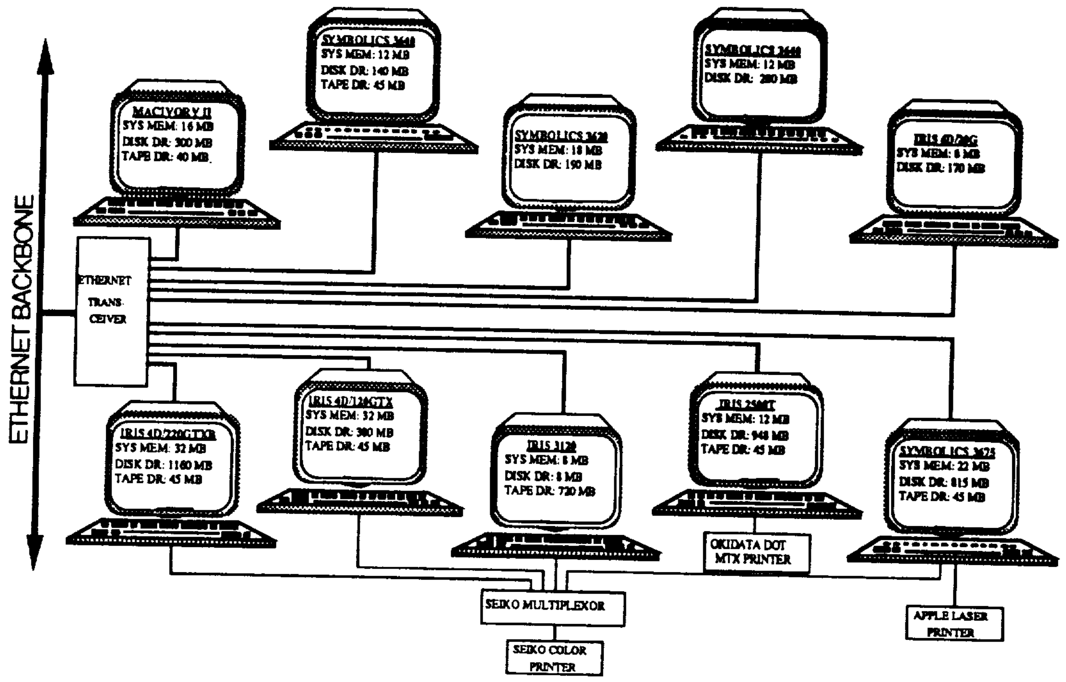


Figure 3. Phase IV Hardware Configuration

4.2.1 Symbolics Lisp Machines

Model 3675 Color Workstation (Barracuda) consisting of:

Monochrome Console with OCLI filter
Keyboard & Mouse
45 MB 1/4" Cartridge Tape Drive
Ethernet Controller and Transceiver
22.5 MB RAM
Enhanced Performance Option
338 MB Fujitsu Eagle Disk
550 MB CDC Disk
Model CG70-FB02 High Resolution, 24-bits/Pixel Color Frame Buffer
Tektronix 19" Color RGB Monitor
Model OP36-FPA1 Floating Point Accelerator
Symbolics # SLAN-FORT Fortran 77 Compiler
Symbolics # STCP-1 TCP/IP Software
S-Group (S-Paint, S-Geometry, S-Render, S-Dynamics, and color 6.0 V405.13)
Genera 7.2

Model 3640 Color Workstation (Puffer) consisting of:

Monochrome Console with OCLI Filter
Keyboard & Mouse
45 MB 1/4" Cartridge Tape Drive
Ethernet Controller and Transceiver
11.25 MB RAM
2-140 MB Disks
CAD Buffer
Tektronix 19" Color RGB Monitor

Symbolics # SLAN-FORT Fortran 77 Compiler
Symbolics # STCP-1 TCP/IP Software
S-Group (S-Paint, S-Geometry, S-Render, S-Dynamics, and color 6.0 V405.13)
Genera 7.2

Model 3640 Monochrome Workstation (Squid) consisting of:

Monochrome Console with OCLI Filter
Keyboard & Mouse
Ethernet Controller and Transceiver
13.5 MB RAM
2-140 MB Disks
Symbolics # SLAN-FORT Fortran 77 Compiler
Symbolics # STCP-1 TCP/IP Software
Genera 7.2
Automated Reasoning Tool (ART) Version 3.2

Model 3620 Monochrome Workstation (Sea Slug) consisting of:

Monochrome Console with OCLI Filter
Keyboard & Mouse
Ethernet Controller and Transceiver
18 MB RAM
190 MB ST506 Disk
Symbolics # SLAN-FORT Fortran 77 Compiler
Symbolics # STCP-1 TCP/IP Software
Genera 7.2

4.2.2 MacIvory Workstation

Apple Macintosh IIx Workstation with Symbolics MacIvory board (Otter) consisting of:

19" Radius Monochrome Monitor
Keyboard & Mouse
Ethernet Interface card
21 MB RAM (5MB SIMM CPU memory + 16MB Nubus memory card)
300 MB External Disk
Internal 3.5" floppy disk drive
40 MB mini cartridge tape drive
MacIvory floating point accelerator
MacIvory 7.4
Macintosh system software version 6.0.2
Microsoft Word 4.0
MacDraw II 1.1

4.2.3 Silicon Graphics Computers

W-2500A Workstation (Orca) consisting of:

19" High Resolution Monitor
Keyboard & Mouse
45 MB 1/4" Cartridge Tape Drive
Ethernet Controller and Transceiver
12 MB RAM
2 474 MB Fujitsu 10.5" Disk Drives
HU-T04 Turbo Option W/4MB RAM
H3-FPA Floating Point Accelerator
H-DM4A 1024x1024x4 Display Memory
H-ZC2 Z Clipping Assy
C-WTCP IP/TCP Software
P-DBX Dial/Button Box
Unix System V with BSD 4.2

NFS
C Compiler
IRIS Graphics Library II and Window Manager

W-3120 Workstation (Manta) consisting of:

19" High Resolution Monitor
Keyboard & Mouse
Ethernet Controller and Transceiver
8 MB RAM
72 MB Winchester Disk Drive
H3-FPA Floating Point Accelerator
H-DM4A 1024x1024x4 Display Memory
H-ZC2 Z Clipping Assy
C-WTCP IP/TCP Software
P-DBX Dial/Button Box
Unix System V with BSD 4.2
NFS
C Compiler
IRIS Graphics Library II and Window Manager

W-4D120GTX PowerSeries Workstation (Coral) consisting of:

19" High Resolution Monitor
Keyboard & Mouse
Ethernet Controller and Transceiver
32 MB RAM
380 MB ESDI Winchester Disk Drive
Double Buffered 1280x1024x4 Display Memory
Double Buffered Alpha
24 bit Z buffer
C-WTCP IP/TCP Software
P-DBX Dial/Button Box
IRIX System V release 4D1-3.1D
NFS
C Compiler
C++ Translator
Fortran 77 Compiler
IRIS Graphics Library II and 4Sight Windowing System

W-4D220GTX PowerSeries Workstation (Starfish) consisting of:

19" High Resolution Monitor
Keyboard & Mouse
Ethernet Controller and Transceiver
32 MB RAM
2 780 MB ESDI Winchester Disk Drives
Double Buffered 1280x1024x4 Display Memory
24 bit Z buffer
C-WTCP IP/TCP Software
IRIX System V release 4D1-3.1D
NFS
C Compiler
C++ Translator
Fortran 77 Compiler
IRIS Graphics Library II and 4Sight Windowing System

W-4D20G Personal IRIS Workstation (Urchin) consisting of:

19" High Resolution Monitor
Keyboard & Mouse
Ethernet Controller and Transceiver
8 MB RAM
170 MB SCSI Winchester Disk Drive

1280x1024x4 Display Memory
Double Buffered Alpha
C-WTCP IP/TCP Software
IRIX System V release 4D1-3.1D
NFS
C Compiler
IRIS Graphics Library II, 4Sight Windowing System, and Environment Manager

4.2.4 Networking Hardware

CableTron MT-800 Ethernet/IEEE 802.3 Transceiver

4.2.5 Peripherals

Okidata Model 2410 Dot Matrix Printer
Apple LaserWriter Plus Laser Printer
Seiko Instruments D-Scan CH5312 Color Printer & Multiplexor
GraphOn GQ-250 ASCII Terminal & Keyboard (2)
Hewlett-Packard HP 700/22 ASCII Terminal & Keyboard (3)

4.3 SOFTWARE ENVIRONMENT

The A^{3}I Program office made a decision to standardize on software, such as system software and languages, rather than on hardware. This factor, coupled with the requirements and technical approach appropriate to the Program, led to the selection of the following software:

	Symbolic Models/Tools	Numeric & Graphic Models/Tools
Operating Systems	Genera	UNIX
Languages	Common Lisp	FORTRAN, C
Object-oriented methods	Flavors	C++
Machine Class	Symbolics	Silicon Graphics IRIS

In addition to the software/OS packages described above, two key software development approaches support the A^{3}I Program goal of providing a flexible environment in which various models of system and human performance can be integrated.

4.3.1 Object-Oriented Programming

The A^{3}I Program has adopted the use of object-oriented programming methods, where practical, in an effort to manage the complexity of the simulation software through modularity and abstraction, as well as to promote graceful incremental software development. While not universally applied to every component, the object-oriented paradigm is a natural match to the structure of the man-machine integration problems investigated. Object-oriented techniques support extensive reuse of software structures and are an evolving standard in the software development community. Additionally, use of object-oriented techniques may reduce the ultimate code bulk by making extensive reuse of software structures and can be written to be virtually self-documenting.

4.3.2 Source Code Control

Software for which the A^{3}I Program does not have access to source code (with modification rights) is generally not permitted in the configuration. Encumbrances to success, since technology transfer is a primary concern. However, to date, some exceptions have been allowed for packages with clearly defined and easily accessible software interfaces that would be inappropriate for the Program to develop, or software development utilities (shells, debuggers, process performance metering, function libraries, etc) that offer superior performance which other organizations have ready access to. Exceptions to the desire to avoid encumbrances are two software packages used in Phase IV, MultiGen™, a commercial 3D modeling package with a hefty license fee and the Generic Expert System Tool, (GEST), a package with a moderate but not inconsequential license fee produced by Georgia Technical Research Institute (GTRI).

Generally, the graphic design and analysis tools were built using C, Unix System V with BSD 4.3 extension, and the IRIS Window Manager/Graphics Library II. The MultiGen™ modeling package was used as the underpinning for the CDE and VEST components, as well as a visualization medium for the Aerodynamics & Guidance Module. Most Lisp tools and models were built using Symbolics Common Lisp, with the Flavors extension, under Genera 7.2. The S-Packages were available for use in displays, although not heavily relied upon during this phase. Communication software used the TCP/IP protocols to communicate between the Silicon Graphics and Symbolics workstations over Ethernet.

Two expert system shells were introduced in Phase IV to facilitate development of knowledge-based approaches in certain modules. The Generic Expert System Tool, (GEST), produced by Georgia Technical Research Institute (GTRI), was used as the basis for the Scheduler. This shell includes a blackboard architecture, which facilitated development of this module. The C Language Information Processing System (CLIPS), developed by NASA-Johnson Space Center, was used for the rule-based portion of the Display Layout Analysis tool. Source code is available for both of these shells and both are available at relatively modest cost to users of MIDAS, although GEST is more expensive than CLIPS.

The distribution of the Phase IV models, tools, and displays across the available workstations is shown in Figure 4 below.

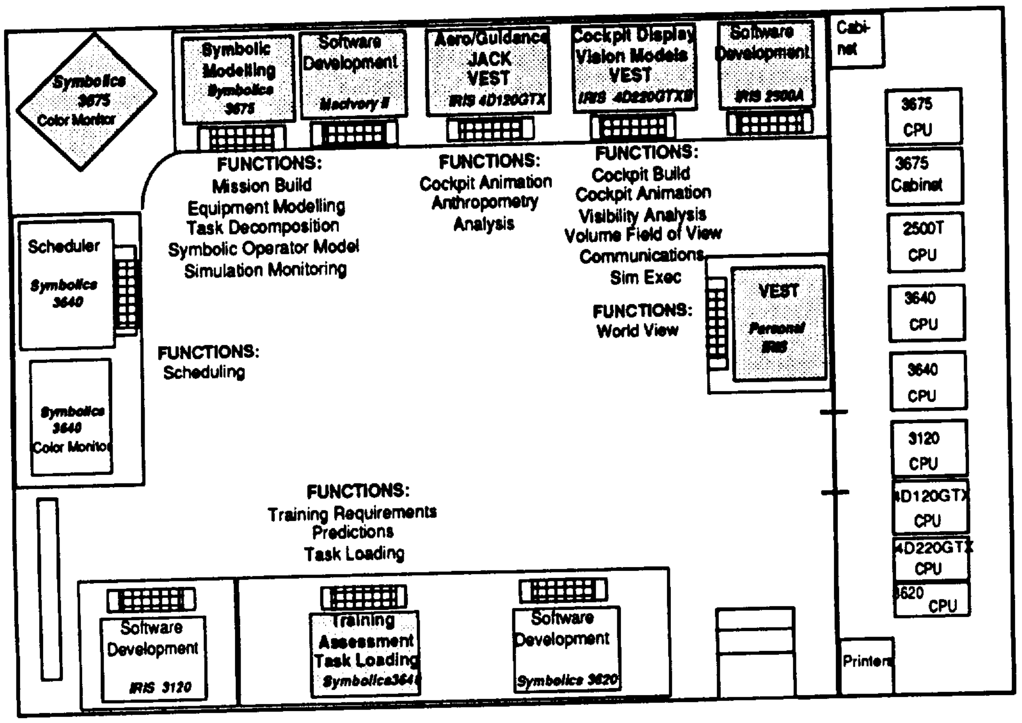


Figure 4. Distribution of Phase IV Software Components and Displays within the A^{3}I Lab

4.4 EXTERNAL INTERFACE REQUIREMENTS

4.4.1 User Interfaces

The user's interface to the computational tools and models of MIDAS is extremely important. However, since the Program's charter is to develop prototype facilities in an architecture that is continually evolving and may or may not be common to prospective delivery platforms, a polished, consistent interface across all of the applications has not been developed or considered a priority in Phase IV. As an interim solution, each application developer was encouraged to use pop-up windows, pull-down menus, and the manipulation of graphic or iconic representations as the principle mechanisms of user interactions.

A high priority in Phase V will be to design a unified, principle-based approach to the implementation of user interfaces that is based on a thorough study of user requirements as well as human-computer interactions (HCI) research (previous and ongoing) at various universities and industry centers.

4.4.2 Integration

A considerable degree of integration among the various modules of MIDAS was achieved in Phase IV. The data flow within MIDAS during Phase IV is shown in detail using an N^2 format in Figure 5. The components involved are portrayed in the shaded boxes located on the diagonal of the figure. As shown in the legend, inputs are on the vertical axes of each box and outputs on the horizontal axes.

MISSION EDITOR	Mission Reqmnts (Goal Form)		Operator Activites (Hierarchical & Evt. Resp.)			Waypoints with desired x, y, z ; Hdg, Alt, Airspeed			
Component Functions (State Achievers)	SYMBOLIC EQUIP. MODELS	Done							Control & Display States
	Tick	SIM. EXEC	Tick			Tick	Tick	Tick	Tick
		Done	SYMBOLIC OPERATOR MODEL	Task List, Constraints, VACP, Horizon	Attributes of Operator, World, Task, & Equip.	Accept/ Reject Controls		Reach-for & Look-at Site Cmds	
			New Task Sequence, & Loads	SCHEDULR (Z)	Tasks, Base VACM Loads				
				Scaled VACMs for Task Combos.	TASK LOADING MODEL				
		Done	Demands for Control Movement			VEHICLE GUIDANCE MODEL	Control Positions		
		Done				Position, orientation, speed	VEHICLE DYNAMICS MODEL		Hdg, Alt, Misc A/C Parameters
	Switch Movements	Done	Hand / Head Position, Reach Status					ANTHRO. MODEL JACK	Switch Movements Body Position
	Location/ Name of C&Ds	Done							VEST

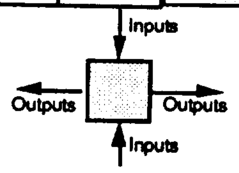


Figure 5. MIDAS Phase IV Integration N² Chart

For example, during each simulation "tick", the Vehicle Guidance Model computes the demands for control movement based on the current vehicle position, orientation, and speed (from the Dynamics Model) together with the next desired waypoint location, airspeed, and altitude (from the Mission Editor). These control requirements are then passed to the Symbolic Operator Model, which either accepts or rejects the controls based on other task demands. If accepted, the Guidance Model's computed controls are input to the Vehicles Dynamics Model and a new aircraft position and attitude are determined.

Integration between components is achieved in part by the sharing of state variables through a "data pool" which exists as part of the Communications Module. In all, there are a few hundred variables in the data pool. The state variables shared among the Phase IV MIDAS modules are logically grouped based on the functions they serve. Listed below are the most frequently referenced groups together with their members.

Jack reach command:

> Type of reach activity,
> Reach site name.

Position and availability of pilot's left and right hands and head orientation:

> Left hand position (x, y, z),
> A flag indicating the availability of the left hand,
> Right hand position (x, y, z),
> A flag indicating the availability of the right hand,
> Head orientation (yaw, pitch).

Ownship's position, orientation, and important aerodynamic parameters:

> Position (x, y, z),
> Orientation (yaw, pitch, roll),
> Altitude above ground level,
> Z component of velocity,
> Airspeed,
> Heading,
> Turnrate.

Convoy vehicle's position and orientation:

> Position (x, y, z),
> Orientation (yaw, pitch, roll).

Waypoint:

> Waypoint ID,
> Position (x, y, z).

Multi-Function Display navigation page:

> Format (Centered or Decentered),
> Scale.

Multi-Function Display aircraft page:

> Torque,
> Specific fuel range,
> Endurance,
> Aft tank fuel quantity,

Forward tank fuel quantity,
Total fuel quantity,
Engine 1 fuel flow,
Engine 2 fuel flow,
Total fuel flow.

Multi-Function Display communication page:

Challenge code,
Reply code,
Frequency Display Buffer.

4.5 REQUIREMENTS SPECIFICATION

4.5.1 Process and Data Requirements

The thrust of the development work in Phase IV was to improve the breadth and depth of existing modules, develop and integrate new components and achieve a higher degree of integration among the various components to implement a dynamic simulation of a mission scenario.

By the end of Phase IV, the requirements for MIDAS had resulted in development, both in-house and through grant/contract, of the following software components, hosted on networked Silicon Graphics IRIS and Symbolics computers:

1) The Symbolic Modeling component, written in Symbolics Common Lisp, contains methods to represent and decompose mission goals to their lowest level, match them to equipment-dictated activities, interact with the scheduling and task loading components, and execute activities subject to resource constraints.

2) The Scheduler makes use of the blackboard architecture available in the Generic Expert System Tool (GEST) from GTRI, to implement a constraint-based scheduler using two alternate scheduling strategies, time minimization and load balancing.

3) The Task Loading Model, written in Common Lisp, classifies tasks according to their demands on the Visual, Auditory, Cognitive and Motor processing dimensions and passes those values of operator loading to the Symbolic Operator Model to establish resource constraints and to the scheduler for use in scheduling tasks.

4) The Symbolic Equipment Model, written in Symbolics Common Lisp, models the cockpit equipment as finite state machines to represent the detailed physical and functional attributes and to specify the operation of equipment during the simulation.

5) The Visual Editor and Simulation Tool (VEST), written in C as an extension to the commercial modeling package, MultiGen™, provides an interactive 3-D tool used to create, control and observe, from several perspectives, the 3-D graphic environment of the mission simulation. The Cockpit Design Editor (CDE), a subsystem of VEST, contains 3-D modeling utilities for prototyping the cockpit geometry, instruments, controls, and displays. Links can be made to other models or data files for animation of selected controls & displays.

6) The Display Layout Analysis (DLA) tool, written in C and also using the Expert System shell CLIPS, developed by NASA, is a prototypical tool intended to provide designers with assistance in determining the spatial configuration of cockpit displays.

7) The Anthropometric Model (Jack), is a 3-D, interactive, anthropometric human model or graphic mannequin. This component is written in C and developed by Dr Norm Badler at the University of Pennsylvania.

8) The Vision Models, written in C and fully integrated into the Jack environment, include the Volume Field-of-View Model, developed by Dr. Aries Arditi at The Lighthouse in New York, and the Cockpit Display Visibility Model, developed by the Drs. J. Bergin and J. Lubin of the SRI/David Sarnoff Research Center.

9) The Aerodynamics & Guidance Model (AGM), written in Fortran, was developed by Dr Anil Phatak of Analytical Mechanics Associates and modified in-house. It contains rather generic equations of motion for a helicopter, providing pitch, roll, yaw, and translation in each axis as a function of decoupled cyclic, collective, and pedal movements. Guidance routines provide outer-loop speed and horizontal/vertical path control feedback as a simple representation of piloting activities.

10) The Communications Module, written in both C and Lisp, provides the data pool through which the simulation components share state variables as well as driving the tick-based simulation by sending ticks to the other modules.

11) The Training Assessment module, written in Symbolics Common Lisp and Automated Reasoning Tool (ART) code, provides heuristic methods to estimate the training media, instructional techniques, and time necessary to train various operators to "initial qualification" based on individual task characteristics. This component was developed in Phase III and not modified during Phase IV; detailed documentation of this module is included in the Phase III design document.

5.0 DESIGN

5.1 ARCHITECTURAL DESIGN

Figure 6 below shows the components of MIDAS at the end of Phase IV and how they relate to one another. The shaded boxes are components of MIDAS and the rounded boxes show the nature of the information flowing between the modules. The Mission Editor and Task Representation modules are essentially components of the Symbolic Operator Model at the present time. Jack is integrated with VEST and the cockpit environment created with the CDE to represent the pilot and CPG in their cockpit with the DMA terrain visible through the windscreen. The Vision Models are not integrated with the simulation, nor is the Display Layout Analysis tool or the Training Assessment Module. The Scheduler and the Task Loading Model are not actually integrated with the simulation, although the communication protocols have been established. The Symbolic Operator Model bypassed the functions that the Task Loading Model and Scheduler were designed to perform during the simulation.

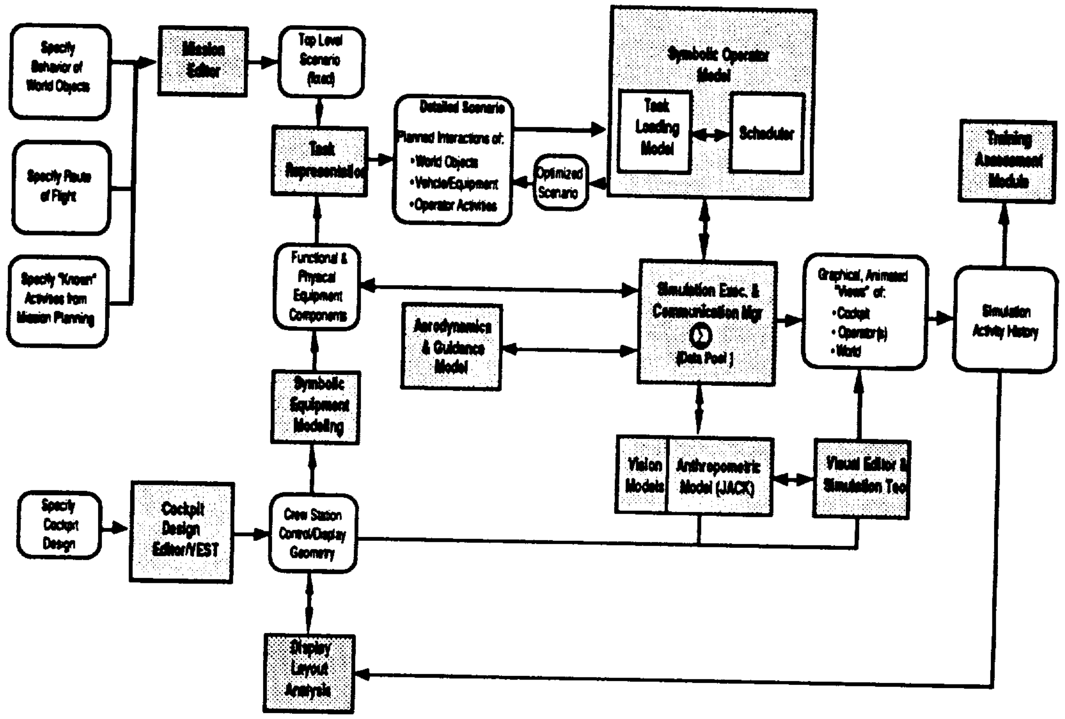

Specify Behavior of World Objects
Specify Route of Flight
Specify "Known" Activities from Mission Planning
Mission Editor
Top Level Scenario (fixed)
Task Representation
Detailed Scenario
Planned Interactions of:
•World Objects
•Vehicle/Equipment
•Operator Activities
Optimized Scenario
Symbolic Operator Model
Task Loading Model
Scheduler
Training Assessment Module
Functional & Physical Equipment Components
Aerodynamics & Guidance Model
Simulation Exec. & Communication Mgr
(Data Pool)
Graphical, Animated "Views" of:
•Cockpit
•Operator(s)
•World
Simulation Activity History
Symbolic Equipment Modeling
Vision Models
Anthropometric Model (JACK)
Visual Editor & Simulation Tool
Specify Cockpit Design
Cockpit Design Editor/VEST
Crew Station Control/Display Geometry
Display Layout Analysis

Figure 6. MIDAS Phase IV Top Level Software Architecture

5.1.1 Phase IV Development Summary

Responses to the Phase III demonstrations, as well as discussion at the Phase IV off-site reinforced the need not only to continue the development of the core set of A^{3}I models and tools but to fully integrate them into a designers workstation. Also, emphasis would have to be placed on explicitly addressing how such models and tools would be sensitive to cockpit design change. In particular, there was interest in determining the potential ramifications of introducing the "glass cockpit" concept, that is the use of Multi-Function Displays (MFD) in place of dedicated displays in the cockpit. Accordingly, the focus of the phase was to use a portion of a typical mission segment as the setting for a comparison of the Apache AH-64A cockpit, with dedicated displays, and the new Apache Longbow model, in which many functions currently performed with dedicated equipment are incorporated into Multi-Function Displays (MFD). The intention was to show how MIDAS could be used to compare operator performance in the same mission scenario with two different cockpit designs. These objectives required a degree of integration and detail previously impossible, and drove the design requirements for the individual MIDAS components. The major components of MIDAS in Phase IV are described briefly below. Further detail may be obtained from Annexes A through J which contain more detailed documentation for individual components.

5.1.1.1 Symbolic Operator Model

This model, coded in Symbolics Common Lisp, contains the data structures and methods used to represent and decompose the required mission, environment and, human performance models of the crew. This component expresses mission activities in terms of goals or states to be achieved, allowing the user to explicitly allocate such tasks to equipment or human operators, as well as providing for event-related operator responses which cannot cleanly be represented in an hierarchical fashion. The focus for the Symbolic Modeling component during this phase was on the design and coding of a generalizable framework for symbolically representing the functions of cockpit equipment used to accomplish mission tasks. This framework allows various cockpit alternatives to be evaluated without completely re-editing the mission decomposition, since the design maintains a distinction between the physical structure (or state operators) of the equipment, and the functional requirements (or inferred goals) required by the task. Previous A^3I symbolic models of the mission and pilot tasks failed to explicitly depict the relationship of the equipment design and the primitive task actions, loading values, or timelines.

This component is understandably one of the most complex and continually evolving. It currently contains two major subcomponents, the scheduling and loading models described below. During a simulation, this model attempts to execute assigned mission activities subject to specified constraints, state variables, and other simulation object requirements. This model accomplishes this action by: 1) updating the simulated operator's goal list to delete terminated or inappropriate goals, 2) examining equipment and world state variables to determine if event-response activities are required, 3) tracing the decomposition of mission goals to their lowest level, finding matching equipment operation patterns or activities which will satisfy them, 4) sorting these matched goal-activity patterns by priority, 5) interacting with the scheduling and loading operator model components as appropriate (although during the demonstrations, this interaction was not performed) and 6) executing these activities subject to physical resource (hand, eye, etc) requirements, Visual Auditory, Cognitive, and Psychomotor (VACP) load limits, and temporal/logical constraints.

Mission, task, environment, or operator objects are instantiated when their conditions are met, executing their assigned procedures and spawning new activities. Contained within the various task objects is information on temporal relationships, preconditions, logical constraints, loading, subtasks, and relative priority. In this manner, the decomposed mission serves as a forcing function, "driving" the interaction of various models used during a simulation. Refer to Annex A for a more detailed description.

5.1.1.1.1 Scheduler (Z)

This constraint-based, opportunistic model of operator scheduling behavior was developed using the blackboard architecture provided as part of the Generic Expert System Tool (GEST). Display portions of this component are written in Symbolics Common Lisp. Provided a task queue of indeterminate length, along with data about each task (such as logical constraints, estimated duration, resource requirements, etc.), Z solves for a near-optimal sequence and schedule based on a strategy of either time minimizing or load balancing, intended to represent possible operator behaviors. The scheduler contains modular components or knowledge sources that represent individual stages in the scheduling process, with an extended task-based decomposition (a "divide-and-conquer" technique) used to partition the overall scheduling problem. Z closely interacts with the MIDAS task loading model for reasoning about resource interactions between plausible concurrent tasks. Refer to Annex B for a more detailed description.

5.1.1.1.2 Task Loading Model

This component, written in Symbolics Common Lisp, is based on current research in multiple resource theory, scaling, workload, and perception. Based on attributes of the mission tasks, world state, operator, and crew station equipment, a resource classification taxonomy is used to classify individual tasks in terms of their demands on the Visual, Auditory, Cognitive, and Motor processing dimensions. In addition, conflict matrices are used to describe the interactions of these resource demands across different processing dimensions and tasks. Refer to Annex C for a more detailed description.

5.1.1.2 Symbolic Equipment Models

These generalizable structures, written in Symbolics Common Lisp, allow characteristics of the crew station equipment to be represented in terms of both physical and functional attributes which are used to specialize the mission tasks prior to a simulation. In this manner, MIDAS can support the generation of explicit operator actions which are sensitive to specific equipment designs. These structures also provide a model-based means to maintain and manipulate equipment state variables which drive the animation of graphical cockpit controls and displays. Refer to Annex D for a more detailed description.

5.1.1.3 Visual Editor and Simulation Tool (VEST)

VEST is an interactive 3-D tool used to create, control, and observe from several visual perspectives, a 3-D graphic representation of vehicles traversing through DMA terrain during a simulation. Users can select by mouse a viewing position from anywhere within the mission gaming area, zoom in on specific controls and displays for study, as well as include a representation of the Jack Anthropometric model within the crew station for visualizing operator movement during a simulation. The Cockpit Design Editor (CDE), a subsystem of VEST, contains interactive 3-D modeling utilities for graphically prototyping the crew station geometry, instruments, controls, and displays using a built-in library of primitive cockpit objects. Links can be made to other models or data files for animation of selected controls & displays. A significant extension has recently been made to this component, allowing the creation and animation of multi-function displays (MFD) containing graphical features, text strings, and dynamic fields. This component is written in C as an extension to a commercial modeling package from Software Systems, Inc. called MultiGen™. Refer to Annex E for a more detailed description.

5.1.1.4 Display Layout Analysis (DLA) tool

The Display Layout Analysis (DLA) tool, is an initial prototype of a tool intended to assist a crewstation designer in laying out the displays which provide the pilot with windows to the information sources which he/she needs to successfully operate the aircraft. Display layout assistance is guided by a set of design metrics incorporated into the tool, which can be operated both in an analytic, design-aiding mode and in an evaluative mode. In the analytic mode, human factors design guidelines form networks of relations between information sources and other information sources, between information sources and controls, and between information sources and regions of the display surfaces. A rule-based advisor can be invoked which issues warnings for detected violations of display layout guidelines. This component is written in C, using the Expert System shell CLIPS for the rule-based advisor. Refer to Annex F for a more detailed description.

5.1.1.5 Anthropometric Model (Jack)

A 3-D, dynamic anthropometric model has been developed through a grant to Dr. Norman Badler at the University of Pennsylvania in order to address fundamental human anthropometry and motion considerations. This model, called Jack, is written in C and runs on the Silicon Graphics 4D series, providing realistic and physically quantifiable human figure motion within a 3-D space environment. Jack allows the user to select different sized human figures or graphic mannequins that include the 5th through 95th percentile male and female, based on NASA astronaut demographics. These mannequins can then be placed within a 3-D object environment created and stored using a number of modeling packages. Articulation is achieved using a goal-solving technique based on specifying body joint orientations or end-effector (limb) goals. Joint limitations have been installed to eliminate unreasonable movements. Kinematic and inverse kinematic controls are applied so that goals and constraints may be used to position and orient the figure, with external/internal forces and torques applied to produce motion. A movement time calculation has been incorporated based on Fitts Law, using reach site distance and target width.

Supporting graphic output in wireframe, solid filled, or smooth shaded modes, key poses can be stored and interpolated for animation, allowing environmental limitations to be detected as a function of human size and movement characteristics. In addition, by attaching the "view" of the environment to the mannequin's eye, Jack displays a perspective corresponding to what the mannequin would "see" while moving in the environment, providing the first step toward further analysis and conclusions about object occlusion and visibility. Refer to Annex G for a more detailed description.

5.1.1.6 Vision Models

Refer to Annex H for a more detailed description of the Vision Models.

5.1.1.6.1 Volume Field of View Model

This model of binocular human visual representation in 3-D space was developed by Dr. Aries Arditi at The Lighthouse of New York. It provides computer graphic methods for delineating and testing hypotheses about the relationship between two-dimensional visual field maps and the three-dimensional visual space they serve, under the conditions of: 1) changing eye position; 2) occlusion by structures that are part of or are mounted on the observer such as facial structures, goggles, or headgear; 3) occlusion by environmental objects; 4) normal and abnormal defects of the visual field such as blind spots and areas of temporarily reduced visibility due to local adaptation and photopigment bleaching; and 5) variables that alter the focus of environmental objects on the retinas (accommodation and pupillary response). Instantaneous field of view volumes based on these factors are visualized by projecting their intersection with the object space in different colors. This model, written in C, is fully integrated with the Jack anthropometric model, which is used to determine the operator's head position and point of regard in the field of view.

5.1.1.6.2 Cockpit Display Visibility Model

This analytical model, written in C and also fully integrated into the Jack environment, was developed by Drs. Jim Bergin and Jeff Lubin at the SRI/David Sarnoff Research Center. It allows the designer to assess the visibility of cockpit objects imaged on the retina in terms of a visual system footprint. This footprint represents the projection onto the crew station of the sensory capabilities of the human visual system when considered as a detector/filter system. The existing MIDAS graphical, anthropometric, and vision modeling capabilities are used to describe the physical characteristics of potential designs and define the instantaneous volume field of view. Based on such information, this component provides methods to project the retinal photoreceptor apertures onto the cockpit model and support empirically-based predictions about the legibility of characters and symbols. Because the human retina is highly inhomogeneous, the retinal footprint produced is also highly inhomogeneous, depicting contours of visual performance data which describe the probability that certain imaged information will or will not be legible. Because factors such as ambient illumination in the cockpit, the adaptive state of the operator, and the reflective /emissive properties of displays are critical to consider in such contexts, this model addresses each of these aspects.

5.1.1.7 Aerodynamics & Guidance Model (AGM)

The MIDAS AGM is a two-part Fortran model, initially developed by Analytical Mechanics Associates Inc., which represents rather generic helicopter guidance and dynamics for uncoupled controls. Given the current position, orientation, and angular rates, the guidance portion of the model determines the control inputs required to fly to the next waypoint with its associated position, altitude, and airspeed. The aerodynamics portion of the model uses the computed controls to determine the helicopter's next position, orientation, etc., based on the simulation tick interval. The AGM's input and output is integrated with the symbolic pilot model and anthropometric model such that during a simulation, the computed flight control requirements are passed to the symbolic pilot model as resource demands, with their actual start times and duration determined by the evaluation of such demands in the context of other pilot psychomotor activities. Flight control movements are graphically depicted by attaching the Jack anthropometric model's end effectors to the appropriate controls and using inverse kinematics to "pull" the appendages to the computed control positions. Refer to Annex I for a more detailed description.

5.1.1.8 Simulation Executive and Communications Module

This component, written in both C and Lisp, uses TCP/IP protocol to facilitate inter-machine communication and dynamic message sharing between all MIDAS components. A "data pool" concept is used during the simulation to synchronize and distribute in excess of 200 operator, world, and equipment state variables among the simulation processes and objects. Refer to Annex J for a more detailed description.

5.1.1.9 Training Assessment Module

The training assessment module provides heuristic methods to estimate the media, instructional techniques, and time necessary to train various operators to "initial qualification" based on characteristics of the operator, task, and crew station equipment. This prototype knowledge-based system is implemented in ART™ (the Automated Reasoning Tool) and Common Lisp on the Symbolics. This tool uses the instructional systems design (ISD) methodology to assign each task a set of learning experiences (such as explanation, demonstration, part-task training, and full task training) along with a medium for each learning experience (such as textbook/workbook, slide/tape, lecture, videodisc, and a wide range of simulation devices). For each learning experience and media assignment, a time to train is computed, based on the task, operator, and equipment attributes. This module was developed in Phase III; detailed documentation is included in the Phase III Detailed Design Document.

5.1.2 Demonstration Scenario

The Phase IV demonstrations consisted of a 15 minute introductory briefing by the program manager, followed by a little more than two hours of demonstrations by the development staff. The demonstration objective was to describe and demonstrate individual components of MIDAS as well as to demonstrate the integrated mission simulation capability. The demonstrations made use of data for controls, displays, mission profiles, and operator task descriptions obtained from the on-going McDonnell Douglas AH-64 Apache Longbow Program.

Beginning with the CDE, the procedures to build and animate a set of displays/controls on the cockpit panel were shown. A detailed demonstration of the new tools for representing and animating multi-function displays was also shown. Then, still in design/specification mode, the Display Layout Analysis tool was demonstrated. There are two modes in the DLA tool; one is the network tension mode where the relationships between displays and other elements in the environment are visualized as springs of various tension. In the other mode, a rule-based system evaluates a given layout and gives advice and warnings about possible violations of human factors principles.

Some new capabilities of Jack were then demonstrated, including lifts and balanced reaches. A sequence was shown in which Jack, sitting in the Apache Longbow cockpit, reached for the left multi-function display with his right hand, showing how Jack could be used to determine feasibility of reaches. The interactive user interface of SAS, a spreadsheet containing anthropometric data, was also demonstrated.

The Vision Models were then demonstrated in the Jack environment. The volume field of view model was shown with Jack sitting in a simplified Longbow cockpit. The ability to change the view point was demonstrated, as well as the ability to project retinal images onto the 3D world. The visibility model was demonstrated using data for the "O" and "Q" characters from the Longbow multi-function display. Probability of correct discrimination between the two characters was projected out onto the cockpit display panel under varying conditions of ambient lighting and multi-function display luminance.

Next, the mission editing component of the Symbolic Modeling component was introduced, showing the terrain map and the ability to enter a route of flight and mission requirements for the mission. The mission to be simulated was an enroute segment with some required radio calls. A briefing was given describing the goal structure and the process of matching a goal to the equipment-dependent operator activities that could satisfy that goal. The point was made that the goal need not change if different equipment were placed in the cockpit, but that different equipment would dictate different activities. The Equipment Models were also described in a briefing, with emphasis on the distinction between physical and functional attributes. Then the goal processing procedures were described, which include 1) updating the simulated operator's goal list to delete terminated or inappropriate goals, 2) examining equipment and world state variables to determine if event-response activities are required, 3) tracing the decomposition of mission goals to their lowest level, finding matching equipment operation patterns or activities which will satisfy them, 4) sorting these matched goal-activity patterns by priority, 5) interacting with the scheduling and loading model components as appropriate (although this was not done during the demonstrated scenario), and 6) executing these activities subject to physical resource (hand, eye, etc) requirements, Visual Auditory, Cognitive, and Psychomotor (VACP) load limits, and temporal/logical constraints.

The Scheduler was then demonstrated, using a set of tasks for starting an APU. A comparison was shown between the two scheduling strategies, time minimization and load balancing. The Task Loading Model was described and a brief demonstration of the computation of load in the four dimensions, visual, auditory, cognitive and motor, for a set of tasks was shown.

A briefing was given explaining how the Simulation Executive and Communications module worked, with one Symbolics machine and three Silicon Graphics IRIS machines networked together during the integrated dynamic simulation run.

The demonstrations were concluded by demonstrating the fully integrated dynamic mission simulation capability of MIDAS. The helicopter, with fully populated cockpits and models of the pilot and copilot/gunner (CPG), was placed in the gaming area, driven by the aerodynamics and guidance model, and viewed from three perspectives, a world view showing the helicopter moving over the DMA terrain, a view of the pilot cockpit from over his shoulder and a view of the CPG cockpit from over his shoulder.

The program goal for Phase IV had been to be able to demonstrate the simulation of a given mission scenario in two cockpit environments, the Apache AH-64A and the Apache Longbow, and to obtain comparative data across the two designs given the same mission goals. In the event, however, Phase IV demonstrations included only the Longbow cockpit configuration. The simulated mission ran at approximately ten times real time, necessitating selection of only a limited segment of the simulated mission scenario for the demonstration. A short segment of the scenario was shown, in which the CPG selected a radio frequency by manipulating the multi-function display pages.

5.1.3 Programmatic Information

5.1.3.1 Constraints

Following the Phase IV off-site at Asilomar Conference Grounds July 26-28, 1989, the group began design and development work for Phase IV. Near the end of the Phase IV development period, a junior C programmer, Dr. Christian Neukom, was hired to become the A^3I in-house Jack expert. Just before the demonstration period, Dr. Betsy Constantine was hired as Task Manager for the Sterling staff. By the end of Phase IV, the in-house staff consisted of 3 Lisp programmers , 4 graphics programmers, a human factors specialist / junior Lisp programmer, a Task Manager with a Lisp background, a system administrator and a CAD draftsperson.

In the computer lab, two machines were acquired. A new Silicon Graphics PowerSeries 4D220GTX, containing parallel graphics and central processors added significant new graphics capability, and a Macintosh II with a Symbolics MacIvory board was purchased to enhance the Lisp programming capabilities.

Funding for the Lighthouse of New York and the SRI/David Sarnoff Research Lab was continued for applied vision models. These components have proven to be extremely valuable models for MIDAS, eliciting considerable interest at the demonstrations.

The new organization plan for A^3I, which established two principal scientist positions and a new technology transfer group called the Industry Liaison Section has been partially implemented. The group is fortunate to have been able to fill the Principal Scientist for Cognition position by hiring Dr. Kevin Corker, a cognitive scientist with a long history of involvement with the A^3I program. He brings exciting new concepts in the simulation and cognitive modeling area to the program and will be directing the research and design efforts in Phase V. Dr. James Larimer is still acts in the role of Principal Scientist for Perception.

5.1.3.2 Risks

Selection of the appropriate level of detail at which MIDAS design and analysis tools are intended to operate remains a difficult issue. The A^3I Program Office has made it clear that MIDAS is intended for the conceptual development phase of crewstation design because of the high "payoff" for properly incorporating human engineering principles during this period. However, it is becoming increasingly clear that most of the human performance models and analysis methods currently or potentially incorporated in MIDAS require as inputs task, equipment, and environmental data which is more appropriate for detailed design. This apparent conflict between the model/analysis needs and the intended use of MIDAS is still unresolved. It may be true that even though MIDAS is intended to be used in the early conceptual stages of design, its use may require a degree of detailed analysis and specification that is not usually performed at the conceptual design stage. This need may require a change in the design process and could have serious implications for the Program's success in developing a prototype workstation which meets the needs of its projected users.

5.1.3.3 Summary of Results

There was a very positive response to the traditional end-of-phase demonstrations. Begun in June 1990, these demonstrations were attended by more than 200 people from NASA, the US Army, other DoD components, several universities as well as from industry, particularly the major helicopter manufacturers.

There was a great deal of interest in the Scheduler with its two scheduling strategies, both as evidence of real progress toward cognitive modeling and for its potential stand-alone usefulness. There were questions about whether the scheduling strategies that were implemented actually represented human operator scheduling behavior. This question will be addressed in Phase V when empirical data will be obtained in an attempt to validate the results obtained from the Scheduler.

The Task Loading Model was of considerable interest, particularly to industry visitors, who must produce loading estimates for their designs. Again, the biggest issue was whether the MIDAS Loading Model had been validated. In Phase V there will be a high priority placed on obtaining empirical data appropriate for use in validating the Task Loading Model.

The Display Layout Analysis tool was a big hit with many visitors, partly because it incorporates a very effective visualization technique that immediately connects with the observer and partly because it was quite clear just how a designer might use such a tool. It demonstrates direct aiding of designers as they solve a specific design problem and provides a clear means of evaluating a proposed display configuration according to established human factors principles.

The Vision Models elicited a great deal of interest from most visitors. Design questions about visibility are easy to understand and are key issues to be addressed during the design process. The Vision Models, integrated into the Jack environment, present attractive and useful visualizations of the kind of answers a designer might want from such a tool. These models were seen as useful interactive tools for a static analysis of visibility.

On the slightly negative side, the integrated simulation ran so slowly that it was difficult to imagine how a designer would interact with the dynamic analysis aspect of MIDAS. A decision was made not to artificially speed up the demonstration by running the graphics from files obtained earlier, because it was important to have visitors understand that true integration of many of the modules involved in the mission simulation had been achieved. The Scheduler and Task Loading Model, however, while integrated with each other, were not integrated with the Symbolic Operator Model during the actual demonstration, although the required communication protocols had been established.

A few important things were missing from MIDAS at the time of the demonstrations. There was virtually no integrated analysis capability, although the Symbolic Operator Model collected data during the simulated mission scenario that could have served as raw data for analysis. Also, the intention had been to demonstrate a comparison of two different cockpit designs, the Apache Longbow and AH-64A, with the same mission scenario. This comparison was not available and the simulated mission scenario was run with only one cockpit, the Longbow.

5.2 DETAILED DESIGN

Please refer to Annexes A-J for detailed design information for each component of MIDAS.

6.0 USER'S GUIDE

Please refer to Annexes A-J for User Guide information for individual modules.

7.0 ABBREVIATIONS AND ACRONYMS

A^3I	Army-NASA Aircrew/Aircraft Integration
AGM	Aerodynamics/Guidance Model
AMA	Analytical Mechanics Associates, Inc.
APU	Auxiliary Power Unit
ART	Automated Reasoning Tool
BBN	Bolt, Beranek and Newman Laboratories, Inc.
CAD	Computer-Aided Design
CBT	Computer-Based Training
CDE	Cockpit Design Editor
CFT	Cockpit Familiarization Trainer
CLIPS	C Language Information Processing System
CPG	Copilot/Gunner
CPS	Computer Program System
CPT	Cockpit Procedures Trainer
CSCI	Computer Software Configuration Item
DTED	Digital Terrain Elevation Data
DMA	Defense Mapping Agency
DOF	Degrees-of-Freedom
EES	Expert-EASE Systems, Inc.
HF/CAE	Human-Factors Computer-Aided Engineering
I/O	Input/Output
ISD	Instructional Systems Development
MFD	Multi-Function Display
MIDAS	Man-machine Integration Design & Analysis System
NFS	Network File Software
NRC CoHF	National Research Council Committee on Human Factors
OFT	Operational Flight Trainer
SCDD	Software Component Description Document
SGI	Silicon Graphics Inc.
TCP/IP	Transmission Control Protocol/Internet Protocol
USGS	United Stated Geological Survey
VEST	Visual Editor and Simulation Tool
WST	Weapon System Trainer

8.0 NOTES

8.1 LIMITATIONS

One critical limitation of Phase IV MIDAS is the lack of any kind of user interface for specifying the mission scenario with its associated goals and tasks. Also lacking was any non-programming means of specifying the physical and functional characteristics of the cockpit equipment. Anyone wishing to use MIDAS to do anything other than run the A^{3}I demonstration scenario with the Longbow cockpit design would have to encode the goals, tasks, equipment models and equipment-dependent activities in Common Lisp.

Another limitation in the usefulness of MIDAS to potential designer/users is the current dependence of the cockpit modeling and simulation graphics on the MultiGen™ package. The license fees for MultiGen™ are high enough to stand as a considerable barrier to the dissemination of MIDAS.

8.2 FUTURE DIRECTIONS

At the transition between Phase IV and Phase V, MIDAS is conceived as having two distinct modes, a dynamic simulation and analysis mode and an interactive static analysis mode. The main focus in the next phase of development will be on the dynamic simulation aspect of MIDAS and, in particular, on the development of cognitive models within the Symbolic Operator Model. Also, since it must be clear how the MIDAS workstation improves the present iterative, man-in-the-loop design process, analysis and evaluation of simulation results will be emphasized and more complete integration of all components of the simulation will be stressed. For the simulation, software will conform to a highly distributed, object-oriented architecture, sometimes called an agents or actors architecture. This architecture will enhance the ability of MIDAS to function as a framework for incorporating a variety of models developed elsewhere, for as long as a model conforms to the message-passing protocols of agents with which it must communicate, it can be introduced without disturbing the functioning of the remainder of the system.

At the end of Phase V, MIDAS is envisioned as running on two machines networked together, one Symbolics machine and one SGI IRIS machine. For the interactive tools, a common graphics environment, running on the SGI IRIS machine, will be chosen. This graphics environment must not have serious encumbrances, like the license fees for MultiGen™. Translators will be written to allow data files from certain CAD packages to be read into the MIDAS environment without being re-drawn. All interactive tools will operate in the chosen graphics environment, with a common user interface. A unified user interface for the Phase V MIDAS will be developed, based on X Windows running on both the Symbolics and IRIS machines.

Additional conceptual design work to be done on the user interface in Phase V will include a careful analysis of who the various kinds of users will be and the way they will want to interact with MIDAS. This study will help guide further development of MIDAS in subsequent phases as well as provide a principled specification of the future user interface.

During Phase V, a high priority will be given to establishing collaborations with organizations with access to empirical data suitable for use in validating some of the MIDAS models, particularly the Task Loading Model and the Scheduler. Attention will be given to validating other models as they are developed.

9.0 HISTORICAL INFORMATION

9.1 PHASE I DEVELOPMENT

9.1.1 Requirements and Design Approach

9.1.1.1 Summary Level

The initial phase of development of the prototype HF/CAE workstation found the Program in serious danger of extinction due to insufficient funding caused by regular and sizable cuts from guidance funding levels. It was believed that a visually-compelling, proof-of-concept demonstration was required to communicate the essence of the Program to individuals unfamiliar with the particulars of the science involved. Maximum visual utility was demanded of every expenditure and development.

A baseline simulation capability was required that demonstrated mission modelling, human performance metrics and helicopter-pilot interactions within a controllable, time-stepped environment that provided multiple graphic "views" into the underlying model(s). The simulation needed to be incrementally extensible, hence only a framework for more elaborate modelling was required for this phase, given the time constraints imposed. Several areas of development emphasis were identified:

9.1.1.2 Mission Modelling

The overall A^{3}I Program model architecture calls for a mission model driving the closed loop pilot-vehicle system. The model would be developed with a dynamic, interactive task analysis framework for systematically describing tasks involved in certain classes of advanced helicopter operations. The framework will provide a feasibility demonstration of the methodology, including all critical mission and flight management functions within a pre-specified sample scenario.

Typically, scout-attack helicopter missions are largely opportunistic or discretionary in nature. Consequently, the mission model generated by the mission decomposition methodology must allow this component to be represented, either by providing conditional branching based on some pilot model parameter, or stochastic event triggering.

Bolt, Beranek and Newman (BBN) had already started development (under a NASA contract initiated prior to the PDR) of a "Mission Decomposition Methodology" that would provide essentially the entire simulation structure for Phase I. Refer to the respective software component description document for the Phase I Mission Editor for details.

In order to drive and manipulate the specific mission model generated by the Mission Decomposition Methodology, a simulation executive was required that allowed greater user control of the simulation than conventional executive programs. Future integration of a vast number and variety of models within this executive structure was projected as well, hence flexibility as well as functionality was required. Refer to the software component description document for the Phase I Modeller for details.

Communications software was required to link the Symbolics 3670 running the mission model with the Silicon Graphics IRIS 2500T displaying dynamic, 3-D graphic views driven by the mission. The link was Ethernet under TCP/IP protocol. Refer to the software component description document for Phase I Communications for details.

9.1.1.3 Graphics

3-D, color, dynamic mission representation displays were required to provide intuitive understanding of simulation progress. Further, these so-called "views" became extremely valuable as debugging aids for programmers developing software on the system. Refer to the software component description document for the Phase I Graphic Views for details.

In addition to view graphics, a state display editor was required to allow designers to select appropriate model variables for run-time observation, and determine how and where the values were to be displayed. This tool allowed designers to individually select which simulation variables were of interest for monitoring, and the nature of their display (i.e. dial, bar, graph, etc.). Refer to the software component description document for the Phase I Icon Editor for details.

9.1.1.4 Human Performance Modelling

The first phase needed to demonstrate the capability to model, structure and analyze the human component of complex and interactive pilot-helicopter systems by elucidating the effects of human performance limitations on mission effectiveness. The mission had to be responsive to changes in pilot performance, and conversely, the pilot's loadings should be reflected in task loadings imposed by execution of the prescribed mission.

It was decided that emphasis should be placed on developing some meaningful demonstration of training effects as provided by profiles of novice and experienced pilot representations. Refer to the software component description document for Phase I Training Implications for details.

9.1.1.5 Demonstration Scenario

The demonstration scenario consisted of the capability to perform multiple consecutive simulations. Variables included:

1) A novice and experienced pilot profile that may be menu-selected prior to a simulation run to illustrate training effects.

2) Convoy return of missile fire at any point in the mission subject to the discretion of the designer.

It was possible to have extensive run-time control over the running of models from menu items. It was also possible to examine data and information both after the run, and during a model freeze state. Menu selection of these capabilities required no programming experience to start, operate and evaluate the simulation.

9.1.2 Hardware Environment

Figure 7 below indicates the Phase I hardware configuration. These components are described in further detail in the subsections which follow.

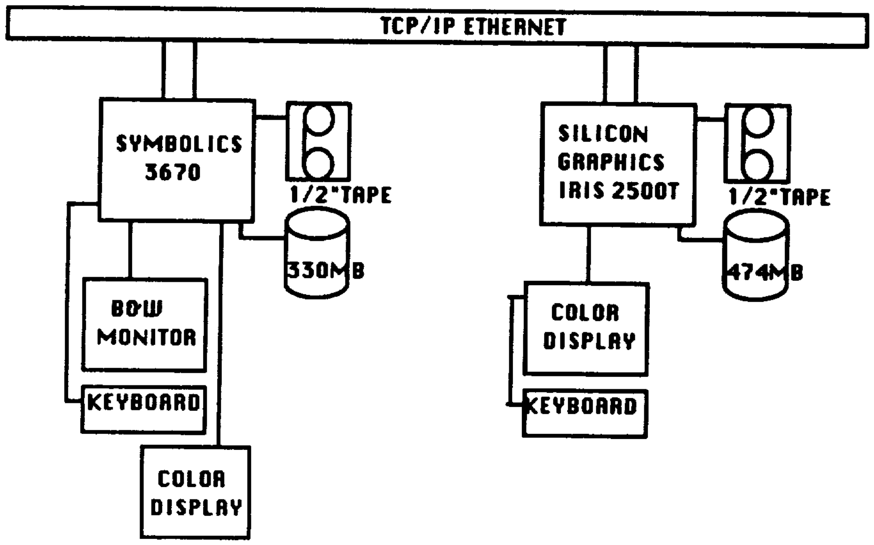


Figure 7. Phase I Hardware Configuration

9.1.2.1 Symbolics Lisp Machines

Model 3670-1433 Color Workstation consisting of:

8MB Main Memory
Ethernet Controller and Transceiver
335 MB Fixed Disk
Monochrome Console
Keyboard & Mouse
SYS36 20 MB System Software
Documentation
Model CG70-FB02 High Resolution, 24-bits/Pixel Color Frame Buffer
Model CGOP-O1L 19" Color RGB Monitor
Model OP36-FPA1 Floating Point Accelerator
Model CGSW-PKG Software Package consisting of:
SCGR-DYNA Dynamic Animation System
SCGR-PAINT Paint System
SCGR-GEOM Geometry System
SCGR-RENDER Rendering System
Symbolics # SLAN-FORT Fortran 77 Compiler
Symbolics # STCP-1 TCP/IP Software

9.1.1.2 Silicon Graphics Computers

W-2500A Workstation consisting of:

1/4" Tape Drive
HU-T04 Turbo Option W/4MB RAM
H3-FPA Floating Point Accelerator
H-DM4A 1024x1024x4 Display Memory
H-ZC2 Z Clipping Assy,
C-WTCP IP/TCP Software

P-DBX Dial/Button Box
R-UNIF Fortran Compiler

9.1.2.3 Other Processors

None.

9.1.2.4 Networking Hardware

TCL Incorporated Model 2010EC Ethernet Transceivers (tap-type)

9.1.2.5 Peripherals

Okidata Model 2410 Dot Matrix Printer

9.1.3 Software Environment

Generally, the graphic design and analysis tools were built using C, Unix, and the Replicore 3-D modelling package. Lisp-based tools and models were built using Symbolics Common Lisp under Genera 6.2, and the S-Packages were often used for Displays. Communication software was written to enable inter-machine message passing and simulation synchronization. The distribution of the Phase II models ,tools, and displays is shown in Figure 8 below.

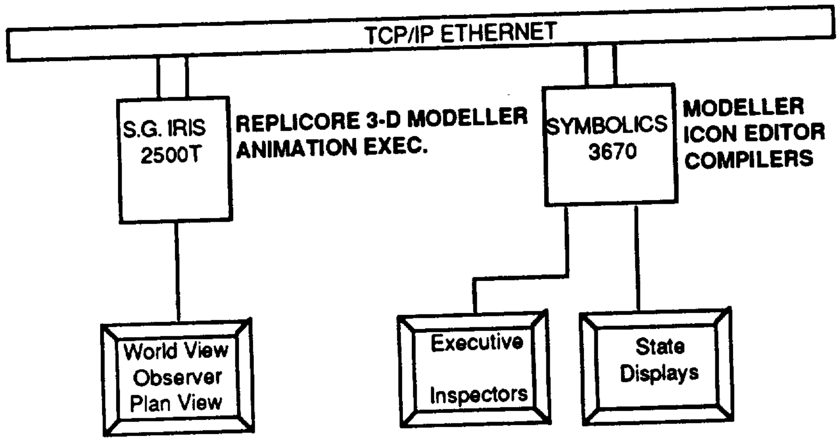


Figure 8. Phase I Software Modules

9.1.4 Programmatic Information

9.1.4.1 Constraints

The demonstrations had to be conducted prior to July, 1986, hence there was little time to develop the demonstration in time to impact the next year's fiscal funding. Technical planning began late in June, 1985. The Phase I Preliminary Design Review (PDR) was conducted on 25 October 1985, where general objectives were established.

Staffing initially included 1 EE (software integration task manager), and 1 systems programmer. A LISP programmer was added in October, 1985, followed by a senior graphics specialist is April, 1986 and an entry-level programmer in May, 1986. Training and familiarity was required for any programmer working with the Symbolic computer (often quoted to be a 6-month learning curve).

Due to the staffing limitations, subcontracts were required to perform some of the necessary work. In particular, 3-D graphic view work was initially subcontracted to a small business marketing a 3-D CAD modelling package. Delivery of the work was ultimately several months late, and incomplete, causing in-house programming staff to be severely pressed to integrate and debug other software prior to the established demonstration deadline. Future contractual endeavors cannot involve organizations that are at risk of being unresponsive, or that are developing components that may be considered critical path items.

The Program owned a Symbolics 3670 LISP computer with 24-bit color system at the time of the PDR. A procurement began shortly afterward to acquire a Silicon Graphics IRIS 2500T graphics computer, which arrived at Ames in January, but was not in working condition until February.

9.1.4.2 Risks

The majority of Phase I applications developments were under contract or subcontract. This condition poses some risk of failure caused by potential incompatibilities at the time of integration, competing priorities inherent within respective organizations, lack of control over developments and progress, and other problems that cannot be treated or corrected by the Program Office. While appropriate for this initial phase (due to constraints), each instance of reliance on organizations that the Program Office does not have direct control over should be carefully considered.

Since the Phase I architecture was minimal, there were few technical risks (outside of meeting deadlines) that might prohibit success. The major source of uncertainty involved networking the Symbolics and IRIS through TCP/IP Ethernet. Unfortunately, late delivery of graphics software forced a suboptimum (almost frenzied) approach to communications debugging, since in-house staff did not have the necessary familiarity with TCP/IP protocol details. Consultants were used to assist with debugging and optimizing the link.

9.1.4.3 Summary of Results

Preliminary demonstrations (of the mission model) were held in February 1986, followed by the first of several formal Phase I demonstrations starting in late June 1986. The demonstrations consisted of three computer displays (two in color):

1) Three-dimensional (3D), color, dynamic, mission representations composed of world, pilot and plan views of the simulated mission.

2) Mission model set-up, control and data display.

3) Color iconic "state displays" providing continuous display of mission model variable values.

The simulation executive (on the Symbolics 3670) controlling the mission model provided appropriate data via EtherNet TCP/IP to 3D graphic "views" resident on the IRIS 2500T to drive each dynamic simulation object. Mission model programming began in November of 1985, state display work began about the same time, while graphic display work started in late January, 1986. Over 50 man-months of combined programming effort was dedicated to this Phase, generating nearly 5000 lines of Fortran code (3000 in-house, 2000 contract), 1200 lines of C (in-house), and 7800 lines of Lisp (3800 in-house, 4000 contract) in this eight month period.

Numerous design and implementation compromises were made in the first phase of development due to the severe time constraints (start 11/1, complete by mid June), and minimal staffing available. Future phases must address more long-term strategic approaches.

9.2 PHASE II DEVELOPMENT

9.2.1 Requirements and Design Approach

9.2.1.1 Summary Level

Phase II was intended to devise more long-term approaches to the development of the workstation. Another primary purpose was to assemble a team of individuals (both in-house and outside) appropriate for this activity. This team would be composed of both researchers and implementers, since the Program's approach is to employ contemporary techniques to integrate the best models of human behavior/performance, vehicle/systems and environment available.

Phase I had succeeded in demonstrating that the concept of a prototype workstation for aiding early helicopter cockpit design with regard to human performance limitation was viable. The implications of attempting to develop such a system were also more clearly understood after Phase I. The purpose of Phase II was to:

1) Develop modelling and simulation tools
2) Develop graphics tools
3) Integrate 6 DOF helicopter dynamics
4) Develop a more representative mission
5) Design and implement a more modular architecture
6) Gain additional insight into modelling and the design process
7) Build an appropriate in-house implementation team
8) Establish working relationships with various centers-of-excellence

At the completion of Phase I it was evident that more long-term strategies to development would have to be adopted if the Program were to ultimately succeed. It was also recognized that a more active effort was necessary in the area of integration of research results if less mature fields such as human-computer interaction and predictive human modelling were to gain any acceptance.

New software development requirements in Phase II centered around modelling environment, pilot models, vehicle/systems models, world models, analysis and decision aiding, and user interfaces Work Breakdown Structure (WBS) elements. The specific applications chosen are described in the subsections that follow.

9.2.1.2 Modelling Environment

9.2.1.2.1 Mission Editor

Development of the Mission Editor continued in Phase II under subcontract with BBN with domain expertise and integration supplied by in-house staff. BBN designed a graphic editing interface utilizing the mouse and pop-up menu templates to relieve the user of Lisp code editing. A manual decision interface was also implemented to override the previous "selection by aspect" method of simulated pilot decision-making. The Mission/Task Editor is an application initially developed BBN that serves as the human performance modelling framework whereby more elaborate computational models of human behavior and performance may be installed or "activated" in the workstation contingent on the type and level of analysis required to answer a particular question. For example, the framework provides a default toplevel directed acyclic graph form of human task modelling that can be used to evaluate task sequencing and resultant human resource loadings (visual, auditory, cognitive and psychomotor) based on empirical data. However, it is possible to integrate detailed predictive computational models of human performance such as dynamic anthropometric models that are able to compute rates, durations, reach, comfort factors and other parameters. These detailed models can be utilized as an alternative to the more subjective toplevel empirical models that the system provides by default. The framework also provides interfaces to simulation models of the vehicle/systems and environment at various levels of abstraction. Most importantly, the framework provides contingent task behavior subject to vehicle and environmental state variables. Refer to Appendix 1 of the Phase II System Architecture Description Document (SADD) for details.

9.2.1.2.2 Modeller

The Modeller serves as the Phase II simulation executive. It provides all modes of interaction with the simulation, including build/edit, test/verify, experiment frame, run, analysis and document/report. The ultimate goal of the Modeller is to provide the complete environment for construction, integration, testing, simulation analysis and reporting of results. The most developed mode of the Modeller is the run mode, whereby the user controls the execution of the simulation models. Model selection, data specification and run-time display (state-displays and views) configuration is provided through the experiment frame mode of the Modeller. Refer to Appendix 2 of the Phase II SADD for details.

9.2.1.2.3 Visual Modeller

The Visual Modeller is a prototype visual programming language for dynamic systems modelling developed under subcontract by Expert-Ease Systems Inc, of Belmont, CA. It allows the user to select components such as dividers, summers, integrators, sources and sinks from an extensible library of components and assemble them on a graphic "worksheet" to form working models. Connections between component input and output ports are make graphically, as is specification of initial conditions and parameters. Models are subsequently run as interpreted code (versus compiled) subject to boundary conditions (start time, end time, step size) supplied by the user. The application was initially developed as a stand-alone tool for evaluation, hence it has yet to be integrated with other MIDAS workstation elements. It is anticipated that some form of visual programming language will be developed for the Modeiler build/edit mode that utilizes the same type of interface as the Visual Modeller. This tool would become the primary means of developing and integrating vehicle/systems and world models for the simulation. Refer to Appendix 3 of the Phase II SADD for details.

9.2.1.2.4 State Display Editor

The State Display Editor is an enhanced version of the Phase I Icon Editor, which is based on the Graphics Editor from Steamer, an application developed by BBN for the Navy Personnel Research and Development Center (NPRDC) in San Diego, CA. The State Display Editor is used to conveniently build displays composed of graphs, bars, dials, sliders, text, etc. for indicating the state of some selected dynamic variable during a simulation. The user interface has been substantially reworked to mimic the Apple Macintosh desktop metaphor. It also takes advantage of improvements to the Symbolics operating system (Genera 7.1) such as infinitely scrollable windows in both vertical and horizontal dimensions. Other features added to the original Icon Editor include "Macintosh-like" text handling, keystroke commands (to supplement pull-down menus) and numerous bug fixes that had existed from the original Steamer code. No programming is required to use this tool. Refer to Appendix 4 of the Phase II SADD for details.

9.2.1.3 Pilot Models

9.2.1.3.1 Anthropometric Model

The A^{3}I Anthropometric Model was developed under a NASA/Ames grant to Dr. Norman Badler at the University of Pennsylvania's Department of Computer and Information Science. The POSIT/HIRES model provides human body dimensions, reach and position based on CAR (Crewstation Assessment of Reach, Boeing Corp.) database link sizes, driven by task/goal specifications (HIRES). Unlike other systems that statistically utilize anthropometric databases, POSIT allows the user to specify each link independently and establish motion constraints for any link or joint in the model. Since the anthropometric model is an ongoing research effort at the University of Pennsylvania, a working version of POSIT was not received by the Program until 1 month prior to the end of Phase II. Consequently, the capabilities of the system were demonstrated off-line as a stand-alone system, although the graphic databases generated by other MIDAS workstation tools were immediately made compatible with POSIT/HIRES, as well and the Mission/Task Editor's output. Future Phases of the Program will see integration of the next generation of POSIT/HIRES (JACK/GOALTENDER), which is expected to include dynamics and field-of-view modelling capabilities. Refer to Appendix 5 of the Phase II SADD for details.

9.2.1.3.2 Loading Model

The loading model currently used to measure human performance is based on data accumulated at the Army Research Institute at Ft. Rucker, AL, that was subsequently committed to computer hardware within a modelling framework developed by BBN. The model generally states that task performance requires human resources from visual, auditory, cognitive and psychomotor (VACP) dimensions. This approach is loosely based on Chris Wickens Multiple Resource Theory and subsequent model implementation by Aldrich & McCracken at Anacapa Sciences Incorporated. The data from ARI is based on a survey of active helicopter pilots who were asked to estimate the VACP loadings for various tasks on a scale of 0-7, given very specific guidelines and criteria for each level in that range. The advantages of this approach are visibility, simplicity, and intuitive appeal. The disadvantages are that VACP values obtained from the survey are context dependent, thus to model variations in loading as a function of vehicle/system or world state, as well as pilot variables (training level, stress, fatigue, workload) there is no empirical data available to support alteration of baseline VACP values as a function of these variables.

9.2.1.4 Vehicle/Systems Models

9.2.1.4.1 Dynamics and Guidance Models

The type of helicopter dynamics and guidance models used in Phase I were discrete point-mass. These models did not provide the necessary data in translational and rotational degrees of freedom to analyze vehicle orientation (pitch, roll, yaw) as a function of control movements. The dynamics model used in Phase II had been used at Ames previously in motion simulation studies on VAX VMS hosts, thus the majority of the effort required to use this model was a port to the Symbolics computer. This, however, was not a simple task due to compiler differences and the Program's need to "package" models in a modular fashion that simplifies integration and interaction with other code, most notably Lisp. The guidance routines were also based on existing code used in motion simulator facilities, although extensive enhancements were required and performed by Anil Phatak and Huan Tran of Analytical Mechanics Associates (AMA), Inc. These included outer-loop speed, horizontal and vertical path control feedback to provide path following that was more representative of real pilot strategies. An important lesson learned in Phase II was that it is not advisable to attempt to integrate models that are still undergoing development into a system that itself is evolving, particularly when there are language compatibility complications (e.g. Fortran and Lisp). Although this was necessary in Phase II due to time constraints, future Phases should avoid this through better planning.

9.2.1.4.2 Cockpit Display Editor

The Cockpit Display Editor (CDE) is a new application developed for the construction and editing of cockpit displays and controls. Its interface is based on the MultiGen™ visual database editing program developed by Software Systems of San Jose, CA. Software Systems and Sterling Software entered a joint development agreement which allowed Sterling source code rights to the MultiGen™ software in exchange for developing enhancements to MultiGen™ (described in the Graphic Views SCDD). The CDE is 3-D, color, dynamic, and has a user interface modelled after the Apple Macintosh. Displays can be conveniently linked to model parameters, or animated from stored datafiles. The CDE contains the database of all positional and geometric attributes of displays and controls in the simulated cockpit. This database will eventually be utilized by human behavior and performance models such as signal detection and visibility to analyze optimal instruments/control placement.

9.2.1.5 World Models

9.2.1.5.1 World Models

World models have not changed significantly over Phase I, with the exception of replacing flat terrain and gaussian hills with Defense Mapping Agency (DMA) terrain data. Architecturally, considerable effort was devoted to separating the simulation of world objects from pilot and vehicle/systems simulation models in anticipation of migration to distributed or parallel processing hardware.

9.2.1.5.2 Views

Geometric representations of world objects have changed appreciably from Phase I with the addition of enhanced MultiGen™ software. As mentioned in world models, flat terrain and gaussian hills have been replaced with (DMA) terrain data, and objects/vehicles have real-world dimensions since they were obtained form a simulator out-the-window visual system database (Singer-Link DIG). The DMA data can be read directly off a tape and transformed into a 3-D, colored image by the graphic modelling system. The simulation views have been enhanced as well, since MultiGen™ features provide interactive, Macintosh-like editing of objects (it is an object-oriented editor) and subsequent viewing with 3 translational and rotational degrees of freedom. Model-driven or stand-alone animation of objects is also provided as a feature that was added to the basic MultiGen™ software.

9.2.1.6 Analysis and Decision Aiding

9.2.1.6.1 Training Resource Requirements Prediction

The impact of training was demonstrated in Phase I by providing 2 types of pilot models (skilled and novice) and comparing the performance results in a simulated mission for each case. The two models were based on the assumption that on the average, a less skilled pilot requires longer and perceives a higher (VACP) load in task performance than a skilled pilot. Phase II endeavored to demonstrate that it would be possible to perform an analysis of a simulated mission's tasks and extrapolate the training resource requirements necessary to train an individual to perform such a mission. The approach drew heavily from Instructional System Development (ISD) methodologies. Hence the focus was upon post-simulation analysis and training requirements estimation, rather that attempting to show the effects of skill level variations on mission performance. Both are considered important for the MIDAS workstation.

9.2.1.7 Demonstration Scenario

The Phase II Scenario involved walking demonstration attendees through a simulated mission and cockpit build, anthropometry analysis, and task loading/timeline inspection. Emphasis was placed on "running" as many components together in an integrated fashion for a slight derivative of the Ambush Scenario first started during Phase I.

9.2.2 Hardware Environment

Figure 9 below indicates the Phase II hardware configuration. These components are described in further detail in the subsections which follow.

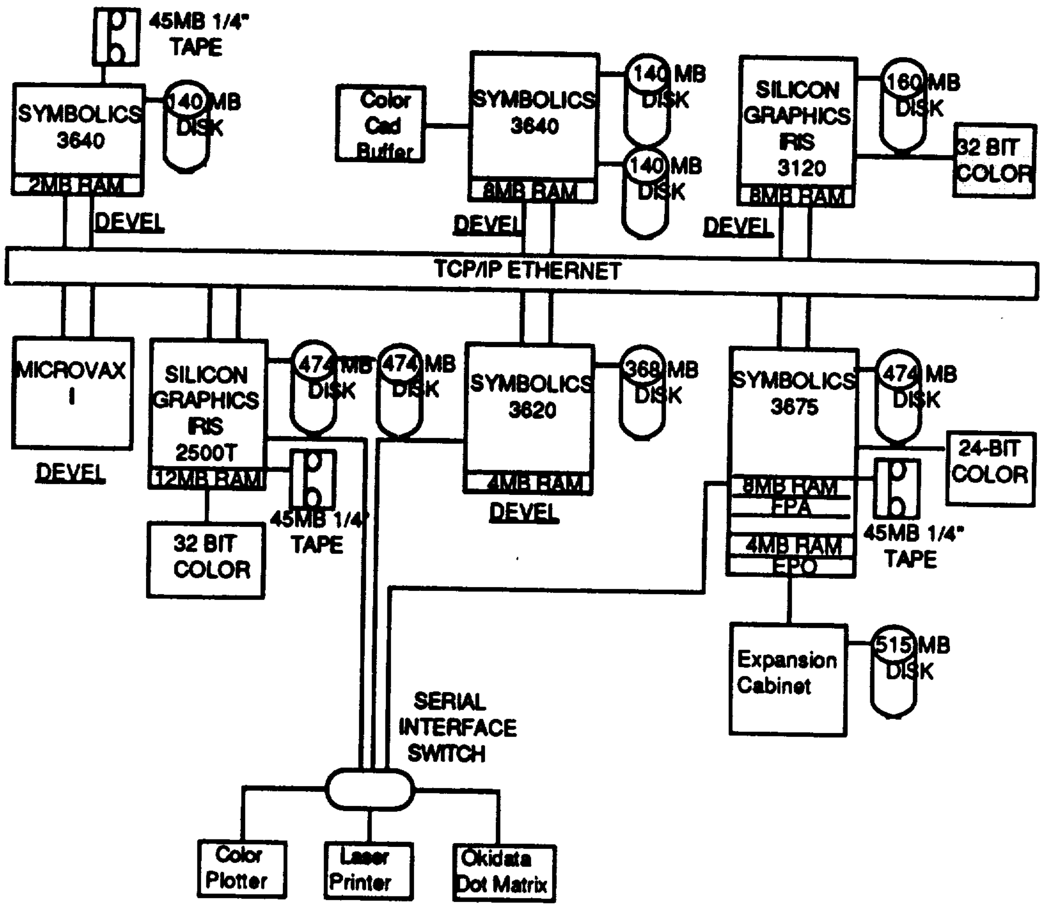


Figure 9. Phase II Hardware Configuration

9.2.2.1 Symbolics Lisp Machines

Model 3675 Color Workstation consisting of:

Monochrome Console
Keyboard & Mouse
45 MB 1/2" Cartridge Tape Drive
Ethernet Controller and Transceiver
12 MB RAM
Enhance Performance Option
474 MB Fujitsu Eagle Disk
515 MB CDC Disk
Model CG70-FB02 High Resolution, 24-bits/Pixel Color Frame Buffer
Tektronix 19" Color RGB Monitor
Model OP36-FPA1 Floating Point Accelerator
Symbolics # SLAN-FORT Fortran 77 Compiler
Symbolics # STCP-1 TCP/IP Software

Model 3640 Color Workstation consisting of:

Monochrome Console
Keyboard & Mouse
Ethernet Controller and Transceiver
6 MB RAM
2-140 MB Disks
CAD Buffer
Tektronix 19" Color RGB Monitor
Symbolics # SLAN-FORT Fortran 77 Compiler
Symbolics # STCP-1 TCP/IP Software

Model 3620 Monochrome Workstation consisting of:

Monochrome Console
Keyboard & Mouse
Ethernet Controller and Transceiver
4MB RAM
368 MB Disk
Symbolics # SLAN-FORT Fortran 77 Compiler
Symbolics # STCP-1 TCP/IP Software

9.2.2.2 Silicon Graphics Computers

W-2500A Workstation consisting of:

45 MB 1/2" Cartridge Tape Drive
Ethernet Controller and Transceiver
HU-T04 Turbo Option W/4MB RAM
H3-FPA Floating Point Accelerator
H-DM4A 1024x1024x4 Display Memory
H-ZC2 Z Clipping Assy
C-WTCP IP/TCP Software
P-DBX Dial/Button Box
R-UNIF Fortran Compiler

W-3120 Workstation consisting of:

Ethernet Controller and Transceiver
H3-FPA Floating Point Accelerator
H-DM4A 1024x1024x4 Display Memory
H-ZC2 Z Clipping Assy
C-WTCP IP/TCP Software
P-DBX Dial/Button Box

9.2.2.3 Other Processors

Digital Equipment Corp. MicroVax II

9.2.2.4 Networking Hardware

TCL Incorporated Model 2010EC Ethernet Transceivers (tap-type)

9.2.2.5 Peripherals

Okidata Model 2410 Dot Matrix Printer
Apple LaserWriter Plus Laser Printer
GraphOn Graphics Terminal

9.2.3 Software Environment

Generally, the graphic design and analysis tools were built using C, Unix BSD 4.3, and the IRIS Window Manager/Graphics Library II. The MultiGen™ modelling package was used as the underpinnings for the CDE and Views components. Most Lisp tools and models were built using Symbolics Common Lisp under Genera 6.2, with the Flavors extension, and the S-Packages were often used for for Displays. Communication software was written to enable inter-machine message passing and simulation synchronization. The distribution of the Phase II models,tools, and displays is shown in Figure 10 below.

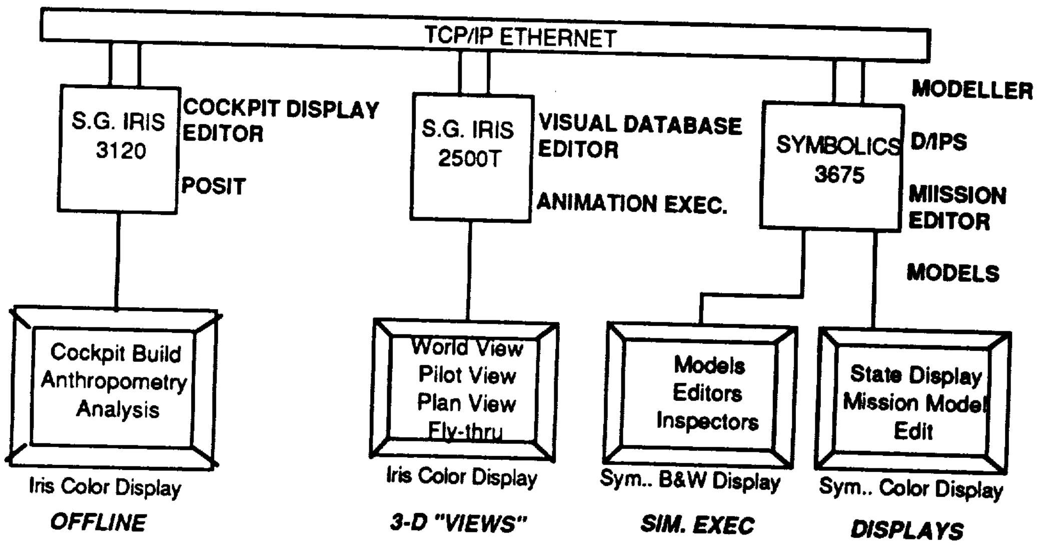


Figure 10. Phase II Software Components & Displays

9.2.4 Programmatic Information

9.2.4.1 Constraints

Following the Phase II offsite planning meeting at Saint Francis Retreat, a schedule was developed for the work that had been identified during the planning meeting. The schedule indicated that the end of Phase II would require a period of approximately 1 year. Although the end of phase demonstrations would as a consequence likely miss affecting the following year's funding cycle, the plan was accepted.

Phase II development did not start until in late December, 1986 due to the need to prepare extensive budget proposals while at the same time developing technical plans. At that time, there were 6 in-house programming staff members composed of 3 Lisp programmers (2 junior, 1 intermediate), 2 graphics programmers (1 senior, 1 junior), a systems administrator and a task manager to perform and coordinate the implementation of the above mentioned 11 applications in cooperation with 2 subcontractors (BBN and AMA). In March, 1987, the senior graphics programmer left the Program to head another project, followed by the systems administrator shortly thereafter. These individuals were replaced, as well as the addition of 4 new staff member and 4 subcontracts by July, 1987.

A decision was make in July to complete Phase II by early October and begin Phase II in an effort to complete Phase III in time to influence the following year's funding. Although the staff required to perform work planned for Phase II were not available until very near the end of the phase, and the length of the phase was reduced by 2 months, sufficient work was completed to demonstrate the essential concepts planned for Phase II.

Equipment was a severe limitation until the delivery of two additional Symbolics computers and an IRIS workstation by June. Additional mass storage, dynamic memory and hardware upgrades were ordered for existing equipment. With the exception of the Symbolics upgrade to a 3675, most of the hardware was received and installed in time to improve the productivity of staff.

9.2.4.2 Risks

By the end of Phase II there were 6 subcontracts and 1 NASA grant under the management of in-house implementation staff. This represents a sizable risk that the contributions of these subcontractors be appropriate, cost-effective and carefully monitored. The limited funds of the Program could not tolerate any waste of manpower or funds.

The decision to attempt to integrate discrete-event Lisp-type and continuous Fortran-type models in Phase II, without the benefit of a specialist in simulation modelling, could have proven to be a disaster. Although successful, the process was inefficient and frustrating. Future attempts at such model integration must be made with the aid of an expert in simulation modelling.

Hardware technological risks were again minimized by the use of standard, proven computers and software. Communications was TCP/IP protocol, and special hardware had been purchased to monitor network data packets in the event of any problems.

The composition of the in-house implementation staff is currently engineers, programmers, mathematicians, computer scientists, physicists and a helicopter pilot. There are no cognitive psychologists or human factors specialists. This is a serious void for a team developing a prototype workstation for human factors analysis. Nonetheless, this void has been temporarily filled by subcontracting for this expertise.

9.2.4.3 Summary of Results

The first set of demonstrations was conducted for NASA and local personnel on October 22, 1987, followed by numerous scheduled and non-scheduled demonstrations over the course of the next 6 weeks. The feedback was without exception positive, although it became clear, after several comments, that in the next phase of development it would be necessary to address some concrete design issues rather that remaining general and abstract with regard to what the MIDAS workstation could accomplish.

Late in Phase II the Program Office obtained a Chief Scientist to direct ongoing research activities that are slated for eventual integration in the workstation. This addition tremendously improves the Program's ability to evaluate and respond to new research results that may be relevant to the design of complex man-machine systems. Similarly, two consultant subcontractors involved with the Program have extensive experience in the field of human behavior and performance modelling.

As Phase II was aimed at more long-term, strategic approaches, there will be a substantial amount of code reuse as well as coding momentum carried into Phase III. This momentum is further facilitated by reasonable success in assembling a team of individuals with appropriate skills and interests for this Program. These interests include human behavior/performance modelling, training, graphical modelling languages, integrative frameworks, decision aiding and user interfaces.

9.3 PHASE III DEVELOPMENT

9.3.1 Requirements and Design Approach

9.3.1.1 Summary Level

Responses to the Phase II demonstrations, as well as discussion at the Phase III off-site reinforced the need to continue the development of the core set of A^{3}I models and tools. However, emphasis would have to be placed on explicitly addressing how such models and tools would be sensitive to cockpit design change. Furthermore, the enhancement of the present applications and the start of new components must be anchored to a "real world" application for effective demonstration positions. The Program office decided to use a detailed, "vertical slice" of the conceptual development process as a method to illustrate the intended use of the workstation. The AH-64A mission and cockpit was the focus of the phase, with empirical flight test results from an AH-1 Cobra Communication Study as a source of specific task data. These objectives required a degree of integration and detail previously impossible, and drove the following development requirements for the individual MIDAS components:

9.3.1.2 Symbolic Modelling CSCI

Consisting of a mission editor, task decomposition aids, as well as state displays for task analysis, the focus for the Symbolic Modelling CSCI during this phase was on the design and coding of a generalizable framework for symbolically representing the functions of cockpit equipment used to accomplish mission tasks. This framework allows various cockpit alternatives to be evaluated without completely re-editing the mission decomposition, since the design maintains a distinction between the physical structure (or state operators) of the equipment, and the functional requirements (or inferred goals) required by the task. Previous A^{3}I symbolic models of the mission and pilot tasks failed to explicitly depict the relationship of the equipment design and the primitive task actions, loading values, or timelines. Results of the "vertical slice" example (report phase line) were used to demonstrate this process and successfully compared to actual data from a similar AH-1 flight test conducted by the Aeroflightdynamics Directorate. Task timeline, resource use, and loading displays were similar in concept to those used during previous phases.

9.3.1.3 Graphic Views CSCI

Phase III development requirements for the Views CSCI are best described as enhancements to the well-received capabilities existing in Phase II. The internal resolution of the geometric modelling package was reduced from 3/8 inch to 1/256 inch, permitting sharper and more detailed rendering of the DMA terrain and moving models made possible.

Also added was the capability for the user to select a viewing position from anywhere within the mission gaming area. Existing low-detail helicopter models were also replaced by the fully populated AH-64A model developed using the CDE. Finally, the Views CSCI was ported to the new IRIS 4D and modified as dictated by a new version of the underlying MultiGen™ software.

9.3.1.4 Cockpit Design Editor (CDE) CSCI

Existing CDE software was ported to the IRIS 4D and improved with the addition of pop-up user windows, an hierarchical data base of instrument information and characteristics for human factors analysis, and improved mouse operations. The CDE was successfully used to build a detailed 3-D model of the complete AH-64A pilot cockpit and instrumentation, providing both a feasible application of its capabilities, as well as cockpit/craft models for the remaining workstation elements. An F-16 cockpit was also created for potential use within the fixed-wing community. A fair amount of debugging of the initial CDE code took place with these full-scale CAD attempts along with code changes brought about by a new version of the underlying MultiGen™ software.

9.3.1.5 Anthropometric Model or JACK CSCI

Under our grant to Dr Norm Badler and the University of Pennsylvania, the 3-D dynamic anthropometric model was also ported to the IRIS 4-D and now includes fully defined body parts, limb joint constraint settings, adjustable eye viewpoints, as well as improved animation capabilities. This model was placed inside the AH-64A cockpit designed with the CDE and saved in a Psurf format, and "instructed" to perform elementary psychomotor operations through mouse operations. An somewhat unrealized goal for this phase involved "driving" Jack through the task decomposition commands on the Symbolics. Although demonstrated in a small scale, this level of integration during a simulation was not pursued due to the lack of a synchronizing "time concept" within Jack. A number of miscellaneous improvements were also added by the developer including a spread-sheet style anthropometric database with editing options, increased functionality in shaded (rather than wire-frame mode), and a new sliding window user interface. Jack was used during the phase to perform reach analyses and a simplified visual occlusion test using 5th and 95th percentile males/females in the Apache Cockpit.

Far more rigorous vision models were kicked-off this phase with a grant to the New York Association for the Blind, and a contract with the David Sarnoff Research Center. These models of volume field of view and legibility, respectively, will be integrated within the Jack environment but are not detailed within this document since preliminary versions are not expected until Phase IV.

9.3.1.6 Aerodynamics & Guidance Module (AGM) CSCI

The AGM CSCI was ported from the Symbolics computer to the IRIS 4D and almost entirely translated from Fortran to C to take advantage of the additional compute power. One of the specific desires this phase was to demonstrate that A^{3}I indeed had a viable aero model which could be used to provide position, orientation, and rate information to other components. Modifications were made to provide user-selectable action points during simulation, a graphical (MultiGen™ based) representation of helicopter pitch, roll, and yaw characteristics, simulation of cyclic, collective, and pedal inputs, and improved interface with simulated pilot models. A high degree of integration between the AGM and Symbolic Modelling CSCI's was attempted but made difficult due to limitations of the AGM in accurately portraying some piloting activities required in the demonstration scenario.

9.3.1.7 Communications CSCI

The IRIS to Symbolics Communications software developed during Phase II was used essentially "as is" for Phase III. Some modification was necessary due to the receipt of two new IRIS's during the phase. Because most of the application developments addressed during this period were "stand-alone" oriented, the major use of this CSCI was to transmit position, orientation, rate, and engine characteristics from the AGM to the Views and CDE CSCI's for animation.

9.3.1.8 Training Assessment CSCI

Training assessment was greatly augmented and refined during Phase III. During Phase II, an instructional system was assigned (by table look-up) to train each task by matching task characteristics with attributes of instructional systems within the task's learning category. The Phase III approach used ART (the Automated Reasoning Tool) and Common Lisp on the Symbolics to develop a prototype knowledge-based system. This processes uses the instructional systems design (ISD) process to assign each task a set of learning experiences (such as explanation, demonstration, part-task training, and full task training) along with a medium for each learning experience (such as textbook/workbook, interactive slide/tape, lecture with visual aids, videodisc/CBT, CFT, CPT, OFT, WST without and with motion, and the actual system). For each learning experience/media assignment, a time to train is computed, based on the task, operator, and equipment characteristics. The Phase III training approach was heavily based on previous work performed by the Logicon Corporation under contract to the Air Force. Training assessment is accomplished on an individual task basis, with no attempt made to address the grouping of tasks into lessons or courses.

9.3.1.9 Scheduler

Work was also begun this phase on a dynamic reactive scheduling component to address the sequencing and scheduling a pilot may perform as a means to control his task performance and timeliness. While not committed to code during Phase III, significant headway into the specific objectives and approaches for this state-of-the art component were achieved. Because it was not completed during the phase, formal documentation for the scheduler is not provided as part of this Phase III SDDD.

9.3.1.10 Demonstration Scenario

The Phase III demonstrations consisted of a 30 minute introductory briefing by the program director, followed by 1.5 hours of application demonstrations by the staff. The "vertical slice" into the development process was emphasized as the attendee was essentially "walked through" the conceptual development process for a potential communication switch change on the AH-64A. The demonstration objective was to describe the potential interactions and conflicts arising from moving the radio select button from the ICS panel on the front bulkhead to the cyclic (as was done in the AH-1 flight test). The capability for A^3I to support three phases of the design process was stressed: specification, static analysis, and dynamic analysis.

Beginning with the Views CSCI, the projected DMA gaming area was portrayed as the mission environment and the inherent capabilities of visualization emphasized. Next, the mission editing component of the Symbolic Modelling CSCI was introduced, along with it's facilities to input the scenario for an unmasking maneuver combined with several radio calls. The Aerodynamics & Guidance CSCI was then demonstrated in a stand-alone fashion, traversing over simulated flat terrain and viewed in several perspectives. Following the AGM, the CDE was demonstrated. Because of the maturity and visual nature of this component, a fair amount of detail was provided—beginning with the procedures to build an individual gauge, attaching it to a control panel, animating it, placing it in a cockpit, building a vehicle structure around the cockpit, and finally, placing the completed helicopter in the world prior to the simulation. As the last demonstration of the MIDAS specification capabilities, the Symbolic Modelling CSCI was returned to. This time it was used to portray a further decomposed mission with the design-dependent operator activities fully described as a result of the symbolic equipment models for the alternative communication switch configurations.

Jack was then used to perform some basic reach sequences in the AH-64A cockpit, using the maximum ranges of the human model data. The fact that the operator could not reach the panel-mounted communications panel when restricted to moving from the shoulder only (simulating a locked inertial reel) was demonstrated. Additionally, the potentially dangerous glare shield occlusion of the tailwheel lock and master arming switch was shown for "taller" pilots by attaching Jack's camera to the mannequin's eye during an animation sequence.

The Training Assessment CSCI demonstration was then conducted for two send-radio message tasks with the alternative radio switch configurations. The input, output, and processing characteristics of this component was described, as the attendees were shown how a knowledge-based system operates.

The newly initiated applied vision models were then introduced through a briefing. An Amiga-based prototype of the New York Association for the Blind's volume field of view model was used to render the binocular retinal maps, facial occlusions, and physiological blind spots for a series of mouse-selected fixation points.

The demonstrations were then concluded with the dynamic analysis capabilities of MIDAS. The fully populated cockpit was placed in the gaming area, driven by the aerodynamics and guidance model, and viewed from several perspectives. A summary output screen from a "simulation run" was then shown on the Symbolic Modelling color monitor. This screen showed the loading and task timelines for the two potential designs and made use of different colors to describe physical resource conflicts between the scenario's flying tasks and the radio selection actions. Approaches for hypermedia-like access to Boff & Lincoln's *Human Engineering Data Compendium* (once it comes out on CD-ROM) was then shown through a simulated key-word search. The intent was to describe how analysts would be able to get extremely valuable context-sensitive information for areas such as "performance under vibration" or "effects of the use of gloves" to make cockpit design and mission decisions.

9.3.2 Hardware Environment

The hardware architecture in place at the end of Phase III is depicted in Figure 11 below. These components, together with their resident software and peripherals are described in further detail in the subsections which follow.

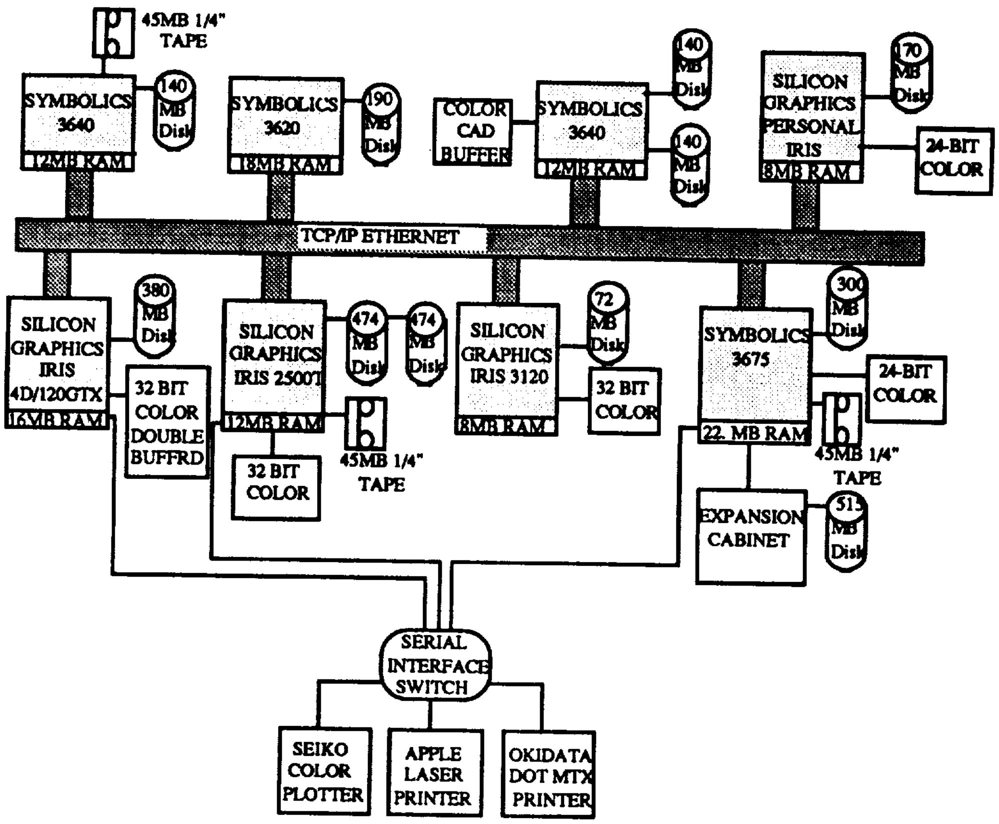


Figure 11. Phase III Hardware Configuration

9.3.2.1 Symbolics Lisp Machines

Model 3675 Color Workstation (Barracuda) consisting of:

Monochrome Console with OCLI filter
Keyboard & Mouse
45 MB 1/4" Cartridge Tape Drive
Ethernet Controller and Transceiver
22.5 MB RAM
Enhanced Performance Option
338 MB Fujitsu Eagle Disk
550 MB CDC Disk
Model CG70-FB02 High Resolution, 24-bits/Pixel Color Frame Buffer
Tektronix 19" Color RGB Monitor
Model OP36-FPA1 Floating Point Accelerator
Symbolics # SLAN-FORT Fortran 77 Compiler
Symbolics # STCP-1 TCP/IP Software
S-Group (S-Paint, S-Geometry, S-Render, S-Dynamics, and color 6.0 V405.13)
Genera 7.2

Model 3640 Color Workstation (Puffer) consisting of:

Monochrome Console with OCLI Filter
Keyboard & Mouse

45 MB 1/4" Cartridge Tape Drive
Ethernet Controller and Transceiver
11.25 MB RAM
2-140 MB Disks
CAD Buffer
Tektronix 19" Color RGB Monitor
Symbolics # SLAN-FORT Fortran 77 Compiler
Symbolics # STCP-1 TCP/IP Software
S-Group (S-Paint, S-Geometry, S-Render, S-Dynamics, and color 6.0 V405.13)
Genera 7.2

Model 3640 Monochrome Workstation (Squid) consisting of:

Monochrome Console with OCLI Filter
Keyboard & Mouse
Ethernet Controller and Transceiver
13.5 MB RAM
2-140 MB Disks
Symbolics # SLAN-FORT Fortran 77 Compiler
Symbolics # STCP-1 TCP/IP Software
Genera 7.2
Automated Reasoning Tool (ART) Version 3.2

Model 3620 Monochrome Workstation (Sea Slug) consisting of:

Monochrome Console with OCLI Filter
Keyboard & Mouse
Ethernet Controller and Transceiver
18 MB RAM
190 MB ST506 Disk
Symbolics # SLAN-FORT Fortran 77 Compiler
Symbolics # STCP-1 TCP/IP Software
Genera 7.2

9.3.2.2 Silicon Graphics Computers

W-2500A Workstation (Orca) consisting of:

19" High Resolution Monitor
Keyboard & Mouse
45 MB 1/4" Cartridge Tape Drive
Ethernet Controller and Transceiver
12 MB RAM
2 474 MB Fujitsu 10.5" Disk Drives
HU-T04 Turbo Option W/4MB RAM
H3-FPA Floating Point Accelerator
H-DM4A 1024x1024x4 Display Memory
H-ZC2 Z Clipping Assy
C-WTCP IP/TCP Software
P-DBX Dial/Button Box
Unix System V with BSD 4.2
NFS
C Compiler
IRIS Graphics Library II and Window Manager

W-3120 Workstation (Manta) consisting of:

19" High Resolution Monitor
Keyboard & Mouse
Ethernet Controller and Transceiver
8 MB RAM
72 MB Winchester Disk Drive
H3-FPA Floating Point Accelerator

H-DM4A 1024x1024x4 Display Memory
H-ZC2 Z Clipping Assy
C-WTCP IP/TCP Software
P-DBX Dial/Button Box
Unix System V with BSD 4.2
NFS
C Compiler
IRIS Graphics Library II and Window Manager

W-4D120GTX PowerSeries Workstation (Coral) consisting of:

19" High Resolution Monitor
Keyboard & Mouse
Ethernet Controller and Transceiver
16 MB RAM
380 MB ESDI Winchester Disk Drive
Double Buffered 1280x1024x4 Display Memory
Double Buffered Alpha
24 bit Z buffer
C-WTCP IP/TCP Software
P-DBX Dial/Button Box
IRIX System V release 4D1-3.1D
NFS
C Compiler
C++ Translator
Fortran 77 Compiler
IRIS Graphics Library II and 4Sight Windowing System

W-4D20G Personal IRIS Workstation (Urchin) consisting of:

19" High Resolution Monitor
Keyboard & Mouse
Ethernet Controller and Transceiver
8 MB RAM
170 MB SCSI Winchester Disk Drive
1280x1024x4 Display Memory
Double Buffered Alpha
C-WTCP IP/TCP Software
IRIX System V release 4D1-3.1D
NFS
C Compiler
IRIS Graphics Library II, 4Sight Windowing System, and Environment Manager

9.3.2.3 Networking Hardware

CableTron MT-800 Ethernet/IEEE 802.3 Transceiver

9.3.2.4 Peripherals

Okidata Model 2410 Dot Matrix Printer
Apple LaserWriter Plus Laser Printer
Seiko Instruments D-Scan CH5312 Color Printer & Multiplexor
GraphOn GQ-250 ASCII Terminal & Keyboard (2)
Hewlett-Packard HP 700/22 ASCII Terminal & Keyboard (3)

9.3.3 Software Environment

Generally, the graphic design and analysis tools were built using C, Unix BSD 4.3, and the IRIS Window Manager/Graphics Library II. The MultiGen™ modelling package was used as the underpinnings for the CDE and Views components, as well as a visualization medium for the Aerodynamics and Guidance CSCI. Most Lisp tools and models were built using Symbolics Common Lisp, with the Flavors extension, under Genera 7.2. The Training Analysis Module uses the Automated Reasoning Tool (ART) as a shell for its inference requirements. The S-Packages were available for use in displays, although not heavily relied upon during this phase. Communication software written in previous phases was used to enable inter-machine message passing and simulation synchronization. The distribution of the Phase III models, tools, and displays is shown in Figure 12 below.

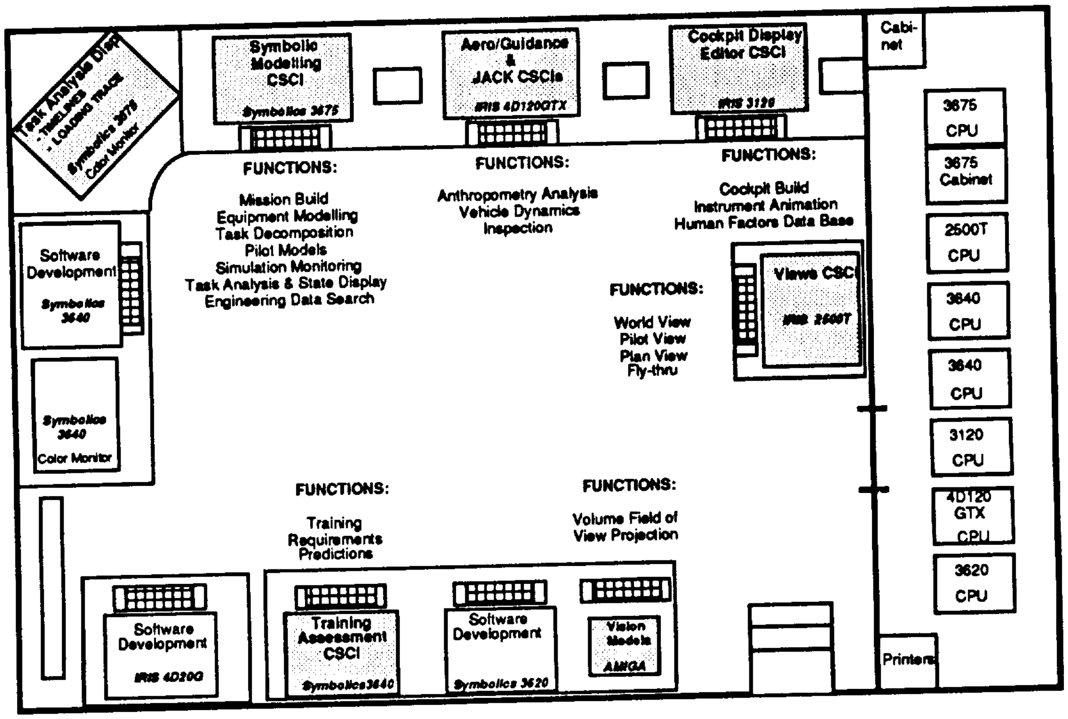


Figure 12. Distribution of Phase III Software Components and Displays within the A^3I Lab

Absent from the figure above is the Communications CSCI. This component is actually distributed among all of the various Symbolics and Silicon Graphics machines as dictated by the integration requirements. Currently, the capability exists to share the following variables among the graphic and symbolic computers during a simulation.

For each Helicopter (Ownship and Wing): *Ownship only

Helicopter position (x, y, z)
Helicopter orientation (yaw, pitch, roll)
Altitude (AGL), Airspeed, Groundspeed
Velocity in each axis
Engine torque
*Cyclic position, pedal position, collective position

For each Truck or Ground Vehicle (up to 6):

Truck position (x,y,z)
Truck orientation (yaw)

For each Missile on board a Helicopter or Ground Vehicle (up to 9):

Missile position (x, y, z)
Missile orientation (yaw & pitch)

Additionally, each object above has several "flags" which can bet set to communicate when an event such as a "hit" or an "explosion" occurs.

9.3.4 Programmatic Information

9.3.4.1 Constraints

Following the Phase III off-site at Asilomar Conference Grounds in December 1987, the group began Phase III in earnest. The demonstrations were initially set for June of 1988, but then slipped to November 1988 for a variety of reasons.

Funding for Phase III essentially dictated a no growth policy for the in-house staff, along with minimal new subcontracts/grants or equipment purchases. In general, the computing equipment available during this phase was well-matched to the development requirements —a rather distinct change from previous phases. In addition, shortly before the demonstrations, the A^{3}I lab was completely remodelled, greatly improving the staff's working conditions through additional table space, improved lighting, and a sharp appearance. Approximately 2 weeks of development time was lost during this period, although all the computing equipment managed to handle the movement well. The additional space enabled us to take receipt of a new SGI 4D70G IRIS and have it up and running quickly. Subsequent upgrades to this configuration allowed A^{3}I to be one of the first users equipped with the new PowerSeries 4D120GTX, containing parallel graphics and central processors. Although this capability was not really exploited during the phase, valuable experience with the new architecture was gained.

In contrast to the above, the impacts to staffing and outside efforts due to the funding level were not so benign. Overall, staffing levels were inadequate to meet all the goals set forth by the Program office. Although a new deputy director for A^{3}I (government employee) was added during August 1988 and his background in simulation/training devices was very helpful with the Training Assessment CSCI, the crunch was felt by the Sterling Software staff. Throughout the majority of the phase, the in-house staff consisted of 3 Lisp programmers (2 senior, 1 junior), 3 graphics programmers (2 senior, 1 intermediate), and a system administrator who doubled as an applications analyst for the Jack CSCI. Problems with identifying and hiring a suitable Task Manger after Mr Lakowske's departure in March 1988 severely complicated any serious integration efforts. The fact that the A^{3}I Program has essentially been without a permanent and competent first line supervisor for over a year has definitely delayed progress in general. It has also made coordination and communications between the Program's diverse efforts nearly impossible.

To mitigate the staffing deficiencies, Dr Yan Yufik (Yufik and Associates) received a subcontract during this phase for assistance in cognitive modelling and training. However, his ideas for an analytical complexity assessment were not universally well received and his support was not renewed after November 1988. Similarly, Dr Anil Phatak from AMA aided the in-house staff with minor AGM improvements, but was also unfunded during FY89. An active subcontract with BBN was also maintained during this phase but not funded.

Despite the relatively weak year in funding, new outside efforts were initiated with the New York Association for the Blind and the David Sarnoff Research Lab for applied vision models. Spearheaded by Dr James Larimer, the chief scientist for A^3I, these new components promise to be extremely valuable models for MIDAS. However, prior to the demonstrations (June 1988), Dr Larimer was promoted to Branch chief, and while still within ASHFRD, his oversight and guidance for these efforts was diminished. Furthermore, his more critical role of overall scientific guidance for A^3I was lost. A suitable replacement has not been found.

Dr Larimer's departure, combined with the fact that virtually no one on the development staff was formally trained in human factors, complicated our ability to develop and convincingly demonstrate the use of MIDAS in a "real world" cockpit design application. The present organizational structure has required each programmer/analyst to be not only a software developer, but a part-time scientist and domain expert as well.

Following the documentation requirements at the end of the phase, two senior graphics programmers left the project to pursue positions outside of Ames. However, this period marked a turning point. Crippled by the loss of in-house personnel, recruiting activities began immediately. By June 1989, two outstanding software engineers were hired. A new CAD draftsperson, a system administrator, a human factors analyst, and a senior Lisp programmer were also hired during the summer, rounding out an A^3I development staff which was now back to full strength. Although a Task Manager had not been hired, we had several promising leads and were well-poised to enter Phase IV.

On a related topic, a program office reorganization plan was approved by the US Army Aeroflightdynamics Director and the NASA Director of Aerospace Systems. This reorganization expanded A^3I's principal scientist positions from one to two, and established a new technology transfer group called the Industry Liaison Section which would be headed by a government project manager. A^3I's core research and development responsibilities would be split between the two scientists, with one steering cognitive aspects and the other perceptual issues. This division is a specific recommendation of the National Research Council study and report entitled *Human Performance Models for Computer-Aided Engineering*. The dual scientist positions would hopefully allow better monitoring and guidance for our extramural grants and contracts, as well as more time devoted to and specific direction for the in-house software development staff. The ILS section would respond to the increasing number of requests to disseminate in-house technology by intensifying efforts to transfer the appropriate A^3I lab products to other government agencies and industry. Unfortunately, the filling of this government post was not authorized during Phase IV, so the benefits of the new organization were not realized.

9.3.4.2 Risks

The majority of the risks faced by A^3I during Phase III were management-oriented and not technical in nature. They stemmed from unclear and conflicting development direction combined with the aforementioned staffing situation. A program with the ambitions and scope of this magnitude cannot tolerate a continuance of these problems.

One quasi-technical risk did exist and should be mentioned however. It centers on the level of detail appropriate for the MIDAS design and analysis objectives. The Program Office has made it clear that MIDAS is intended for the conceptual development phase of crewstation design because of the high "payoff" for properly incorporating human engineering principles during this period. However, most of the known human performance models and analysis methods require as inputs task, equipment, and environmental data which is more appropriate for detailed design. This apparent conflict between the model/analysis needs and the intended use of MIDAS was (and still is) unresolved. Its resolution will have serious implications for the Program's success in developing a prototype workstation which meets the needs of its projected users.

9.3.4.3 Summary of Results

The program had a tremendous response to the traditional end-of-phase demonstrations. Begun in November 1988, these demonstrations were attended by approximately 170 people from NASA, the US Army, other DoD components, as well as several universities. We were then asked by the Aeroflightdynamics Director to extend invitations to industrial sources, particularly the major helicopter manufacturers. The detailed demonstrations which resulted were actually conducted more as joint working groups and continued periodically through April 1989. This activity precipitated a significant amount of effort which can best be described as technology transfer. Lockheed Missiles and Space Company used our CDE package and vehicle dynamics interface to demonstrate a proposed Autonomous Underwater Vehicle concept. Boeing Commercial Aircraft Company spent three days at our facilities, understanding the tools, walking through code, and taking both software and documentation back to Seattle to set up a MIDAS-like design workstation at their company. Finally, the Marine Advanced Amphibious Attack Vehicle (AAAV) program office became very interested in the MIDAS capabilities, and we completed some vehicle prototyping design work for their review. Similar activities with the Fiber-Optic Guided Missile (FOG-M) program office and Boeing Helicopter Company also may evolve.

The few criticisms which were levied essentially boiled down to three areas. First, a number of people expected to see more explicit human performance models, especially in the cognitive area. Functions such as decision making, planning, scheduling, etc were not emphasized this phase. Even where present, such models were often embedded within the mission decomposition/simulation component complicating their observation. Additionally, a number of attendees indicated they wanted more "hard analysis." People wanted to know specifically how MIDAS could enable them to find the "best" design in terms of any number of measures—both quantitative and qualitative. They also wanted to get to a "bottom line" in terms of mission success, etc. While "bottom line" aspects have never been a particular focus of MIDAS, significant room does exist for us to improve the analysis capabilities included for design evaluation. Finally, a number of folks dug deeply enough to see that we haven't yet reached the level of integration among the CSCIs which is intimated. Distinct equipment models exist on both the graphic and symbolic sides. Task information needed by the Training Assessment CSCI is not contained in the task objects under the Symbolic Modelling CSCI. Jack was only demonstrated in a stand-alone mode. The lack of a properly functioning simulation capability at the start of the demonstrations certainly contributed to this criticism. However, the point was generally accurate. The degree of integration among the CSCIs was not where is should have been at that point in the overall development—primarily because it is the most difficult area of all to manage.

10.0 ANNEXES

Annex A

Army-NASA Aircrew/Aircraft Integration Program: Phase IV

A^3I

Man-Machine Integration Design and Analysis System (MIDAS) Software Detailed Design Document

Symbolic Operator Model

prepared by

Jerry Murray

Table of Contents

Table of Contents

Table of Contents

MAN-MACHINE INTEGRATION DESIGN & ANALYSIS SYSTEM (MIDAS) SOFTWARE DETAILED DESIGN DOCUMENT PHASE IV:

SYMBOLIC OPERATOR MODEL

1.0 INTRODUCTION

1.1 IDENTIFICATION OF DOCUMENT

This is the Software Product Specification for the Symbolic Operator Model module of the Man-machine Integration Design and Analysis System (MIDAS).

1.2 SCOPE OF DOCUMENT

The material in this document is directed toward three categories of readers:

1) those who wish to learn what the MIDAS Symbolic Operator Model module does,

2) those who wish to use the Symbolic Operator Model software to investigate the interactions between an operator and a specific crew station design within the context of a given mission,

3) those who might want to modify and update the Symbolic Operator Model software.

1.3 PURPOSE AND OBJECTIVE OF DOCUMENT

This document attempts to describe the methodologies used to represent an object-oriented simulation environment in which crew member activities are sensitive to crew station design. The primary purpose of this document is to present the Phase IV methodology for representing mission goals and crew member activities and their interactions with other models during a simulation. This methodology provides a framework in which models of human performance may be integrated for the purposes of evaluating alternative crew station designs.

This document presents the results of work accomplished in Phase IV. It is important to remember when using this document that the primary objective for symbolic modeling in Phase IV was to demonstrate a general framework for representing mission requirements and crew member activities and how these representations could interact with other models to provide a driving function for a simulation and task history for later analysis. Many of the functions and data structures used were selected on the basis of how they contributed to achieving this objective. As is typical in any rapid prototyping environment, many of these functions and structures will not be appropriate in future phases. However, attempts have been made to present what is available in a manner which will support development in future phases.

A majority of the code developed in Phases I, II and III was not required to meet the current objectives and not incorporated into Phase IV. A significant portion of that code, however, does address issues which remain major concerns of the MIDAS program and may be useful in future phases. This document does not intend to supersede previous documentation which address many issues not addressed in Phase IV, especially in the areas of decision modeling. In many cases, techniques developed in previous phases may be incorporated in future work with little or no modifications.

2.0 RELATED DOCUMENTS

2.1 APPLICABLE DOCUMENTS

James Allen, "Maintaining Knowledge about Temporal Intervals", *Communications of the ACM* **26** (11), 832-843, 1983.

Army-NASA Aircrew/Aircraft Integration Program (A^3I) Software Detailed Design Document: Phase III, Contractor Report 177557, NASA Ames Research Center, Moffett Field, California 94035-1000, June 1990.

Symbolics Genera 7.2 Documentation, Symbolics Publication Number 999079, Symbolics, Inc., Cambridge, Massachusetts, 1988.

Development of an Advanced Task Analysis Methodology and Demonstration for Army-NASA Aircrew/Aircraft Integration, BBN Laboratories, NASA Contract No. NAS2-12035.

Boff, Kenneth R. and Lincoln, Janet E., *Engineering Data Compendium, Human Perception and Performance*, Harry G. Armstrong Aerospace Medical Research Laboratory, Wright-Patterson Air Force Base, Ohio, 1988.

Operator's Manual for Army AH-64A Helicopter, TM 55-1520-238-10, Headquarters, Department of the Army, 28 June 1984.

A Computer Analysis to Predict Crew Workload During LHX Scout-Attack Missions, Anacapa Sciences, Inc. October 1984

2.2 INFORMATION DOCUMENTS

A Comprehensive Task Analysis of the AH-64 Mission with Crew Workload Estimates and Preliminary Decision Rules for Developing an AH-64 Workload Prediction Model, Anacapa Sciences, Inc., October 1986.

Sacerdoti, E.D., The Non-linear Nature of Plans, Advance Papers of IJCAI-1975, Tbilisi, USSR.

Stefik, M.J., Planning with Constraints, *Artificial Intelligence*, 16, pp 111-140. 1981.

Sussman, G.J., A Computational Model of Skill Acquisition, Ph.D. Thesis, Massachusetts Institute of Technology, Cambridge, MA., 1973.

3.0 CONCEPT

3.1 DEFINITION OF SOFTWARE

3.1.1 Purpose and Scope

The development of a dynamic and interactive analysis methodology for investigating environment, design, mission and crew interactions in the context of a mission simulation requires that the designer or planner be provided the freedom to modify any of the significant elements being analyzed at varied levels of detail. In response, the simulation must provide for a reorganization and reordering of the tasks executed by the pilot, and a recalculation of performance and workload metrics as a function of the designer-imposed changes. Standard decomposition procedures tend to have a "Bottom-up" structure. The Symbolic Operator Model takes a different approach. In order to capture the characteristics that pilots bring to task performance, and to provide the flexibility described above, this module employs two interacting perspectives to guide the mission decomposition.

- The first perspective views the mission from the aircrew's goal structure in mission performance.

- The second perspective is that of the software implementation architecture providing an object-oriented representation for the fundamental task units which are derived from the system design style.

The Symbolic Operator Model combines a goal / activity oriented mission perspective and an object-oriented software structure, with a detailed description of the tasks performed in service of the mission's operational goals. This description includes characterization of task performance time, performance load, models of pilot's decision and response characteristics and mission specific doctrine pertaining to task performance. The module provides for the integration of models of the environment, equipment design, mission and crew as the forcing function for a mission simulation.

Although current features of the Symbolic Operator Model, and the A^{3}I system in general, have been accepted as useful for evaluating various design alternatives, the evolving design has not been limited to purposes of evaluation but it is intended to remain open to investigations of other applications of the system to the conceptual design process.

The code developed in Phase IV has probably not progressed far enough to attempt applying it to an actual domain problem; however, the general framework for using goals and templates is clearly presented. It was clear during the Phase IV demonstrations that these techniques provide a significantly more powerful task decomposition methodology than was employed in previous phases. However, this methodology is also significantly more complex and labor intensive; it was not clear during the demos that the advantages the approach provides were perceived to justify the additional effort. Since many functions, methods and structure were at various stages of development at the end of Phase IV, it is recommended that the basic approach be reviewed and the requirements for a general framework be clearly stated before additional development or modification is attempted.

3.1.2 Goals and Objectives

Goals for the Symbolic Operator Model in Phase IV included development of:

- A domain independent representational framework to which domain-specific details could be added without writing system level code.

- Operator activities using multiple levels of abstraction with the lowest level defined in terms of primitive actions specified by equipment component functions.

- Explicit representations of mission specific, design-independent goals.

- Mechanisms for function allocation based on context.
- An integration framework for utilizing anthropometric, loading and scheduling models.

Objectives for Phase IV included:

- Demonstration of methods for representing key attributes of goals and activities as instance variables instead of determining attributes by inheritance from flavor components. This establishes an architecture which permits a wide range of user interfaces and editing mechanisms.
- Development of basic attributes for goal representation as opposed to development of detailed data structures for future phases. Although the structures for goal representation used in Phase IV will need to be modified or reimplemented, the mechanisms developed provide a framework for specifying future goal representation requirements.
- Demonstration of context-dependent methods for agent allocation. This objective focussed on the control of when the agent allocation functions would be evaluated: in the definition, initialization, or simulation phases.
- Demonstration of integrating Jack commands into the activities. Methods for the automatic generation of Jack commands were demonstrated in Phase IV.

3.1.3 Description

The Symbolic Operator Model provides a model-based simulation mechanism for representing an operator's interaction with complex crew station design in an attempt to satisfy mission requirements. The mechanism provides a design independent representation of mission requirements and generates a time-ordered set of operator activities. The system is unique in its ability to represent environmental context sensitivity in computing performance and load information.

3.2 USER DEFINITION

The user of the Symbolic Operator Model is considered to be a member of the design team interested in using the Symbolic Operator Model to represent simulated operator activities to evaluate the design of a crewstation.

3.3 CAPABILITIES AND CHARACTERISTICS

A requirement for the MIDAS simulation capability is that changes in mission requirements or crew station design affect mission performance and operator activity load. Mission requirements are represented as goals and provide performance metrics suitable for comparing alternative crew station designs. Operator activities control the interaction of operator models with equipment system models and provide the Task Loading Model with the data necessary for calculating the load values representing the "cost" associated with performing a given set of activities. Operator activities also provide a means of integrating other components, such as the anthropometric model, into the simulation as well as generating data necessary for graphical simulation displays.

A major architecture revision in Phase IV was the explicit representation of goals as separate data structures. In previous phases, mission requirements were implicitly represented in a task activities. In practice this made it extremely difficult to compare missions performed with alternative crew station designs since it proved difficult to separate design-independent mission requirements from design-dependent tasks and preconditions (cf.. AH-64A vs. AH-64 Longbow vs. AH-1 task analyses).

Another problem associated with implicitly representing mission requirements in tasks concerned the actual implementation used in previous phases. In these implementations, failing to perform a task (due to load constraints) resulted in less load with no loss of performance since the requirement was expressed in a activity structure never activated and only activated activities were stored for later analysis. By representing mission requirements as separate data structures, all mission requirements are visible during post simulation analysis.

In previous phases, the operator's activities were determined solely by a single hierarchical task decomposition. An assumption stated with this design were the constraints on how mission tasks could be decomposed. The mission (the top node of the hierarchy) was decomposed into phases, phases were decomposed in segments, segments into functions, functions into tasks, and tasks into subtasks (with continuing levels of subtasks into subsubtasks). A constraint on the task decomposition representation was that an operator would only be concerned with the functions and tasks of a single segment at one time. This constraint assumes that the functional interaction of defined segments can be managed in a sequential order. For example, all tasks associated with egress functions are accomplished only in the egress segment after the ingress and mission segments have been completed. However, it is extremely common to have functions clearly associated with one segment to have subtasks more appropriately performed in earlier segments. For example, entering a navigation egress waypoint during the ingress segment.

Another side effect of having a single hierarchical decomposition is that contingent behaviors associated with responding to the environment (e.g., evade radar lock) which are appropriate throughout the mission are required to be modeled as either separate segments or with multiple representations in each segments with dramatic effect on the computational tractability of the task network.

The structures developed and implemented in Phase IV address these problems while allowing these constraints as heuristics when permitted by the specific details of the selected domain application. The solution provided by the Phase IV design results from allowing multiple hierarchical decompositions of functions in which hierarchies are allowed to overlap during execution and provides structures for which a single hierarchy could be created during post-simulation processing to aid analysis.

4.0 REQUIREMENTS

4.1 REQUIREMENTS APPROACH AND TRADEOFFS

The MIDAS system is intended to be a general tool for analysis of mission requirements and cockpit design. In order to do this, it is necessary to account for the component processes of the environment, aircraft systems, mission and crew and the relational complexity of these processes. The requirements for the Symbolic Operator Model include:

- Environmental models including representation of terrain elevation, simple features such as roads, and miscellaneous friendly and hostile vehicles and systems.

- Aircraft systems models to represent the basic aircraft and component systems at adjustable levels of detail.

- A mission model which is the representation of actual mission requirements and associated doctrine and procedures as normally presented in a operations briefing. It is not a representation of the execution of a mission but rather a model which helps drive a mission simulation and the standard by which the results of a simulation are evaluated.

- Crew models to generate performance capabilities and task decomposition structures that are sensitive to context and crew station design.

The Symbolic Operator Model is built upon the premise that the design of a crew station must be evaluated within the contexts in which it will be operating. It is necessary for the designer to be able to vary characteristics of the pilot, mission and environment in order to provide the necessary contexts needed to evaluate design alternatives.

The MIDAS system contains a tick-based simulator which provides a discrete time model-based simulation of the interactions of the operator (agent), equipment and environment. The simulator has the form of a tree consisting of objects. Each object in it has a parent (or is the top-level object) and each object has component objects. At each beat of a driver clock, a tick message is sent to the top-level object (in this case, the MIDAS simulator). It performs whatever procedures it has been programmed to carry out during a tick, then passes the tick to each of its component objects. They carry out their tick procedures, then pass the tick downward, and so on. This approach was taken to meet the design goal of modularity. Anything that handles a tick message may be added to a list of component objects. This architecture allows for great flexibility and modularity in building simulations.

4.2 EXTERNAL INTERFACE REQUIREMENTS

The Symbolics Operator Model is required to communicate with external models through the Communications module, which is described in Annex J.

4.3 HARDWARE ENVIRONMENT

The Symbolic Operator Model software runs on the Symbolics 3600 series of workstations with 22.5 MB of RAM and 338 and 550 MB hard drives and an Ethernet Controller and Transceiver.

4.4 SOFTWARE ENVIRONMENT

The Symbolic Operator Model software is written under the Genera 7.2 operating system, with extensive use of Flavors for object-oriented programming. TCP/IP software communication software is required for integration with other modules of MIDAS.

A decision was made in the beginning of the phase to continue to use the Symbolics Flavor System for object-oriented programming for Phase IV development. However, the general acceptance of the Common Lisp Object System (CLOS) and the release of Symbolics Genera 8.0 with CLOS indicate that future development should probably be developed within CLOS. It was apparent throughout Phase IV that the Flavor System was probably not appropriate for long term development and that a transition to a more standard Common Lisp and CLOS should be accomplished. It is clear, in retrospect, that the decision to stay with the Flavor System for Phase IV was correct since time and resources would not have permitted a major transition. Rather than produce production/deliverable quality code, the emphasis in Phase IV was directed at exploring basic concepts including the introduction of explicit goals, the use of templates for activities and goals, and the integration of external components such as the anthropometric model. The real value of code developed in Phase IV is not how a specific solution was implemented but rather the identification of the problems. For example, a template mechanism developed in CLOS will probably be significantly different than the approach use in Phase IV but it is clear that careful attention will need to be made to how values are passed from templates to goals or activities.

4.5 REQUIREMENTS SPECIFICATION

The requirements for the Symbolic Operator Model are:

- Symbolic representation of the environment, including terrain models and other vehicles and operators to the extent that they affect the primary operator's activities.
- Integration with the representations of the operator's vehicle, crew station and associated subsystems.
- Symbolic representation of a mission for a complex, rotary wing aircraft.
- Symbolic representation of an operator performing tasks in a complex crew station in which the operator's activities are sensitive to the design of the crew station.
- Symbolic representation of concurrent activities within constraints of resources and load factors.
- Activities generated by mission requirements sensitive to context and not limited to planned sequence.
- Activity sequence, duration and load sensitive to human performance models and equipment design.
- Integration of the symbolic operator model with anthropometric, loading and scheduling models.
- Integration of symbolic models with graphical models existing on other hardware by means of an independent communications program.
- Provide a task decomposition and a simulation driving function which are dependent on the interactions of the mission, operator, vehicle and environmental models
- Task allocation based on context.
- Mechanisms for adding domain specific functions and attributes without requiring system level code development.
- Mechanisms for crew interaction.

5.0 DESIGN

5.1 ARCHITECTURAL DESIGN

5.1.1 Architectural Design Description

The symbolic modeling components of a MIDAS simulation are developed in three phases: definition, initialization and simulation phases. These phases must be performed in sequence. If a definition for a default value is changed, the initialization and simulation phases should be redone. These three phases are illustrated in Figure 1.

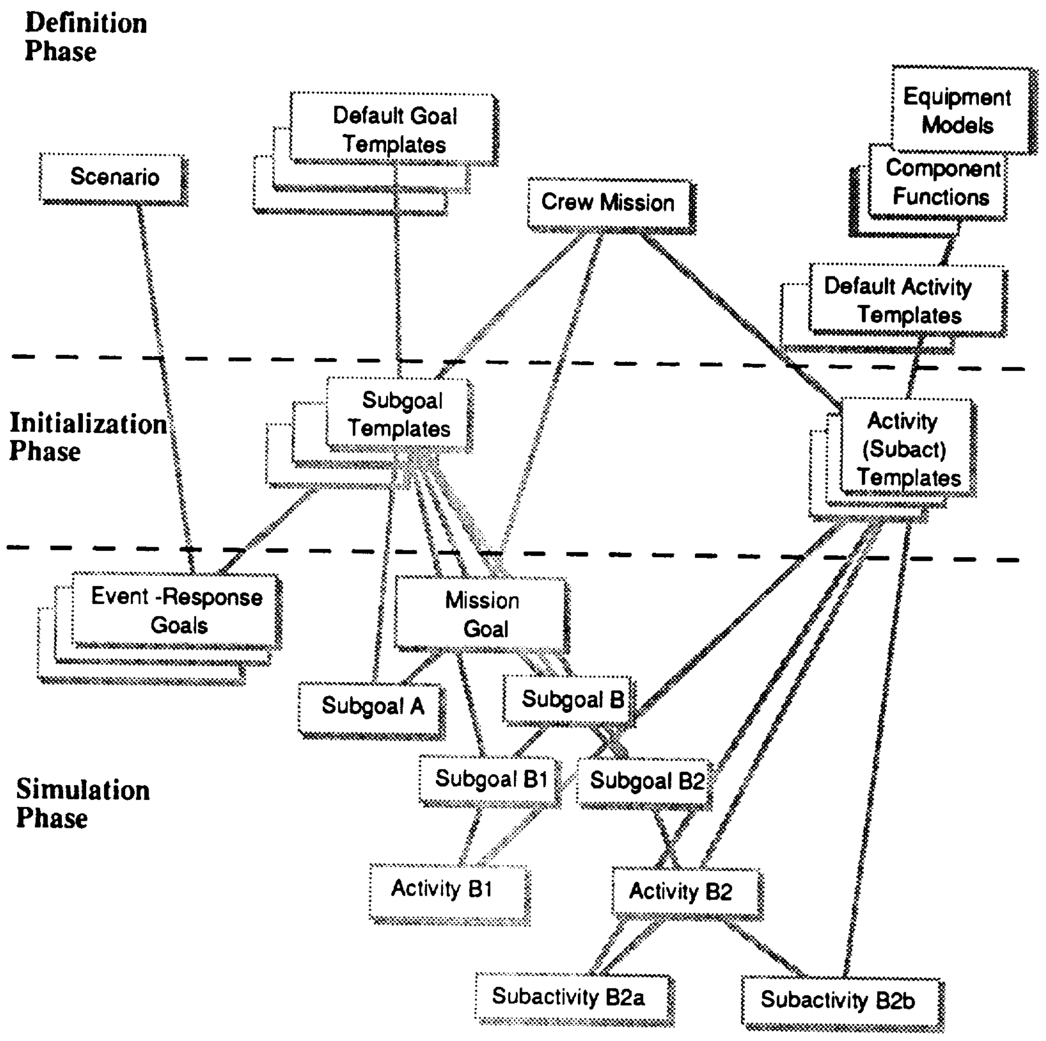


Figure 1. Symbolic Modeling Development

5.1.1.1 Environment Modeling

Although not a requirement, defining the environment model is typically the first step in developing a MIDAS simulation. The environment is modeled using a digital elevation terrain model (DEM) and a collection of static, abstract and active objects.

5.1.1.1.1 Terrain

The DEM model is defined as an object with an area specified in reference to the Universal Transverse Mercator (UTM) coordinate system and provides elevation in feet above sea level (MSL) for given x and y values. The DEM model also provides a transformation between the UTM coordinates and an internal coordinate system of x and y (in feet) used for graphics and aerodynamic modelling. A method of the DEM object is a predicate function which determines if line of sight is possible between two 3-D points and is used to determine line of sight between environmental objects. The actual structure of the DEM object may be varied as desired and terrain elevation values may be stored within the structure itself or accessed with foreign function calls to terrain models provided by the Silicon Graphics. Simulation objects should reference the terrain model by the same DEM methods regardless of whether the elevation values are stored within the DEM object or accessed by DEM methods from the Silicon Graphics machines

5.1.1.1.2 Static Objects

Static objects are used to represent environmental features whose characteristics (i.e. location, shape, etc) remain constant during a simulation. Roads and buildings are examples of potential static objects for a MIDAS simulation. Static objects require definition of their location in terms of x, y, and z values. Additional object features may be added as required by the domain.

5.1.1.1.3 Abstract Objects

Abstract objects are features of the environment, such as phase lines and operations orders, which do not necessarily physically exist and normally will not have a graphical representation in the Views module. Abstract objects may be implemented as actual object instances or implied by reference. When implied by reference, the object should be accessible by a clearly defined method (e.g. the method "(phase-lines)" provided access to the phase lines defined in Phase IV although they are actually implemented as values in a list instead of as an object).

Two abstract objects used in a MIDAS simulation include the scenario and crew-mission objects.

5.1.1.1.3.1 Scenario

The scenario object is a data structure used to store initialization values and logic for controlling the simulation during execution. This abstract object specifies the initial position of physical objects in the domain and provides a mechanism for controlling external events during the simulation.

Scenario instances are created during the simulation initialization and a new instance is created for each simulation run.

Parameters for scenario instances include the following (full structural details provided in later sections):

sim-name	=> a string representing the scenario name
sim-id	=> an unique id for each instance
envir-context	=> list of <environmental-parameter> <value> pairs For example:

```
:envir-context
          '(:phase-lines
                  ;;; A phase line called RED from grid location
                  ;;; 48008400 to 48007900 and controlled by
                  ;;; D-1/26CAV
                  ((RED 48008400 48007900 D-1/26CAV)
                  ;; A phase line called BLUE from grid location
                  ;;; 50008400 to 50007900 and controlled by
                  ;;; B-1/26CAV
                  (BLUE 50008400 50007900 B-1/26CAV))
            :acps  ;;; Air Control Points (ACP)
                  ;;; An air control point called ACP-DELTA
                  ;;; located at grid 48208030 and controlled
                  ;;; by the tactical operations center (TOC)
                  ((ACP-DELTA 48208030 TOC))
            :farp 47008000  ;;; Forward Arm and Refuel Point
                                     (FARP)
            :flot  ;;;  Forward Line Own Troops (FLOT)
                  '(54008400 54007900))
```

world-objects	=> used to designate active world objects
e-transmissions	=> time-tagged electronic transmissions. Transmitting a mission causes the message to be posted here with frequency and time tags.
initial-time	=> indicates what scenario time should be at tick 0.
current-time	=> provides central location for current time representation
current-tick	=> provides central location for current tick representation
control-events	=> events used to control simulation flow
int-cued-events	=> identifies when events should happen during the simulation
int-active-events	=> scenario controlled events in progress
int-terminated-events	=> history of terminated scenario events

5.1.1.1.3.2 Crew-mission

The crew-mission object is a data structure used to provide information normally provided by operations orders, standard operating procedures (SOPs), and doctrine. Callsigns and assigned frequencies, required reports, and planned route waypoints are among the information provided by the crew-mission. The actual representation of the planned means of executing the mission is the result of integrating the crew-mission information with the procedures of the mission goal hierarchy into a task hierarchy. This task hierarchy is created by instantiating subgoal templates with arguments supplied by the associated default goal templates and crew-mission details. These subgoal templates are used during the simulation to create the actual instances of goals. It should be noted that one subgoal template may create many different instances of a goal since values for the goal's instance variables may be determined at run-time according to context. For example, a goal for "turn-to-assigned-heading" will have different values for the heading to achieve. In fact, even the agent allocated the goal may vary during since the subgoal template may specify that the goal should be allocated to the crew member currently flying. At the time the goal is instantiated during the simulation (i.e. as result of an radio call assigning a new heading) the value for the agent-allocation parameter is determined simply by which agent is currently flying.

Crew-mission instances are created during the simulation initialization and a new instance is created for each simulation run. Parameters for crew-mission instances include the following (full structural details provided in later sections):

mission-name	=> string representing mission name
assigned-crew	=> instances of agents
mission-role	=> role symbol (e.g. 'TM1-AC1 = Aircraft 1 of Team 1)
callsign	=> assigned communication callsign (e.g. 'Y5T46)
goal-net	=> specifies top-level goal node for planned mission goals
point-of-departure	=> grid location of mission's point of departure. Note: since the simulation may represent only a segment of a mission, the aircraft may never actually be at the point of departure during the simulation.
waypoints	=> preplanned navigation waypoints
routes	=> preplanned routes
planned-route	=> preplanned route with altitude information
default-comm-assign	=> planned communication task assignments
ceoi	=> Communications Electronic Operating Instructions (CEOI) Callsigns, frequencies and communications nets.
comm-req	=> required communication tasks (e.g. report-crossing-phase-lines)

Information that is typically available to a crew member performing a mission in the real world is normally provided to the agent model representing that crew member by the crew-mission object. Information that is necessary for a simulation and does not correspond to real life missions is normally provided by the scenario object.

5.1.1.2 Active Objects

Active objects represent items which can respond to changes in the environment or have internally defined behavior resulting in state changes during the simulation. Active objects may represent functional systems, agents, or complex man-machine systems.

Functionally active objects, such as electro-mechanical systems, may be represented by defining a "tick" function to determine changes in the object's state for each increment of time during the simulation. This function is associated directly with the object and is evaluated once each time the object is sent a "tick" message. Inputs to this function may be state parameters of other objects or parameters changed by agents by means of activities.

5.1.1.2.1 Agents

Human operators are represented as agents whose behaviors are defined in terms of activities. Activities are implemented as instances of action using an object-oriented data structure and may be predefined as a set of actions in a partially ordered sequence (equivalent to a pre-defined script) or driven by goal directed behavior. Activities provide a mechanism for an agent to interact with equipment and the environment and will be discussed in detail in later sections.

Complex man-machine systems, such as other vehicles and radar sites, may be represented with explicit agent models linked to functional equipment models or abstracted to functional objects in which the crew member's control of the system is implicit in the "tick" function or by adding activities to a functional object. It is important that a MIDAS user is not constrained to one representational method and the decision of which technique is used should be determined by the domain requirements. In previous phases, convoy vehicles and other aircraft were adequately modeled with one object structure (as opposed to a "truck driver" object interacting with a "truck" object) by adding activities to the vehicle object.

Agents interact with their environment (and other agents) by means of activities. Each agent is permitted to perform concurrent and overlapping activities when resources and loading limitations permit. Agent resources in Phase IV consisted of the left and right hands and visual reference as defined by a 3D point in space. These resources were defined to be compatible with Jack, the anthropometric model provided by the University of Pennsylvania. Load limitations were based on a system which describes load in terms of visual, auditory, cognitive and motor (VACM) components. Each action has associated VACM components and an agent's VACM load components are computed by the Task Loading Model based on the agent's activities. The Task Loading Model is presented in detail in Annex C. An agent is allowed to perform concurrent activities if resource and load limitations are not exceeded and within the physical limitations of the crew station design. Extensive mechanisms have been implemented and demonstrated in Phase IV to provide a framework which insures agent activities are consistent with crew station design. This is accomplished by defining equipment specific activities with component functions defined by the Symbolic Equipment Model.

Two mechanisms were developed in Phase IV to control the selection and sequencing of an agent's activities. The first involved a generalization of mechanisms developed in previous phases for controlling activities from within other activity structures and is a logical progression of work done in those phases. The second mechanism involved controlling activities by explicitly representing goals. Goals were introduced for multiple purposes. First, to provide a means of more clearly comparing separate designs since activities of one design often do not easily map into the alternative design. Second, goals provide a means of delaying until run-time the construction of a task hierarchically (or multiple hierarchies). This allows the use of context to reduce search space. This appears especially necessary in domains where extensive contingency behavior is required. In domains where the sequencing of an operator's actions is generally predictable, the use of goals may not provide a computational advantage. The use of goals to control activities is presented in detail in a later section.

An AGENT's instance variables include the following:
(see later sections for full structure details)

ID:	=> unique id made by "AGENT"+ gensymed number (e.g. AGENT-323)
ROLE:	=> function role (e.g. PILOT or CPG)
E-R-GOALS:	=> list of instances of event-response goals
GOALS:	=> list of instances of decomposable goals
TERMINAL-GOALS:	=> list of instances of terminal goals
GOAL-FILE:	=> history of goals from the above lists which have terminated
ACTIVITIES:	=> list of instances of current decomposable activities
CUED-ACTIVITIES:	=> list of activity instances waiting to start - (activity wait list)
TERMINAL-ACTS:	=> list of instances of terminal activities.
ACT-FILE:	=> list of terminated activity instances
INT-TEMP-VACP:	=> used internally to allocate resources

RESOURCE-COMMIT:	=> record of resource allocations
VACP-HISTORY:	=> time-tagged history list

====== The remaining variables concern anthropometric modeling ==========
(see Annex G for details)

Separate instance variables for both right and left hand for:

<RH or LH>-REACH-ID:	=> reach type
<RH or LH>-REACH-SITE:	=> predefined site
<RH or LH>-X:	=> x location of hand
<RH or LH>-Y:	=> y location of hand
<RH or LH>-Z:	=> z location of hand
<RH or LH>-STATUS:	=> estimated reach time remaining - 0 = hand at site
<RH or LH>-LAST-SITE:	=> site reach initiated from

HEAD-TURN-ID:	=> specialization of reach-id
SITE-TO-LOOK-AT:	=> predefined site
HEAD-YAW:	=> direction head is turned
HEAD-PITCH:	=> angle head is tilted up or down
LAST-SITE-LOOKED-AT:	=> last predefined site to which the head was fixed

5.1.1.2.2 Activities

Activity structures are used to describe agent behavior. An activity is an object-oriented data structure which represents a specific instance of an agent's action. Agent behavior may be represented with many levels of abstractions since each activity may be decomposed into a hierarchical network of subactivities. Activity structures are either decomposable or terminal. Decomposable activities have subactivities which in turn may be decomposable or terminal. Terminal activity structures, the lowest level of activity decomposition, provide mechanisms for integrating an agent's action with other objects. The mechanisms enable a terminal activity to change either the agent's or other object's state by executing the component functions describe in the activity's structure. These component functions are defined based on equipment system design and provide the means of making an agent's behavior sensitive to design. Component functions are described in Annex D.

Each activity has a discrete temporal extent with associated times for initiation, start, and termination. Activity initiation time represents the earliest time an activity could have started. Initiation time may vary from start time due to delay in starting an activity because of resource constraints. Activities which have been initiated but not yet started are classified as cued-activities. Activities which have started are classified as active activities. Termination time represents when an activity finished which could result from completion or interruption. Since activities have discrete temporal extent, an action which is interrupted and resumed is represented with two activity object instances with the relationship between the two expressed explicitly in the activity at the next higher level of abstraction.

Control of activity initiation, starting and termination is accomplished with initiation, start and termination conditions defined in the activity structure. A number of mechanisms have been provided which reduce the effort necessary to define activities by allowing initiation, starting and termination conditions to be defined once for a set of activities in the structure of the activity at the next higher level ("parent activity") of abstraction. These mechanisms, which include definitions of sequential, parallel and rotational activities, are described in detail in later sections.

5.1.1.2.2.1 Script-Based Agent Behavior

As in previous phases, control of an agent may be accomplished by providing the desired control logic within the structures of activities. This is accomplished by specifying an activity as the top node in a hierarchy. This activity specifies subactivities and the temporal relations between each of the subactivities. The temporal relations may be expressed as a general relation which applies to the subactivities in the order that they are specified. For example, top-level activity X is specified as a sequential activity and specifies subactivities A, B, and C. In this example, subactivity A will start when the activity X starts. Subactivity B will start after subactivity A has been completed and C will follow B likewise. Rotation activities would work in a similar manner. Parallel activities actually are implemented in a different manner than in previous phases. In this phase, a default logic was implemented in which all subactivities will be started unless otherwise constrained (i.e. by a sequential relation specification, resource constraints, etc). As a result, an activity specified as a parallel activity simply allows each subactivity to start. The main difference between this approach and that taken in previous phases concerns the specification of what had been called parallel-stop activities. In previous phases, a parallel activity would continue to be active until all subactivities were completed. Parallel-stop activities, however, stop as soon as any subactivity completed. This mechanism was provided by specifying parallel-stop as a flavor component of the activity. In this phase, however, parallel-stop activities are specified by including in the termination conditions a test which returns "true" if any subactivity is completed. The differences between parallel and parallel-stop activities in this phase from those in other phases are related to implementation and conceptually remain the same.

This new default logic, however, provides a more complex and flexible mechanism for controlling subactivities than those provided previously. In previous phases, temporal relationships between the subactivities were limited to a single term which was applied to all subactivities. Using the new default logic, it is now possible to express temporal relations (actually any type of constraint) which can be applied to a subset of the subactivities (i.e. C follows A independent of the status of B). These temporal relations can be point-based or interval-based (using James Allen's temporal intervals). Also, non-temporal constraints, such as resource constraints, can be used to control the sequencing of activities. This basic mechanism provides a means of implementing any type of script-based scenario.

5.1.1.2.2.2 Goal-Directed Agent Behavior

A major development in Phase IV was the introduction of explicit goals. Goals were introduced as a means of controlling the generation of top-level activities enabling an agent to have multiple activity hierarchies. There were many factors that influenced this decision. First, activities are intended to represent actions that the agent performs and a separate structure was desired to represent states that should be achieved during a mission. An alternative approach would be to use the same activity structures. This raises problems, however, when tasks are shed or neglected. Additional problems arise during multi-agent interaction when a side effect of one agent's action achieves the states that another agent is trying to achieve. In this case, the "goal" has been achieved but it is unclear which agent's activity structure should represent this. By explicitly representing goals as separate structures, it is easy to represent shed/neglected tasks and, in the multi-agent example, represent one agent's activity achieving another agent's goal. Second, it seem advantageous to have an representation that explicitly represented the standards since alternative crew station designs may have different activities for the same mission. Hence, the goals are intended to represent design independent standards by which competing designs could be compared.

Goals in Phase IV were represented in a manner similar to activities. A goal may have subgoals and similar expression of temporal relations was provided. A goal which has subgoals specified is referred to as a decomposable goal. At the lowest level in a given goal hierarchy are terminal goals which have no subgoals but specify states that need to be achieved or maintained. These states are expressed in terms which are mappable to states achieved by an agent's activities or equipment systems.

Goals are characterized from a number of perspectives. First, as explained above, goals are either decomposable or terminal. Second, goals are either achievement or maintenance goals. Third, goals are classified as to whether they are satisfied. Fourth, goals may be either current, active or inactive. Fifth, goals may have either abstract or explicit function allocation.

Achievement goals specified a state (e.g. "target detected") and the goal is considered satisfied as soon as this state is achieved. Achievement goals are classified as either satisfied or not-satisfied.

Maintenance goals specify states (e.g. "maintain heading 135") which must be maintained for a specified duration. Maintenance goals are classified as:

currently-satisfied	=	state maintained but duration of goal has not yet been completed
not-currently-satisisfied	=	state not maintained but duration of goal has not yet been completed
satisfied	=	state maintained and duration completed
not-satisfied	=	state not maintained and duration completed.

Goals are considered current when it is possible to test if the desired state has been achieved. A goal to "pass over point A at 5000 feet" can only be tested at the time the aircraft passes over point A.

Goals are considered active when the desired states of the goal influence actions currently being taken. In order to achieve "pass over point A at 5000 feet", some actions (i.e. establish descent) may need to be achieve prior to the time of passing A. The requirements for satisfying the preconditions of a goal are identified when a goal becomes active. Inactive goals are merely goals which do not influence current behavior.

Goals have explicit function allocation when the goal identifies a unique agent (e.g. "the pilot"). A new feature was introduced in Phase IV which enables goals to be dynamically allocated based on context by specifying an abstract function allocation. An abstract function allocation is accomplished by specifying a set of agents and a Lisp form to be evaluated at run time which will select one of these agents based on context. For example, a goal could be assigned to the crew (i.e. the set of pilot and cpg) and provide a form which selects the crew member with the lowest task requirements.

During a simulation, each agent responds to a tick message in the following manner:

1. Update Current-Time.
2. Delete any goals no longer justified or completed from active and matched goal lists.
3. Identify new event-response goals.
4. Add newly activated goals from the mission hierarchy.

7. Identify matched-goals with terminated or completed activities. Move goal from matched goal list to active goal list.
7. Map activities to goals in active goal list. Add action to activity wait list.
8. Sort activity wait list based on goal priorities.
9. For each activity in wait list, start activity when resource, load, and temporal constraints permit.
10. Test termination conditions to each current activity - delete activity when conditions indicate.
11. Execute tick procedure for each active activity.
12. Signal simulation executive cycle completed.

In previous phases, relationships between activities were determined by the details of a single hierarchy and contingency behavior was represented as branches of the hierarchy. If an event that indicated some activities should be performed had the potential of occurring at different segments of the mission, the activity for responding to this event had to either be modeled as a separate preemptive segment or modeled repetitively in each segment that the event potentially could occur. As a result, adding contingency behavior to respond to multiple events possibly occurring throughout the mission would produce an activity hierarchy that would be hard to manage if not intractable.

The approach used in Phase IV addresses this problem by allowing multiple activity hierarchies. From a planning perspective, the mechanisms provide means of resolving conflicts arising from resource competition. The approach does not resolve conflicts arising from conflicting state sequences such as the Sussman Anomaly (Sussman, 1973), however, in many situations this does not present a problem since many concurrent activities do not involve states that intersect. For example, the interactions between concurrent activities of transmitting a radio message and updating a navigation system involves resources more than equipment state dependencies. Since resource commitments are made at the time an activity is activated, the approach is in some ways similar to planners allowing partial ordering (Sacerdoti, 1975) and constraint posting (Stefik, 1981). The objective in Phase IV was not to develop or implement an actual planner but to demonstrate how such systems could be integrated into the simulation. The state dependencies of different functions on a Multi-Function Display clearly indicate a requirement for approaches that address the issues of conflicts between subgoal state sequences. It is believed that such approaches could be integrated into the framework of the agent's tick method presented above.

5.2 DETAILED DESIGN

5.2.1 Detailed Design Approach And Tradeoffs

5.2.1.1 Activity Definition

A key objective of the Symbolic Operator Model is the development of an agent model whose behavior is sensitive to changes in crew station design. In order to achieve this, terminal activities are defined using equipment component functions as the mechanisms for the agent to interact with equipment models. Equipment component functions are defined during the process of developing an equipment model (see Annex D). These component functions specify state changes which should occur as result of a specified action. By defining terminal activities using these component functions, any change in the crew station equipment models will have a direct effect on the agent's behavior. The state changes resulting from executing an equipment component function are specified in a manner that clearly represents the duration of the action and how state is changed during each "tick" of time during the action. Within an terminal activity, equipment component functions are executed in sequential order.

In order to provide an activity modeling environment in which activity definitions can be described and changed without encountering the problems associated with changing mentioned previously concerning flavor definitions, a mechanism was developed in Phase IV to aid the process of activity definition. The key feature is the use of template data structures instead of flavor definitions. Templates are used to define default activity parameters and specify global changes to an activity class definition, for specialization of both class and instances to provide exceptions to activity class default values, and to make activity instances context sensitive.

Templates are provided at two levels:

- First, default templates which provide a definition of a general activity class.
- Second, subact templates which provide specialization of default activity templates based on the context of the mission and the environment.

Default activity templates are defined by the users during a pre-simulation activity definition phase which must be accomplished after the equipment models have been defined. Subact templates are later automatically generated during a pre-simulation activity initialization which must be performed after activity definition and mission definition phases have been completed.

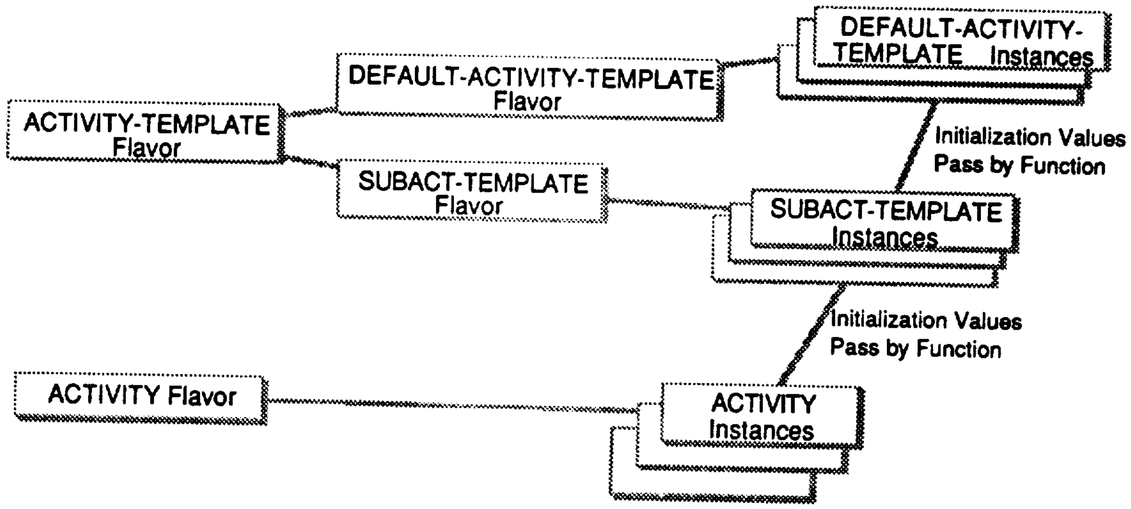
If a user wants to change a parameter value associated with all instances of a specific activity class, the specifications in the default activity template should be changed. The default activity templates for decomposable activities indicate how sub-act templates should be created for its subactivities. If the subactivities are to be created with the values specified by a default activity template, the user need only specify the name of the activity and a sub-act template will be created using the default values. If the user, however, wants to make changes to the default values when certain conditions apply or for a subset of an activity class, the user can specify conditions in the parent default activity template that one its subactivities should be created from a template with values other than the defaults (see later sections for details). These mechanisms provide logic for controlling inheritance both globally and locally. The alternative approach of defining separate activity templates for both global and local definitions would require producing a significant larger number of user defined templates and possibly obscure whether the intention was to redefine the class or just define a subset with minor variations.

Default activity parameters are defined using DEFAULT-ACTIVITY-TEMPLATES. These templates are instances of the flavor DEFAULT-ACTIVITY-TEMPLATE and the desired activity parameters are specified as values of instance variables. These activity template instances define a class definition for typical activities. Unlike previous phases in which changes in the definition of a class of activities required recompiling flavor definitions, class definitions can be modes. Class definitions can be modified simply by changing an instance variable.

> It should be noted that the approach using flavor definitions provides a means of changing previously defined instances when a class definition is changed. This, however, does not seem to offer any significant advantage since reinitializing activity instances after changing class definition appears to be computationally tractable and logically in sequence.

After default activity templates and the mission have been defined, sub-act templates are generated during an initialization process. These templates are used to create the instances of activities which are instantiated throughout a simulation. Since these templates may provide functions to be evaluated at run-time for determining the value of an activity parameter, this approach provides a flexible mechanisms for context dependent activities.

Structurally, any default activity template can be used to represent the top-level node in an activity hierarchy. Normally one default activity template is defined to represent a procedure associated with a specific domain function (e.g. "enter NAV waypoint"). This activity template will specify the next lower level activities in the activity hierarchy with lower level activities defined in the default activity templates for the indicated subactivities. As described in previous sections, terminal activities represent the bottom nodes of an activity hierarchy which is the level at which an agent's actions interact with equipment. This process is recursive until the indicated subactivities are terminal activities. The resultant activity hierarchy represents the desired procedure.



DEFAULT-ACTIVITY-TEMPLATE Instances
DEFAULT-ACTIVITY-TEMPLATE Flavor
ACTIVITY-TEMPLATE Flavor
Initialization Values Pass by Function
SUBACT-TEMPLATE Flavor
SUBACT-TEMPLATE Instances
Initialization Values Pass by Function
ACTIVITY Flavor
ACTIVITY Instances

Figure 2. Activity Data Structures

5.2.1.2 Activity Templates

Both DEFAULT-ACTIVITY-TEMPLATES and SUB-ACT-TEMPLATES are defined with ACTIVITY-TEMPLATES as a component flavor, as illustrated in Figure 2. ACTIVITY-TEMPLATES include the following parameters (full, detailed structure is describe in later sections):

act-name	=> a string representing the activity name
act-short-name	=> a string abbreviating activity name - for graphing
agent-allocation	=> which agent should perform
equip-context	=> specifies required crew station design
act-keys	=> provides a generalized parameter - value list structure: <parameter-1> <value> ...<parameter-n> <value> This enables new parameters to be defined without recompiling.
act-goal-keys	=> used to pass parameters to activity instances.

act-type	=> indicates general temporal relationships of subactivities "sequential" specifies subactivities should be processed in sequence. "parallel" specifies all subactivities should start when activity starts assuming constraints allow. Indicates subactivities are anticipated to be all started at once. "rotation" specifies sequential order which is to be repeated until termination conditions cause activity to stop. "complex" specifies all subactivities should try to start. Indicates constraints are anticipated to delay the start of some activities.
required-resources	=> specifies required-resources. activities may be constrained from starting when a required resource is not available. These resources include: visual auditory cognitive motor left / right hand point-of-fixation (look-at site)
functions	=> high level text description of activity for display purposes.
priority	=> currently user defined priority.
initialization-procedures	=> procedures to be run when activity is initialized.
preconditions	=> state-activity pairs for indicating possible preconditions/actions.
com-fcn-procs	=> defined only for terminal activities. Specifies a list of equipment component functions which to be executed in sequential order.
explicit-local-constraints	=> will be tested when activity tries to start. Indicates subactivities are anticipated to be delayed from starting.
estimated-duration	=> expressed in ticks - 100 ms.
start-procedures	=> procedures to be run when activity is started.
tick-procedure	=> specifies forms to be evaluated each tick.
termination-conditions	=> specifies forms to evaluated each tick. When one of the forms returns "TRUE", the activity will be will be terminated.
termination-procedures	=> specifies forms to be evaluated when the activity terminates.
vacp-data	=> context-free estimate of VACP values for this activity.
matching-pattern	=> used to match activities to goals.
sub-act-list	=> used only for decomposable activities.

5.2.1.3 Activity Instances

Activity instances are created during the simulation and represent a unique instance of an agent performing action. Parameters for activity instances include the following (full structural details provided in later sections):

act-name	=> a string representing the activity name.
act-id	=> string produced by ACT<gensym-number> (e.g. ACT425).
agent-allocation	=> which agent should perform.
act-keys	=> provides a generalized parameter - value list structure: <parameter-1> <value> ...<parameter-n> <value> This enables new parameters to be defined without recompiling the template definition.
act-goal-keys	=> used to pass parameters to activity instances.
act-type	=> indicates general temporal relationships of subactivities "sequential" specifies subactivities should be processed in sequence. "parallel" specifies all subactivities should start when activity

	starts assuming constraints allow. Indicates subactivities are anticipated to be all started at once. "rotation" specifies sequential order which is to be repeated until termination conditions cause activity to stop. "complex" specifies all subactivities should try to start. Indicates constraints are anticipated to delay the start of some activities.
required-resources	=> specifies required-resources. activities may be constrained from starting when a required resource is not available. These resources include: visual, auditory, cognitive, motor, left-hand, right-hand, point-of-fixation (look-at site)
priority	=> currently user defined priority.
initialization-procedures	=> procedures to be run when activity is initialized.
preconditions	=> state-activity pairs for indicating possible preconditions/actions.
com-fcn-procs	=> defined only for terminal activities. Specifies a list of equipment component functions which to be executed in sequential order.
estimated-duration	=> expressed in ticks - 100 ms.
start-procedures	=> procedures to be run when activity is started.
tick-procedure	=> specifies forms to be evaluated each tick.
termination-conditions	=> specifies forms to evaluated each tick. When one of the forms returns "TRUE", the activity will be will be terminated.
termination-procedures	=> specifies forms to be evaluated when the activity terminates.
vacp-data	=> context-free estimate of VACP values for this activity.
mapped-goal	=> used only for top level activities - an instance.
tick-started	=> time (in ticks) activity was started.
tick-ended	=> time (in ticks) activity ended.
sub-activities	=> used only for decomposable activities - active instances.
activity-history	=> used to store terminated subactivities.

Mechanisms have been developed which provide for automatic generation of anthropometric model commands. When a terminal sub-act template is instantiated, the "make-instance after" methods tests each procedure in the list of equipment component function procedures and appropriate anthropometric commands are inserted.

EXAMPLE:

Assume an activity is defined with the following component function procedures:

comp-fcn-procs => (cfp-1 cfp-2 cfp-3 cfp-4)

cft-1 representing a component function of switch "X"
cfp-2 representing a component function of display "Y"
cfp-3 representing a component function of switch "Z"
cfp-4 representing another component function of switch "Z"

Testing the first procedure, "cfp-1", the equipment model for switch "X" is queried for a predefined site location and a command is generated for the appropriate "reach-for" action. Testing "cfp-2" would generate a "look-at" command. It should be noted that both "cfp-3" and cfp-4" will generate the same reach command, resulting in the following list (code simplified):

```
comp-fcn-procs => ( (reach-for "X")
                    cfp-1
                    (look-at "Y")
                    cfp-2
                    (reach-for "Z")
                    cfp-3
                    (reach-for "Z")
                    cfp-4 )
```

It should be noted that unless some other activity is overlapping, the last reach command will be asking for a hand to move to its current position. As detailed in the Jack documentation, Annex G, a request for a movement that is currently achieved will be signaled by a completed message during that tick. This results in the mechanism being context sensitive in that only movements that have to be made will have a duration beyond one tick. The one tick duration of the null movements has not seemed significant and this mechanism appears sufficient for modeling movement which is conditional upon the initial state of the required resources.

It should also be noted that at the end of the activity, one hand would remain on switch "Z". If it is desired for the hand to return to a neutral position, it can be handled in two ways. First, the user defining the activity can specifically include the movement during the initial specification as follows:

```
comp-fcn-procs =>  ( cfp-1 cfp-2 cfp-3 cfp-4  (reach-for <site-n>) )
```

Second, the user can define an event-response goal indicating that the hand should return to a predefined neutral position after a specified period of inactivity.

5.2.1.4 Goal Definition

In order to provide goals which definitions can be described and changed without encountering the problems associated with flavor definitions, a mechanism was developed in Phase IV to aid the process of goal definition that is closely related to the process used for activity definition. As with activity definition, the key feature is the use of template data structures instead of flavor definitions. Templates are used to define default goal parameters and specify global changes to a goal class definition, for specialization of both class and instances to provide exceptions to goal class default values, and to provide context sensitive activity instances.

Templates are provided at two level:

- First, default templates which provide a definition of a general goal class.
- Second, subgoal templates which provide specialization of default goal templates based on the context of the mission and the environment.

Default goal templates are defined by the users during a pre-simulation goal definition phase. Subgoal templates are later automatically generated during a pre-simulation initialization which must be performed after the mission definition phase has been completed.

If a user wants to change a parameter value associated with all instances of a specific goal class, the specifications in the default goal template should be changed. The default goal templates for decomposable goals indicate how sub-goal templates should be created for its subgoals. If the subgoals are to be created with the values specified by a default template, the user need only specify the name of the goal and a sub-goal template will be created using the default values. If the user, however, wants to make changes to the default values when certain conditions apply or for a subset of a goal class, the user can specify conditions in the parent default goal template that one its subgoals should be created from a template with values other than the defaults (see later sections for details). This provides logic for controlling inheritance both globally and locally.

Default goal parameters are defined using DEFAULT-GOAL-TEMPLATES. These templates are instances of the flavor DEFAULT-GOAL-TEMPLATE and the desired goal parameters are specified as values of instance variables. These template instances define a class definition for typical goals. After default goal templates and the mission have been defined, sub-goal templates are generated during an initialization process. These templates are used to create the instances of goals which are instantiated throughout a simulation. Since these templates may provide functions to be evaluated at run-time for determining the value of a goal parameter, they provides mechanisms for context dependent goals.

In a manner similar to activities, structurally, any default goal template can be used to represent the top-level node in a goal hierarchy. Normally one default goal template is defined to represent the planned mission. This goal template will specify the next lower level goals in the goal hierarchy by specifying the desired subgoals in a list structure. This list structure is compose of symbols and/or sublists. The symbols in the list refer to names of default goal templates which in turn can specify a lower level of subgoals. The sublists provided a default goal template name as the first element and the remaining elements are used as arguments to override default values provided by the default template. This process is recursive until the indicated subgoals are terminal goals. The resultant goal hierarchy represents the planned mission.

Although some contingencies can be incorporated into the planned goals, another mechanism was developed in Phase IV using event-response goals. Event-response goals are similar to other goals with the exception that a state is specified that determines when the goal becomes active for an agent. When activated, an event-response goal creates an additional goal hierarchy (although possibly only one level deep) for an agent. The number of event-response goals which may be assigned an agent is unlimited and the agent's ability to handle multiple goal hierarchies is constrained primarily by the agent's resources. The current system requires the user to define resolution methods for conflicting goal state requirements.

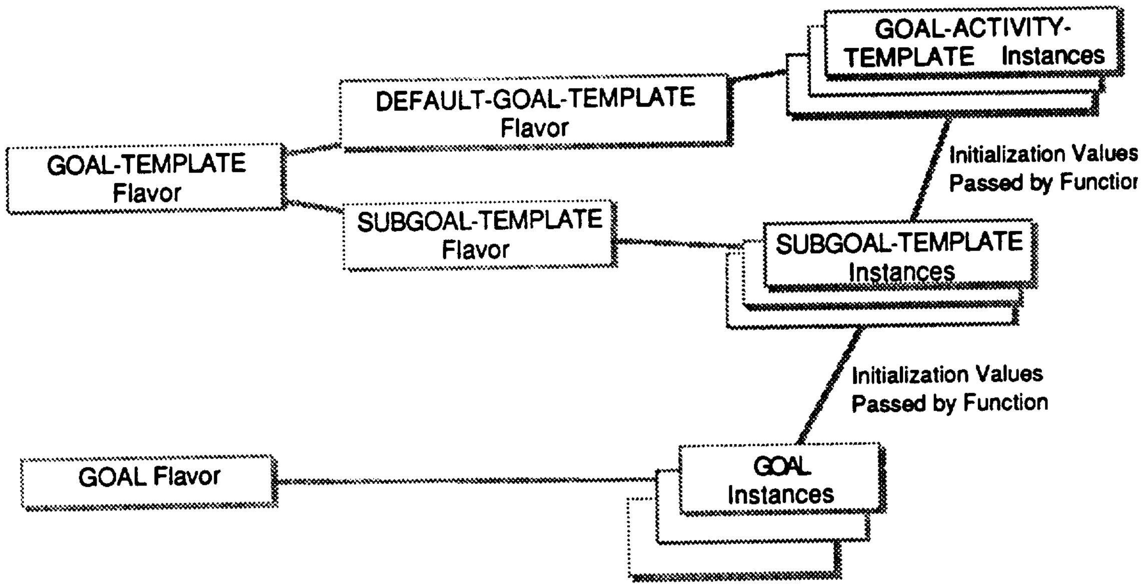

GOAL-ACTIVITY-TEMPLATE Instances
DEFAULT-GOAL-TEMPLATE Flavor
GOAL-TEMPLATE Flavor
Initialization Values Passed by Functior
SUBGOAL-TEMPLATE Flavor
SUBGOAL-TEMPLATE Instances
Initialization Values Passed by Function
GOAL Flavor
GOAL Instances

Figure 3. Goal Data Structures

5.2.1.5 Goal Templates

GOAL-TEMPLATE is a flavor that is used as a flavor component of the flavor definitions for both DEFAULT-GOAL-TEMPLATE and SUB-GOAL-TEMPLATE definitions, as illustrated in Figure 3. The following instance variables are common to both:

goal-name	=> a symbol representing a goal name.
context-name	=> name with context added
goal-id	=> creates unique id
goal-ref-id	=> a unique id set by the default goal templates
goal-level	=> highest goal is level 1
goal-priority	=> larger values have priority
terminal-p	=> achieve or maintain if terminal
goal-keys	=> misc keys
matching-pattern	=> pattern to match goal to design activities
matched-act-template	=> template indicated by pattern above
event-response	=> what happens to template after event is true
goal-test	=> function indicating achievement status
agent-allocation	=> who is responsible for this goal
agent	=> instance of agent
functions	=> intended state changes achieved by activity
activation-tests	=> event tests for event-response goal
currency-tests	=> when is this goal actually a requirement
initialization-procedures	=> to be completed when goal is instantiated
preconditions	=> list of state/goal-templates pairs necessary
contin-conditions	=> list of state/goal-templates pairs necessary
sub-goal-list	=> list of goal names and keywords
tick-procedures	=> what to do each tick
termination-conditions	=> when to end

termination-procedures	=> what to do when I end
temporal-relations	=> describes temporal relations of goals

DEFAULT-GOAL-TEMPLATEs include the above instance variables and the following two variables:

source-file	=> file defining this goal
subgoals-func	=> functions for determining mission subgoals

SUB-GOAL-TEMPLATEs include the instance variable defined by GOAL-TEMPLATE and the following two instance variables:

parent-template	=> goal templates creating this subgoal
sub-goals	=> subgoals

5.2.1.6 Goal Instances

Goal instances are created during the simulation and represent a unique instance of a task requirement. Parameter values are inherited from the related goal template or derived by an function provided by the template. Parameters for goal instances include the following (full structural details provided in later sections):

goal-name	=> goal name.
context-name	=> name with context
goal-id	=> creates unique id
parent-goal	=> higher-level goal
mission	=> instance of crew mission
goal-level	=> highest goal is level 1
goal-priority	=> larger values have priority
terminal-p	=> nil when decomposable, t if terminal
goal-keys	=> misc keys
act-goal-keys	=> used to pass arguments to methods creating activities
matching-pattern	=> pattern to match goal to design activities
matched-activity	=> instance of activity result of pattern above
event-response	=> t if this goal is an event response goal
status	=> describes activity response to goal
achievement-status	=> specifies if goal is satisfied
template	=> goal template specifying default values
agent	=> who is responsible for this goal
sub-goal-templates	=> templates used to create sub-goals
sub-goals	=> instances of goals
activation-tests	=> event tests for event-response goal
currency-tests	=> when is this goal actually a requirement
current-p	=> tick when this goal became current
tick-created	=> tick goal created and initialized
tick-activated	=> when goal became an active goal
tick-terminated	=> when goal became inactive
precondit	=> list of state/goal-templates pairs necessary
contin-condit	=> list of state/goal-templates pairs necessary
goal-test	=> function indicating achievement status
achieved-p	=> t or nil list for maintenance goals
tick-procedures	=> what to do each tick

termination-conditions	=> when to end
termination-procedures	=> what to do when goal ends
data-pit	=> used to store data concerning state values

5.2.1.7 Goal/Activity Matching

A key feature of the Phase IV approach to agent modeling is the mechanism developed for the agent to select an activity as an attempt to satisfy a given goal. Since a primary objective of the MIDAS system is to have agent behavior sensitive to changes in either mission or crew station design, a flexible mechanism was needed to match each mission goal to a variety of activities associated with alternative crew station designs. This mechanism should also be sensitive to situations in which the mission is varied and specific design activities can be used for a variety of missions. The approach used in Phase IV involves the use of matching patterns to map activities to goals. Matching patterns are defined for each goal and this pattern is used by a matching function to identified the appropriate activity. Matching patterns may be explicitly defined in the default goal templates during the definition phase:

```
matching-pattern => '(cpg send :FM1 msg)
```

They may also be defined by functions specified during the definition phase which will produce a context sensitive pattern at run time :

```
matching-pattern => (,(comm-task-assign *mission* 'flt-coord)  ;;; backquote implied
                     send
                    ,(task-radio-type *mission* 'flt-coord
                            (comm-task-assign *mission* 'flt-coord))
                    msg)
```

In this example, the forms preceded by the comma (backquote implied in code not shown) are evaluated at run time within the context of the mission scenario. The first form evaluated, ",(comm-task-assign *mission* 'flt-coord)", accesses the crew-mission object by the global variable *mission* and checks to see which agent is currently asssigned flight coordination ("flt-coord") responsibilities. The second form evaluated: ",(task-radio-type *mission* 'flt-coord (comm-task-assign *mission* 'flt-coord))", identifies which radio should be used for this task. The resulting pattern could be one of the following:

```
matching-pattern => '(pilot send :FM1 msg)
matching-pattern => '(pilot send :VHF msg)
matching-pattern => '(pilot send :UHF msg)
matching-pattern => '(cpg send :FM1 msg)
matching-pattern => '(cpg send :VHF msg)
matching-pattern => '(cpg send :UHF msg)
```

These matching patterns are then passed as an argument to a agent's matching method, (MAP-ACTIVITY-TO-GOAL AGENT) which returns the appropriate activity instance. In Phase IV, simple hashing was used in the matching method since it serve the intended purpose of demonstrating how the mapping could be context sensitive. For more complex problems, a more powerful mapping function may be desired such as one using a many sorted logic. New mapping techniques may be easily integrated simply by redefining the agent's "MAP-ACTIVITY-TO-GOAL" method.

An important feature of using the matching-pattern approach is that goals and activities can be defined in a modular manner and links between goals and activities defined later and will allow separate and parallel development. This is important since it became clear during Phase IV that a symbolic modeling approach to simulation is extremely labor intensive and will require a significant team to work on all but the most trivial design models.

5.2.1.8 Integrating Jack

An anthropometric model developed at the University of Pennsylvania, commonly referred to as Jack, was integrated with the Symbolic Operator Model in Phase IV. Jack was used to model body movements and drive graphical displays during the simulation. By incorporating Fitt's Law into the reach movements, Jack was able to provide a model-based estimate of the duration required to perform various reaches. Head movements were also provided for the simulation; however, these movements were either made by having the head follow the hand during a movement or by simple shift of position which would be accomplished in one tick (100 ms). Integration of Jack was accomplished by predefining a set of three dimensional sites and having terminal activities generating commands to reach (or look) at a given site.
These commands were implemented in the form of component functions and could be intermixed with component functions defined by equipment. The duration of these commands would be determined by the Fitt's Law estimate provided by Jack. The commands generated a data set which was passed to a data pool (see Annex J).

The data set generated by these commands have the form:

(<data-pool-identifier> <reach-type> <reach-site>)

<data-pool-identifier>	=> an integer
<reach-type>	=> an integer
<reach-site>	=> a string. The first character of the string is significant and limited to the following rules: "R" identifies the site is on the right multifunction display "L" identifies the site is on the left multifunction display "x" is used for all other sites

When a command is generated to reach to a new site, the reach type determines if the reach is made from the shoulder or the waist by checking the reach type code which also determines if the head will follow the hand during the movement. After the initial tick, the data pool will update the appropriate variables for the agent including a variable for the "X", "Y", and "Z" position of the hand and the status variable. The status variable indicates the remaining time to finish the movement and will indicate "0" when the movement has been completed. Agent objects can check to see if a hand resource is committed simply by checking the the status variable for that hand.

It should be noted that many movements are affected by the last movement performed. For example, during testing a command was generated which cause the CPG agent to reach down and to the rear of where the hand had been located. This cause the anthropometric model to generate a bending of the wrist which seemed appropriate for that movement. However, the next command generated a reach to a site on the multi-function display. During the execution of this movement, the wrist was not straightened and, since the reach was for the tip of the finger to touch the screen, most of the hand was "pushed" through the display so the finger could make contact. There were also some movements of the head to the rear that should have been possible but appear illegal since the head would fall off. During Phase IV, these problems were interesting but not serious problems since realistic movements could be generated after some trial and error. A more serious issue arose from the requirement to predefine 3-D sites. The number of sites in a typical crew station could be extremely large and the requirement to define a "site" after having defined symbolic and graphical models seems redundant. Generating predefined sites to "Look-at" outside the aircraft presented obvious problems if it was desired to have the agent look at a moving object. (See the Jack documentation, Annex G, for a complete discussion of the anthropometric model.)

5.2.1.9 Integrating the Task Loading Model

One intended use of a MIDAS simulation is the comparison of the workload imposed by alternative crew station designs. This is accomplished by associating load attributes with each activity and providing a function which determines how these individual activity loads are combined to represent the agent's load. Currently, load attributes are characterized into one of four channels: visual, auditory, cognitive and motor (VACM). As in previous phases, the Symbolic Operator Model provides the user a means of specifying the VACM values of an activity during the definition phase by either specifying the individual values or providing a function which will be evaluated at run time to produce context sensitive values. The module also provides a means of combining these activity load attributes into agent load attributes by a simple function which is typically a additive function.

This approach requires extensive effort on the part of the user if it is desired to reasonably show how activities interact to produce combined load values that are not simply additive. A new module, the Task Loading Model (TLM), was developed in Phase IV to address this issue. The TLM module can be dynamically integrated with the Symbolic Operator Model by passing the list of concurrent activities to the TLM module as each new activity is started. The TLM module then returns a set of values representing the combine VACM load for the agent. This combine load value is then used by the agent model to determine if load requirements are sufficiently low to allow other activities to start.

The TLM module can also be statically integrated with the Symbolic Operator Model by allowing a full simulation to proceed use the default additive method of combining load attributes and then using the TLM model to perform a post-simulation analysis to indicate where overloads would have been experienced.

Both of these approaches require that base estimates of individual load attributes for each activity be estimated during the definition phase. Additional requirements are imposed by the TLM models concerning requirements in the definition phase. For full details of these requirements, see the Task Loading Model documentation, Annex C.

5.2.1.10 Integrating the Scheduler

The agent model allocates resources to activities as the activities are activated. For activities which are activated at the same time, resources are allocated according to the priority of the associated goal. In situations where some goals may have a window of opportunity, this strategy may prove inefficient since reordering of activities may allow more effective allocation of resources. In Phase IV, a scheduler was developed to provide additional strategies for resource allocation. The default strategy provided by the agent model is essentially a constraint-based, least-commitment approach which will sequentially allocate any available resources to a set of activities which have sorted by goal priority. The new Z-Scheduler was designed to be integrated into the simulation by allowing the scheduler to post additional constraints prior to the actual allocation of resources. In this way, the scheduler can direct resources to be first allocated to an activity with a lower priority if the higher priority activity competing with that resource will still have time to be completed within a specified time window. This provides of means of intermixing the subactivities associated with multiple goals in a manner to minimize time or to maximize use of resources.

Although the integration of the scheduler was not shown during the demo, a simple mechanism was tested at various times during development. This mechanism is clearly inefficient but, since the mechanism can be replaced with more efficient modules without having to make other changes, it provides means of evaluating the use of this type of scheduler in a simulation.

The integration approach used during Phase IV development was to modify the agent's tick method in the following manner. Normally, after the activities are sorted by goal priority, resources are allocated to each activity in sequence unless that activity is restricted from starting by a constraint. The modified agent's tick method added a new step after the activities were sorted and before the activities were allocated resources. This new step wrote activity details, including constraint descriptions, to a file which was read by the Z-Scheduler. At this point the agent model waits until a new file is written by the Scheduler. The Scheduler reads the activity description file and evaluates it as a scheduling problem. The solution developed by the Scheduler is expressed as a set of additional constraints which are written to the file that the agent model is waiting to read. Upon reading the new constraints from the file written by the Scheduler, the agent proceeds in the normal manner of activating activities and allocating resources as permitted by constraints which now includes the new constraints. Although the method of passing data by writing to a file is clearly simplistic and inefficient, it can be replaced by a more sophisticated method without requiring changing other mechanisms.

It should be noted that the Z-Scheduler used in Phase IV takes a series of tasks and constraints as input and returns a amended constraint list as its output. Additional approaches will need to be developed to address situations in which the ordering/reordering of activities result in task shedding. Also, since the scheduler requires estimates of each task's duration which is independent of task sequencing, the scheduler is not sensitive to issues concerning how the duration of various activity schedules are effected by reach times or preconditions.

5.2.1.11 Output Available for Analysis

During a MIDAS simulation, each terminated goal and activity is stored for later analysis. Goals provide details needed for performance since goal satisfaction is stored within each goal object.

Activities provide information required to analyze how crew station design affects crew member workload. Agent load attributes are not currently recorded for analysis but can be reconstructed from activity histories.

5.2.2 Detailed Design Description

Details concerning the design of the basic objects are provided in the following sections.

5.2.2.1 Scenarios

Objects of flavor SIM-SCENARIO have the following instance variable values:

SIM-NAME:	User defined string "Enroute Segment 3".
SIM-ID:	Not currently used.

ENVIR-CONTEXT:	Used to record and access context dependent information. This variable is a list with the following format:

```
( <parameter-1> <parameter-1-value>
  <parameter-2> <parameter-2-value>
       :      :      :      :
  <parameter-n> <parameter-n-value>)
```

Example:

```
(:PHASE-LINES
        ((RED 48008400 48007900 D-1/26CAV)
         (BLUE 50008400 50007900 B-1/26CAV))
  :ACPS
        ((ACP-DELTA 48208030 TOC))
  :FARP 47008000
  :FLOT '(54008400 54007900))
```

WORLD-OBJECTS:	Objects which should be given a tick message.
E-TRANSMISSIONS:	Instance of radio messages.
INITIAL-TIME:	Time tick 0 represents (e.g. tick 0 is 0630 - 6:30 AM).
CURRENT-TIME:	Time in military format.
CURRENT-TICK:	Time in ticks.
CONTROL-EVENTS:	Not currently used. Included to provide a structure for adding events controlled by the simulation. The structure is of the form:

```
( (<test-1> <event-1>)
  (<test-2> <event-2>)
     :     :     :
   (<test-n> <event-n>)
```

Each form is tested every tick: if the test evaluates to true, the event is executed (e.g. a message being sent).

INT-CUED-EVENTS:	Not currently used.
INT-ACTIVE-EVENTS:	Not currently used.
INT-TERMINATED-EVENTS:	Not currently used.

5.2.2.2 Crew-Missions

Objects of flavor CREW-MISSION have the following instance variable:

MISSION-NAME:	User defined string (e.g. "Enroute-1").
MISSION-ID:	Not currently used - provided for managing multiple missions.
ASSIGNED-CREW:	List of agent instances.
CREW:	Instance of the crew flavor.

MISSION-ROLE:	Symbol designating role (e.g. TM1-AC1 => Aircraft 1 in Team 1) This symbol is also referenced in the CEOI.
CALLSIGN:	Standard military format: <alpha><numeric><alpha><numeric><numeric> Y5T46 => "Yankee Five Tango Four Six"
EQUIP-SYSTEMS:	Not currently used.
GOAL-NET:	Symbol used to reference top level activity used to represent the planned mission (e.g. ENROUTE-1).
WAYPOINTS:	Hash table of waypoints.
POINT-OF-DEPARTURE:	Grid coordinates of take-off point (e.g. 43407910).

ROUTES: List of predefined routes in the following format:

```
   ( (<route-number> ( <waypoint-a> ...<waypoint-n>))
     (<route-number> ( <waypoint-a> ...<waypoint-n>))
          :      :      :      :      :      :
     (<route-number> ( <waypoint-a> ...<waypoint-n>)) )
Example:
   ((1 (1 2 3 4)) (2 (1 4)) (3 (1 2 3 5 6 7)))
```

DEFAULT-ROUTE-A/S: Not currently used - but provided for interface purposes.

DEFAULT-ROUTE-ALT: Not currently used - but provided for interface purposes.

PLANNED-ROUTE: List of planned route in the following format:

```
   ( (<route-number>
      ( <waypoint-a>  <altitude-to-wp> <airspeed-to-wp>))
      ( <waypoint-b>  <altitude-to-wp> <airspeed-to-wp>))
          :      :      :      :      :      :      :
      ( <waypoint-a>  <altitude-to-wp> <airspeed-to-wp>))
Example:
   (1  (1 50 50)
       (2 50 50)
       (3 50 50)
       (4 50 0))
```

ASSIGNED-ROUTE:	List of currently assigned route. This will be the same as the planned-route unless the route changes during the mission.
LAST-PLANNED-ROUTE:	For debugging purposes.
WAYPOINTS-FLOWN:	List of waypoint id integers for waypoints overflown.
NEXT-WP:	Should be deleted - use next-waypoint instead.
NEXT-WAYPOINT:	Integer representing id of next waypoint.
CURRENT-ROUTE-HDG:	Not currently used - provided for future display purposes.
CEOI:	Provides information relating to communication tasks. Represents information normally provided by the

Communications Electronic Operating Instructions (CEOI).
Uses the following format:

```
(<parameter-1> <parameter-1-value>
 <parameter-2> <parameter-2-value>
      :        :        :
 <parameter-n> <parameter-n-value> )
```

During Phase IV the format was specialized to:

```
( :callsigns    (<entity-1> <callsign-1>
                 <entity-2> <callsign-2>
                 :        :        :        :
                 <entity-n> <callsign-n>)
  :net  ( (<comm-net-1 <net-radio-type> <net-freq> <string-for-display>)
```

Example from Phase IV demo:

```
  (:CALLSIGNS
          ((BN-CDR P7F78)          ; Battalion Commander
           (CO-CDR Y5S92)          ; Company Commander
           (TOC Y5M14)             ; Tactical Operations Center
           (TM1-LD Y5T46)          ; Lead - Team 1
           (TM1-AC2 Y5T52)         ; Aircraft 2 - Team 1
           (TM1-AC3 Y5T73)         ; Aircraft 3 - Team 2
           (TM2-LD Y5V43)          ; Lead - Team 2
           (TM2-AC2 Y5V67)         ; Aircraft 2 - Team 2
           (TM2-AC3 Y5V25)         ; Aircraft 3 - Team 2
           (FARP Y5D37)            ; Forward Arm Refuel Point
           (D-1/26CAV G3W54)       ; D Troop, 1/26 Cavalry
           (B-1/26CAV H6L39)       ; B Troop, 1/26 Cavalry
           (ADA J2Q87))            ; Air Defense Net Control
  :NET  ; communication nets
          ((COMPANY :FM1 38.5 (C0-CDR TM1-LD TM2-LD)
                   "Company FM1 net")
           (COMPANY :VHF 126.5 (CO-CDR TM2-LD TM2-AC2 TM2-AC2 )
                   "Company VHF net")
           (TEAM1-CMD :UHF 234.1 (CO-CDR TM1-LD TM1-AC2 TM1-AC2)
                   "Team 1 UHF command net")
           (TEAM1 :FM1 40.9 (TM1-LD TM1-AC2 TM1-AC3)
                   "Team 1 internal FM net")
           (TEAM1 :VHF 140.7 (TM1-LD TM1-AC2 TM1-AC3)
                   "Team 1 internal VHF net")
           (TEAM2 :FM1 44.0 (TM2-LD TM2-AC2 TM2-AC3)
                   "Team 2 internal FM net")
           (TEAM2-CMD :VHF 147.9 (C0-CDR TM2-LD TM2-AC2 TM2-AC3)
                   "Team 1 VHF command net")
           (TEAM2 :UHF 237.2 (TM2-LD TM2-AC2 TM2-AC3)
                   Team 1 internal UHF net")
           (TOC :FM1 54.9 (TOC BN-CDR CO-CDR)
                   "Bn command net")
           (ADA :FM1 48.5 (ADA TM1-AC2 TM2-AC2)
                   "ADA net")
           (D-1/26CAV :FM1 34.8 (D-1/26CAV TM1-AC2 TM2-AC2)
                   "D-1/26CAV command net")
           (B-1/26CAV :FM1 46.4 (D-1/26CAV TM1-AC2 TM2-AC2)
                   "B-1/26CAV command net")))
```

DEFAULT-COMM-ASSIGN: Describes how communication tasks were initially allocated. In the Phase IV demo, The pilot was assigned to monitor the command net on UHF and the team net on VHF. The CPG was assigned to monitor the company FM net and make required flight coordination calls on FM which would require changing frequency between monitoring company FM and making required flight coordination calls.

Example from Phase IV demo:

```
(:PILOT
        ((TEAM1-CMD :UHF)
         (TEAM1 :VHF))
 :CPG
        ((COMPANY :FM1)
         (FLT-COORD :FM1)))
```

COMM-ASSIGN: This should be different from default-comm-assign only if there has been a reallocation of tasks during the mission (i.e. function allocation for load leveling, etc.)

COMM-REQ: List of symbols indicating a requirement. This provides an easy means of editing and changing an agent's requirements of editing and changing a crew task requirements.

Example:

```
COMM-REQ: (REPORT-CROSSING-PHASE-LINES)
```

LAST-POS-REPORT: Time (in ticks) of last position report. This is hard-coded as instance variable only to demonstrate capabilities. In the future, this should probably be one of many parameters in a more generalized variable.

5.2.2.3 Agents

An object of flavor AGENT has the following instance variables:

ID: Automatically generated id string in the format of Agent-<a gensymed number> (e.g. "Agent-355").

CREW: Not currently used - intended for agent abstraction. Agent abstraction specifies that one agent of the crew should be allocated this goal but does not indicate if it is the pilot or cpg.

ROLE: Role of agent - a symbol - Pilot or CPG

```
    ;;; The following instance variables are specific to the anthropometric model.
    ;;; See the documentation for this model for details concerning the values.
    ;;; Reach/Head-turn ids specify how the model should move (e.g. from the
    ;;; waist or shoulder, etc). Sites specify a predefined 3-d location to which a
    ;;; movement/look should be made.
LH-REACH-ID:
LH-REACH-SITE:
LH-X:
LH-Y:
```

```
LH-Z:
LH-STATUS:
LH-LAST-SITE:
RH-REACH-ID:
RH-REACH-SITE:
RH-X:
RH-Y:
RH-Z:
RH-STATUS:
RH-LAST-SITE:
HEAD-TURN-ID:
SITE-TO-LOOK-AT:
HEAD-YAW:
HEAD-PITCH:
LAST-SITE-LOOKED-AT:
        ;;; end of anthropometric variables
        ;;;

E-R-GOALS:           Instances of event-response goals not yet activated.

GOALS:               Instances of activated decomposable goals.

TERMINAL-GOALS:      Instances of activated terminal goals

GOAL-FILE:           Instances of terminated goals

        ;;; The following variables were included to keep track of where the agent had
        ;;; been looking. These values could be tested to see if the agent had been
        ;;; monitoring required items. This was not fully developed in the demo and
        ;;; probably should be removed as hardcoded variables and developed within
        ;;; the representation of a cognitive architecture.
LAST-CHECKED-ALT:
LAST-CHECKED-A/S:
LAST-CHECKED-HDG:
LAST-CHECKED-PWR:
LAST-CHECKED-TRAN:
LAST-CHECKED-FREQ:
LAST-CHECKED-CW1:
LAST-CHECKED-LWIN:
LAST-CHECKED-RWIN:
LAST-CHECKED-FWIN:
LAST-CHECKED-NAV:
        ;;; end of variable group
        ;;;

ACTIVITIES:          Decomposable activity instances

CUED-ACTIVITIES:     Activities not yet started

TERMINAL-ACTS:       Terminal activity instances currently executing

ACT-FILE:            List of terminated activities

SCHEDULER-TRIGGER: Not currently used.
```

VISUAL-REF:	Not currently used.
INT-TEMP-VACP:	Used to allocate resources to activities.
RESOURCE-COMMIT:	Used to record how resources are allocated.
VACP-HISTORY:	Not currently used - had been simple recording of time-tagged additive VACM values.
ASSERTIONS:	Not currently used - intended for conclusions based on rule interpretation of perceptual data fusion.
MSGS-REC:	Not demonstrated in demo - but provided to handle the processing of messages monitored.

An object of the flavor agent has the following methods:

Methods for accessing and setting the all local instance variables.

MAP-ACTIVITY-TO-GOAL -see code for documentation

TICK - see code for documentation

HEAD-MOVE-COMPLETED-P
RH-REACH-COMPLETED-P
LH-REACH-COMPLETED-P
LOOK-AT
RH-REACH
RH-BLIND-REACH
RH-REACH-FROM-WAIST
RH-REACH*
LH-REACH
LH-BLIND-REACH
LH-REACH-FROM-WAIST
LH-REACH*

5.2.2.4 Goals

Objects of flavor GOAL have the following instance variables:

GOAL-NAME:	User defined string (e.g. "FLY-TO-WP").
CONTEXT-NAME:	User defined format statement which produces a context sensitive string (e.g. "FLY-TO-WP2").
GOAL-ID:	Automatically generated id string in the format of G-<a gensymed number> (e.g. "G-355").
PARENT-GOAL:	Instance of a goal which specifies this goal as a subgoal.

MISSION:	Instance of crew-mission.
GOAL-LEVEL:	Integer - higher numbers represent lower levels in a hierarchy.
GOAL-PRIORITY:	Integer - higher numbers represent higher priority.
TERMINAL-P:	T or NIL - T if terminal goal, NIL otherwise.
GOAL-KEYS:	Used to record and access context dependent information. This variable is a list with the following format: (<parameter-1> <parameter-1-value> <parameter-2> <parameter-2-value> : : : : <parameter-n> <parameter-n-value>)
ACT-GOAL-KEYS:	Only used by terminal goals. Used to pass parameters to activities.
MATCHING-PATTERN:	Only used by terminal goals. A pattern that is defined by the user and is passed to a matching function to identified an activity that may satisfy the goal (e.g. FLY-COURSE-TO WP)
MATCHED-ACTIVITY:	Only used by terminal goals.
EVENT-RESPONSE:	Not currently used - flag now indicated by having a form in the activation-tests variable.
STATUS:	Not currently used.
ACHIEVEMENT-STATUS:	Not currently used - achieved-p used instead.
TEMPLATE:	For debugging purposes.
AGENT:	Instance of agent - the goal was allocated to this agent.
SUB-GOAL-TEMPLATES:	Only used by decomposable goals. Used to store references to subgoals which should not be created when the goal is initially created (e.g. all but the first subgoal for a sequential goal) but will be created later.
SUB-GOALS:	Only used by decomposable goals. List of goal instances which are subgoals of this goal.
ACTIVATION-TESTS:	Only used by event-response goals. Specifies a form to be evaluated each tick. When the form evaluates to true, the goal is activated. Event-response goals are stored in the event-response goal instance variable (i.e. e-r-goals) of the agent and the activation test of each goal in this list is tested each

tick by the agent's tick method. When the goal is activated, it is removed from the list and placed in the appropriate decomposable or terminal goal list.

CURRENCY-TESTS:	Used to test satisfaction of the goal. Defined by the user. The use of backquotes is common here since the user can specify when a subform should be evaluated. During development, the use of embedded forms using backquotes was tested so evaluation of user defined forms is performed in two stages. The first stage, completed during the initialization phase, results in the specialization of the embedded, second form which is evaluated during the simulation. For example, the goal for sending a required message has a currency tests variable. During the definition phase, the user defines the goal with the following form (the backquote is not shown):

```
:currency-tests (,(list '> '(utm-x *aircraft*)
 (- (cadr report) 20)))
```

After initialization, the currency-test has been specialized to the following by evaluating the backquoted form which produces a form which can test the if the x-coordinate location of the aircraft is greater than 20 meters before the phase line :

```
:currency-tests (> (utm-x *aircraft*) 4780)
```

The form, (> (utm-x *aircraft*) 4780), is then tested during the simulation.

CURRENT-P:	Indicates if the goal is current - T or nil.
TICK-CREATED:	Time (in ticks) goal was created.
TICK-ACTIVATED:	Time (in ticks) goal was activated.
TICK-TERMINATED:	Time (in ticks) goal was terminated.
PRECONDIT:	Not currently used.
CONTIN-CONDIT:	Not currently used.
GOAL-TEST:	Appropriate only for terminal goals. Specifies how goal satisfaction should be tested. A list with the following structure:

```
( <goal-temporal-type> <goal-test-form>)
```

Goal temporal type can be either ACHIEVE or MAINTAIN. The goal test form can be any valid lisp form which returns T or nil.

Example:

```
(MAINTAIN
   (EQL (COURSE-IND *AIRCRAFT*) 76))
```

ACHIEVED-P:	Indicates if goal has been satisfied. Achievement goals are classified as either satisfied or not-satisfied. Maintenance goals specify states (e.g. "maintain heading 135") which must be maintained for a specified duration. Maintenance goals are classified as:

currently-satisfied =	state maintained but goal has not yet been completed
not-currently-satisified =	state not maintained goal has not yet been completed
satisfied =	state maintained and goal completed
not-satisfied =	state not maintained and duration completed.

TICK-PROCEDURES:	Not typically needed but retained for special purposes.
TERMINATION-CONDITIONS:	Tested each tick - if evaluates to True the goal is terminated. Example: ((< (DISTANCE-TO *AIRCRAFT* 10170 3445) 400))
TERMINATION-PROCEDURES:	Procedures which should be run when goal terminates. Not typically needed but retained for special purposes.
DATA-PIT:	Used to store terminated subgoals.

5.2.2.5 Activities

Activity instances are created during the simulation and represent a unique instance of an agent performing action. Parameter values are inherited from the related activity template or derived by an function provided by the template. Parameters for activity instances include the following:

act-name	a string representing the activity name.
context-name	a string including context dependent descripters.
act-short-name	a string abbreviating activity name - for graphing.
act-id	string produced by ACT<gensym-number> (e.g. ACT425).
agent-allocation	which agent should perform.
act-template	used for debugging.
equip-context	specifies crew station design.
act-keys	provides a generalized parameter - value list structure: <parameter-1> <value> ...<parameter-n> <value> This enables new parameters to be defined without recompiling

the template definition.

act-goal-keys	used to pass parameters to activity instances.
act-type	indicates general temporal relationships of subactivities "sequential" specifies subactivities should be processed in sequence. "parallel" specifies all subactivities should start when activity starts assuming constraints allow. Indicates subactivities are anticipated to be all started at once. "rotation" specifies sequential order which is to be repeated until termination conditions cause activity to stop. "complex" specifies all subactivities should try to start. Indicates constraints are anticipated to delay the start of some activities.
required-resources	specifies required-resources. activities may be constrained from starting when a required resource is not available. These resources include: visual auditory cognitive motor left-hand right-hand point-of-fixation (look-at site)
functions	high level text description of activity for display purposes.
priority	currently user defined priority.
initialization-procedures	procedures to be run when activity is initialized.
preconditions	state-activity pairs for indicating possible preconditions/actions.
com-fcn-procs	defined only for terminal activities. Specifies a list of equipment component functions which to be executed in sequential order.
remaining-comp-fcn-procs	used internally to keep track of which component function has been completed. Initially this list is set to a copy of comp-fcn-procs. As component functions are completed, they are deleted from this list.
explicit-local-constraints	will be tested when activity tries to start. Indicates subactivities are anticipated to be delayed from starting.
estimated-duration	expressed in ticks - 100 ms.
start-procedures	procedures to be run when activity is started.
cf-procedures	not currently used - use com-fcn-procs instead.
tick-procedure	specifies forms to be evaluated each tick.
termination-conditions	specifies forms to evaluated each tick. When one of the forms returns "TRUE", the activity will be will be terminated.

termination-procedures	specifies forms to be evaluated when the activity terminates.
compute-vacp-data	flag indicating VACP values should be determined by form in the vacp-load-form slot.
vacp-load-form	form used to produce context dependent VACP values
vacp-data	context-free estimate of VACP values for this activity.
vacp-data-history	time-tagged VACP values.
mapped-goal	used only for top level activities - an instance.
tick-cued	time (in ticks) activity was initially tested for starting.
tick-started	time (in ticks) activity was started.
tick-ended	time (in ticks) activity ended.
sub-activities	used only for decomposable activities - active instances.
activity-history	used to store terminated subactivities.

6.0 USER'S GUIDE

6.1 INSTALLATION AND INITIALIZATION

Files for the Symbolic Operator Model are maintained in a directory called MIDAS2. This directory is organized into subdirectories as follows:

BASIC1	- Defines flavor definitions and global variables.
BASIC2	- Defines basic functions and methods.
COMM	- Defines data pool as required by the communications module.
DEMO	- Defines mission, objects, agents, goals and activities specific to the the Phase IV demonstration.
DISPLAYS	- Defines the Symbolic Operator Model display constraint frame.

The Symbolic Operator Model has been defined as a system using the Genera Development System Tools in a file named "midas.system" in the MIDAS2 directory.

Loading the system requires that the Equipment Modeling and LONGBOW systems be loaded in sequence. The Phase IV demonstration also requires a binary file to be loaded from the MIDAS directory. This file defines a terrain digital elevation model (DEM) which was used during development. Since it was the stated objective of the Program Office to maintain DEM's only on the IRIS machines, the code should be changed to access elevation values from the IRIS instead of from the structure created by the binary file in the MIDAS directory.

To install the Symbolic Operator Model at a new site, the newest lisp files from the MIDAS2 directory should be copied to the new site. Besides the Symbolic Operator Model, files defining the Equipment Models and the Longbow System will be required. Details for installing these systems are provided in Annex D. Files required for communications are detailed in Annex J and these files must be loaded prior to any simulations. Also, the current Symbolic Operator Model requires one binary file, "terrain>p4-dem.bin", from the MIDAS directory. This file should be recreated and moved from the MIDAS directory into the MIDAS2 directory for future development (the MIDAS directory was used as the development directory).

Although it is not required prior to loading the system, prior to running any simulation it is necessary to load whatever files are required by the Communications modules and initialize the IRIS machines as instructed in the Communications module documentation, Annex J.

After the required files have been loaded onto the fileserver, the file "midas.system" must be edited to reflect any changes in logical pathnames.

Programmers should be aware that many of the files in the "DEMO" directory were created to produce a small demonstration in Phase IV. Any future experimentation with code developed will require changes to many of the definitions provided in these files. Experimentation with Jack can be accomplished without changing files in other directories.

The Symbolic Operator Model may then be loaded by the following top level commands:

```
prompt> Load System Midas2
```

The system will load the Equipment Modeling and Longbow systems if they are not loaded into the current environment. Any files required by the Communications module should then be loaded.

The Symbolic Operator Model display may be selected by:

```
SELECT Symbol-Shift-E
```

Initialization is accomplished by selecting the "RESET SIM" menu item in the MIDAS display. The system is ready for a simulation after this has been completed.

6.2 STARTUP AND TERMINATION

Simulation starting and termination is accomplished by Simulation Executive commands. Refer to Annex J for details.

Annex B

Army-NASA Aircrew/Aircraft Integration Program: Phase IV

A^{3}I

Man-Machine Integration Design and Analysis System (MIDAS)
Software Detailed Design Document

Scheduler (Z)

prepared by

Renuka Shankar

Table of Contents

Table of Contents

Table of Contents

Table of Contents

MAN-MACHINE INTEGRATION DESIGN & ANALYSIS SYSTEM (MIDAS) SOFTWARE DETAILED DESIGN DOCUMENT PHASE IV:

Z - SCHEDULER

1.0 INTRODUCTION

1.1 IDENTIFICATION OF DOCUMENT

This document establishes the requirements and detailed design of the Z-Scheduler module of the MIDAS system. The Z-Scheduler document has to be read within the broader context of the documentation of the Symbolic Operator Model (Annex A) and the Task Loading Model (Annex C).

1.2 SCOPE OF DOCUMENT

The material in this document is directed towards three categories of readers as shown in Figure 1 below.

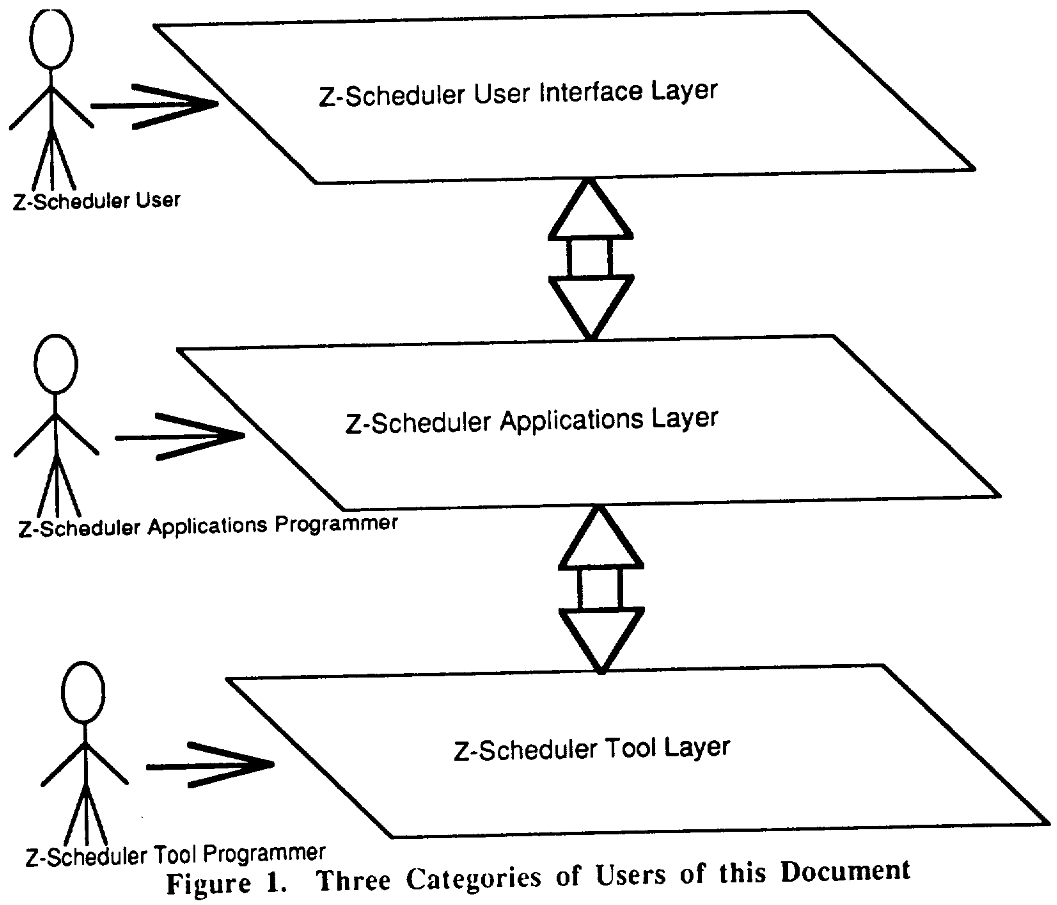

Z-Scheduler User Interface Layer
Z-Scheduler User
Z-Scheduler Applications Layer
Z-Scheduler Applications Programmer
Z-Scheduler Tool Layer
Z-Scheduler Tool Programmer

Figure 1. Three Categories of Users of this Document

The three categories of readers are:

1) Z-Scheduler Users: Those who are interested in learning what Z-Scheduler Module in MIDAS does. The users interact with the Z-scheduler's user interface layer primarily to input a set of tasks and view the schedule through the user interface. No programming skills are required to interact with the scheduler in this capacity.

2) Z-Scheduler Applications Programmers: Those who wish to extend the capabilities of the Z-Scheduler software. For example, when one wants to build upon or modify the existing set of scheduling strategies in the scheduler. Some lisp programming and basic GEST programming knowledge is recommended for using Z-Scheduler for this purpose.

3) Z-Scheduler tool programmer: Those who might want to modify and update the Z-Scheduler tool layer. Extensive knowledge of the Symbolics programming environment, object-oriented programming, knowledge-based designing, and some experience in programming with GEST is required to program Z-Scheduler in this capacity.

1.3 PURPOSE AND OBJECTIVE OF DOCUMENT

This document attempts to describe the methodologies used to represent the Z-Scheduler of the MIDAS system. The Z-Scheduler was proposed, designed and implemented in Phase IV of the A^{3}I project. This implies that though there has been some references to the Scheduling model in the previous phase documentation there has never been a powerful and separate scheduling model in the earlier phases of this project.

2.0 RELATED DOCUMENTS

2.1 APPLICABLE DOCUMENTS

The following documents are referenced herein and are directly applicable to this volume:

Symbolics Genera 7.2 Documentation, Symbolics Publication Number 999079, Symbolics, Inc., Cambridge, Massachusetts, 1988.

GEST ver4.0 manual, GTRI, Atlanta, GA, 1989.

2.2 INFORMATION DOCUMENTS

The following documents amplify or clarify the information presented in this volume:

James Allen, "Maintaining Knowledge about Temporal Intervals", *Communications of the ACM* 26 (11), 832-843, 1983.

David Ben-Arieh, "Knowledge Based Control System for Automated Production and Assembly", Purdue Univ, Ph.D Dissertation, Aug 85.

S. Hart, "Research papers and publications 1981-1987," NASA Technical memorandum 10001b, 1987.

Barbara Hayes-Roth, "A BlackBoard Architecture for Control" , *AI* **26** (1985) 251-321.

Barbara Hayes-Roth and Frederick Hayes-Roth, "A Cognitive Model of Planning", *Cognitive Science* **3**, 275-310 (1979).

N.P. Keng, D.Y.Y. Yun and M. Rossi, "Interaction Sensitive Planning System for Job-shop Scheduling". In *Expert Systems and Intelligent Manufacturing*, Michael D.Oliff editor. Elsevier Science Publishing Co., 1988.

Johan de Kleer, "An Assumption Based TMS", *AI* **28** (1985) 127-162.

Jay Liebowitz, Patricia LightFoot, "Expert Scheduling Systems: Survey and Preliminary Design Concepts", *Applied AI* **1**:261-283, 1987.

N. Morray, M. Dessouky, "Scheduling Models for Workload Prediction and Guidance", EPRL report, U. of Illinois at Urbana-Champaign, Aug 89.

Jeff Rickel, "Issues in the Design of Scheduling Systems". In *Expert Systems and Intelligent Manufacturing*, Michael D.Oliff editor. Elsevier Science Publishing Co., 1988.

P. Sanderson, N. Morray, "The Human Factors of scheduling Behavior", EPRL report # 90-09, U. of Illinois at Urbana-Champaign.

Peng Si Ow, Stephen Smith, and Randy Howie, "A Cooperating Scheduling System." In *Expert Systems and Intelligent Manufacturing*, Michael D.Oliff editor. Elsevier Science Publishing Co., 1988.

Peng Si Ow, Stephen Smith , "Viewing Scheduling as an Opportunistic Problem-solving Process", *Annals of OR. Approaches to Intelligent Decision Support*", R. G. Jeroslow (editor), Baltzer Scientific Publishing Co., 1987.

3.0 CONCEPT

3.1 DEFINITION OF Z-SCHEDULER

Z-Scheduler is a constraint-based, opportunistic scheduling tool that represents one of MIDAS' first cognitive models. The Z-Scheduler is provided with a task queue along with data about each task (such as temporal constraints, estimated duration, resource requirements, etc). The scheduler solves for a "near-optimal" sequence and schedule based on a strategy of time minimization or load balancing, intended to represent possible operator behaviors. Z-Scheduler uses a blackboard architecture — the closest thing to a cognitive architecture to date. Also, this architecture allows its components to be modular and easily extensible.The components of Z-Scheduler are called knowledge sources and each component represents a stage in the scheduling process. Z-Scheduler itself is modular and is called on an "as-needed" basis by the task decomposition with a variable temporal horizon. Z-Scheduler closely interacts with the Task Loading Model for learning about resource interaction between plausible concurrent tasks. Z-Scheduler uses the extended task-based decomposition (a "divide-and-conquer" technique) to partition the scheduling problem space into subspaces, finds the local solutions for each of the subspaces and threads the individual local solutions along to derive the global scheduling solution.

3.1.1 Purpose and Scope

The primary purpose of the Z-Scheduler is to extend the power and applicability of the MIDAS symbolic operator model. As mentioned above, Z-Scheduler works at modeling the pilot's strategic scheduling behavior. Specifically the Z-Scheduler takes a set of tasks generated by the planner as input and outputs a specific order on the execution of the tasks. Z-Scheduler has been keeping strict modular principles in mind and hence the scope of Z-Scheduler's applicability is therefore wide.

3.1.2 Goals and Objectives

The primary goal of the Scheduler was to capture the scheduling behavior of an operator into a computational model. The focus of the work has been modeling the scheduling strategies that the operator applies while scheduling behaviors to achieve desired results. The Z-Scheduler is currently geared to simulate two such operator scheduling strategies namely, the minimize-time strategy and the balance-load strategy. The former strategy is when the operator is concerned primarily with performing all the allocated tasks within the shortest time window by extending his/her resources to the maximum limit possible. The balance-load strategy simulates the operator delaying the execution of the tasks so as to continue being at his/her comfortable resource load limit.

In the earlier phases of this project, the pilot activities whose preconditions are satisfied at any moment in time, begin executing at that moment. There was minimal and in some cases no mechanism to do the resource allocation in order to arrive at an opportune moment to perform the task. In other words, there was no "look ahead" mechanism by which the operator analyzes the currently active activities and decides on an order for performing the task such that the tasks all get done in a more efficient manner. This is the aspect of the operator model that the Z-Scheduler was targeted to address. Hence, a more general goal of the Z-Scheduler was to enhance the symbolic operator model.

Four major objectives have been outlined for the Z-Scheduler for its first developmental stage during Phase IV of the A^3I project. They are the following:

1. Simulate varied operator scheduling strategies.

2. Reflect changes in schedule with a change in environment.

3. Provide for predictive task analysis.

4. Assist in enhancing the operator model with a look-ahead feature.

The first objective of the Z-Scheduler was that Z-Scheduler should be able to demonstrate scheduling behavior obtained as a result of applying varied scheduling strategies. It has been proven in the human factors research on strategic behavior that a human's schedule is dependent on the heuristics applied during scheduling and these heuristics are what gives rise to strategies. In this phase of development of the scheduler, we isolated two such proven strategies. We then, scheduled tasks using the two strategies as objective functions to evaluate among alternate schedules and hence, demonstrate the difference in the schedule generated by the two strategies.

Since, the broadest goal of MIDAS is to show the impact of the man-machine interface design on human performance, it follows that all modules of MIDAS obey this guiding principle. Since, Z-Scheduler assigns a time-line order to a set of tasks performed by humans while interacting with the man-machine system design (in Phase IV the man-machine system was the Apache helicopter), it can be easily seen that if the environment changes the set of tasks will change and so will the schedule. Hence, using this logic we can conclude that Z-Scheduler reflects changes in the schedule with a change in environment.

The tasks input to the scheduler are the tasks that generated based on the predictive capacity of the task generator and hence are the tasks are to be performed in the near future (temporal horizon). Hence, it can said that the time window that the scheduler is working on fitting all the tasks is in some projected future time.

By incorporating a scheduler in the MIDAS symbolic operator model, questions pertinent to our Phase IV objectives is addressed such as:

a. How close to an analyst-specified time was the pilot able to perform mission related tasks?
b. What is effect on the schedule, when the set of imposed constraints are changed?
c. What happens when the pilot changes his scheduling strategy (like optimizing time, or making sure the loading is constant etc.)?
d. What are the changes in the schedule when the tasks are performed in a different environment, like moving from a traditional environment to a MultiFunction Display (MFD) environment?

Z-Scheduler is designed to handle the above queries; hence justifying the requirement for a scheduler in the MIDAS symbolic operator model.

3.1.3 Description

Z-Scheduler attempts to capture varied operator scheduling strategies into a computational model. Two such strategies have been implemented in this phase of development of the Z-Scheduler - the minimize time strategy and the balance load strategy. During the application of the minimize time strategy the operator tries to perform the specified tasks in as short a time window as possible by extending the resource capabilities to the maximum level possible. The balance load strategy is one in which the operator is working towards being at a comfortable resource limit and hence delays the execution of some tasks. In the literature, especially work done by Sandy Hart, (Hart, 1987) these strategies are well accepted.

The input to Z-Scheduler is a set of tasks to be scheduled. These tasks are generated by the planning component of the symbolic operator model. Each task description contains details on the estimated duration for the task, the priority (the relative importance of a task to the other tasks on the stack) and the resource utilization (the resources required for the effective performance on a task). The resource structure for the Z-scheduler is a vector of four elements, namely, the Visual, Auditory, Cognitive and Motor loadings and are defined in the Task Loading Model.

Along with the stack of tasks, Z-Scheduler also gets a stack of constraints which are defined as some restriction placed on the execution of the tasks. Z-Scheduler can handle both qualitative and quantitative time constraints. The qualitative constraints in the literature are called interval-based constraints and are useful in describing the temporal relation between two tasks. For example, we can specify task A is before task B or any of the 13 relationships defined in (J. Allen, 1983). Also, tasks can be directly anchored to the absolute time by specifying quantitative relations like 'Start-at task-A 5 secs". The effort of merging both the qualitative and quantitative temporal constraints into a uniform framework is one of the most significant contributions of the Z-Scheduler.

Before getting into the details of the dynamics in the Z-Scheduler, it would be important to know the rationale for choosing the selected architecture and approach. On analyzing the MIDAS scheduling problem based on the nature of input, we concluded that it falls into the category of dynamic-multistage-routing problems. Simply stated this category of problems are at the complex end of the spectrum of difficulty where traditional OR techniques fail to provide solutions and hence, we turned to knowledge-based techniques for the answers.

Just as the approach taken in the majority of past works, we have modeled the problem as a constraint satisfaction problem (CSP). Generally, whenever a problem is modeled in this manner, the solution space is divided into subspaces and the local solutions are found for each space and these local solutions are threaded along to get the global solution. The method in which the problem space is divided is crucial to the efficiency of the model. We have chosen task-based scheduling to solve the CSP.

Now moving to the heart of the scheduling process, the blackboard architecture is central to Z-scheduling process. The blackboard is a central structure that contains all the scheduling state data and is divided into sections, each section representing a stage in the schedule generation process. The knowledge required to perform the schedule is also partitioned and each partition, called a knowledge source, works on each stage of the scheduling process.

The first knowledge source to be triggered is the constraint solver knowledge source. The primary focus of this knowledge source is to translate the hybrid representational constraints into a uniform framework for the temporal reasoning process. Also, because of the nature of inputs to Z-scheduler, the tasks are oblivious of their constraints. Hence, the constraint solver contains rules for solving for one constraint at a time that includes caching the constraint in the relevant tasks and incrementally building networks called partial dependency graphs.

The Z-Scheduler uses task-based scheduling for partitioning the scheduling problem space. Task-based scheduling is a two step process. The first step involves the selection of a single task from the task stack that contains only the unscheduled tasks. The second step involves the commitment of a resource time slice to the selected task by evaluating all alternative resource allocations for that task and selecting the best allocation based on certain decision metrics. Each of these two steps is carried out in separate knowledge sources, the first being implemented by the task selector knowledge source and the second being carried out by the resource allocator knowledge source.

The metric in selecting tasks for resource allocation is called the task criticality measure. This measure has three factors, one of them being the slack on that task. The more the slack on the task less its task criticality. The heuristic applied in the task selector knowledge source is that the task with the least amount of slack becomes the candidate for resource allocation. The metric for allocating a resource time slice for the selected task is to pick the solution that has minimum resource competition, the logic being that the solution must not eat away into the opportunity of the tasks that have not yet been scheduled. This metric is called the resource competition measure and is implemented by the resource allocator knowledge source.

The constraint propagator knowledge source is a record keeping knowledge source. Every time a task is scheduled it updates rest of the task windows accordingly by propagating the temporal constraints in the partial dependency graphs.

The truth maintainer knowledge source is a high level monitor of the constraint propagating activities. Whenever a conflict occurs during the process of constraint propagating, this knowledge source halts the execution of the other knowledge sources by sending a message via the blackboard. After halting the process that was responsible for the conflict (an example of the conflict is that the window duration of the task becomes smaller than the estimated duration of the task) the truth maintainer attempts to find the reason for the conflict's occurrence and, on finding it out, resets all the knowledge sources to the point in time that brought about the conflict. In other words, the truth maintainer is built on the principle of chronological backtracking, where the amount backtracking is the previous decision point.

3.2 USER DEFINITION

The Z-Scheduler resides within the symbolic operator model of the MIDAS system and specifically is sandwiched between the Task Generator and the Simulation module as shown in Figure 2 below:

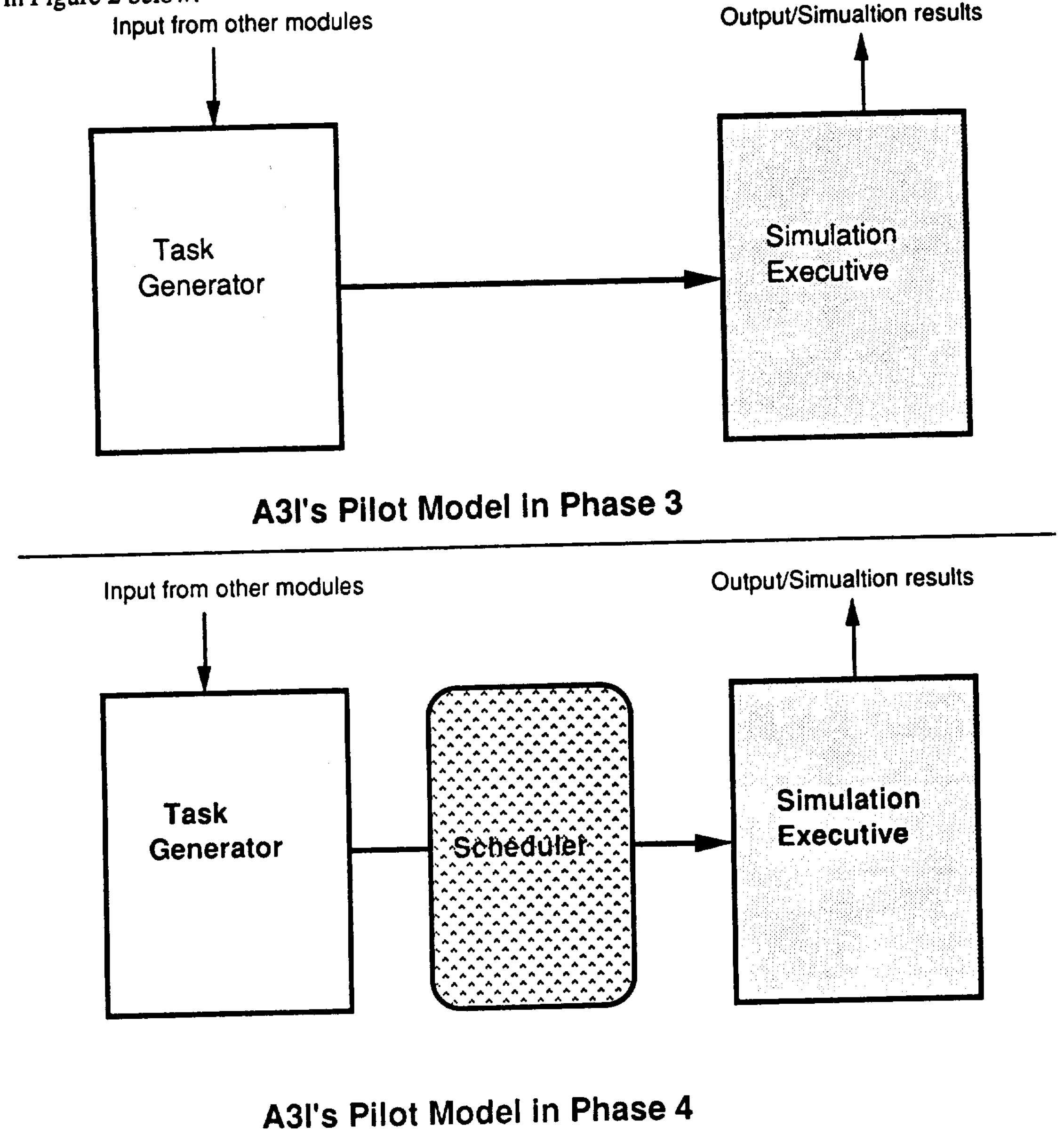


Figure 2. Scheduler in Relation to Symbolic Operator Model

It is important to distinguish Z-Scheduler from the task generator and the simulation executive components of MIDAS. The responsibility of determining *what* activities need to be performed to accomplish the generated/specified goals remains in the task generator. Similarly, the functions of tick passing and *synchronization* with the other models will remain within the simulation executive.

As mentioned above, Z-Scheduler *only* determines the optimum order and times for a list of tasks, dictated by some objective function (such as time or load minimization) which represents the pilot's strategic control of his/her behavior. To perform this function, the Z-Scheduler receives as input a set of tasks with some specified constraints. The scheduler uses these constraints to search through the solution spaces of all feasible schedules to arrive at the optimal time-line order of the tasks.

3.3 CAPABILITIES AND CHARACTERISTICS

Z-Scheduler is a constraint-based, opportunistic scheduling tool that attempts to capture varied operator scheduling strategies into a computational model. Provided with a task queue along with data about each task (such as temporal constraints, estimated duration, resource requirements, etc), the Z-Scheduler solves for a "near-optimal" sequence based on the specified operator scheduling strategies. In Phase IV, we have implemented two such operator scheduling strategies and they are the minimize time strategy and the balance load strategy. The minimize time strategy is one in which the operator works towards performing all the specified tasks in as short a time window as possible. The balance load strategy is one in which the operator works towards being at a comfortable resource level and hence delays the execution of some of the tasks.

3.4 SAMPLE OPERATIONAL SCENARIOS

The operational scenario for demonstrating the capabilities of the Z-Scheduler during Phase IV was tasks extracted from the preflight segment of the AH64 task analysis. This segment included two functions — one being the communications function and the other being the auxiliary power unit (APU) starting function. The former function included two tasks, 'transmit-cockpit-communication and ' receive-cockpit-communication. The APU starting function included seven tasks, the higher priority tasks being set-gen1-switch and set-gen-2-switch. The rest of the five tasks were routine monitoring tasks the details of which are in file "puf:>Renuka>Z>tests>apu-starting-tasks.dryrun" .

Some of these tasks were related to each other by the specifications of four constraints. The primary constraints were that the set-gen-2-switch must be after the set-gen-1-switch, and that transmitting communications comes before receiving communications. For obtaining the details of the other constraints, refer to the above mentioned file.

On starting the execution of the blackboard controller (for the exact commands, please refer to Appendix A), the first stage of the scheduling process, the constraint solving, is performed. The user will notice all the input tasks stacked as cards in the window titled "Unscheduled Task Stack". Similarly the constraints specified by the user will be shown in the "Unsolved Constraint Stack" window. As the constraint solver solves the constraints incrementally, the user will notice constraints being pulled out of the constraint stack and the respective partial dependency graphs (PDG) being drawn. When the constraints are all solved, the constraint solver redraws the original constraint stack, but this time under the heading of "Applied constraint stack". After the constraint solver completes its function of creating the PDGs, the constraint propagator knowledge source knowledge fires up.

At this point the users attention should be directed towards the GANTT chart window. The user will see the window of opportunity being formed for each task by the propagation of constraints between the tasks. The task selector knowledge source steps in after the constraint propagation and works on calculating the task criticality. As a result of this the user will notice the tasks being shuffled and reorganized in the task stack window. The tasks that have changed their relative positions in the stack are highlighted for a small instant. The task that gets to be on the top of the stack becomes the candidate for resource allocation. When the resource allocator knowledge source gains control of the execution, the user will notice the selected task get highlighted in the GANTT chart and a sliding bar shift across the span of the task window. The length of the moving bar signifies the estimated duration of the selected task and the different positions indicates possible resource allocation positions for that task. The resource allocator completes assigning the resources when the user notices the task that was highlighted earlier is transformed from a double-headed arrow into a shaded box. The user will also notice the cumulative resource curves showing up in the four resource panes.

The process then iterates between the task-selector, resource allocator and constraint propagator knowledge source and, as the scheduling progresses, the user will notice the task stack getting smaller and when all the tasks are scheduled the stack becomes empty. At the same time, the GANTT chart contains all the scheduled task and all are marked by solid boxes.

For the sake of clarifying a display that looks complex at first glance, all the objects are color coded. Each task gets assigned a color and the same color shows off the task across the many windows. Also, at any point during the scheduling process, the user is encouraged to inspect the tasks and constraints. This may be performed by suspending the knowledge source execution if many details are required or can be performed simultaneously along with the scheduling process. While choosing the latter option the user be warned there are chances that s/he may be viewing data that might have already changed since the information was displayed.

4.0 REQUIREMENTS

Z-scheduler requires a blackboard architecture for its implementation. Rather than develop the blackboard framework in house, we decided use an existing shell in order to save on time and effort. Among, the few available alternatives we choose Generic Expert System Tool (GEST), since it outperformed the other two alternative in terms of cost to features ratios. GEST was cost effective, indeed its run time version was available on zero-cost basis to the group and it offered a host of features that was lacking with the other two frameworks, namely, the BB1 software and the GBB tool. During the time of evaluation of the 3 alternates, GEST showed more maturity than the other two tools. GEST had a reasonably good user-interface, which was among one its selling points too.

The Symbolics platform was desired for a host of reasons. The major reason being that, during the past few phases, the operator models were all implemented on the Symbolics machines. Since, Z-Scheduler is a part of the operator model, it followed that it would be appropriate for Z-Scheduler to implemented on the Symbolics. Also, since GEST's favored platform was the Symbolics machines, we decided that Z-Scheduler being built on top of GEST should be on the Symbolics machine.

4.1 HARDWARE ENVIRONMENT

Z-Scheduler currently runs on a Symbolics 3640. It also requires a Color Monitor - presently uses Techtronix color monitor with supporting CAD Buffer boards.

The details of the machine configuration is shown below. The details include the circuit boards present in the Symbolics 3640 (host-name is 'Puffer and is in the A^{3}I site).

Chassis
Datapath
Sequencer
Memory Control
Front End
2Meg Memory
CAD buffer
512K Memory
IO
FEP
IO Paddle Card

Summarizing the important cards shown above, Puffer includes 2.5 M Word of RAM and CAD buffer among the other default cards. Alternatively, Puffer is a 3640 Processor with 2560K words Physical memory and 27177K words Swapping space. The disk drive for Puffer has 30 MB capacity.

The Ethernet Address of Puffer is : 08-00-05-03-20-20

Additionally, Z-Scheduler may be ported to hardware that support Common lisp, Color and the GEST system. Some of the possible platforms that Z-Scheduler may be ported to are MacIvory, XL1200, SUN4. It must be remembered here that these machines are only candidates for porting and the author has not ported the software across these platforms and hence will not make any commitments to the feasibility and ease of porting etc.

4.2 SOFTWARE ENVIRONMENT

Z-Scheduler currently uses a world load with containing the following systems along with their versions.

System	Version
Genera	7.2
System	376.158 (ECO level 6)
Utilities	27.29 (ECO level 4)
Server Utilities	28.5 (ECO level 1)
Hardcopy	118.17 (ECO level 1)
Zmail	165.20 (ECO level 1)
LMFS	102.7 (ECO level 1)
Tape	82.18 (ECO level 3)
Nsage	27.227 (ECO level 1)
Documentation Database	62.1
IP-TCP	67.8 (ECO level 3)
Experimental News	5.0
Generic Expert System Tool	296.0
Color	405.13
Color Support	409.13
Color Doc	408.0
SGD Book Design	2.6

The world for the above combination of environment is " FEP0:>Inc-Ocean-Color-Gest.load.1".

The interface between Z-Scheduler and the task generator component of the Symbolic Operator Model is through the file input and output on the Chaos Net connection using the Nfile protocol. The interface to the Task Loading model is achieved by having one image of the model resident within Z-Scheduler.

Z-Scheduler has also been successfully ported to Genera 8.0.

4.3 EXTERNAL INTERFACE REQUIREMENTS

The task generator (TG) projects into a near future (temporal horizon), tries to anticipate tasks likely to happen and instantiates instances of these projected tasks. Along with the set of tasks it also generates a set of possible temporal constraints that might exist between the tasks and appends this to a global constraint list. Each task spawned by the Task Generator contains some task characteristics such as estimated duration, amount of resources required etc. The Task Generator feeds the set of tasks and the set of constraints to Z. Also, the Task Generator specifies the total time available for the tasks and the scheduling strategy.

Z-Scheduler generates all the possible ordering of the tasks by inferencing on the constraints into Partial Dependency Graphs (PDG), evaluates all the possible resource assignments for each task by performing a task-based decomposition and then converges on the best resource assignment for each task, thus forming the schedule. Z-Scheduler closely interacts with Task Loading Model to learn about the resource interactions between tasks that may be performed concurrently. The output of Z-Scheduler is a list of additional constraints that appends to the global constraint list or is a list of constraints that is a subset of the input constraint list - the extra elements being rejected by Z. The output constraints are generated by reverse-parsing of the converged schedule, that is, translating the schedule into set of constraints.

4.3.1 Interface with Task Generator

4.3.1.1 Input from the Task Generator to Z-Scheduler

The following are global variables that are shared between the task generator and the Z-Scheduler

tasks-to-be-scheduled ::= instances of tasks to be scheduled

constraints ::= list of constraints on the tasks. Each constraint is a list with unspecified number of elements. The structure of the list is defined as follows. The first element is the category of the disjunction in the constraints. Each of the other elements consist of three sub-elements - the first is a type of constraint. second is the task that the constraint belongs to and the third is the constraining element- can be a task,a constant time and also a form to be evaluated.

primary-strategy ::= Minimize-time or level-load.

time-available ::= time allowed for the tasks to be executed.

secondary-strategies ::= a list of secondary strategies.

4.3.1.2 Output to Task Generator from Z-Scheduler

```
*output-constraint-list* ::= list
                        ; Similar in syntax to *input-constraint-list*
                        ; Subset of the disjunction constraints in the
                        ; *input-constraint-list* plus all the valid
                        ; singular constraints plus all the new singular
                        ; constraints generated by Z-Scheduler.
```

The Z-Scheduler can modify the input constraints list to create the output constraint list in the following way:

1. Prune the number of choices in the disjunctions - if possible just leave only one behind.
2. Leave the singular constraints undisturbed.
3. For tasks that have no constraints, add constraints relating the task to other tasks.

4.4 REQUIREMENTS SPECIFICATION

4.4.1 Process and Data Requirements

From the above section, it can be seen that the data requirements to the scheduler is a set of tasks and a set of constraints. The set of tasks fed into Z-Scheduler are flavor objects. Since GEST can only infer on the frames, beliefs and what-if structures (refer to GEST manual for more details), Z-Scheduler had to convert the flavor objects to GEST frames. Z-Scheduler also has mechanism to convert the elements in the input constraint list into constraint frames. The following section will give the structure description of the data requirements for the Z-Scheduler as it is represented in GEST.

4.4.1.1 Input Representation

On an abstract level **Z-Scheduler** is a system whose input is described as (I, R, P, S, C) where,

I- is the set of tasks to be scheduled(For example,. REACH-FOR-RADIO, TUNE-RADIO).
R- is the set of available resources(For example,. Left-hand).
P- a single high level(primary) strategy for the schedule(the scheduling strategy). For example, MINIMIZE-TIME.
S- is a set of secondary performance measures for the schedule. For example, MINIMIZE-MOVEMENT-TIME-BETWEEN-TASK.
C- is the set of constraints on R and I.(For example,. REACH-FOR-RADIO *is-before* TUNE-RADIO).

4.4.1.1.1 Input Task Representation (I)

The task representation for the tasks to be scheduled is an extension of the *task formalism* that has already been proven in planning systems such as NON-LIN and the more recent O-Plan(Tate, Drummond, 89). Successful scheduling such as OPIS, ISIS, ISA and many others too use the same idea as in O-PLAN, i.e, a frame representation for the tasks. The basic idea present in these systems, of representing each task as a frame, is also present in the Z-Scheduler. The task frame in the Z-scheduler is described below. It contains slot names whose values are either initially set during the time of conversion from flavors to frames or is set during the manipulations of the scheduling decisions.

```
Task
    name
    agent                        ; The person/system performing the task.
    priority                     ;  The importance of the task with respect to the other tasks.
    estimated-duration           ; The estimated duration on the task.
    cognitive-resource           ; The VACP values on this task. This slot
                                 ; will interface to Task Loading Model.
    EST-value                    ; Earliest Start Time value
    LET-value                    ; Latest End Time value
    ST-value                     ; Start Time value
    ET-value                     ; End Time value
    EST-unscheduled-filters
    EST-scheduled-filters
    LET-unscheduled-filters
    LET-scheduled-filters
    Scheduling status            ; Indicates whether the task has been scheduled or not.
    member-PDG                   ; Indicates which Partial Dependency Graph the task belongs to.
```

4.4.1.1.2 Resource Representation (R)

The set of resources (R) that have to be allocated to the tasks use the representation shown below. The explicit representation of the exhaustive list of all resources is not present in our current implementation, and will be added, since it will be required for the scheduling process.

```
Resources
    list-of-all-cognitive-resources          ; Cognitive resources are resources that
                                             ; may be shared. This set is dictated by
                                             ; Task Loading Model. Hence, if Task
                                             ; Loading Model's model has 4 (V A C P)
                                             ; channels, then this will be a list of 4
                                             ; elements. In the event Task Loading
                                             ; Model  decides to have a 5
                                             ; channel model, then, it will be
                                             ; reflected in the slot. This slot helps
                                             ; in the integration.
                                             ; Cognitive resources are analog in
                                             ; nature. Hence, small chunks of such
                                             ; resources can be allocated to separate
                                             ; tasks that are performed in parallel.
     instantaneous-cumulative-visual-load          ; the total visual resources allocated
                                                   ; at each instant.
     instantaneous-cumulative-auditory-load        ; same as above, but for the auditory
                                                   ; resource
     instantaneous-cumulative-cognitive-load
         instantaneous-cumulative-motor-load
```

4.4.1.1.3 Primary Strategy Representation (P)

The primary strategy (P) of the scheduler depends on the pilot's scheduling strategy. If the pilot wants to complete all tasks as soon as possible, then this strategy, translated as MINIMIZE-TIME, is the primary strategy. Z-Scheduler scheduling process always applies the primary strategy at all decision points in its scheduling cycle. Hence, the primary strategy will be the primary means of evaluating between the alternate schedules, and in selecting the best schedule.

```
Primary-strategy
        strategy                ; MINIMIZE-TIME etc.
```

4.4.1.1.4 Secondary Strategy Representation (S)

The secondary strategy is the set of secondary performance measures that are applied to measure the 'goodness' of alternate schedules. Assume that there are two alternate schedules that equally satisfy the primary strategy. In such situations, the secondary performance measures are applied to select the more optimal schedule among the two. Although this structure is not represented explicitly in the design of the Z-Scheduler, its structure as shown below will be implemented during future extensions.

```
Secondary-strategy
        list-of-strategies      ; Each element is a list that has 2 elements.
                                ; The first is an evaluation function and the
                                ; second is the desirable value for it. For example,
                                ; the evaluation function may be
                                ; TOTAL-MOVEMENT-TIME-BETWEEN-TASKS = sum of
                                ; movement time of all tasks from itself to its
                                ; succeeding task, and the desirable value may be
                                ; MINIMAL.
```

4.4.1.1.5 Constraint Representation (C)

Constraints may be classified as shown below in Figure 3. Z-Scheduler takes the approach that constraints from different sources like equipment, causal etc can all be translated into uniform temporal constraints.

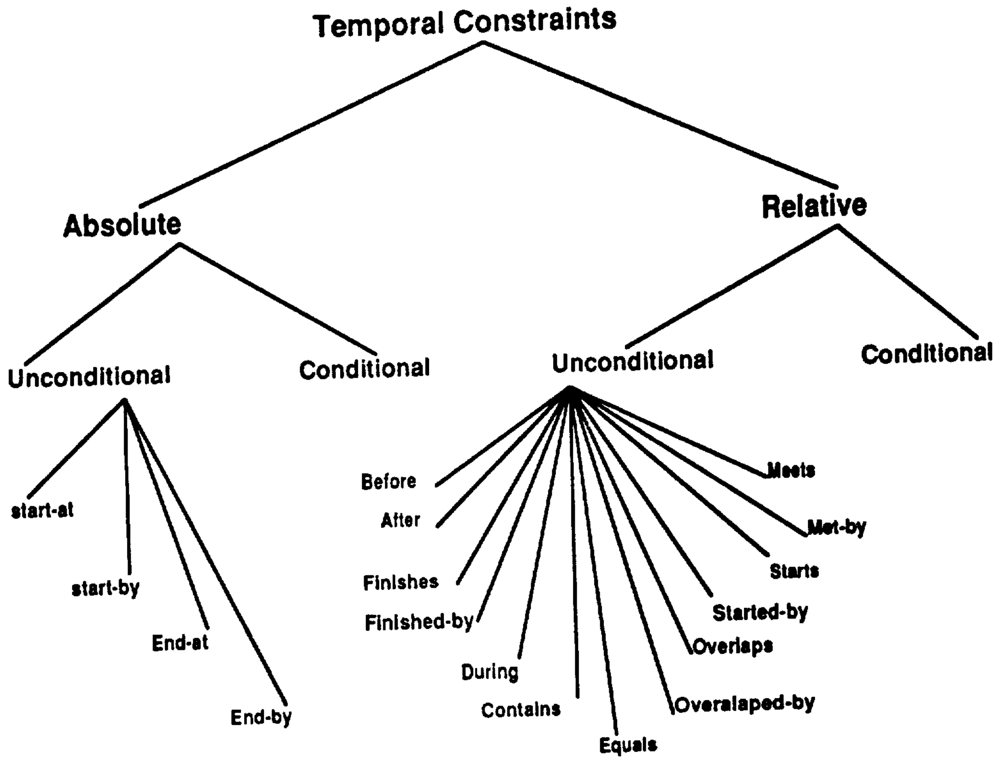


Figure 3. Temporal Constraint Categories in Z-Scheduler

Z-Scheduler differentiates between local and global temporal constraints. The local constraints can be represented in any of the categories shown above and are defined as being local to a particular task. For example, specifying a constraint like, " Task-A before Task-B" relates only to Task-A and Task-B and not to other tasks on the stack . Global constraints are constraints that are applicable across all tasks, need never to be defined by the user of the Z-scheduler and are internal to the scheduler. These constraints are shown below

1. LET > EST
2. LET-EST >= Duration
3. ET > ST
4. ET-ST = Duration

The global constraints are used by the Truth maintainer knowledge source to check for possible conflicts in the current schedule.

Among the local constraint categories, Z-Scheduler is currently geared to handle the "Unconditional-Absolute" and the "Unconditional-Relative" categories. These local constraints may be compounded to give complex constraints. The connection between these local singular constraints can be made using logical relations like OR, AND etc.

Considering the OR logical connective between singular constraints, we have disjunctions among constraints, within which we can specify categories as shown in Figure 4 below.

# of tasks considered # of constraint categories	One	Many
One	eg. (before A B) SAME-RELATION-WITH-SAME-TASK	eg. (OR(before A B) (before A C) (before A D)) SAME-RELATION-WITH-MANY-TASKS
Many	eg. (OR(before A B) (equals A B) (overlaps A B)) MANY-RELATIONS-WITH-SAME-TASK	eg. (OR(before A B) (equals A D)) MANY-RELATIONS-WITH-MANY-TASKS

Figure 4.. Categories of Disjunction Relationships

In short, Z's local constraint representation is a merger of both the qualitative and the quantitative time constraints and hence is more expressive than just any one of the representations alone. It must be remembered here that Z-Scheduler uses the hybrid structure of temporal constraints only for representation, and translates these constraints into quantitative constraints for temporal reasoning, as will be clear later in the section on the Constraint Solver Knowledge Source.

4.4.2 Performance and Quality Engineering

During a benchmarking session for the Z-Scheduler, the time taken for scheduling nine tasks having four constraints is about 9 minutes. The 70% of the time taken for the schedule generation was spent on performing input/output to the displays. Also, for this instance, the scheduling solution space was large, or, in other words the solution density was high and hence there was an increase in the deliberation time because the Z-Scheduler was trying many more of the promising start time and end times for each task. Given below are some of the performance improvements the author has made to fine tune the Z's deliberation time to a minimum.

1. Local Blackboards with each KS: It is a well known fact that greater a databases' size the more time it takes to access objects from it. Similarly, viewing the Blackboard as a common shared database among the knowledge sources, it becomes computationally expensive to access and add objects in the Global blackboard. One of the solutions implemented to

overcome the above problem was to strictly use only the information that is common across two or many knowledge sources in the blackboard. The information that is required only by a single knowledge source is stored in its own local blackboard and only information pertaining to other knowledge sources are posted on the Blackboard. The concept is similar to performing intermediate calculations on a scratch pad and only use the results of the calculation in the next step while solving problems. Since, in GEST the knowledge sources and their local blackboard are spaced closer in the physical memory, the access of an object is twice as fast as when the same access is from the global blackboard.

2. Dynamic Priority of the knowledge sources: Although the blackboard architecture was initially chosen for its intuitive parallel processing architecture, it was noticed that most of the problem decomposition into knowledge sources turned out to be functional. This functional decomposition of the knowledge sources dictated that most of them to be activated serially. The mechanism in GEST to order knowledge sources is in terms of shared and unshared queues. The shared queues are structures that allow simultaneous execution of the knowledge sources (KS). For example when KS1 and KS2 belong to the same shared queue, and at any instant in time they both have rules that are active — assume it is rule11 and rule12 in KS1 and rule 21 and rule22 in KS2. Then the execution will allow rule11 in KS1 to fire, fire rule21 in KS2, fire rule12 in KS1 and then fire rule22 in KS2. If KS1 and KS2 belonged to the same unshared queue, then order of rule firings will be rule11, rule 12, rule 21, rule22. Also, in both cases in every instance of time the controller orders all the knowledge sources to inspect within itself the rules ready for firing which means that each of the knowledge sources in the same queue must match in the patterns in its rules to the contents in the Blackboard and its corresponding local blackboard to identify the candidate active rules. Since most of our knowledge sources can only be activated in sequence, this uniform instant to instant matching, conflict resolving in each KS was seen to be expensive in terms of computational time. Hence, the design of the KS was such that any time one KS was invoked, the first action would be to increase its own priority so as to thwart any other KS to get the processing time. Also, when the knowledge source realizes its function to be nearing completion it gracefully and voluntarily decreases its priority so as to allow for other activated knowledge sources to run. The only exception to the above dynamic priority to the KS is the Conflict resolver (Truth Maintainer) KS, since the very function of this KS is to quickly identify conflict and this function is an alltime one and hence this KS constantly maintains higher priority than the rest of the knowledge sources.

3. Partitioning the Blackboard: As mentioned in above, one of the bottlenecks in the performance of a blackboard application is the retrieval of objects from the blackboard. In the Z-Scheduler, we have divided up the space using the frame hierarchy concept provided in the GEST framework.

4. Choosing Extended Task-based scheduling: One of the crucial metrics for CSP algorithms is the number of backtracks — the greater the number lesser its time-efficiency. The CSP algorithm chosen for the Z-scheduler is an extended version of the task-based scheduling approach — extended to schedule for multiple shareable resources with lower deliberation times. This approach takes care of the interaction that arises between the scheduled tasks and the unscheduled ones up front and hence, minimizes on the causes for the conflicts, thereby reducing the number of backtracks.

4.4.3 Implementation Constraints

The Z-Scheduler is built on top of a knowledge-based tool from Georgia Tech Research Institute called Generic Expert System Tool (GEST). Hence, appropriate access to GEST tool is required for running the Z-Scheduler.
Details on acquiring GEST should be directed to

Stefan Roth
GTRI, Atlanta, GA

spr@prism.gatech.edu

Since GEST keeps track of all the changes made to the working memory of all the knowledge sources during the execution of the Z-Scheduler, 10000 blocks of paging space is suggested for an acceptable response of the scheduler. This quantity of paging space may be higher for an input containing more than 100 tasks.
Also it is suggested that between few runs of the Scheduler an immediate GC be performed.

5.0 DESIGN

5.1 ARCHITECTURAL DESIGN

The architecture for the Z-Scheduler is shown in Figure 5.

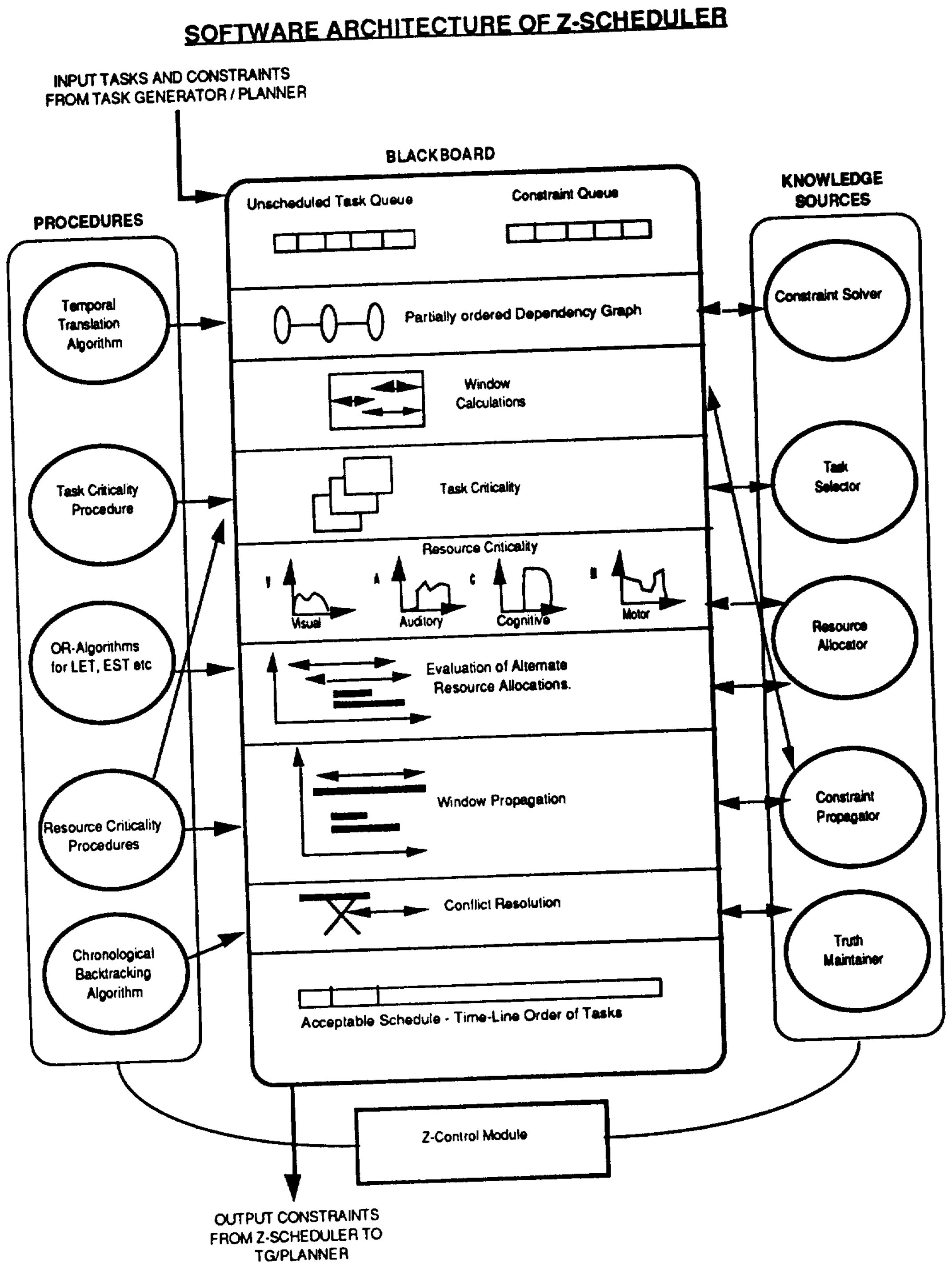

SOFTWARE ARCHITECTURE OF Z-SCHEDULER
INPUT TASKS AND CONSTRAINTS FROM TASK GENERATOR / PLANNER
BLACKBOARD
PROCEDURES
KNOWLEDGE SOURCES
Unscheduled Task Queue
Constraint Queue
Temporal Translation Algorithm
Partially ordered Dependency Graph
Constraint Solver
Window Calculations
Task Criticality Procedure
Task Criticality
Task Selector
Resource Criticality
V
A
C
M
Visual
Auditory
Cognitive
Motor
OR-Algorithms for LET, EST etc
Evaluation of Alternate Resource Allocations.
Resource Allocator
Resource Criticality Procedures
Window Propagation
Constraint Propagator
Chronological Backtracking Algorithm
Conflict Resolution
Truth Maintainer
Acceptable Schedule - Time-Line Order of Tasks
Z-Control Module
OUTPUT CONSTRAINTS FROM Z-SCHEDULER TO TG/PLANNER

Figure 5. Software Architecture of Z-Scheduler

Z-Scheduler uses a blackboard architecture. This architecture allows for its components to be modular and easily extendible.The components of Z-Scheduler are called knowledge sources and each component represents a stage in the scheduling process. Z-Scheduler itself is modular and is called on an "as-needed" basis by the decision making module. Z-Scheduler uses the extended task-based decomposition (a "divide-and-conquer" technique) to partition the scheduling problem space into subspaces, finds the local solutions for each of the subspaces and threads the individual local solutions along to derive the global scheduling solution.

Z's scheduling problem is been classified into sub-problems using the interaction-sensitive task-based scheduling paradigm. This task-based scheduling problem decomposition has been extended to apply to multiple shareable resources, to handle rescheduling and tuned for faster performance. Incorporating Z-Scheduler into the symbolic operator model allows the model to reason about performance on tasks projected into the future (the temporal horizon), thereby enhancing the predictive capability of the MIDAS task analysis.

Before discussing the details of the Z-Scheduler, we need to characterize our scheduling problem. Doing so, will help the design of the scheduler. Based upon the nature of the input we can fit our scheduling problem into the category of *dynamic multi-stage routing* problem (Ben-Arieh, 1985). It is dynamic because the set of input tasks change (due to unexpected events) over time. It is multi-stage because all the tasks to be scheduled may not be homogeneous (in the same level of the mission decomposition structure); hence, the more abstract tasks may have to be expanded further by the scheduler. It is a routing problem because there may be situations where a task can be performed by alternate resources.

Traditional OR techniques have failed to provide effective solutions for such complex problems (Si Ow *et al*, 1987,1988). Hence, we must turn to the more recent approaches to scheduling for solutions. These applications have used AI techniques such as constraint satisfaction algorithms, temporal reasoning mechanisms, search space pruning heuristics, frame-based schedule representation, and problem space partitioning with considerable success.

5.1.1 Design Approach and Tradeoffs

This section explains the categorization of scheduling problems. Specifically this section identifies the category that the A^3I scheduling problem belongs to.

Scheduling problems are classified based on the following categories.

1. Static/Dynamic : This classification is based upon the arrival of tasks to be scheduled. If all the tasks to be scheduled are available initially, then it is a static problem. If the tasks arrive continuously then it is a dynamic problem.

2. Single-Stage/Multi-Stage : This classification is based on the number of subtasks comprising a task. If a task cannot be further divided into subtasks then it is a single-stage problem. If a task can be divided into subtasks then it is a multistage problem.

3. Queuing/Routing : This classification is based on the execution resources possessed by the tasks. If a task can be executed only by a unique resource then it is a queuing problem (queue for that particular resource). If a task can be executed on alternate resources than it is a routing problem (route to an alternate resource to process a task).

The first categorization can also be viewed as a Scheduling/Rescheduling (static) *vs* Reactive scheduling (dynamic) problem. At this point we want to distinguish between Rescheduling and Reactive scheduling. Rescheduling is the scheduling process in which every time the task queue changes, the ongoing scheduling process is interrupted, the schedule so far is dropped and the scheduling is started afresh again. Reactive scheduling is the scheduling process where every time the task queue changes, the scheduling process is interrupted to consider the addition to the task queue and scheduling this task by working around the existing schedule. It has been clear from the beginning that the MIDAS scheduling problem is a Reactive (dynamic) scheduling problem. Additionally, we make the following assumptions:

Assumption 1: In the MIDAS scheduling problem the tasks arrive continuously.

Assumption 2: At present, we consider that the tasks to be scheduled are primitive units which cannot be decomposed further. Hence we have a single-stage problem. One of the future directions will be to schedule higher levels of the mission plan. Then our problem extends to being a multi-stage one.

Assumption 3: Also, currently we assume that a task can primarily be performed only by one resource. This assumption makes our problem a queuing one. In the future when this assumption is lifted then problem becomes a routing one.

Therefore, the current MIDAS scheduling problem is a *dynamic single-stage queuing problem* with the possibility of extending it to be a *dynamic multi-stage routing problem*. Scheduling can always be viewed as a constraint satisfaction problem (J. Allen, 1983).

Given the above description of Z-Scheduler's problem, it has been concluded in past research that traditional OR algorithms fail to arrive at an acceptable schedule in an acceptable time. Hence, we turned to knowledge based techniques to solve this scheduling problem. Knowledge based techniques employ heuristics to arrive at an acceptable schedule in an acceptable time by using these heuristics to limit the number of possible schedules explored. The general approach while using this technique is to model the scheduling problem as a constraint satisfaction problem (CSP). CSP has a structure as shown in Figure 6 below.

CONSTRAINT SATISFACTION PROBLEM

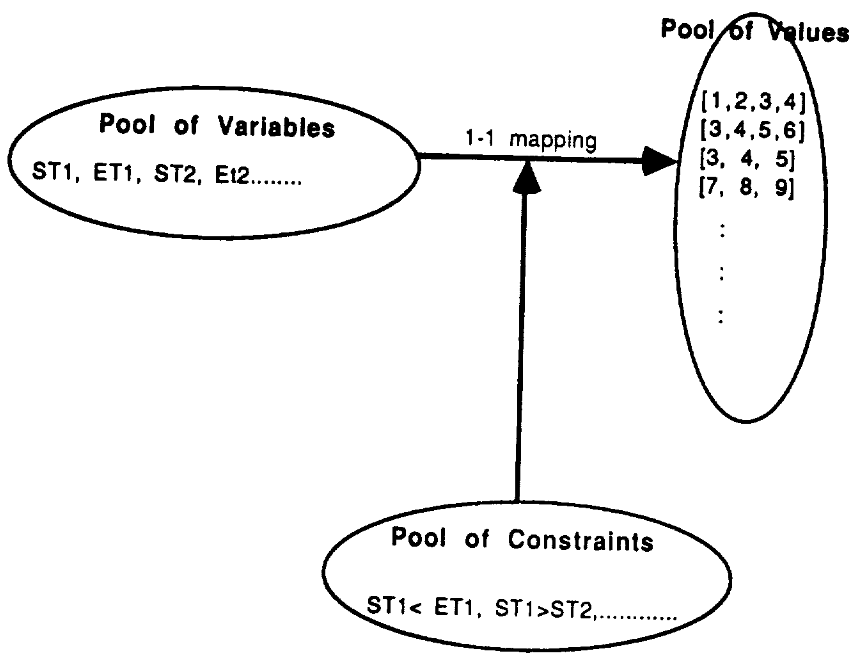


Figure 6. Constraint Satisfaction Problem

CSP contains a pool of variables that have a corresponding domain of values and the aim of the algorithm that solves the CSP is to assign for all the variables a value from its corresponding domain such that all the assignments do not violate any constraints between the variables defined in the constraint pool. Whenever a problem is modeled in this manner, the solution space is divided into subspaces and the local solutions are found for each space and these local solutions are threaded along to get the global solution. The method in which the problem space is divided is crucial to the efficiency of the algorithm that solves the CSP. One of the metrics for the CSP algorithms efficiency is the number of backtracks performed; lower the number of backtracks better the algorithm. In the Z-Scheduler we have chosen the more recent task-based scheduling to partition the scheduling space, since the algorithm that can solve the subproblems takes care of the interaction between variables due to the constraints up front and hence minimizes on the backtracks to handle the conflicts. The process is explained in depth in the Task Selector and Resource Allocator knowledge source sections, Sections 5.1.2.2 and 5.1.2.3.

5.2 DETAILED DESIGN

5.2.1 Detailed Design Description

This section contains the descriptions of the procedures present in each of the knowledge sources along with a description of the controlling mechanism that monitors the execution of these knowledge sources.

5.2.1.1 Constraint Solver Knowledge Source

The Constraint Solver knowledge source primarily does a pre-processing function that transforms the given constraints into a form that the other knowledge sources can infer over. Preprocessing here means that the process is performed before any of the actual scheduling processes and is performed only once during each run of the Z-Scheduler. The constraint solving process can be divided into two phases, the constraint ordering phase and the partial dependency graph (PDG) generation phase.

During the constraint ordering phase, the input constraint list is ordered according to the following heuristics:

1. Order the tasks according to their user defined priority.
2. Group all the singular constraints of a task in the order specified in earlier step.

In the current version of the Z-Scheduler this step is not implemented, but the author sees this as an important part of the constraint solver knowledge source that may help in avoiding conflicts in the schedule later in the scheduling process.

The creation of PDGs is one of the main functions of the constraint solver KS. The PDG is created by inferencing on the set of temporal constraints specified in the above order. This process involves accessing the slot values contained in each constraint frame one at a time and shaping the PDGs. Concurrently it also encodes filters for the EST-value, LET-value, ST-values and ET-values by performing look-ups to the temporal transform tables. The notion of applying constraints one at a time is generally attributed to the work by Mark Stefik in MOLGEN and is termed constraint posting. Additionally, during the PDG generation process, the hybrid constraint representation is translated into point-based filters by performing lookups to the temporal transform tables as shown below.

\ Constraints \ \ Task-A's \ Slot-values \	Before	During	Overlaps	Meets	Starts
Example	(Before A B)	(During A B)	(Overlaps A B)	(Meets A B)	(Starts A B)
EST-value					
EST-constraints-unscheduled		EST(A) = EST(B) +1	EST(A)=EST(B) +1-D(A)	EST(A) = EST(B)-D(A)	EST(A) = EST(B)
EST-constraints-scheduled		EST(A) = ST(B) +1	EST(A)= ST(B) +1 -D(A)	EST(A) = ST(B)-D(A)	EST(A) = ST(B)
LET-value					
LET-constraints unscheduled	LET(A) = LET(B) -D(B)-1	LET(A)= LET(B) -1	LET(A)=LET(B) -D(B)-1+D(A)	LET(A) = LET(B)-D(B)	LET(A)= LET(B)- D(B) +D(A)
LET-constraints scheduled	LET(A) = ST(B) -1	LET(A)= ET(B) -1	LET(A)= ST(B) -1 +D(A)	LET(A) = ST(B)	LET(A) = ST(B) +D(A)
Start-time-value					
Start-time-cons- -traints		ST(A) > ST(B)	ST(A) < ST(B)	ST(A) = ST(B)-D(A)	ST(A) = ST(B)
End-time-value					
End-time-cons- traints	ET(A) < ST(B)	ET(A) < ET(B)	ET(A) > ST(B) ET(A) < ET(B)	ET(A) = ST(B)	ET(A) = ST(B)+D(A)
Duration constraints		D(A) >= D(B)+2	D(A) >= 2		
PDG-constraints	P(A) < P(B)	P(A) = P(B)	P(A) = P(B)	P(A) = P(B) - 1	P(A) = P(B)

where P(A) = position of task A in its PDG.

.c8.Table 1. Temporal Transform — Unconditional-Relative Constraint Propagation

Constraints \ Task-A's Slot-values	Equals	Started-by	Finished-by
Example	(Equals A B)	(started-by A B)	(finished-by A B)
EST-value			
EST-constraints -unscheduled	EST(A)=EST(B)	EST(A)=EST(B)	EST(A)=EST(B)+D(B)-D(A)
EST-constraints -scheduled	EST(A)=ST(B)	EST(A)=ST(B)	EST(A)=ET(B)-D(A)
LET-value			
LET-constraints -unscheduled	LET(A) = LET(B)	LET(A)=LET(B)-D(B)+D(A)	LET(A)=LET(B)
LET-constraints -scheduled	LET(A) = ET(B)	LET(A)=ST(B)+D(A)	LET(A)=ET(B)
Start-time-value			
Start-time-cons- -traints	ST(A) = ST(B)	ST(A)=ST(B)	ST(A)=ET(B)-D(A)
End-time-value			
End-time-cons- -traints	ET(A) = ET(B)	ET(A)=ST(B)+D(A)	ET(A)=ET(B)
Duration-const- -raints	D(A)=D(B)		
PDG-constraints	P(A) = P(B)		

Table 1(cont). Temporal Transform — Unconditional-Relative Constraint Propagation

\ Constraints \ \ Task-A's \ Slot-values \	After	Contains	Overlapped-by	Met-by	Finishes
Example	(After A B)	(Contains A B)	(Overlapped-by A B)	(Met-by A B)	(finishes A B)
EST-value					
EST-constraints unscheduled	EST(A)=EST(B) +D(B)+1	EST(A) = EST(B) + D(B)+1-D(A)	if D(A)<D(B) EST(A) = EST(B) +D(B)+1- D(A) otherwise EST(A)=ST(B)+1	EST(A) = EST(B) +D(B)	EST(A)= - EST(B) +D(B) -D(A)
EST-constraints scheduled	EST(A) = ET(B) +1	EST(A)= ET(B) +1-D(A)	if D(A)<D(B) EST(A)=ET(B)+1 -D(A) otherwise EST(A)=ST(B)+1	EST(A) = ET(B)	EST(A)= - ET(B) -D(A)
LET-value					
LET-constraints		LET(A)= LET(B) -D(B)-1 +D(A)	LET(A) = LET(B) -1 +D(A)	LET(A)= LET(B) +D(A)	LET(A)= LET(B)
LET-constraints scheduled		LET(A)= ST(B) -1+D(A)	LET(A)= ET(B)-1 +D(A)	LET(B) = ET(B) +D(A)	LET(A)= - ET(B)
Start-time-value					
Start-time-cons-traints	ST(A) > ET (B)	ST(A) < ST(B)	ST(A) < ET(B) ST(A) > ST(B)	ST(A) = ET(B)	ST(A)= - ET(B) -D(A)
End-time-value					
End-time-cons-		ET(A) > ET(B)	ET(A) > ET(B)	ET(A) = ET(B) +D(A)	ET(A)= traints ET(B)
Duration constraints		D(A) <= D(B)-2	D(A) >= 2		
PDG-constraints	P(A) > P(B)	P(A) = P(B)	P(A) = P(B)	P(A) = P(B) + 1	P(A)= P(B)

Table 1(cont). Temporal Transform — Unconditional-Relative Constraint Propagation

\ Constraints \ \ \ Slot-values \	Start-at	Start-by	End-at	End-by
Example	(Start-at A 5)	(Start-by A 6)	(End-at A 7)	(End-by A 8)
EST-value				
LET-value		LET = 6 + D(A)		LET = 8
Start-time-value	5			
End-time-value			7	

.c8.Table 2 Temporal Transform — Unconditional Absolute Constraint Propagation

The PDG generation stage includes taking one constraint at a time and inserting it into each of the tasks and at the same time updating the partial dependency graph. The process will result in strips of partial dependency graphs being formed. The function that forms the PDG takes in two tasks, say, task-A and task-B, and PDG that the tasks belong to, say PDG-1 and PDG-2, and works in the manner as shown below.

CASE-1 : When PDG-1 = PDG-2 = nil
form a new-PDG frame named PDG-AB with the task-order-list as (A B).
insert into member-PDG slot of each of the tasks the new-PDG frame i.e., PDG-AB.

CASE-2: When PDG-1 = nil and PDG-2 = PDG-B
Insert Task-A into PDG-B and update Task-A's member-PDG slot to point to PDG-B.
Insert Task-A into PDG-B

CASE-3: When PDG-1 = PDG-A and PDG-2 = nil
Insert Task-B into PDG-A and update Task-B's member-PDG slot to point to PDG-B.
Insert Task-B into PDG-A

CASE-4: When PDG-1 = PDG-2 = PDG-AB
Do not update the member-PDG slots in the respective task frames.

CASE-5: When PDG-1 = PDG-A and PDG-2 = PDG-B and PDG-A != PDG-B
Merge PDG-A and PDG-B into a new PDG = PDG-AB
update the member-PDG slots of task-A and Task-B.

5.2.1.2 Task Selector Knowledge Source

The Z-Scheduler uses task-based scheduling for partitioning the scheduling problem space. Task-based scheduling is a two step process. The first step involves the selection of a single task from the task stack. The second step involves the commitment of a resource's time slice to the selected task by evaluating all alternative resource allocations for that task and selecting the best allocation based on certain decision metrics. Each of these two steps is carried out in separate knowledge sources, the first being implemented by the task selector knowledge source and the second being carried out by the resource allocator knowledge source.

The metric in selecting tasks for resource allocation, is called the Task Criticality (TC) measure. Task Criticality in its most simple form is the ratio of the execution time to the window duration — that is, it is a measure of the slack of a task. The more the slack on the task less its task criticality. If TC equals 1, then it means that the task is very critical because it has only a single solution for the task and does not have alternate solutions. Since, the task-based scheduling (Rossi, Keng,1989) can only handle single resource domains (schedules tasks that use a single machine), we had to extend this approach to handle the multi-resource domain (dictated by the nature of HCI tasks). Each of these resources were shareable (at any moment in time, a resource can handle more than 1 task by giving x-units of resource to task-1 and y units of resource to task-2). Hence, the TC calculations had to be extended to handle the multi-shareable resources.

The task criticality extended in the Z-Scheduler is a measure having three factors, slack on the task, the user defined priority of the task and the resource requirement for the task. The heuristic applied in the task selector knowledge source is that the task with highest task criticality becomes the candidate for resource allocation.

Along with the task to be scheduled, the heuristic had to select a resource that needed to be focussed upon and the logic is outlined below.

Task-criticality ::= TC ::= Priority * (Max[V A C P] * (duration / (LET - EST)))

Task-selected ::= t ::= Task with Max TC

Resource-selected ::= R ::= Max [V A C P] of t

TC-for-tasks-that-use-resource-R-in-the-window-duration-of-task-t ::= (TC)r ::= Priority * R * (duration/LET-EST))

5.2.1.3 Resource Allocator Knowledge Source

Similar to calculating the criticality of task, the resource criticality is calculated in the resource allocator knowledge source. The metric for allocating a resource time slice for the selected task is to pick the solution that has minimum resource competition, the logic being that the solution must not eat away into the opportunity of the tasks that have not yet been scheduled. This metric is called the resource competition measure.

Resource-Competition-in-a-time-period ::= (RC)t ::= SUM of (TC)r of tasks in that time-period

Resource-Competition-for-solution ::= (RC)s ::= SUM of (RC)t for the solution.

Resource-allocated-to-task-t ::= Min (RC)s ::= The solution selected is the one having the minimum resource competition

Hence, if there are alternate solutions to allocating the resource to a task then, Z-Scheduler picks the solution that has the least impact on the rest of the tasks, i.e, it selects the time period of the resource that is least competitive.

5.2.1.4 Constraint Propagator Knowledge Source

The constraint propagator knowledge source is a record keeping knowledge source. Every time a task is scheduled the constraint propagator knowledge source updates the rest of the task's window of opportunity by propagating the temporal constraints in the partial dependency graphs.

The effect of allocating time units of a resource to a task is propagated in two stages. The first stage involves the propagation through the single PDG that contains the scheduled task and is called the Intra-PDG-propagation stage. The second stage includes the propagation between all the PDGs that have tasks that overlap the time units of the resource assigned and is defined as the Inter-PDG-propagation stage.

5.2.1.4.1 Intra-PDG-propagation

The local constraints that relate two or more tasks help identify the preceding-tasks, succeeding-tasks and concurrent-tasks for every task and this is useful for the purposes of generalizing the constraint propagation mechanisms. The order in which the constraints are propagated also uses the grouping below. The constraints belonging to the preceding-tasks of the task just scheduled are first propagated, followed by the constraints belonging to the succeeding-tasks and lastly, the constraints belonging to the concurrent-tasks are propagated. The reason for doing so is that the preceding- and the succeeding-tasks reduce the window of opportunity much more than the constraints belonging to the concurrent-tasks.

The following task categories are formed by the given constraints and are as shown below.

Preceding-tasks	= tasks that are constrained by the 'Before or Meets constraint types.
Succeeding-tasks	= tasks that are constrained by the 'After, Met-by constraint types.
Concurrent-tasks	= tasks that are constrained by the Overlaps, Overlapped-by, During, Contains, Starts, Started-by, Finishes, Finished-by, and Equal constraint types

The windows are initially calculated for the first time immediately after the constraint solving knowledge source has finished execution. Incremental OR algorithms calculate the initial window of opportunity for all the tasks. The window of opportunity is represented as a time slice that is bound on the left by Earliest Start Time (EST) value and is bound on the right by the Latest End Time (LET) value. The EST for each task is calculated by finding the tasks that do not have any preceding tasks and assigning to the EST value the start of the input temporal horizon and forward chaining to calculate the EST of the rest of the tasks and is shown in detail below.

Initial window calculations:

Forward chaining to calculate the EST

```
For all tasks with

    Preceding-tasks = nil
    If ST or ET = nil i.e they are not scheduled yet
    Then  EST = start of the given temporal horizon
       Loop for all succeeding-tasks of task
         apply constraint-propagate-unscheduled (test only EST-constraints)
    Else Loop for all succeeding-tasks of task
         apply constraint-propagate-scheduled (test ST constraints and if fails
                                   then EST-constraints)
```

Backward chaining to calculate the LET

```
For all tasks with
    Succeeding-tasks = nil
    If ST or ET = nil i.e they are not scheduled yet
    Then  LET = end of the given temporal horizon
       Loop for all preceding-tasks of task
         apply constraint-propagate-unscheduled (test only LET-constraints)

    Else Loop for all succeeding-tasks of task
         apply constraint-propagate-scheduled (test ET constraints and if fails
                                   then LET-constraints)
```

The windows are updated after the execution of the resource allocation stage using a recursive algorithm as shown below.

```
Recursion
    Loop for each task constrained by the just scheduled task
        calculate the EST and LET using the EST and LET constraint formulas
        If EST and LET changed - then recursively do the recursion.
```

Calculation of the EST and LET from the constraint formulas is invoked by demon procedures provided by the GEST tool. Depending which demon invokes the propagation of the values, one of the following procedures are called. If the value being changed is the EST or LET values, then 'constraint-propagation-unscheduled'is called. If the values were ET or ST then the 'constraint-propagation-scheduled' function is called. Each of these functions is explained below.

constraint-propagation-scheduled

```
For each Predecessor in the preceding-tasks
    If predecessor is scheduled
    then do nothing.
    else  Identify ST-filter that relates the scheduled-task and the Predecessor
          Apply that filter
          If ST-value changes for the Predecessor
          Then call constraint-propagation-scheduled          ;; recursion!!
          Identify ET-filter that relates the scheduled-task and the Predecessor
          Apply that filter
          If ET-value changes for the Predecessor
                  Then call constraint-propagation-scheduled    ;;recursion!!
    Identify EST-filter that relates the scheduled-task and the Predecessor
    Apply that filter
        If EST-value changes for the Predecessor
                Then call constraint-propagation-unscheduled
    Identify LET-filter that relates the scheduled-task and the Predecessor

    Apply that filter
        If LET-value changes for the Predecessor
                Then call constraint-propagation-unscheduled

For each Successor in the succeeding-tasks
    If successor is scheduled
    then do nothing.
    Identify ST-filter that relates the scheduled-task and the successor
    Apply that filter
        If ST-value changes for the successor
                Then call constraint-propagation-scheduled    ;;recursion!
    Identify ET-filter that relates the scheduled-task and the successor
    Apply that filter
        If ET-value changes for the successor
                Then call constraint-propagation-scheduled    ;;recursion!
    Identify EST-filter that relates the scheduled-task and the successor
    Apply that filter
```

```
            If EST-value changes for the successor
                    Then call constraint-propagation-unscheduled
        Identify LET-filter that relates the scheduled-task and the successor
        Apply that filter
            If LET-value changes for the successor
                    Then call constraint-propagation-unscheduled

    For conc-task in the concurrent-tasks
        If conc-task is scheduled
        then do nothing.
        Identify ST-filter that relates the scheduled-task and the conc-task
        Apply that filter
            If ST-value changes for the conc-task
                    Then call constraint-propagation-scheduled     ;;recursion!
        Identify ET-filter that relates the scheduled-task and the conc-task
        Apply that filter
            If ET-value changes for the conc-task
                    Then call constraint-propagation-scheduled     ;;recursion!
        Identify EST-filter that relates the scheduled-task and the conc-task
        Apply that filter
            If EST-value changes for the conc-task
                    Then call constraint-propagation-unscheduled
        Identify LET-filter that relates the scheduled-task and the conc-task
        Apply that filter
            If LET-value changes for the conc-task
                    Then call constraint-propagation-unscheduled
```

<u>constraint-propagation-unscheduled</u>

```
    For each Predecessor in the preceding-tasks
        If predecessor is scheduled
        then do nothing.
        Identify EST-filter that relates the unscheduled-task and the Predecessor
        Apply that filter
            If EST-value changes for the Predecessor
                    Then call constraint-propagation-unscheduled
        Identify LET-filter that relates the scheduled-task and the Predecessor
        Apply that filter
            If LET-value changes for the Predecessor
                    Then call constraint-propagation-unscheduled

    For each Successor in the succeeding-tasks
        If successor is scheduled
        then do nothing.
        Identify EST-filter that relates the unscheduled-task and the successor
        Apply that filter
            If EST-value changes for the successor
                    Then call constraint-propagation-unscheduled
        Identify LET-filter that relates the scheduled-task and the successor
        Apply that filter
            If LET-value changes for the successor
                    Then call constraint-propagation-unscheduled
```

```
For conc-task in the concurrent-tasks
    If conc-task is scheduled
    then do nothing.
    Identify EST-filter that relates the unscheduled-task and the conc-task
    Apply that filter
        If EST-value changes for the conc-task
                Then call constraint-propagation-unscheduled
    Identify LET-filter that relates the unscheduled-task and the conc-task
    Apply that filter
        If LET-value changes for the conc-task
                Then call constraint-propagation-unscheduled
```

5.2.1.4.2 Inter-PDG-propagation:

Calculate the loading profile for the duration of the task-just-scheduled. For every peak (the max resource capacity) form a frame for it, give it a name and assign its start and end times.

```
ALGORITHM-1:
This algorithm saves initial procedures for iteration but does more checking.
   Loop for each task in the Inter-DG-concurrent tasks
    let EST-temp(task) = EST (task)
        LET-temp(task) = LET (task)
        Loop for peak and next-peak in peaks for all odd numbered peaks
            If EST-temp (task) < = ST(peak)
            Then If LET-temp > ST(next-peak)
                 Then MAY-BE-EST-n = EST-temp
                      MAY-BE-LET-n = ST(peak)
                      MAY-BE-EST-n+1 = ET(peak)
                      MAY-BE-LET-n+1 = LET-temp (task)
                 Else                                ;;; i.e LET-temp >ST (next-peak)
                  MAY-BE-EST-n = EST-temp
                  MAY-BE-LET-n = ST(peak)
                  MAY-BE-EST-n+1 = ET(peak)
                  MAY-BE-LET-n+1 = ST(next-peak)
                  EST-temp = ET(next-peak)
              Else If LET-temp > ET (peak)        ;;; i.e., EST-temp > ST (peak)
                  Then If LET-temp < ST(next-peak)
                         Then
                           MAY-BE-EST-n = ET(peak)
                           MAY-BE-LET-n = LET-temp
                           EST-temp = ET(next-peak)
                         Else
                           No Solution - hence backtracking required.
         Find the validity of each of the windows by checking if
         LET-x(task) - EST-x (task) >= Duration ( task)
         If false, then drop that window duration from the list
         possible-windows.
         Among the valid windows select the largest one as the prime
         candidate.
         Call Intra-DG propagation using the largest window duration as
         the changed reference.
```

5.2.1.5 Truth Maintainer Knowledge Source

The Truth Maintainer knowledge source is a high level monitor of the constraint propagating activities. Whenever a conflict occurs during the process of constraint propagating, this KS halts the execution of the other knowledge sources by sending a message via the blackboard. After halting the process that was responsible for the conflict- — an example of the conflict is that the window duration of the task becomes smaller than the estimated duration of the task — the Truth Maintainer attempts to find the reason for the conflict's occurrence and on finding it out, it resets all the knowledge sources to that point in time that brought about the conflict. In other words, the Truth Maintainer is built on the principle of chronological backtracking, where the amount backtracking is to the previous decision point.

The conflicts are described as any violations of the global constraints as described in the section on constraint representation. For example, the global constraint 'LET-EST >= Duration' when violated means that the window of opportunity for a task is less than the duration of the task, hence making the task impossible to schedule. The truth maintenance algorithm, applies all the global constraints whenever any task's EST-value, LET-value, ST-value or ET-value changes during the constraint propagation stage. Currently, Z-Scheduler performs only the chronological backtracking mechanism to handle the conflicts and hence, does not have any domain knowledge to relax the constraint without backtracking. One of the future extensions of the Z-Scheduler is to include domain dependent strategies that handle the conflicts without backtracking, which happens to be a more psychologically valid method for handling the conflicts.

5.2.1.6 Z-Scheduler Display Design

The following section describes several types of displays used in visualizing the scheduling process. Each of these displays in distinct and can be implemented separately. The only place in which they overlap is that clicking on a presentation in one display will bring up a different type of display.

Please note that :

Flavor classes appear in *italic* type.
presentation types appear in ***bold-italic*** type.
Function and variable names appear in **bold** face.

5.2.1.6.1 Task Stack Display

This display shows the order in which the tasks will be performed. This display will be dynamic and will update itself any time the scheduler reorders the tasks. The window will contain a presentation object for each task. A task object will look like a playing card with a label on the top. The task objects will stack on on top of another to show the task ordering.

```
        ---------------
        | Task C     |
      --------------- |
      | Task B     |--
    ---------------- |
    | Task A     |--
    |            |
    |____________|
```

task-stack-window: window flavor which will be used for task stack display.

task-card: presentation type for card like display of task object.

Clicking the mouse on this presentation type will pop up a window describing the task object (see "Task Frame Display" below).

***default-task-stack-character-style*:** Default font used to print task names in task card presentations.

(**show-task-stack** ordered-task-list &key stream
(character-style *default-task-stack-character-style*)
returns a task-stack-window displaying the task ordering. If you specify a stream the stream must be of type task-stack-window. If stream is not specified you'll be prompted with the mouse for the window size and location. Character style specifies font used to print task names.

(**update-task-stack** task-stack-window ordered-task-list &key highlight-time)
The scheduler must call this function to update the task-stack-window whenever it reorders the tasks. If highlight-time is supplied, each task presentation which has changed its position in the occlusion stack will be highlighted. These task presentations will remain highlighted for highlight-time seconds.

5.2.1.6.2 Task Frame Display

This is a simple display which shows the slot values of a task. It will look something like this:

```
Name:  Task A
Agent:  ...
Resource: ...
```

object-frame-window: window flavor for pop up windows showing slots of frame.

constraint: presentations type whose object is a list of (reference-task related-task)
mouse-left action pops up task frame window for related-task.
mouse-middle action pops up window with constraint frame slots and values.

***default-frame-character-style*:** default font used to print slot values.

(**show-task-frame** task superior
&optional (character-style *default-frame-character-style*))

pops up a temporary window showing slot values. Window will disappear when you click the mouse outside it. The function get-task-slots-for-display determines which slots are displayed. superior is the parent window of the pop-up window (can be either color screen or main screen).

(**get-task-slots-for-display**) returns a list of slot-printer-pairs (see description below).

(**show-constraint-frame** constraint superior
&optional (character-style *default-frame-character-style*))

pops up a temporary window showing slot values. Window will disappear when you click the mouse outside it. The function get-constraint-slots-for-display determines which slots are displayed. superior is the parent window of the pop-up window (can be either color screen or main screen).

(**get-constraint-slots-for-display** reference-task related-task)
returns a list of slot-printer-pairs (see description below).

The generic function for popping up a window showing the slots of a frame is **print-frame**. This is the function called internally by **show-task-frame** and **show-constraint-frame**.

(**print-frame** (frame slot-printer-pairs superior
&optional (character-style *default-frame-character-style*))

slot-printer-pairs is a list of pairs
(slot-name printer)
printer specifies how to print the slot value
it must be one of
(non-mouseable-print default-print constraint-print pdg-print)

non-mouseable-print slot value will not be mouseable.
default-print slot value has same mouse functionality as values in lisp listener
constraint-print slot values will be printed as constraint presentations.
pdg-print slot values will be printed as dependency-graph presentations.
(see description of dependency-graph in "Schedule Chart" below)

5.2.1.6.3 Dependency Graphs

This display will contain a directed graph showing the dependency relationships between task execution. For example, the display below shows that Task A must occur before Task B and Task C.

Task B

Task A

Task C

default-task-node-character-style: default font used for labeling nodes in partial dependency graph

default-color-cycle: list of colors we cycle through when drawing links.

pdg-task-node: presentation type for nodes in the graph.
They will be shown as an oval with a label in the middle.
Clicking the mouse on this presentation type will pop up a window describing the task object (see "Task Frame Display" above).

(**show-dependency-graph** pdg &key stream (character-style *default-task-node-character-style*))
returns a window containing the partial dependency graph. If you do not specify a stream you'll be prompted with the mouse for the window size and location.
pdg is an object which contains a list of the tasks. Task objects contain a succeeding-tasks slot which contains a list. Each element in the list is either

1. (task) we draw a single thickness line to the task node

2. (m-t-c-w-s-t task1 .... taskn)
   we draw a double thick line to each task node

3. (s-t-c-w-m-t task1 ... taskn)
   we draw a dashed line to each task node

For all tasks (task1 ... taskn) in case 2 or 3 we draw lines using the same color. ***default-color-cycle*** is a list of colors we cycle through for all succeeding tasks.

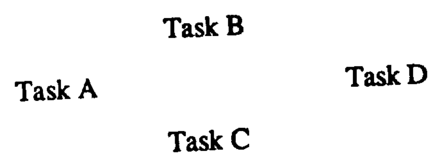
Task objects also contain a concurrent-tasks slot whose value is either nil or a list of tasks.

To construct the partial dependency graph we look at the succeeding-tasks slot and construct the links (drawing lines with arrowheads). We also look at the concurrent-tasks slot and construct those links (drawing plain lines).

For example, if we have pdg-1 with tasks (A B C D) with the following task descriptions we will get the partial dependency graph shown to the right.

task A
succeeding-tasks: ((B) (C))

task B
succeeding-tasks: ((D))



task C
succeeding-tasks: ((D))

task D
succeeding-tasks: nil

5.2.1.6.4 Scheduling Chart (Gantt Chart)

This display is a dynamic display showing the scheduling of tasks.

Initially, for each task within a partial dependency graph there will be a presentation showing the time slice where it is possible to schedule the task. This presentation will look like a line with arrows on the ends (<---> in diagram).

As tasks are scheduled we update the display to show the scheduled times for the tasks. Once a task is scheduled the presentation for that task will change to a solid rectangle (==== in diagram).

The labels on the y-axis are ***dependency-graph*** presentations. Clicking left on them brings displays the partial dependency graph in the proper pane of the frame.

The display will look something like this:

```
    |------------------------------------------------------------------
    |
    |    <-------->
    |         <------->
DG1|                   ========
    |
    |------------------------------------------------------------------
    |
    |
DG2|    ======
    |        <----->
    |           <------------------>
    |
    |
    |____|____|____|____|____|____|____|____|____|____|____|____|____|____|____
```

schedule-window: window flavor. Clicking left in this type of window will bring up a
snapshot-schedule-window which shows the entire Gantt chart.

snapshot-schedule-window: window flavor which show entire Gantt chart. Clicking the
mouse outside this window will make it disappear.

unsched-task-box: presentation for an unscheduled task within
a partial dependency graph. It will look like an thick line with arrowheads on
the ends.
Clicking left on this presentation type will pop up a
window describing the task object (see "Task Frame Display" above).

sched-task-box: as above but for scheduled tasks. Will look
like a solid rectangle.

default-pdg-label-character-style*:** character style of ***dependency-graph
presentations
on left edge of Gantt chart.

***default-task-load-character-style*:** character style used to label the load on task bars
when you are overlaying a load plot on the Gantt chart.

(**show-schedule-window** pdg-list start-time end-time &key stream load-type
cummulative-load-points)
returns a schedule-window. pdg-list is a list of dependency graphs,
start-time and end-time are used for labeling the x-axis.
We get the start and end times for each task in each pdg from the
task objects. If load-type and cummulative-load-points are supplied
we overlay a load plot on this display.

(**modify-schedule-window** task schedule-window)
The scheduler must call this function when it changes the start and/or end
times for a task. Updates presentation for task to reflect time changes.

(**schedule-task** task schedule-window)
The scheduler must call this function when it schedules a task.
Changes presentation for task to be a ***sched-task-box*** presentation.

(highlight-current-task task schedule-window)
The scheduler must call this function each time it begins to work on scheduling a task. Puts a box around the presentation for task.

(snapshot-schedule-window pdg-list start-time end-time)
pops up a snapshot-schedule-window showing everything.

5.2.1.6.5 Visual Loading Plot

This display shows a plot where the x-axis is the same time frame as in the Gantt scheduling chart and the y-axis is the load.

Each task object has slots
st-value: start time
et-value: end time
resource: (V A C P) list containing visual, auditory, cognitive and psychomotor loads.

For each task we put a circle on the plot showing (time, load). The circles will be the color specified in the color slot of the task object.

Also, we construct a line which shows the cumulative load for all tasks at each time tick.

task-load-plot-window: window flavor. Clicking shift-left in this type of window will bring up a menu which allows you to change parameters such as x-min, x-max, number of tick marks etc. and replot the points.

task-load-point: presentation type for load point of a task.
Clicking left on this presentation type will pop up a window describing the task object (see "Task Frame Display" above).

cumulative-point: presentation type for a cumulative load point.
Clicking left will show a cumulative load display. (this display needs to be defined).

default-x-axis-character-style: default character style of labels on x axis
default-y-axis-character-style: default character style of labels on y axis
default-axis-color: default color used for drawing axis and tick marks
default-axis-label-color: default color used for labeling axis
default-cumulative-load-line-color: default color of load line in load plot window

```
(show-load-plot task-list cumulative-load-points load-type
                &key stream
                (x-axis-character-style *default-x-axis-character-style*)
                (y-axis-character-style *default-y-axis-character-style*)
                (axis-color *default-axis-color*)
                (axis-label-color *default-axis-label-color*)
                (load-line-color *default-cumulative-load-line-color*))
```

If stream is provided, displays load plot in stream, otherwise you'll be prompted with mouse for size and location of window. Loads of each task are presented as ***task-load-point*** presentations. Points in cumulative-load-points are presented as ***cumulative-point*** presentations.

Color:

When using the color monitor it is recommended that you change the default character style variable to be be larger than those you'd use on black and white screen. You must also set the color variables.

The variables you should set are enumerated below. See the function **init-color-vars** in file color.lisp.

default-interface-screen default screen on which the interface frame appears
When displaying on a color monitor set to (color:find-color-screen).

All Windows:

default-label-character-style

Task Stack Display:

default-task-stack-character-style

Dependency Graphs:

default-task-node-character-style

Load Plots:

default-x-axis-character-style
default-y-axis-character-style
default-axis-color
default-axis-label-color
default-cumulative-load-line-color

Gantt Schedule Window:

default-pdg-label-character-style
task-load-character-style

dw::*default-ruler-normal-character-style* used for labeling axis margins is Gantt chart
dw::*default-ruler-small-character-style*

Files:

ruler.lisp: margin component for displaying a scale in the margin of a window
presentations.lisp: all presentation types defined for above displays, includes code for presentation actions also.
prim.lisp: some primitive functions and variables common to several display types.
plot.lisp: code for generating load plots.
Gantt.lisp: code for scheduling Gantt chart.
pdg.lisp: code for dependency graphs.
task-stack.lisp: code for task stack display.
print-frame.lisp: code for frame displays.
frame.lisp: constraint frame flavor which includes panes for each display type and possible frame configurations.

color.lisp: variables and functions for displays on color monitors.

5.2.2 External Interface Detailed Design

Getting into the details of the interface requirements for Z-Scheduler, we have outlined them in the implementation-independent BNF form.

The following are Global Variables that should be set by the Task Generator during Z-Scheduler's invocation.

```
Z-invoker ::= #<Z-invoker>
                    ;Flavor instance that has a method (like "make-instance
                    ;after" method) that invokes Z-Scheduler.

*primary-strategy* ::= MINIMIZE-TOTAL-TIME | BALANCE-LOAD
                                        ; The scheduling strategy.

*time-available* ::= integer ; the time period within which all the
                                        ; input tasks need to be performed.

*secondary-strategies* ::=((<performance-measure> <desirable-value>) ... ..)
                              ; Each element is a list that has 2 elements.
                              ; The first is an evaluation function and the
                              ; second is the desirable value for it. For example,
                              ; the evaluation function may be
                              ; TOTAL-MOVEMENT-TIME-BETWEEN-TASKS = sum of
                              ; movement time of all tasks from itself to its
                              ; succeeding task, and the desirable value may be
                              ; MINIMIZE.

<performance-measure ::= ( <function-name> )

<desirable-value> ::= Integer | minimize | maximize

*tasks-to-be-scheduled* ::= (<task-1> <task-2> ....<task-j>...... <task-n>)

<task-j> ::= #<task-j>
                              ; Is an instance of flavor task1.

#<task-j> must have the following accessors.
        name ::= symbol
        agent ::= #< pilot>
        estimated-duration ::= integer          ; integral multiple of tick.
        priority ::= integer                    ; [1-10: default is 5]
        resource-required ::= (<V> <C> <A> <M>)
        constraints-by-Z ::= nil        ; the output will be placed
                                        ; here according to the BNF
                                        ; format specified by Jerry.

#<pilot> := Instance of a pilot flavor.

<V>, <C>, <A>, <M> ::= integers
```

```
*input-constraint-list* ::= (       <singular-constraint-1>
                                    <singular-constraint-2>
                                              :
                                    <singular-constraint-i>
                                              :
                                    <singular-constraint-n>

                                    <disjunct-constraint-1>
                                    <disjunct-constraint-2>
                                              :
                                    <disjunct-constraint-j>
                                              :
                                    <disjunct-constraint-m>)
                                    ; List of n # of singular constraints and m # of
                                    ; disjunct constraints - where n and m are variables.

<singular-constraint-i> ::= (<type> <reference-task>
                                    <relative-task/time>)

<disjunct-constraint-j> ::= (OR <singular-constraint-1>
                                    <singular-constraint-2>
                                              :
                                              :
                                    <singular-constraint-r>)

<type> ::= Before V After V During V Contains V Overlaps V
           Overlapped-by V Meets V Met-by V Starts V Ends V
           Equals V Starts-at V Starts-by V Ends-at V Ends-by

<reference-task> ::= #<task-a>      ; instance of task that is a member
                                    ; of *tasks-to-be-scheduled*

<related-task/time> ::= If <type> = Starts-at V Starts-by V Ends-at V Ends-by
                        Then <related-task/time> ::= integer
                        Else <related-task/time> ::=  #<task-b>
```

Although the above interface principles have been implemented in the Z-Scheduler its soundness remains to be tested.

6.0 USER'S GUIDE

Please refer to the demo manual for the details in using the Z-Scheduler.

6.1 OVERVIEW OF FILE STRUCTURES

All the files constraining the source code for Z-Scheduler is neatly organized under the directory

Puf:>Renuka>Z>.

Each knowledge source has its own directory. Hence the above directory has five sub directories, viz.,

>constraint-solver-ks>

>task-selector-ks>
>Resource-allocator-ks>
>constraint-propagator-ks>
>truth-maintainer-ks>

Each knowledge source directory contains the corresponding files, viz.,

*.rule
*.frame
*.log
*.lisp

6.2 INSTRUCTIONS FOR RUNNING Z-SCHEDULER DEMONSTRATION

The following instructions guide the user through the Z-Scheduler demonstration. The demonstration is performed on Puffer, a Symbolics 3640 in the A^3I lab.

Please note that:

The commands to be typed in by the user are <u>underlined</u>.

The prompt provided by Symbolics is printed in *italics*.

The keys that are to be pressed are **outlined**.

6.2.1 Initial Setup

6.2.1.1 Halt Machine

Type <u>Halt Machine</u> at the *Command:* prompt. The machine comes back with the following prompt (figure 1) -
Do you really want to halt the machine y/n? Just type in <u>yes</u> or <u>y</u> and hit **return**.

6.2.1.2. Boot

Type <u>Boot</u> at the *FEP Command:* prompt, **spacebar** and then type <u>fep0:>Ocean-Gest-Color.boot</u> and hit **return**. A lot of messages will be printed on the screen. This means that the machine is in the process of booting. The whole process will take about 3 minutes.

6.2.1.3 Load Z-Scheduler files

At the *Command:* prompt type <u>Load File Puf:>renuka>z>data-interface>begin-z.lisp</u> and hit **return**.

At the *Command:* prompt type <u>(initial-setup-1)</u> and hit **return**. When a menu asking the type of color monitor present pick *"tectronix color monitor"* from the menu.

At the *Command:* prompt type <u>(initial-setup-2)</u> and hit **return**.

6.2.1.4 Run Z-Scheduler partially

To select the GEST controller, hit the **select** key and then depress the **symbol** and G keys simultaneously and you'll be in the blackboard controller frame.

To start the execution of the Knowledge sources, move the mouse up to the command menu pane and mouse on *KSactions* and click the right button. A menu appears. Drag the mouse down to *Start Execution* and click mouse left button.

This will lead to a lot of activities happening on the Z-frame on the color monitor. Let the Z-Scheduler run for a few minutes. Poise the mouse over the *Suspend* option and immediately after the cumulative resource curves are shown in the VACM panes, click mouse left on this suspend option.

Now, you are all set for demonstrating the Z-Scheduler. You allow the Scheduler run this far in order to focus on the visually compelling parts of Z-scheduling activity. Also, since the time is limited, that is, the whole demonstration should fit the 15 minutes time slot allocated for Z-Scheduler, you want the demo attendees to focus upon the most interesting and intuitive parts of the scheduling cycle.

Move the mouse to the color screen by typing **Function** key and **X** letter key in the respective order. The mouse will now be on the color window.

6.2.2 Input Representation Demo

6.2.2.1 Show a task frame

Mouse and click left on one of the tasks in the stack shown in the "Unscheduled Tasks" window (placed in the upper left hand corner). This action will bring up a button- which looks like a small right angle that moves with the movement of the mouse. This response implies that the window that displays the task description needs to be shaped. In order to do this, click mouse right and then, holding the mouse left button down, outline a window about on quarter the size of the whole color screen and click the mouse left button again. This action leads a display of the details of the chosen task in the window just created with the mouse actions.

After describing the task's input, remove the task window by clicking mouse-left outside the window.

6.2.2.2 Show a constraint frame

To show the constraint frame follow the same procedure as the above, but this time click on one of the cards in the "Used-up Constraint Stack" window (the window place on the upper right hand corner). Also, remember to size the constraint description window only 3 inches in length and breadth. The smaller size of the window is recommended since very little information will be displayed in the constraint description window. Once, again click mouse-left outside the constraint description window to make that window disappear.

6.2.3 Schedule Generation Demonstration

6.2.3.1 Resume Z-Scheduler

Move the mouse over to the B&W monitor by hitting the Function and X keys. Once the mouse is over the B&W screen, mouse on the *Resume* menu option and click mouse-left. This action will spur the Z-Scheduler to continue the scheduling process from the point where it left off due to the previous *suspend* action. After explaining the scheduling process with the aid of the dynamic displays, mouse over the *Suspend* menu option and click mouse-left again.

6.2.4. Scheduling Strategies Comparison Demonstration

6.2.4.1 Bring up the display

In order to bring up the strategies comparison displays, move to the Lisp Listener by hitting Select and L keys simultaneously. At the *Command:* prompt type (display-history-comparison). After about 2 minutes the Gantt charts and the resource loading plots are displayed for the two strategies. Point to the difference and this marks the end of the demo.

7.0 ABBREVIATIONS AND ACRONYMS

A^3I	Army Aircrew Aircraft Integration
APU	Auxiliary Power Unit
BB	Blackboard
CSP	Constraint Satisfaction Problem
EST	Earliest Start Time
ET	End Time
KS	Knowledge Source
LET	Latest End Time
MIDAS	Man-machine Interface Design Assistant
PDG	Partial Dependency Graph
ST	Start Time

8.0 NOTES

8.1 MISCELLANEOUS

During the many demos questions were raised about scheduler also performing the function allocation and viewing the equipment as a resource. Although in the original design of Z-scheduler, equipment were considered as resources, in Phase IV there was already a separate resource-handling mechanism for equipment allocation.

8.2 LIMITATIONS

Although the interface between the Task Generator and the Z-Scheduler is implemented, it has not been exercised and hence its efficiency remains unknown.

8.3 LESSONS LEARNED

The blackboard architecture is a modular architecture and facilitates good software design. However, a lot of analysis of the problem needs to be performed a-priori to coding in order to capitalize on the benefits of this architecture. Z-Scheduler currently uses task-based scheduling and each of the stages in this process is implemented as a separate knowledge source. With a little analysis of this approach it can be seen that there is mostly serial control between the execution of the knowledge sources although there is room for parallelism within each knowledge source.

The concept of considering a time-line for scheduling forces one to always schedule tasks around the chosen time unit. For example when the time line of length 10 seconds is divided into a 10 units of 1 sec each, then each task must start at any of those 10 markings and cannot start at 1. 5 sec etc. Hence, considering a time line of discrete intervals constrains the granularity of the schedule by the time interval of the smallest chosen interval. Of course, the foremost reason behind discretizing the time line is for putting an explicit bound on the time and space complexity of the scheduling algorithm.

8.4 FUTURE DIRECTIONS

As in most research work, there are always numerous ways in which the work can be extended. Below are some of the enhancements that can make Z-Scheduler more powerful.

° **Task Hierarchies**: Currently Z-Scheduler handles tasks that are the leaves of the task hierarchy. It does not have the knowledge to either abstract the tasks into higher groupings nor can it handle tasks given to it as a hierarchy. Z-Scheduler can neither create hierarchies nor can it handle ready-made hierarchies. If Z-Scheduler were to be extended to schedule task hierarchies, I foresee an impact on the time required for scheduling:, i.e., this extension will shorten the time required for scheduling.

° **Resource Hierarchies**: Having an abstraction of the resources might lead to a more sophisticated and cognitively closer resource allocation model. Once again, scheduling these type of resources might reduce the deliberation time (time required for scheduling).

° **Schedule Repair:** Assuming that there will be a tighter interaction between the planner and the scheduler — or the scheduler and the simulation in Phase V of this project — there will be a need to quickly repair the existing schedule to allow to new tasks generated in response to some unexcepted events.

° **Logical relations relating temporal constraints:** In the current version of Z-Scheduler, its constraint mechanism can only handle singular temporal constraints and not the logical relations (ORs, ANDs, EXORs, NOTs) that relate the singular temporal constraints. In step with Kevin Corker's definitions of goals (logical relations between goals) and procedures (temporal relations between procedures), if Z-Scheduler were to handle both goals and procedures uniformly, its constraint mechanism must be extended to reason over logical constraints too.

° **Learning Techniques in the Scheduler:** There have been proven learning techniques in the literature and some of them are yet to be tried in the scheduling domain. From discussions with Sandy Hart's group, it was concluded that investigating in this area too might lead to directions that will lead to a closer cognitive model of human scheduling behavior.

° **Sophisticated backtracking mechanism:** In its present version, the Z-Scheduler uses chronological backtracking (refer to section 5.2.1.5)- Simply stated, it backtracks to the last point where a decision regarding resource allocation was made, finds the alternate choices available and picks the next best alternate and continues to schedules from that point. Instead, if we were to have a dependency directed backtracking — in the event a conflict occurs — the scheduler can backtrack directly to the source of the conflict rather than just blindly backtracking in time. This again is an efficiency measure of the scheduler's behavior.

There are also certain desirable extensions to GEST — the underlying tool that Z-Scheduler uses:

- Time-travel function in the BB-Controller: This function is very useful while debugging an expert system. Although in GEST, this function is available for individual knowledge sources it is not available for the BB-controller. It would be useful to backtrack to a known state whenever some knowledge sources triggered by the controller does not behave as expected.

- Justification Network for each of the knowledge sources: This feature exists in ART, a commercial expert system shell , and maps the nodes as objects in databases with the arcs being specific rules that explain why a specific object was changed the way it did. It is a useful graphical explanation utility which is lacking in GEST and hence a candidate for future extension feature in GEST.

- Functions for viewing the conflict set: Before the conflict resolution stage and the firing stage of the rules it would be useful to view all the rules that match the contents of the working memory at any instant in time. Viewing the conflict set will be helpful in eliminating some of the totally unexpected behavior in the knowledge sources.

- User defined memory reclamation functions: This is very specific to the internal implementation structures used by GEST. One function that provides the user control on how much of the past record for each KS will be useful especially while running large applications programs in GEST.

9.0 APPENDIX A

APPENDIX A — SPEECH FOR Z-SCHEDULER PRESENTATION

Z-Scheduler attempts to capture varied operator scheduling strategies into a computational model. In this demonstration I will be showing you two such strategies- the minimize time strategy and the balance load strategy. During the application of the minimize time strategy the operator tries to perform the specified tasks in as small a time window as possible by pushing his/her resource capabilities to the maximum extent possible. The balance load strategy is one in which the operator is working towards being at a comfortable resource limit and hence delays the execution of some tasks. In the literature, especially work done by Sandy Hart, these strategies are well accepted.

As you have just heard from the previous presenter (Presenter's name), the input to Z-Scheduler is a set of tasks to be scheduled. These tasks are generated by the planning component of the operator model when the operator tries to predict the future tasks within a specific time horizon and hands over these tasks to Z-Scheduler to come up with an acceptable order to perform the tasks. Shown here is the tasks to be scheduled bunched together in this stack. Each task description contains details on the estimated duration for the task, the priority- the relative importance of this task to the other tasks on the stack, the resource utilization: the resources for Z-scheduler is the Visual, Auditory, Cognitive and Motor as defined in the Task Loading Model which you will be seeing next.

Along with the stack of tasks, Z-Scheduler also gets a stack of constraints- constraints are defined as some restriction placed on the execution of the tasks. Z-Scheduler can handle both qualitative and quantitative constraints. (*point to the temporal constraint categories graph on the poster*); qualitative constraints in the literature are called inter-based constraints and are useful in describing the temporal relation between two tasks. For example., we can specify task A is before task B or any of the 13 relationships defined here. Also, we can directly anchor the task to the metric time by specifying quantitative relations like "Start-at tasks-A 5 seconds". In the AI literature these are termed point-based constraints. Hence, as you can see, there has been lots of work put into providing an expressive set of constraints for the user of this tool.

Now that the inputs of the Z-Scheduler are clear, I will explain the scheduling process. But before that the scheduling problem needs to be analyzed. Depending on the nature of input, Ben Arieh in his PhD dissertation from Purdue University, says that scheduling problems can be classified and for each category of classification there are specific solutions that apply. Following the same framework we have concluded that Z's scheduling problem is dynamic-multistage-routing problem. Simply stated, this problem is in the complex end of the spectrum and hence traditional OR techniques fail to solve the problem and hence we turned to knowledge-based techniques for the answers.

As the majority of works in the past have handled this problem, we have modeled the problem as a constraint satisfaction problem. Hence, whenever a problem is modeled in this manner, the solution space is divided into subspaces and the local solutions are found in each space and these local solutions are threaded along to get the global solution. The way the problem space is divided is crucial to the solution. Among the three alternatives we have chosen the more recent task-based scheduling to solve the CSP. This technique will be clear as I get into the scheduling process itself.

Now moving to the heart of the scheduling process, the blackboard architecture is central to Z-scheduling process. The blackboard here is a central structure that contains all the scheduling state data and is divided into stages of the schedule generation. The knowledge required to perform the schedule is also partitioned and each partition called knowledge sources works on each stage of the scheduling process.

Going into the first knowledge source is the constraint solver knowledge source. The primary focus of this knowledge source is to translate the hybrid representational constraints into a uniform base for the temporal reasoning. Also, because of the nature of inputs to Z-scheduler, the tasks are oblivious of their constraints. So the constraint solver has rules that grab one constraint frame at a time and cache the constraint in the relevant tasks and incrementally build these partial dependency graphs (*point to the graphs in the color monitor*).

Since we are using task-based scheduling, this is a two step process. The first step is to pick a task from the tasks to schedule and once the task is decided all alternative solutions for that task are evaluated and, based on some metric, one of solutions is chosen and the task gets scheduled — that is, some resource is promised to that task. These two stages are reflected in the task-selector and resource-allocator knowledge sources.

The metric in selecting tasks for resource-allocation is called the Task-criticality measure. This measure has few factors, one of them being the slack on that task — the more the slack the less the criticality. The task with least amount of slack becomes the candidate for resource allocation. The metric for selecting a solution for the task selected is to pick the solution that has minimum resource competition, the logic being that the solution must not eat away into the opportunity of the tasks that have not yet been scheduled.

The constraint propagator knowledge source is a record keeping knowledge source. Every time a task is scheduled it updates rest of the task windows accordingly. The truth maintainer knowledge source is a high level monitor of the constraint propagating activities. Whenever a conflict occurs during the process of constraint propagating, this knowledge source halts the execution of the other knowledge source by sending a message via the blackboard. After halting the process that was responsible for the conflict — an example of the conflict is that the window duration of the task becomes smaller than the estimated duration of the task — the Truth maintainer attempts to find the reason for the conflicts occurrence and on finding it out, it resets all the knowledge sources to that point in time before the conflict was instantiated. In other words, the Truth maintainer is built on the principle of chronological backtracking, where the amount backtracking is up to the last decision point.

Annex C

Army-NASA Aircrew/Aircraft Integration Program: Phase IV

A^3I

Man-Machine Integration Design and Analysis System (MIDAS)
Software Detailed Design Document

Task Loading Model

prepared by

Lowell Staveland

Table of Contents

Table of Contents

Table of Contents

MAN-MACHINE INTEGRATION DESIGN & ANALYSIS SYSTEM (MIDAS) SOFTWARE DETAILED DESIGN DOCUMENT PHASE IV:

TASK LOADING MODEL

1.0 INTRODUCTION

There are five main parts to this document: (1) the scope, purpose and objectives of the document; (2) the reference list; (3) the scope, purpose, objectives and details of the model; (4) the current and future detailed software and hardware requirements; and (5) the user's guide.

1.1 IDENTIFICATION OF DOCUMENT

This document is the Software Product Specification for the Task Loading Model (TLM) module of the Man-machine Integration Design and Analysis System (MIDAS).

1.2 SCOPE OF DOCUMENT

This document describes a computational approach to modeling human information processing, and the objective task demands that are imposed on an operator of complex systems.

The approach described in this document pertains to its use as one tool in the MIDAS workstation, and its dependence on and relationships to the other tools that are integrated within the MIDAS workstation.

The description contains details of the psychological theories and experimental results underlying the model, and the computational methods by which the theories and results are incorporated into the model and applied to the MIDAS simulation environment.

This document also provides the details of the software architecture and Lisp code that were used to implement the TLM on the Symbolics computers. The functional and procedural aspects of the TLM human-computer interface will be discussed together with the associated Lisp code such that Lisp programmers will be able to change and modify the code, and crew station design team members will be able to use the TLM.

Descriptions of the detailed processing requirements, structure, I/O, and control are provided for each lower level Task Loading Model Software Component , unit, or function contained within the TLM.

The readers of this document should have knowledge of different human performance modeling techniques or human information processing theories, along with a familiarity with the Symbolics programming environment and object-oriented programming.

1.3 PURPOSE AND OBJECTIVE OF DOCUMENT

1.3.1 Purpose of Document

The purpose of this document is to provide the users of the TLM the background of the MIDAS Task Loading Model (TLM) necessary to understand and configure this model to their domain of choice, and to use TLM to evaluate alternative crewstation configurations during the conceptual phase of crewstation design.

This document provides the software engineering specifications for future TLM development efforts.

1.3.2 Objectives of Document

This document provides a detailed description of the model's approach in order to provide a metric that allows a member of a design team to compare the impact of different design configurations on the information processing capabilities of a pilot or operator of a complex system.

The material in this document is directed toward three categories of readers:

1) Those who wish to learn what the MIDAS TLM accomplishes.

2) Those who wish to use the TLM software to investigate the interactions between an operator and a specific crew station design within the context of a given mission

3) Those who might want to modify and update the TLM.

2.0 RELATED DOCUMENTATION

2.1 APPLICABLE DOCUMENTS

The following documents are referenced herein and are directly applicable to this volume:

Baron, S., Kruser, D. S. & Huey, B. M. (Eds.). (1990) *Quantitative Modeling of Muman Performance in Complex, Dynamic Systems*. Washington, D.C.: National Academy Press.

2.2 INFORMATION DOCUMENTS

The following documents amplify or clarify the information presented in this volume:

Aldrich, T. B., Szabo, S. M., & Bierbaum, C. R. (1989). The development and application of models to predict operator workload during system design. In G. MacMillan, D. Beevis, E. Salas, M. Strub, R. Sutton & L. Van Breda (Eds.), Applications of human performance models to system design. (pp. 65-80). New York: Plenum Press.

Andre, A. D., & Wickens, C. D. (1989). Information processing and perceptual characteristics of display design: The role of emergent features and objects (Report No. ARL-89-8/AHEL-89-4). Urbana-Champaign: University of Illinois, Aviation Research Lab.

Barnard, P., Wilson, M. & Maclean, A. (1988). Approximate modelling of cognitive activity with an expert system: A theory-based strategy for developing an interactive design tool. The Computer Journal, 31(5), 445-456.

Boff,K.R., Kaufman, L. & Thomas, J. P. (Eds.). Handbook of perception and human performance (Vols 1-2). New York: John Wiley and Sons.

Boff,K.R. & Lincoln, J.E. (Eds.). (1988). Engineering data compendium: Human perception and performance. (Vols 1-4). WPAFB, OH: AAMRL/HE/CSERIAC.
Chase, W. G. (1986). Visual information processing. In K.R. Boff, L. Kaufman, & J. P. Thomas (Eds.), Handbook of perception and human performance: Vol II. Cognitive processes and performance. (pp. 28-1 - 28-71). New York: John Wiley and Sons.

Derrick, W. (1988). Dimensions of operator workload. Human Factors, 30, 95-110.

Dunn-Rankin, P. (1983). Scaling methods. Hillsdale, New Jersey: Lawrence Erlbaum Associates.

Elkind, J., Card, S., Hochberg, J. & Huey, B. (Eds.). Human performance models for computer aided engineering. Washington, D. C.: National Academy Press.

Flach, J. M. (1989). The ecology of human-machine systems (Report EPRL-89-12). Urbana: University of Illinois, Engineering Psychology Research Laboratory.

Fleishman, E. A. & Quaintance, M. K. (1984). Taxonomies of human performance: The description of human tasks. Orlando: Academic Press.

Gibson, J. J. (1979). The ecological approach to visual perception. Boston: Houghton Mifflin.

Gopher, D. & Kimchi, R. (1989). Engineering psychology. Annual Review of Psychology, 40, 431-55.

Gopher, D. & Donchin, E. (1986). Workload - An examination of the concept. In K.R. Boff, L. Kaufman, & J. P. Thomas (Eds.), Handbook of perception and human performance: Vol II. Cognitive processes and performance. (pp. 41-1 - 41-49). New York: John Wiley and Sons.

Hart, S. G. (1989). Crew workload-management strategies: A critical factor in system performance. Proceedings of the Fifth International Symposium on Aviation Psychology (pp.). Colombus, Ohio.

Hart, S. G. & Staveland, L. E. (1988). Development of NASA-TLX (Task Load Index): Results of empirical and theoretical research. In P. A. Hancock & N. Meshkati (Eds.), Human mental workload (pp. 139-1830. Amsterdam, The Netherlands: North Holland.

Holley, C. D. (1989). A model for performing system performance analysis in predesign. In G. MacMillan, D. Beevis, E. Salas, M. Strub, R. Sutton & L. Van Breda (Eds.), Applications of human performance models to system design. (pp. 91-102). New York: Plenum Press.

Hulme, A. J. & Hamilton, W. I. (1989). Human engineering models: A user's perspective. In G. MacMillan, D. Beevis, E. Salas, M. Strub, R. Sutton & L. Van Breda (Eds.), Applications of human performance models to system design. (pp. 487-500). New York: Plenum Press.

Kantowitz, B. H. & Roediger, H. L. (1980). Memory and information processing. In G. N. Bower & E. R. Hilgard (Eds), Theories of learning.Englewood Cliffs, NJ: Prentice-Hall

Lachman, R., Lachman, J. & Butterfield, E. C. (1979). Cognitive psychology and information processing: An introduction. (pp. 89-127). Hillsdale, NJ: Erlbaum.

Linton, P. M., Plamondon, B. D., Dick, A. O., Bittner, A. C. & Christ, R. E. (1989). Operator workload for military system acquisition. In G. MacMillan, D. Beevis, E. Salas, M. Strub, R. Sutton & L. Van Breda (Eds.), Applications of human performance models to system design. (pp. 21-46). New York: Plenum Press.

McCracken, J. H. & Aldrich, T. B. (1984). Analyses of selected LHX mission functions: Implications for operator workload and system automation goals (Technical Note ASI479-024-84). Fort Rucker, AL: Army Research Institute Aviation Research and Development Activity.

Miller, R. A. & Jagacinski, R. J. (1989). The organization of perception and action in complex control skills (Final Report Grant No. NAG 2-195). Moffett Field, CA: Ames Research Center, NASA (RF Project 763264/714826).

North & Riley (1989). W/INDEX: A predictive model of operator workload. In G. MacMillan, D. Beevis, E. Salas, M. Strub, R. Sutton & L. Van Breda (Eds.), Applications of human performance models to system design. (pp. 81-90). New York: Plenum Press.

O'Donnell, R. D. & Eggemeier, F. T (1986). Workload assessment methodology. In K.R. Boff, L. Kaufman, & J. P. Thomas (Eds.), Handbook of perception and human performance: Vol II. Cognitive processes and performance. (pp. 42-1 42-49). New York: John Wiley and Sons.

Pachella, R. G. (1974). The interpretation of reaction time in information processing research. In B. Kantowitz (Ed.), Human information processing: Tutorials in performance and recognition. Potomac, Md.: Erlbaum.

Polsen, M. C., Wickens, C. D., Klapp, S. T. & Colle, H. A. (1989). Human interactive informational processes. In P. A. Hancock & M. H. Chignell (Eds.). Intelligence interfaces: Theory, research and design. (pp. 129-164). Amsterdam: Elsevier Science Publishers B.V.

Posner, M. I. (1986). Overview. In K.R. Boff, L. Kaufman, & J. P. Thomas (Eds.), Handbook of perception and human performance: Vol II. Cognitive processes and performance. (pp. V-3 - V-10). New York: John Wiley and Sons.

Rasmussen, J. (1983). Skills, rules, and knowledge; Signals, signs, and symbols, and other distinctions in human performance models. IEEE Transactions on Systems, Man, and Cybernetics, 13(3), 257-266.

Roth, E. M. & Woods, D. D. (1988). Cognitive task analysis: An approach to knowledge acquisition for intelligent system design. In G. Guida & C. Tasso (Eds.), Topics in expert systems design. (pp.). Amsterdam: North Holland.

Sanders, A. F. (1989). Human performance models and system design In G. MacMillan, D. Beevis, E. Salas, M. Strub, R. Sutton & L. Van Breda (Eds.), Applications of human performance models to system design. (pp. 475-486). New York: Plenum Press.

Schneider, W. & Fisk, A. D. (1982). Attention theory and mechanisms for skilled performances (Rep. HARL-ONR-8201). Champaign: University of Illinois, Human Attention Research Laboratory.

Staveland, L.E. (1988). Combinatorial rules for generating workload ratings. Unpublished master's thesis, San Jose State University, San Jose, CA.

Treisman, Anne. (1986). Properties, parts and objects. In K.R. Boff, L. Kaufman, & J. P. Thomas (Eds.), Handbook of perception and human performance: Vol II. Cognitive processes and performance. (pp. 35-1 - 35-70). New York: John Wiley and Sons.

Vicente, K. J. & Rasmussen J. (1990). The ecology of human-machine systems II: Mediating "direct perception" in complex work domains (Report EPRL-90-01). Urbana: University of Illinois, Engineering Psychology Research Laboratory.

Wickens, C. D. (1984). Processing resources in attention. In R. Parasuraman & D. R. Davies (Eds.), Varieties of attention (pp. 63-102). Orlando, FL: Academic Press.

Wickens, C. D. (1987). Attention in Aviation. Proceedings of the 4th Conference on Aviation Psychology. Columbus: Ohio State University.

Wickens, C. D. (1989a). Models of multitask situations. In G. MacMillan, D. Beevis, E. Salas, M. Strub, R. Sutton & L. Van Breda (Eds.), Applications of human performance models to system design. (pp. 259-274). New York: Plenum Press.

Wickens, C. D. (1989b). Resource management and time sharing. In J. Elkind, S. Card, J. Hochberg & B. Huey (Eds.), Human performance models for computer aided engineering. (pp. 180-202). Washington, D. C.: National Academy Press.

Wickens, C. D. & Andre, A. D. (1989). PAWES multiple resource analysis. (Report No. AF Dayton RI-59630X). Urbana-Champaign: University of Illinois, Aviation Research Lab.

Wickens, C. D. & Flach, J. M. (1988). Information Processing. In E. Weiner (Ed.), Human factors in aviation. (pp. 111-155).

Woods, D. D. & Hollnagel, E. (1987). Mapping cognitive demands in complex problem-solving worlds. International Journal of Man-Machine Studies, 26, 257-275.

Woods, D. D. & Roth, E. M. (1988) Cognitive systems engineering. In M. Helander (Ed.), Handbook of human-computer interaction. (pp. 3-43). Amsterdam: Elsevier Science.

3.0 CONCEPT

This section provides the purpose, scope, objectives and details of the TLM.

3.1 DEFINITION OF TLM

MIDAS loading model is an output, normative, bottom-up, multi-task human performance model. It is an output model because it generates the loading values after being fed a task description. It is normative because it assumes that the aircrews or system operators are highly skilled and motivated and would perform in a manner that is rational and consistent with the information available, and with the constraints, risks and objectives that exist. It is bottom-up because it generates the values based on rank orderings of the interactions and combinations of basic perceptual, cognitive, and motor activities. In this sense, it also has some process and prescription characteristics, because the basic activities can be diagnostic of the problems in the conceptual designs (indicated by high loading values). It is a multi-task model because it evaluates the loading of a variety of tasks performed concurrently as well as performed serially.

Throughout the presentation of the concept, the term "user" refers both to humans (e.g., systems designers, engineers, psychologists) and to other MIDAS interfacing information systems and components (e.g., the scheduling module and task decomposition module).

3.1.1 Purpose and Scope

3.1.1.1 Purpose of TLM

The MIDAS loading model is a computational model of human performance whose purpose is to aid engineers in the conceptual design of aircraft crewstations. The model provides the users with a tool to evaluate and predict the characteristics and amount of load imposed on an operator while performing mission tasks.

3.1.1.2 Scope of TLM

The MIDAS-TLM expands the previously used MIDAS loading model to include an evaluation of the loads that are placed on pilots interacting with a given design within the context of a series of flight activities or tasks that are generated during a simulated flight.

The model uses a definition of task loading as the aircrew's or operators' capabilities to perceive and process the information imposed on their perceptual, cognitive and motor systems by task demands. This approach is based on an information processing analysis which holds that human performance can be objectively and quantitatively described with information processing structures in conjunction with the mental processes that act on those structures (Wickens & Flach, 1988; Lachman, Lachman & Butterfield, 1979; Kantowitz & Roediger, 1980; Posner, 1986: Chase, 1986).

This definition and the embodied approach should enable the model to be applied to any conceptual design configuration in the aerospace domain. The model maps the tasks that are spawned by the design under consideration to human performance measures which may be used for evaluation.

3.1.2 Goals and Objectives

3.1.2.1 Goals of TLM

The goal of the MIDAS-TLM is to predict the loads on each of four psychological dimensions for each task and task combination associated with a design, thus allowing the designer to evaluate the loads a conceptual design imposes on a system operator.

3.1.2.2 Objectives of TLM

The objective of the TLM is to model the information processing capabilities of aircrews such that systems designers can use it in conjunction with other MIDAS workstation software components. These other modules are used to render a conceptual crewstation design, and analyze the design to elicit the tasks that need to be performed. These tasks are used as input to the TLM, which evaluates and predicts the characteristics and amount of perceptual, cognitive and motor load that the tasks might impose on an average aircrew member.

The MIDAS-TLM generates these values of load by classifying tasks according to a taxonomy of perceptual, cognitive, and motor attributes. This is accomplished by mapping information processes and structures underlying human performance to the taxonomy of perceptual, cognitive and motor attributes underlying task performance. Memory, attention, and time demands and conflicts between attributes in the taxonomy are compared in matrices and ranked. The ranks are combined according to a model of workload that generates load estimates on visual, auditory, cognitive, and motor dimensions.

3.1.3 Description of TLM

The description of the TLM provides the experimental and theoretical details underlying the development TLM. The description also provide the details of the operation and structure of the model.

3.1.3.1 Background

The model of loading previously used in the MIDAS Workstation was developed by McCracken & Aldrich (1984). It is based on Wickens' (1984) theory of multiple resources that partitions loading into Visual, Auditory, Cognitive, and Psychomotor (VACP) dimensions and uses them as component scales to measure pilot workload, which is defined as "the attentional demands imposed on a pilot during flight" (Aldrich, Szabo & Bierbaum, 1987).

This model does not meet the needs of the MIDAS Workstation because the loading values are assigned by the design team through a task analysis either before or after a simulation. This reduces the model's sensitivity to the contexts that interact with task performance. It is important to incorporate contextual effects because task performance depends on the environment in which it is performed, and the presence of other tasks, which can have a dramatic effect on the loads imposed on aircrews. Without accounting for context, predicted loads may reflect arbitrary differences between design alternatives.

The MIDAS-TLM will enhance the current loading model by rank-ordering task loadings during a simulation, providing sensitivity to the context within which those tasks are embedded, and by anchoring tasks to a timeline that matches a task to its load value. These enhancements will provide some contextual basis for generating predictive estimates of Visual, Auditory, Cognitive, and Motor (VACM) load based on certain pilot, world, and environmental attributes that are generated during a simulation.

3.1.3.2 Definition of Loading

The definition of loading used in the model is the pilots' capabilities to perceive and process the information imposed on their perceptual, cognitive, and motor systems by task demands.

3.1.3.3 Approach to development

The developmental approach is based on an information processing analysis which holds that human performance can be objectively and quantitatively described as the mental structures and mental processes that act on those structures (Wickens & Flach, 1988; Lachman et al, 1979; Kantowitz & Roediger, 1980; Posner, 1986: Chase, 1986). Applying this analysis to the aerospace design environment yields the basic assumptions underlying the MIDAS-TLM:

- An abstraction hierarchy can represent a set of invariant qualities (Vicente & Rasmussen, 1990; Flach, 1989) that are common to the information processing system's capability to represent (structures) and manipulate (processes) information symbolically, the capability which Lachman et al, (1979) consider to be the essence of the information processing paradigm.

- An abstraction hierarchy of these structures and processes can be mapped to a wide range of pilot task interactions with cognitive task analysis (Woods & Hollnagel, 1987; Woods & Roth, 1988; Roth & Woods, 1988; Barnard, Wilson & Maclean, 1988).

- The structures and processes that map to a task can be compared in a conflict matrix to rank the use and conflicts (North & Riley, 1989; Holley, 1989; Wickens & Andre, 1989) among their memory, temporal, and attentional resources.

- Algorithms can be derived to calculate loading values from the ranking procedure (Dunn-Rankin, 1984; North & Riley, 1989, Wickens & Andre, 1989).

3.1.3.4 Conceptual Structure of the MIDAS-TLM

Developing the MIDAS-TLM requires two frameworks that describe the objective aspects of loading imposed by task demands: a taxonomy (Fleishman & Quaintance, 1984; Andre & Wickens, 1989, Flach, 1989) and a conflict matrix (Linton, Plamondon, Dick, Bittner & Christ, 1989; Sanders, 1989; Hulme & Hamilton, 1989; Wickens, 1988). These two frameworks were selected as a means to describe certain attributes and interactions of one or more tasks and are necessary to provide context sensitivity and to assign values to task interactions. These frameworks incorporate the notions of invariant and variant properties that are borrowed from Gibson's ecological approach to perception (1979).

The first framework abstracts the informational structures and processes into a hierarchical taxonomy: structures are the symbolic representations of information present in the perceptual, cognitive and motor systems that are connected by a set of relations; processes are the perceptual, cognitive or motor activities that have structures as inputs or outputs (Lachman et al, 1979).

This first framework, a taxonomy of dimensions and elements (Table 1), represent different structures and processes in information processing, and incorporate the notion of invariant properties. These dimensions and elements are abstracted from the invariant properties of underlying structures and processes (Table 2).

The second framework incorporates the notion of variant properties to categorize the interactions of the structures and processes. These properties vary with respect to the structures and processes and their interactions.

This framework, a matrix of conflict values (Figure 1), compares and contrasts the variant properties — memory, attentional, and temporal resources — underlying the interactions of the elements in each dimension used in the MIDAS-TLM. The values in Figure 1 represent the degree of conflict between the memory, attentional, and temporal resources on a relative scale of 0 to 4.

3.1.3.5 Dimensions of the MIDAS-TLM

The top level of the taxonomy represents the dimensions that are abstracted from the invariant properties relating the paired elements (Table 2). These dimensions and their properties result from research on mental workload that suggests that loading is multidimensional in nature (Hart & Staveland, 1988; Gopher & Donchin, 1986). Loading has been measured in many different ways by many different researchers (see O'Donnell &

Eggemeier, 1986 for a thorough review). Most researchers now conclude that loading is not a uni-dimensional construct and can not be measured by only one technique, but must be measured by several techniques used in conjunction and combined at some theoretical level to derive a loading value (Derrick, 1988).

Dimensions used in previous models. The set of dimensions used in the TLM was developed by McCracken & Aldrich (1984) as a means to evaluate the U.S. Army's AH-64 (Apache) helicopter flight tasks. It is based on Wickens' theory (1984) of multiple resources and partitions loading into dimensions of separable resources used during information processing. Wickens treats resources as attentional demands — that is, the capacity of the system to attend to processing demands.

The McCracken & Aldrich workload assessment technique partitioned workload into the Visual, Auditory, Cognitive and Psychomotor dimensions (VACP's) suggested by Wickens' theory (1984). They used the VACP's as component scales to measure pilot workload, which they defined as the attentional demands imposed on a pilot during flight (Aldrich, Szabo & Bierbaum, 1987).

The MIDAS-TLM incorporates essentially the same dimensions that McCracken & Aldrich (1984) use with psychomotor (P) changed to motor (M) , but it treats resources as the memory, time, and attentional demands (see framework #2 - Figure 1).

<u>Visual</u>	<u>Auditory</u>	<u>Cognitive</u>	<u>Motor</u>
near visual far visual	orient discriminate	direct transformation	verbal spatial
scan fixate	signal speech	single choice multiple choice	near far
integral separable	salient masked	verbal spatial	discrete continuous
objects features		planned unplanned	gross fine
salient masked			mouth head eye hand feet finger
static dynamic			right left both

Table 1. Taxonomy of Dimensions and Paired Elements Classifying Pilot-task Interactions
(Dimensions are underlined)

Dimension	Paired Elements	Property	Structure /Process	Stage
Visual	Near Visual / Far Visual	Spatial Area	Structure	Perceptual input
	Scan / Fixate	Spatial Search	Process	
	Integral / Separable	Spatial Proximity	Structure	
	Objects / Features	Spatial Grouping	Structure	
	Salient / Masked	Spatial Discriminability	Structure	
	Static / Dynamic	Motion Cues	Structure	
Auditory	Orient / Discriminate	Auditory Localization	Process	Perceptual Input
	Signal / Speech	Auditory codes	Structure	
	Salient / Masked	Auditory Discriminability	Structure	
Cognitive	Direct / Transformation	Automatic / Controlled Processing	Process	Cognitive Processing
	Single Choice / Multiple Choice	Levels of Processing	Process	
	Verbal / Spatial	Processing Codes	Structure	
	Planned / Unplanned	Priming	Process	
Motor	Verbal / Spatial	Response Codes	Structure	Motor Output
	Near / Far	Response Proximity	Structure	
	Discrete / Continuous	Control Dynamics	Process	
	Gross / Fine	Control Dynamics	Process	

Table 2. The Properties, Structures and Processes, and Stages corresponding to the dimensions and paired elements of loading used in the MIDAS-TLM

3.1.3.6 Invariant properties of the dimensions

To incorporate the multidimensional nature of loading into the MIDAS-TLM, the paired elements are grouped into dimensions according to their invariant properties. The invariant properties represent stages of processing, which Kantowitz & Roediger (1980) define as roughly corresponding to the transformations of information.

There are essentially three stages — input, processing, and output — to which each of the four dimensions (Visual, Auditory, Cognitive, Motor) correspond. The visual and auditory dimensions correspond to the input stage because they are input modalities. The cognitive dimension corresponds to the processing stage because cognition involves processing information to various depths. The motor dimension corresponds to the output stage because outputs are generated with various motor effectors.

3.1.3.7 Definition of Resources used in the MIDAS-TLM

In the MIDAS-TLM, the resources are the amount of memory, the amount of attentional demands, or the amount of time each structure or process requires to process information. All three types of resources are considered to be variant or perspective structures in the Gibsonian sense (Gibson, 1979) because they vary as functions of the change in the information extracted from the perceptual array. Consequently, these resources represent the variant properties of the structures, processes, and stages comprising the elements and dimensions in the taxonomy.

3.1.3.8 Rank Order of Resource Use

Comparing and contrasting memory, attention, and time assumes that the resource requirements of each element can be explicitly rank-ordered in relation to the implicit resource requirements of the other elements, an assumption based on research on the resource requirements of the structures and processes underlying the elements.

3.1.3.8.1 Ranking procedure

Each element is entered into a conflict matrix and compared with itself and with all others. Matrices are constructed for all the elements within a dimension (within-matrix) and for all elements between dimensions (between-matrix). This results in four within-matrices and four between-matrices.

The ranking process is simplified by separately comparing and contrasting the paired elements, which are explicitly rank-ordered relative to the implicit resource requirements of each element in the pair and relative to each element in the other pairs. This means that for one pair of elements compared with itself or with another pair, there are four interactions that are rank-ordered.

3.1.3.8.2 Ranking scale

The interactions among elements are ranked from 0 to 4: 0 means few resources are required or their interactions are meaningless (e.g. near speech/far speech) and 4 means the greatest amount of resources are required. The rankings are strictly ordinal. Some interactions were treated as inviolable, and assigned a rank of 9 as a flag that the interaction shouldn't be used to calculate loading values. For example, focusing at an object in the near visual field and at an object in the far visual field at the same time is considered to be inviolable.

3.1.3.9 Generating Conflict Matrix Values

To construct a within-matrix for the visual dimension, the six pairs of elements in the visual dimension are entered into both the rows and the columns of the matrix (Figure 1). The point of intersection of each row and column defines an interaction of two elements: an element in a row with an element in a column. The interactions of the six pairs are ranked two pairs at a time until each of the pairs have been compared with each of the other pairs. Comparisons are achieved by rank ordering the four interactions generated by the four possible combinations of the four elements within the two pairs.

VISUAL DIMENSION	NE	FA	SC	FI	SE	IN	OB	FE	SA	MA	ST	DY
NEAR	1											
FAR	999	2										
SCAN	1	2	5									
FIXATE	2	3	999	999								
SEPARABLE	1	3	3	2	2							
INTEGRAL	2	4	2	3	3	4						
OBJECTS	1	3	2	1	3	3	2					
FEATURES	2	4	1	2	4	2	3	4				
SALIENT	1	2	2	1	1	1	1	1	1			
MASKED	2	3	3	2	2	2	2	2	3	5		
STATIC	1	1	1	1	1	1	2	1	1	2	1	
DYNAMIC	3	2	2	3	2	3	1	2	1	1	2	3

Figure 1. Conflict Matrix for the Visual Dimension
(NE=near, FA=far, SC=scan, FI=fixate, SE=separable, IN=integral, OB=object, FE=features, SA=salient, MA=masked, ST=static, DY=dynamic).

3.1.3.9.1 Generating within-matrix values

The procedure for generating the values (ranks) for the within-matrix of the visual dimension (Figure 1) requires ranking the 78 possible combinations of the six pairs, taking two pairs at a time, which are entered in a triangular matrix that is symmetric about the main diagonal.

For example, the first pair of elements (near visual/far visual) is entered into the matrix, generating four possible interactions that are rank ordered: near visual/near visual, near visual/far visual, far visual/near visual, far visual/far visual. However, since the near visual/far visual and far visual/near visual interactions are redundant, only three interactions need to be ranked. In the within-matrix, each pair of elements involves one redundant interaction so that only three rankings are required for each pair.

This Ranking procedure is repeated to compare each pair of elements with itself and with the other pairs in the same dimension in order to construct the within-matrices.

3.1.3.9.1.1 Example 1: Two tasks with two near visual requirements

Two tasks with two near visual requirements (the intersection of near visual and near visual elements) requires fewer resources than two tasks with two far visual requirements. The time demands are less because it probably takes less time to conduct a visual search near than far (less search area). Less memory is needed because with less time there are probably fewer items that need to be stored in short term memory or encoded into long term memory. The attentional demands may be less because they are needed for less time, even though the same amount may be required for a given unit of time.

3.1.3.9.1.2 Example 2: Two tasks with a near visual and a far visual requirement

Two tasks with a near visual and a far visual requirement will probably require the most resources because it is hard to divide attention between the near and far visual fields. This will probably increase the visual search time because a strategy of switching attention back and forth between the near and far fields needs to be employed, and this requires more time than focusing on only one or the other field. This strategy probably has more memory requirements because information from each field has to be stored and recalled with each switch in attention.

3.1.3.9.2 Generating between-matrix values

In order to construct the between-matrices the procedure for generating within-matrix values is repeated to compare each pair of elements in one dimension with each pair of elements in the other dimensions. This repetition of procedures results in eight conflict matrices (four within and four between) constructed in a similar fashion to the matrix in Figure 1 (see Appendix A, Figures A-1 through A-7).

There are only four between-matrices instead of the six that would result from all combinations of the four dimensions, because information is assumed to be processed serially from the perceptual input stage through cognitive processing to motor output stages (Wickens &Flach, 1988). Therefore, the auditory and visual modalities interact within the input stage (1 matrix), both interact with the cognitive processing stage (2 matrices), and the cognitive processing stage interacts with the motor stage (1 matrix). Rank ordering all possible interactions within the eight matrices results in around 500 conflict values that can be used to generate the VACM loading values.

The rankings are based on the results of experiments that tested the structures and processes underlying the interactions among the elements. Each interaction was assigned a rank that matched experimental results testing either that interaction or a similar interaction. In the cases where interactions could not be matched to specific experiments, rankings were extrapolated from the most appropriate research. Boff, Kaufman & Thomas (1986) and Boff & Lincoln (1988) were used extensively to obtain the experimental results used to generate the rankings.

3.1.3.10 Classifying the Pilot-Task Interaction

In order to classify the pilot-task interaction with the taxonomy, the relevant attributes of each task are mapped into the taxonomy using cognitive task analysis techniques.

3.1.3.10.1 Cognitive Task Analysis (CTA)

Cognitive task analysis provides the means to map a detailed representation of the pilot-task domain because its main goal is to reveal the interactions among the current and desired states of the world (tasks), the state of the agent (pilot), and the representations of the world available to the agent (Woods & Roth, 1988) which are explicated and therefore applicable to modeling.

The cognitive analysis results in "...an umbrella structure of domain semantics that organizes and makes explicit what particular pieces of knowledge mean about problem solving in that domain" (Roth & Woods, 1988 pg.44-45). This umbrella structure provides a framework with which to capture the cognitive demands that are transportable across reasoning situations, yet relevant to the domain because they are represented in the domain semantics (Woods & Hollnagel, 1987; Woods & Roth, 1988; Roth & Woods, 1988).

3.1.3.10.2 MIDAS-TLM as a CTA Structure

The elements and dimensions of the taxonomy used in the MIDAS-TLM is essentially a structure of domain semantics that organizes the perceptual, cognitive, and motor demands in the pilot-task interaction. Since the domain semantics explicitly define the representations of the structures and processes of the tasks that are available to the pilot, the MIDAS-TLM taxonomy can guide the cognitive task analysis.

3.1.3.10.3 Mapping a Task to the Taxonomy

Task attributes are mapped to the elements and dimensions in the taxonomy through a binary (present or absent) classification scheme. The result of constructing the principles in this fashion is a taxonomy of dichotomous principles that represent the range of the structures and processes that a pilot might possibly require to perform a task. Table 2 describes the properties, structures and processes, and stages that correspond to each pair of elements and their respective dimensions.

The taxonomy contains seventeen pairs of elements, plus two other sets of elements in the motor dimension that represent the anatomical constraints (mouth, head, eye, hand, feet, fingers) and their combinations (right, left, both).

3.1.3.10.4 Taxonomic Subsets Used to Classify Pilot-Task Interaction

Not every element of the taxonomy will be used to classify a task and to characterize an interaction because the taxonomy represents a range of the structures and processes that a pilot might use, and different tasks require different structures and processes. Therefore, only a subset of the elements in the taxonomy will be used to characterize a pilot-task interaction, and that subset, one of the many possible subsets that can characterize the interaction, will be selected according to how the attributes that define the task, world, and equipment states are mapped to the taxonomy.

The subset characterizing the pilot task interaction results from selecting one element from a pair that best characterizes the pilot-task interaction being mapped. Examples of some of the possible criteria for selecting a task are listed in Table 3. Only one element can be chosen since they are structured as two distinct but related properties. Therefore, each one is either present or absent and, while both cannot be present at the same time, both can be absent. Consequently, the subset can contain fewer than twenty elements, but not more. This avoids forcing a task to be classified with elements that are irrelevant or inappropriate.

Dimension	Elements	Selection Criteria
Auditory	Orient / Discriminate	Subsystem: Comm; Sensory: interpret words, tones
	Signal / Speech	Subsystem: Comm; Object: messages, alarms
	Salient / Masked	World model state variables: jamming, high ambient noise
Visual	Near Visual / Far Visual	If subsystem = internal visual field then near else far
	Scan / Fixate	If sensory=(vb-list:orient) then scan; else if (vl:read)
	Integral / Separable	CDE: object defs.; World model: external object defs.
	Objects / Features	CDE: object defs.; World model: external object defs.
	Salient / Masked	World model state variables: night/day, rain/clear
	Static / Dynamic	Aero model state variables: stationary, in transit
Cognitive	Direct / Transformation	If no change in inst., controls, then direct else trans.
	Single Choice / Multiple Choice	If element= (list:direct,fixate-sep) then s-choice else if element=(list:trans, scan-sep) then m-choice
	Verbal / Spatial	If speech, read (digits,words) then verbal else spatial
	Planned / Unplanned	Scheduler, (Planner): interrupted task schedule
Motor	Verbal / Spatial	If subsystem = com then verbal else spatial
	Near / Far	If jack-reach < ?-distance then near else far
	Discrete / Continuous	If control = (object list) then discrete else continuous
	Gross / Fine	If control = (type list) then gross else fine

Table 3. Definitions of Paired Elements in terms of some possible criteria (examples of state variables from other A^{3}I models and the AH-64 task analysis)

3.1.3.11 Calculating the Loading Values

The subset of elements characterizing the interaction between pilot and task determines which ranks will be combined to calculate the loading values according to an algorithm derived from one developed by North & Riley (1989) and by Wickens & Andre (1989). The algorithm calculates a loading value for each dimension: L_V, L_A, L_C, and L_M. The ranks of the paired elements in the conflict matrices are used to calculate these values.

The subset of elements that are selected via the mapping process to characterize a task are the elements in the matrices whose intersections define their ranks. The ranks for the appropriate subset of elements are extracted from each conflict matrix and used to calculate the loading values for the within-dimension-matrix (C_W) and between-dimension-matrix interactions (C_B). After the loading values from the between- and within-dimension-matrices have been calculated, they are combined to calculate the values for each visual, auditory, cognitive, and motor dimension.

When there is only one task, s=1, the equation for the within-dimension-matrix values, C_W, is given by

$$C_W = \sum_{t=1}^{s} \sum_{i=1}^{n-2} \sum_{k=i+1}^{n-1} \sum_{j=k+1}^{n} \left[(a_{tki}a_{tji}) + (a_{tji}a_{tjk})\right]$$

a_{tki}, a_{tji}, a_{tjk} = individual conflict rankings
i, j, k = indices to the columns and rows in the matrices
s = number of concurrent tasks
n = number rows and columns
t = task index

When there is more than one task performed concurrently, s>1, the equation for the within-dimension-matrix values, C_W, is given by

$$C_W = \sum_{t=1}^{s} \left[\sum_{i=1}^{n-1} \sum_{k=1}^{n-1} \sum_{j=k+1}^{n} (a_{tki}a_{tji}) + \sum_{i=1}^{n-1} \sum_{k=i+1}^{n} \sum_{j=i+1}^{n} (a_{tji}a_{tjk}) \right]$$

a_{tki}, a_{tji}, a_{tjk} = individual conflict rankings
i, j, k = indices to the columns and rows in the matrices
s = number of concurrent tasks
n = number rows and columns
t = task index

The between-matrix value is the same for either s=1 or s>1, the equation for the within-dimension-matrix values, C_B, is given by

$$C_B = \sum_{t=1}^{s} \left[\sum_{k=1}^{n} \sum_{i=1}^{l-1} \sum_{j=i+1}^{l} (a_{tki} a_{tkj}) + \sum_{i=1}^{l} \sum_{k=1}^{n-1} \sum_{m=k+1}^{n} (a_{tki} a_{tmi}) \right]$$

a_{tki}, a_{tmi}, a_{tkj} = individual conflict rankings
i, j, k, m = indices to the columns and rows in the matrices
s = number of concurrent tasks
n = number of rows
l = number of columns
t = task index

The load value for the Visual dimension is given by

$$L_V = V_{cw} + [(VA_{bw} + VC_{bw}) / 2]$$

V_{cw} = loading value for the visual within-matrix
VA_{bw} = loading values for the visual-auditory between-matrix
VC_{bw} = loading values for the visual-cognitive between-matrix

The load value for the Auditory dimension is given by

$$L_A = A_{cw} + [(VA_{bw} + AC_{bw}) / 2]$$

A_{cw} = loading value for the auditory within-matrix
VA_{bw} = loading values for the visual-auditory between-matrix
AC_{bw} = loading values for the auditory-cognitive between-matrix

The load value for the Cognitive dimension is given by

$$L_C = C_{cw} + [(AC_{bw} + VC_{bw} + CM_{bw}) / 3]$$

C_{cw} = loading value for the cognitive within-matrix
AC_{bw} = loading values for the auditory-auditory between-matrix
VC_{bw} = loading values for the visual-cognitive between-matrix
CM_{bw} = loading values for the cognitive-motor between-matrix

The load value for the Motor dimension is given by

$$L_M = M_{cw} + CM_{bw}$$

M_{cw} = loading value for the motor within-matrix
CM_{bw} = loading values for the cognitive-motor between-matrix

The appropriate ranks within each matrix are cross-multiplied in pairwise combinations to penalize task(s) to the extent that the interacting elements have high mutual conflicts (Wickens & Andre, 1989) — cross-multiplying high ranks adds more to the resulting load values than cross-multiplying low ranks. The values from the pairwise cross-multiplications are summed within each matrix. When there are single tasks (s = 1), each conflict matrix is a 2-dimensional array of ranks that are cross-multiplied across rows and columns to produce eight matrix values — four within-matrix and four between matrix values — to which an averaging and summing algorithm is applied.

When there are combined tasks (s = the number of tasks), each conflict matrix becomes a 3-dimensional array, with the matrix of ranks for each concurrent task represented as a level of the third dimension. Matrix values for a set of tasks are calculated by cross-multiplying, in a pairwise procedure meeting serial constraints, the ranks in the rows and columns in one level (representing one task), with the ranks in the rows and columns in the other levels (representing the other tasks). This procedure also produces eight values to which an averaging and summing algorithm is applied.

This averaging and summing algorithm is based on workload research that suggests averaging models under-estimate loads and additive models over-estimate loads, while the "best-fitting" model incorporates an averaging model, plus an "additional cost" (Staveland, 1980). Consequently, the matrix values are averaged across the between-matrices meeting serial constraints to which the appropriate within-matrix value is added.

The serial constraints used to compute the individual matrix values and the averages of the between matrices are determined by the dimensions that interact during serial processing of the information. The visual and auditory dimensions (the input modalities) interact with central processing dimension (cognitive dimension) since they feed in the information, and they interact with each other since they operate concurrently. The central processing dimension interacts with motor performance (the motor dimension) since it determines and monitors the course of action.

The final load value for each dimension is computed by adding the value of the within-matrix calculation for a dimension to its corresponding averaged value. Adding the value for the within-matrix to the average of the between-matrix values represents the "additional cost" in the "best-fitting" model.

For example, the averaged load value for the auditory dimension is calculated by averaging the auditory-visual and auditory-cognitive between-matrix values. Serial processing constrains the auditory dimension to interactions with the cognitive and visual dimensions but not motor dimensions. The final load value for the auditory dimension is computed by adding the value of the within-matrix calculation for the auditory dimension to the initial load value.

The resultant load values increase exponentially with each additional task because of the number of multiplications associated with the number of interactions that are allowed. Consequently, the load values are re-scaled using a log transform, then multiplied by ten to generate a scale ranging from 0 to 100. Using this approach, this model should be able to calculate the loading values for any single task, as well as any number of interacting tasks.

3.2 USER DEFINITION

A user is defined as either a member of the design team, or as a MIDAS component that interfaces with the MIDAS TLM.

Members of the design team can range from high level managers to low level technical staff including draftspersons, psychologists, engineers, programmers and human factors analysts. Any member of the design team may be required to utilize the tools of the

MIDAS workstation including the TLM. All members should be able to understand the output of the TLM sufficiently to judge the quality and efficacy of a conceptual design. The TLM structures the output to achieve this goal, yet also provides sufficient information to allow for a more detailed analysis of a conceptual design by specialists in the psychology and human factoring of design. In order to understand the output in full, a member of the design team will need to have a background in psychology or human performance modeling, especially in information processing theory, and human perception and performance.

There are only two MIDAS interfacing systems and components that use the TLM: the Scheduler and the task representation methods within the Symbolic Operator Model. In the future, a planning model would also use the TLM. The scheduler calls the TLM during run-time of a simulation to obtain the load values necessary to implement a load balancing strategy. The task representation methods currently call the TLM before a simulation to obtain the load values necessary to sequence the tasks that need to be simulated. In order to call the TLM, the scheduling component is only be required to make one function call. However, the task representation methods have to pass the tasks to be sequenced to obtain their single task load values, or pass the sequenced tasks to obtain their concurrent values.

3.3 CAPABILITIES AND CHARACTERISTICS

This section describes the major operational capabilities to be provided by the component, and identifies which users' needs are supported by each capability.

3.3.1 Architecture

There are four parts to the TLM component: the task editor, the task adjustor, the load calculator, and the user interface. The task editor classifies the tasks and the adjustor dynamically changes the classifications to accommodate changes in the context of the simulation. The load calculator generates the load values based on the classifications, and the user interface displays the load values, tasks and task classifications.

3.3.2 Process Capabilities

The TLM can continually process tasks throughout the simulation, and can process from one to n number of tasks simultaneously to simulate concurrent task performance. Realistically, the TLM would only be required to process two to four tasks simultaneously because processing any more would be simulating unrealistic human performance capabilities.

3.3.3 Performance

The performance of the TLM is unknown to date because it hasn't been tested. Testing and validation will occur in the next phase of development.

3.3.4 Interfaces

The design team members interact with the TLM via mouse sensitive items on the monitor of a Symbolics 3640 machine. By selecting certain objects on the screen, designer can select either single- or multi-task loading values of the aircrew members. The loading values of the tasks and task combinations are displayed along with the taxonomic subset classifying the task (s), the name(s) of the task(s), and the time corresponding to when the task performance was simulated. The designer can scroll the display horizontally and vertically to display the complete timeline with the complete task classification and loading profiles.

3.3.5 Error recovery capabilities

As of yet the TLM does not incorporate models of human error, and does not have error recovery mechanisms for simulation crashes or interruptions.

3.3.6 Reliability

The reliability of the TLMSCI is not known because it hasn't been tested or validated.

3.3.7 Maintainability

Currently the TLMSCI can be maintained only by an experienced Lisp programmer because the interface code is limited, and the error recovery code is limited. In other words, the TLM is still not very user friendly, and any maintenance and changes require delving into the code and programming.

3.3.8 Flexibility and Expansion

The TLM is currently not very flexible from a hardware point of view. The software is in Lisp and runs only on Lisp machines, although it is portable between Lisp machines. Whether or not the TLM will be expanded to be portable between different computers is undecided, but this may be required to port the TLM, within the MIDAS environment, to industrial and commercial applications. From a designers point of view the TLM is more flexible, because it can be applied to a wide range of design environments.

3.3.9 Transportability

The TLMSCI is not a stand-alone tool. It was designed to be used within the MIDAS modeling and design environment. As such, it is only as transportable as the MIDAS workstation. The goal is develop the MIDAS workstation to the point that it can be transferred to industrial and commercial applications, but when this will occur is unknown.

3.3.10 Quality

The quality of the TLMSCI is currently unknown, and will remain unknown until it has been tested and validated.

3.3.11 Adaptation to Various Operational Sites

The TLMSCI can be adapted to various sites, but since it is generic by nature, any tailoring to specific sites will have to be done by the user. This will require creating a library of tasks and simulation state variables in sufficient detail to be fed into the TLM. Once the library is built, the TLM should run fairly autonomously. Designers would be required to monitor task classifications to insure they are valid, and only make limited changes as needed.

3.3.12 Phased Implementation

The TLM results from work conducted during only one phase, phase IV, the last phase that was just finished as of June 1990. During this phase the TLM was created and a functional model implemented in Lisp on a Symbolics 3640 Machine. The next phase is to begin sometime in November of 1990 and will continue for 12 to 18 months. During this next phase the TLM will be tested and validated, and will be expanded to automatically classify tasks and adjust task classifications, both of which are currently done manually by the designer.

3.3.13 General Flow of Data

The flow of data occurs between the TLM and two other MIDAS models: the Scheduler (the task sequencing model) and the task representation methods in the Symbolic Operator Model (goal decomposing model).

The Scheduler calls the TLM during run-time of a simulation to obtain the load values necessary to implement a load balancing strategy.

The task representation methods in the Symbolic Operator Model currently call the TLM before a simulation to obtain the load values necessary to sequence the tasks that need to be simulated.

3.3.14 General Flow of Execution Control

The flow of execution is controlled by the task representation methods in the Symbolic Operator Model. The Symbolic Operator Model determines the task sequences that will be simulated, and in doing so decides when each task or concurrent tasks become active. At this time, the TLM is called to evaluate the loads of the active tasks. The Symbolic Operator Model also calls the TLM prior to the simulation for load values to help decide how to sequence the tasks. The Symbolic Operator Model also can call the Scheduler to help decide how to sequence the tasks. In turn, the Scheduler then calls the TLM for the task loading values the Scheduler requires to implement a load balancing strategy for scheduling tasks. The potential task schedules are then used by the Symbolic Operator Model to help decide how to sequence the tasks.

3.3.15 Networking Requirements

Currently, the Scheduler, Symbolic Operator Model and the TLM use the CHAOS medium, the Nfile protocol and the File service to communicate between the Symbolic Operator Model on the Symbolics file server, and the Scheduler and TLM on two other Symbolics machines.

3.4 SAMPLE OPERATIONAL SCENARIOS

The following scenario illustrates the use of the TLM within the MIDAS environment. To begin with, the user (member of a design team) formulates alternate cockpit designs that would enable a pilot to fly a specific type of mission. The user then renders the design using MIDAS CAD tools. Next, the user would use the Symbolic Operator Model to describe and decompose the tasks for the simulation. The mission is decomposed down to subgoals that are then mapped to various flight activities, which are in turn, decomposed into subactivities that are mapped to their respective pieces of equipment. Mapping subactivities to equipment allows the designer to compare the alternate designs that were formulated.

The equipment-dependent subactivities are the tasks that will be simulated, but at this stage the tasks are lists, not task sequences. To sequence the tasks of a simulation, the tasks and task combinations are evaluated using the TLM to predict the Visual ,Auditory, Cognitive and Motor (VACM) loads that would be imposed on a pilot flying an aircraft using one of the formulated designs. If the predicted values for a given sequence of tasks are acceptable (they fall within tolerance limits determined by the design team), then the task sequence is simulated. If the predicted values are outside the acceptable limits, then the task sequence(s) are evaluated using the Scheduler to generate alternate task sequences.

The Scheduler evaluates several sequences, and determines the best one using a load balancing strategy that specifies the acceptable limits of the task load values. The optimal schedule based on the load balancing strategy is passed to the task representation methods of the Symbolic Operator Model which hands the list to the Simulation Executive component, which executes the simulation of the selected task sequence.

The TLM then evaluates the loads of the tasks that are active at each increment (tick, in the current tick-based system) in the simulation and displays the predicted visual, auditory, cognitive and motor loads of these active tasks along with their task classifications.

This process is repeated for each of the formulated designs. The respective task loading profiles generated by the TLM are analyzed by the users to determine the design that best matches the design criteria. The design criteria are pre-determined by the design team and depend on the design objective, but the criteria could be whether load profiles were within upper and lower boundaries, and whether the task combinations required unreasonable pilot control inputs.

4.0 REQUIREMENTS

The TLM was developed to meet the following Phase IV requirements:

1) to provide some contextual basis for augmenting subjective estimates of projected Visual, Auditory, Cognitive and Motor, (VACM) load, based on certain characteristics or attributes of the task, world state, operator state, and equipment.

2) to develop a method of resource description that is general enough to describe any helicopter aviation task.

3) to develop aa classification taxonomy and methods (e.g., conflict matrices) to describe the interaction of resource demands across processing dimensions (VACM) and tasks.

There are four major components of the TLM: the Task Editor , the Task Adjustor, the Calculator, and the User Interface. These four components represent the functional structure of the software, however, the physical implementation of the code involves an overlapping file structure. There are eight files containing the data object types and the operations performed on them (e.g., flavors, structures, methods, and functions) that are required to run the TLM. The data object types are not separated in the files according to the kind of component, but are separated according to the operations that are performed on the data objects.

4.1 REQUIREMENTS APPROACH AND TRADEOFFS

The Lisp environment was used because object-oriented programming supports the conceptual representation of human perception and performance, and supports the manipulation of the state variables representing specific attributes of perception and performance. The Lisp environment supports rapid development of conceptual approaches for modeling an operator's behavioral states (perceptual, cognitive and motor performance) that can be used to determine operator task loading. Different behavioral states can be represented as objects and the specific attributes of those behaviors can be modelled as state variables. The same can be done for the physical system being modeled. In particular, the physical designs rendered with the MIDAS workstation can be modeled as objects with display attributes modeled as state variables with dynamically changing values.

The result of modeling both the operator's behavioral states and the system's physical states is that the interactions between the two can be modeled and the interactions explored. The values of state variables from other components in the MIDAS workstation can be accessed and integrated into the TLM, making the interactions sensitive to changing contexts in the simulation.

4.2 HARDWARE ENVIRONMENT

The Task Loading Model software runs on the Symbolics 3600 series with 3 MWords CPU memory, 2 14MB fixed disk drives (127MB formatted), an Ethernet Interface card, and a 17" landscape, monochrome monitor with OCLI filter and 1152 * 879 pixel resolution. A minimum of 8 megabytes of memory is recommended to run the TLM.

4.3 SOFTWARE ENVIRONMENT

The Task Loading Model software was written under the Genera 7.2 operating system. Common Lisp with the Flavor System for object oriented programming and the Symbolics Dynamic Windows and Presentation Substrate Systems.

The Genera 7.2 and 8.0 software environment includes Zmail, Zmacs Editor, Symbolics Common Lisp, Document Examiner, Genera windowing environment, CHAOSnet networking software, and Namespace system. The software environment also includes a FORTRAN compiler, IP-TCP communication software 4.2, V67.5, and LMFS (Lisp Machine File System). All the software mentioned is used by the TLM except for the FORTRAN compiler.

The TLM is run in interpreted mode during development efforts, but is run in compiled mode when demonstrating the TLM.

4.4 EXTERNAL INTERFACE REQUIREMENTS

There are two categories of external interfaces, each with different requirements. These external interfaces map directly to the two user groups: (1) a member of the design team, or (2) a MIDAS component that interfaces with the MIDAS TLM.

4.4.1 The Designer's Interface

The designer's interface allows designers to interact with the TLM when evaluating conceptual designs.

4.4.1.1 Purpose of the Interface

The purpose of the interface is to facilitate the use of the TLM by (1) displaying the information about operator performance that is sufficient for a user to judge the quality and efficacy of a conceptual design, and (2) by providing the user with the means to interact with the information that is displayed.

4.4.1.2 Information Content of Displays

The interface displays detailed information about the tasks and the impact of those tasks on an operator. The names of the conceptual design environment, crew selection menu, task names, psychological dimensions and attributes, task loadings, task classifications, and simulation timeline are displayed. The type of design environment and crew members are displayed at the top of the interface. The task names are displayed for each crew member. Under each task name, the imposed task loads are listed with the label for each load, and the attributes used to classify the task are listed along with attribute labels. The simulation timeline is displayed at the bottom of the interface.

4.4.1.3 Interacting with the Interface

The user interacts with the display with the mouse. The user points at mouse-sensitive items that display momentary pop-up menus. These menus allow the user to select the task load data for each crew member, for single-tasks or combined-tasks, and whether the task load data should be re-calculated or re-displayed. The user can use the mouse to point at a scroll bar in the left margin to scroll vertically through the task load and task classification data. The user can also use the mouse to point at a scroll bar in the bottom margin to display the task load and classification data at each time or tick increment in the simulation.

4.4.1.4 Interface Constraints

The current interface is for conceptual designs of helicopter cockpits. There are very few displayed items in the interface that are mouse-sensitive, which limits the application of this version of the interface to different design environments. The user cannot change the names of the system domain, the system operator labels, or the names of the tasks with the interface. The user cannot change the task classifications or the task taxonomy. The user also cannot change the amount of information that will be displayed. To effect any changes, the user has to evaluate and change the appropriate underlying code and then recompile it. Future versions of the interface will increase the number of mouse sensitive items in the display providing the user with capability to adapt the display parameters to different environments.

4.4.2 The Interface to MIDAS Components

There are two MIDAS interfacing systems and components that use the TLM: the Scheduler and the Symbolic Operator Model.

4.4.2.1 Purpose of the Interface

The purpose of the interface is to provide the TLM with the information required to predict the values of the imposed loads on system operators, and to pass the values to the MIDAS components that require them.

4.4.2.2 Information Content

The scheduling component calls the TLM with a call to a single function, using the names of the tasks as the arguments to the function. The function invokes the TLM, and finally returns the load values associated with the tasks.

In order to obtain single task load values, the task decomposition component of the Symbolic Operator Model writes the tasks that will be sequenced to a file that the TLM reads. To obtain the concurrent task load values, the list of sequenced, or active tasks are written to the same file. The TLM returns the values for each task or task combination to the Symbolic Operator Model, or displays them on the TLM interface.

4.4.2.3 Information Flow

There are four ways for data to flow between the Symbolic Operator Model, the Scheduler and the TLM:

1) Symbolic Operator Model to TLM to Symbolic Operator Model: used when the Symbolic Operator Model needs task load values to decide task sequences. The task names are the data passed to the TLM, which passes back four load values for each task.

2) Symbolic Operator Model to Scheduler to TLM to Scheduler to Symbolic Operator Model: used when the Symbolic Operator Model needs alternate schedules to decide the task sequences. The data passed by the Symbolic Operator Model to the Scheduler are lists of instances of task objects and constraints. The Scheduler doesn't pass data to the TLM but calls functions that classify the task objects and evaluates their loads. The Scheduler then passes back a scheduled list of instances of task objects.

3) Scheduler to TLM to Scheduler: used when the Scheduler needs task load values for load balancing independent of the Symbolic Operator Model. The Scheduler doesn't pass data to the TLM but calls functions that classify the task objects and evaluates their loads.

4) Symbolic Operator Model to TLM: used to display the task load values for the task sequence that is being simulated. The task names are passed to the TLM, which displays the four load values for each task.

4.4.2.4 Implementation Constraints

Currently, the TLM cannot obtain information about the states of the simulation environment to adjust the task classifications because the TLM does not interface with any models on the Silicon Graphics machines, such as the aeroguidance model, jack model, vision model, world model, or the data pool. The TLMS communicates only with other Symbolics machines, and only uses information written to a file by the Symbolic Operator Model. Consequently, in Phase IV, all classifications are done manually.

5.0 DESIGN

This section discusses the design of the TLM in detail. It describes the architecture from the system to the file to the individual functions in the files. It also discusses the rational for implementing the architecture of the TLM.

5.1 ARCHITECTURAL DESIGN

The TLM consists of four major components: the Task Editor , the Task Adjustor, the Calculator, and the User Interface.

5.1.1 Design Approach and Tradeoffs

The implementation of the TLM began with developing the TLM Calculator. This component was considered to be the core because it calculated and produced the loading values based on lists of element indices for each dimension. This was coded by Hank Bushnell, as the author at the time was not a programmer. The author then built the current implementation of the TLM as a series of files that fed the input to the calculator in the correct format and output the task data to the display in the correct format. As a result of this approach, the pieces of code were functionally organized into files after the bulk of the code was developed.

5.2 DETAILED DESIGN

There are sixteen files in seven functional groups that implement the four major components of the TLM. The definition of the seven functional groups (FG's) follow:

FG1) Three files contains the code that implements the Task Editor
FG2) One file contains the code that implements the Calculator
FG3) Two files defines the windows, window-panes, and menus that implement the User Interface
FG4) Two files contain the code that formats the task loading data for the User Interface
FG5) Four files contain the code that link the TLM to the Symbolic Operator Model and Scheduler
FG6) One file intializes the TLM
FG7) Three files implement the TLM as a system

This section on detailed design describes the code, logic flow and compilation units in the seven functional groups.

5.2.1 FG1: The task editor and data-input functional group

FILE: SQUID:>stav>midas-tlm>task-names.lisp

DESCRIPTION: This file contains the functions and hash tables containing the names and indices of all the tasks used in the simulation

VARIABLES DEFINED: *CPG-TASK-NAME-TABLE* *PILOT-TASK-NAME-TABLE*
PARAMETERS DEFINED: CPG-LONG-NAMES PILOT-LONG-NAMES
FUNCTIONS DEFINED: CPG-SHORT-NAME PILOT-SHORT-NAME CPG-SHORT-NAMES-TABLE PILOT-SHORT-NAMES-TABLE

NAME: *PILOT-TASK-NAME-TABLE*
DESCRIPTION: defines and initializes the hash table for the short names of the pilot's tasks
INITIAL-VALUE: nil

NAME: *CPG-TASK-NAME-TABLE*
DESCRIPTION: defines and initializes the hash table for the short names of the cpg's tasks
INITIAL-VALUE: nil

NAME: PILOT-SHORT-NAMES-TABLE
DESCRIPTION: makes the hash table containing the short names for the long task names for the pilot
:size 200
INITIAL-VALUE: List of pilot task long-names

NAME: CPG-SHORT-NAMES-TABLE

DESCRIPTION: makes the hash table containing the short names for the long task names for the copilot
:size 200
INITIAL-VALUE: List of copilot task long names

NAME: PILOT-SHORT-NAME
DESCRIPTION: retrieves the short name of a long task name from pilot-task-name-table
INPUTS: *pilot-task-name-table*
CALLING FUNCTIONS: (METHOD PRINT-TO-TASK-ELEMENT-AND-LOAD-PANE TASK-LOAD-WINDOW-FRAME) (METHOD DISPLAY-TASKCOMBO-LABELS TASK-TITLE-PANE)

NAME: CPG-SHORT-NAME
DESCRIPTION: retrieves the short name of a long task name from cpg-task-name-table
INPUTS: *cpg-task-name-table*
CALLING FUNCTIONS: (METHOD PRINT-TO-TASK-ELEMENT-AND-LOAD-PANE TASK-LOAD-WINDOW-FRAME) (METHOD DISPLAY-TASKCOMBO-LABELS TASK-TITLE-PANE)

NAME: PILOT-LONG-NAMES
DESCRIPTION: creates the list of long names for pilot's tasks and assigns it to the variable pilot-long-names
INITIAL-VALUE: nil

NAME: CPG-LONG-NAMES
DESCRIPTION: creates the list of long names for cpg tasks and assigns it to the variable cpg-long-names
INITIAL-VALUE: nil

FILE: SQUID:>stav>midas-tlm>tlm-task-load-data.lisp

DESCRIPTION: This file contains the functions required to access the task classifiaction indices using the tasks names from the task name and index hash table
VARIABLES DEFINED: *CPG-TASK-LOAD-DATA-TABLE* *PILOT-TASK-LOAD-DATA-TABLE*
FUNCTIONS DEFINED: GET-CPG-TASK-INDICES-FROM-TABLE GET-PILOT-TASK-INDICES-FROM-TABLE GET-CPG-TASK-NAMES-FROM-TABLE GET-PILOTTASK-NAMES-FROM-TABLE GET-CPG-ACTIVE-TASKS GET-PILOT-ACTIVE-TASKS CPG-TASK-LOAD-DATA-TABLE PILOT-TASK-LOAD-DATA-TABLE

NAME: *PILOT-TASK-LOAD-DATA-TABLE*
DESCRIPTION: defines and initializes the hash table containing the long task names and their load indices for the pilot
INITIAL-VALUE: nil

NAME: *CPG-TASK-LOAD-DATA-TABLE*
DESCRIPTION: defines and initializes the hash table containing the long task names and their load indices for the copilot
INITIAL-VALUE: nil

NAME: PILOT-TASK-LOAD-DATA-TABLE
DESCRIPTION: creates the hash table containing the long task names and their load indices for the pilot

:size 200
:initial-contents: list of pilot task names and indices

NAME: CPG-TASK-LOAD-DATA-TABLE
DESCRIPTION: creates the hash table containing the long task names and their load indices for the copilot
:size 200
:initial-contents: list of copilot task names and indices

NAME: GET-PILOT-ACTIVE-TASKS
DESCRIPTION: gets a list of the task-load-indices of the active task from the *pilot-task-load-data-table*
INPUTS: TASK-LIST
OUTPUTS: '(current-time list-of-active-tasks current-task)
CALLING FUNCTIONS: GET-AND-DISPLAY-PILOT-COMBINED-TASK-LOADS

NAME: GET-CPG-ACTIVE-TASKS
DESCRIPTION: gets a list of the task-load-indices of the active task from the *cpg-task-load-data-table*
INPUTS: TASK-LIST
OUTPUTS: '(current-time list-of-active-tasks current-task)
CALLING FUNCTIONS: GET-AND-DISPLAY-CPG-COMBINED-TASK-LOADS

NAME: GET-PILOT-TASK-NAMES-FROM-TABLE
DESCRIPTION: gets the keywords from the hash-table - *pilot-task-load-data-table*
FUNCTIONS CALLED: maphash

NAME: GET-CPG-TASK-NAMES-FROM-TABLE
DESCRIPTION: gets the keywords from the hash-table - *cpg-task-load-data-table*
FUNCTIONS CALLED: maphash

NAME: GET-PILOT-TASK-INDICES-FROM-TABLE
DESCRIPTION: gets the pilot's task names and indices from the hash table *pilot-task-load-data-table*
OUTPUTS: '(pilot-single-task-name-list pilot-single-task-index-list)
CALLING FUNCTIONS: DISPLAY-PILOT-SINGLE-TASK-LOADS

NAME: GET-CPG-TASK-INDICES-FROM-TABLE
DESCRIPTION: gets the cpg's task names and indices from the hash table *cpg-task-load-data-table*
OUTPUTS: '(cpg-single-task-name-list cpg-single-task-index-list)
CALLING FUNCTIONS: DISPLAY-CPG-SINGLE-TASK-LOADS

FILE: SQUID:>stav>midas-tlm>tlm-editor.lisp

DESCRIPTION: This file contains the code that was used to input the task indices to the calculator (CCSCI). This file is no longer used in the TLM because of extensive changes. The new input code and procedures are detailed in Section 6: USER'S GUIDE

Future changes and versions will modify this file to search for input data values in the data pools distributed throughout MIDAS.

5.2.2 FG2: The task-load calculator functional group

FILE: SQUID:>Stav>midas-tlm>tlm-calculator.lisp

AUTHOR: Hank Bushnell
DESCRIPTION: This file contains the functions, constants, and variables required to calculate the task load values
VARIABLES DEFINED: *COGNITIVE-MOTOR-CM* *AUDITORY-COGNITIVE-CM* *VISUAL-COGNITIVE-CM* *VISUAL-AUDITORY-CM* *MOTOR-CM* *COGNITIVE-CM* *AUDITORY-CM* *VISUAL-CM*
CONSTANTS DEFINED: These constants are the indices into the conflict matrices for each element in each dimension: *MOTOR-BOTH* *MOTOR-LEFT* *MOTOR-RIGHT* *MOTOR-FINGER* *MOTOR-FEET* *MOTOR-HAND* *MOTOR-EYE* *MOTOR-HEAD* *MOTOR-MOUTH* *MOTOR-FINE* *MOTOR-GROSS* *MOTOR-CONTIN* *MOTOR-DISCRETE* *MOTOR-FAR* *MOTOR-NEAR* *MOTOR-SPATIAL* *MOTOR-VERBAL* *COG-UNPLANNED* *COG-PLANNED* *COG-SPATIAL* *COG-VERBAL* *COG-MULTIPLE* *COG-SINGLE* *COG-TRANSFORM* *COG-DIRECT* *AUD-MASKED* *AUD-SALIENT* *AUD-SPEECH* *AUD-SIGNAL* *AUD-DISCRIM* *AUD-ORIENT* *VIS-DYNAMIC* *VIS-STATIC* *VIS-MASKED* *VIS-SALIENT* *VIS-FEATURES* *VIS-OBJECTS* *VIS-INTEGRAL* *VIS-SEPARABLE* *VIS-FIXATE* *VIS-SCAN* *VIS-FAR* *VIS-NEAR*
FUNCTIONS DEFINED: SCALE-LOAD-VALUES GENERATE-LOAD MOTOR-LOAD COGNITIVE-LOAD AUDITORY-LOAD VISUAL-LOAD BETWEEN-SUM-OVER-TASKS-2 BETWEEN-SUM-OVER-TASKS-1 COGNITIVE-MOTOR-LOAD AUDITORY-COGNITIVE-LOAD VISUAL-COGNITIVE-LOAD VISUAL-AUDITORY-LOAD WITHIN-SUM-OVER-TASKS-2 WITHIN-SUM-OVER-TASKS-1 WITHIN-SUM-OVER-ONE-TASK WITHIN-MOTOR-LOAD WITHIN-COGNITIVE-LOAD WITHIN-AUDITORY-LOAD WITHIN-VISUAL-LOAD PRINT-2D-ARRAY PRINT-SYM-ARRAY MAKE-SYM-ARRAY ASET-SYM

NAME: ASET-SYM
DESCRIPTION:Sets both symmetric elements of a symmetric array whenever one is set.
INPUTS: SYMMETRIC-ARRAY VALUE I J
CALLING FUNCTIONS: MAKE-SYM-ARRAY

NAME: MAKE-SYM-ARRAY
DESCRIPTION: Creates and initializes a symmetric array.
INPUTS: &REST REST &KEY INITIAL-CONTENTS
OUTPUTS: SYM-ARRAY
FUNCTIONS CALLED: ASET-SYM

NAME: PRINT-SYM-ARRAY
DESCRIPTION: Print out a symmetric array, checking that it really is symmetric.
INPUTS: SYM-ARRAY &OPTIONAL (STREAM T)
OUTPUTS:

NAME: PRINT-2D-ARRAY
DESCRIPTION: Print out a 2-dimensional array.
INPUTS: ARRAY &OPTIONAL (STREAM T)

NAME: *VISUAL-CM*
DESCRIPTION: The conflict matrix for the Visual dimension
INITIAL-VALUE: A matrix of conflict and demand values'

NAME: *AUDITORY-CM*

DESCRIPTION: The conflict matrix for the Auditory dimension.
INITIAL-VALUE: A matrix of conflict and demand values

NAME: *COGNITIVE-CM*
DESCRIPTION: The conflict matrix for the Cognitive dimension.
INITIAL-VALUE: A matrix of conflict and demand values

NAME: *MOTOR-CM*
DESCRIPTION: The conflict matrix for the Motor dimension.
INITIAL-VALUE: A matrix of conflict and demand values

NAME: *VISUAL-AUDITORY-CM*
DESCRIPTION: The conflict matrix for the Visual-Auditory dimension.
INITIAL-VALUE: A matrix of conflict and demand values

NAME: *VISUAL-COGNITIVE-CM*
DESCRIPTION: The conflict matrix for the Visual-Cognitive dimension.
INITIAL-VALUE: A matrix of conflict and demand values

NAME: *AUDITORY-COGNITIVE-CM*
DESCRIPTION: The conflict matrix for the Auditory-Cognitive dimension.
INITIAL-VALUE: A matrix of conflict and demand values

NAME: *COGNITIVE-MOTOR-CM*
DESCRIPTION: The conflict matrix for the Cognitive-Motor dimension.
INITIAL-VALUE: A matrix of conflict and demand values

NAME: TASK-ELEMENTS (structure)
DESCRIPTION: Structure describing the elements of a task.
INITIAL-VALUE: '(visual auditory cognitive motor)

NAME: WITHIN-VISUAL-LOAD
DESCRIPTION: Calculate the visual load of a set of tasks.
INPUTS: ELEMENTS-OF-TASKS
FUNCTIONS CALLED: TASK-ELEMENTS-VISUAL WITHIN-SUM-OVER-ONE-TASK WITHIN-SUM-OVER-TASKS-1 WITHIN-SUM-OVER-TASKS-2
CALLING FUNCTIONS: VISUAL-LOAD GENERATE-LOAD

NAME: WITHIN-AUDITORY-LOAD
DESCRIPTION: Calculate the auditory load of a set of tasks.
INPUTS: ELEMENTS-OF-TASKS
FUNCTIONS CALLED: TASK-ELEMENTS-AUDITORY WITHIN-SUM-OVER-ONE-TASK WITHIN-SUM-OVER-TASKS-1 WITHIN-SUM-OVER-TASKS-2
CALLING FUNCTIONS: AUDITORY-LOAD GENERATE-LOAD

NAME: WITHIN-COGNITIVE-LOAD
DESCRIPTION: Calculate the cognitive load of a set of tasks.
INPUTS: ELEMENTS-OF-TASKS
FUNCTIONS CALLED: TASK-ELEMENTS-COGNITIVE WITHIN-SUM-OVER-ONE-TASK WITHIN-SUM-OVER-TASKS-1 WITHIN-SUM-OVER-TASKS-2
CALLING FUNCTIONS: COGNITIVE-LOAD GENERATE-LOAD

NAME: WITHIN-MOTOR-LOAD
DESCRIPTION: Calculate the motor load of a set of tasks.

INPUTS: ELEMENTS-OF-TASKS
FUNCTIONS CALLED: TASK-ELEMENTS-MOTOR WITHIN-SUM-OVER-ONE-TASK WITHIN-SUM-OVER-TASKS-1 WITHIN-SUM-OVER-TASKS-2
CALLING FUNCTIONS: MOTOR-LOAD GENERATE-LOAD

NAME: WITHIN-SUM-OVER-ONE-TASK
DESCRIPTION: Sum the contributions to a load from one tasks: a[t,k,i]*a[t,j,i]) + (a[t,j,i]*a[t,j,k])
INPUTS: CONFLICT-MATRIX TASK-ELEMENTS-ACCESSOR ELEMENTS-OF-TASKS I J K
CALLING FUNCTIONS: WITHIN-VISUAL-LOAD WITHIN-AUDITORY-LOAD WITHIN-COGNITIVE-LOAD WITHIN-MOTOR-LOAD

NAME: WITHIN-SUM-OVER-TASKS-1
DESCRIPTION: Sum the column-wise contributions to a load from all tasks (a[t,k,i]*a[t,j,i])
INPUTS: CONFLICT-MATRIX TASK-ELEMENTS-ACCESSOR ELEMENTS-OF-TASKS I J K
CALLING FUNCTIONS: WITHIN-VISUAL-LOAD WITHIN-AUDITORY-LOAD WITHIN-COGNITIVE-LOAD WITHIN-MOTOR-LOAD

NAME: WITHIN-SUM-OVER-TASKS-2
DESCRIPTION: sum the row-wise contributions to a load from all tasks (a[t,j, i]*a[t,j,k])
INPUTS: CONFLICT-MATRIX TASK-ELEMENTS-ACCESSOR ELEMENTS-OF-TASKS I J K
CALLING FUNCTIONS: WITHIN-VISUAL-LOAD WITHIN-AUDITORY-LOAD WITHIN-COGNITIVE-LOAD WITHIN-MOTOR-LOAD

NAME: VISUAL-AUDITORY-LOAD
DESCRIPTION: Calculate the visual-auditory load for a set of tasks.
INPUTS: ELEMENTS-OF-TASKS
FUNCTIONS CALLED: TASK-ELEMENTS-VISUAL TASK-ELEMENTS-AUDITORY BETWEEN-SUM-OVER-TASKS-1 BETWEEN-SUM-OVER-TASKS-2
CALLING FUNCTIONS: VISUAL-LOAD AUDITORY-LOAD GENERATE-LOAD

NAME: VISUAL-COGNITIVE-LOAD
DESCRIPTION: Calculate the visual-cognitive load for a set of tasks.
INPUTS: ELEMENTS-OF-TASKS
FUNCTIONS CALLED: TASK-ELEMENTS-COGNITIVE BETWEEN-SUM-OVER-TASKS-1 BETWEEN-SUM-OVER-TASKS-2
CALLING FUNCTIONS: COGNITIVE-LOAD GENERATE-LOAD

NAME: AUDITORY-COGNITIVE-LOAD
DESCRIPTION: Calculate the auditory-cognitive load for a set of tasks.
INPUTS: ELEMENTS-OF-TASKS
FUNCTIONS CALLED: TASK-ELEMENTS-COGNITIVE BETWEEN-SUM-OVER-TASKS-1 BETWEEN-SUM-OVER-TASKS-2
CALLING FUNCTIONS: AUDITORY-LOAD COGNITIVE-LOAD GENERATE-LOAD

NAME: COGNITIVE-MOTOR-LOAD
DESCRIPTION: Calculate the cognitive-motor load for a set of tasks.
INPUTS: ELEMENTS-OF-TASKS
FUNCTIONS CALLED: TASK-ELEMENTS-COGNITIVE TASK-ELEMENTS-MOTOR BETWEEN-SUM-OVER-TASKS-1 BETWEEN-SUM-OVER-TASKS-2

CALLING FUNCTIONS: COGNITIVE-LOAD MOTOR-LOAD GENERATE-LOAD

NAME: BETWEEN-SUM-OVER-TASKS-1
DESCRIPTION: Sum the row-wise contributions to a load from all the tasks.a[t,k,i]*a[t,k,j]
INPUTS: CONFLICT-MATRIX ROW-TASK-ELEMENTS-ACCESSOR COL-TASK-ELEMENTS-ACCESSOR ELEMENTS-OF-TASKS K I J
CALLING FUNCTIONS: VISUAL-AUDITORY-LOAD VISUAL-COGNITIVE-LOAD AUDITORY-COGNITIVE-LOAD COGNITIVE-MOTOR-LOAD

NAME: BETWEEN-SUM-OVER-TASKS-2
DESCRIPTION: Sum the column-wise contributions to a load from all the tasks.a[t,k,i]*a[t,m,i]
INPUTS: CONFLICT-MATRIX ROW-TASK-ELEMENTS-ACCESSOR ELEMENTS-OF-TASKS I K M
CALLING FUNCTIONS: VISUAL-AUDITORY-LOAD VISUAL-COGNITIVE-LOAD AUDITORY-COGNITIVE-LOAD COGNITIVE-MOTOR-LOAD

NAME: VISUAL-LOAD
DESCRIPTION: Calculate the average of the sums of the three possible sources of visual loads.
INPUTS: ELEMENTS-OF-TASKS
FUNCTIONS CALLED: WITHIN-VISUAL-LOAD VISUAL-AUDITORY-LOAD VISUAL-COGNITIVE-LOAD
CALLING FUNCTIONS: GENERATE-LOAD

NAME: AUDITORY-LOAD
DESCRIPTION: Calculate the average of the sums of the three possible sources ofloads.
INPUTS: ELEMENTS-OF-TASKS
FUNCTIONS CALLED: WITHIN-AUDITORY-LOAD VISUAL-AUDITORY-LOAD AUDITORY-COGNITIVE-LOAD
CALLING FUNCTIONS: GENERATE-LOAD

NAME: COGNITIVE-LOAD
DESCRIPTION: Calculate the average of thesums of the four possible sources of cognitive loads.
INPUTS: ELEMENTS-OF-TASKS
FUNCTIONS CALLED: WITHIN-COGNITIVE-LOAD AUDITORY-COGNITIVE-LOAD VISUAL-COGNITIVE-LOAD COGNITIVE-MOTOR-LOAD
CALLING FUNCTIONS: GENERATE-LOAD

NAME: MOTOR-LOAD
DESCRIPTION: Calculate the average of the sums of the two possible sources of motor loads.
INPUTS: ELEMENTS-OF-TASKS
FUNCTIONS CALLED: WITHIN-MOTOR-LOAD COGNITIVE-MOTOR-LOAD
CALLING FUNCTIONS: GENERATE-LOAD

NAME: GENERATE-LOAD
DESCRIPTION: takes one or more task objects as its argument, and prints out the respective individual and multiple task loads
INPUTS: TASKS
OUTPUTS: '(WITHIN-AUDITORY-LOAD WITHIN-COGNITIVE-LOAD WITHIN-MOTOR-LOAD VISUAL-AUDITORY-LOAD AUDITORY-COGNITIVE-LOAD

VISUAL-COGNITIVE-LOAD COGNITIVE-MOTOR-LOAD VISUAL-LOAD AUDITORY-LOAD COGNITIVE-LOAD MOTOR-LOAD)
FUNCTIONS CALLED: (READ-INSTANCE-VARIABLE INDEX-V TASK INDEX-V) (READ-INSTANCE-VARIABLE INDEX-A TASK INDEX-A) (READ-INSTANCE VARIABLE INDEX-C TASK INDEX-C) (READ-INSTANCE-VARIABLE INDEX-M TASK INDEX-M) WITHIN-VISUAL-LOAD SCALE-LOAD-VALUES WITHIN-AUDITORY-LOAD WITHIN-COGNITIVE-LOAD WITHIN-MOTOR-LOAD VISUAL-AUDITORY-LOAD AUDITORY-COGNITIVE-LOAD VISUAL-COGNITIVE-LOAD COGNITIVE-MOTOR-LOAD VISUAL-LOAD AUDITORY-LOAD COGNITIVE-LOAD MOTOR-LOAD
CALLING FUNCTIONS: SHOW-TASK-LOAD DISPLAY-SINGLE-TASK-LOADS DISPLAY-COMBINED-TASK-LOADS MULTIPLE-TASK-LOAD SCHEDULER-LOAD-LIST

NAME: SCALE-LOAD-VALUES
DESCRIPTION: scales the loading values
INPUTS: LOAD-VALUE
OUTPUTS: SCALED-VALUE
CALLING FUNCTIONS: GENERATE-LOAD

5.2.3 FG3: The display windows and menus functional group

FILE: SQUID:>stav>midas-tlm>task-load-interface-frame.lisp

DESCRIPTION: This file contains the flavors and methods required to create the windows and panes that are used to display the task load data
VARIABLES DEFINED: *HW1*
FLAVORS DEFINED: TASK-LOAD-WINDOW-FRAME TIME-LINE-LABEL-PANE TIME-LINE-PANE TASK-COMBINATION-LOAD-PANE SINGLE-TASK-LOAD-PANE
TASK-LOAD-PANE-PARENT TASK-ELEMENT-LABEL-PANE ELEMENT-TITLE-PANE TASK-TITLE-PANE CREW-SELECTION-MENU-PANE TLM-TITLE-PANE
METHODS DEFINED:
(DISPLAY-MOTOR-ELEMENT-VALUES TASK-LOAD-PANE-PARENT)
(DISPLAY-MOTOR-ELEMENT-LABELS TASK-ELEMENT-LABEL-PANE)
(DISPLAY-COGNITIVE-ELEMENT-VALUES TASK-LOAD-PANE-PARENT)
(DISPLAY-COGNITIVE-ELEMENT-LABELS TASK-ELEMENT-LABEL-PANE)
(DISPLAY-AUDITORY-ELEMENT-VALUES TASK-LOAD-PANE-PARENT)
(DISPLAY-AUDITORY-ELEMENT-LABELS TASK-ELEMENT-LABEL-PANE)
(DISPLAY-VISUAL-ELEMENT-VALUES TASK-LOAD-PANE-PARENT)
(DISPLAY-VISUAL-ELEMENT-LABELS TASK-ELEMENT-LABEL-PANE)
(DISPLAY-VACM-LOAD-VALUES TASK-LOAD-PANE-PARENT)
(DISPLAY-VACM-LOAD-LABELS TASK-ELEMENT-LABEL-PANE)
(DRAW-TIME-LINE TIME-LINE-PANE)
(DISPLAY-CREW-LABEL ELEMENT-TITLE-PANE)
(DISPLAY-TASKCOMBO-LABELS TASK-TITLE-PANE)
(DISPLAY-TASK-LABELS TASK-TITLE-PANE)
(PRINT-THE-INITIAL-DISPLAY TASK-LOAD-WINDOW-FRAME)
(PRINT-TO-TASKCOMBO-ELEMENT-AND-LOAD-PANE TASK-LOAD-WINDOW-FRAME)
(PRINT-TO-TASK-ELEMENT-AND-LOAD-PANE TASK-LOAD-WINDOW-FRAME)
(Y-SCROLL-TO TASK-ELEMENT-LABEL-PANE AFTER)
(X-SCROLL-TO SINGLE-TASK-LOAD-PANE AFTER)

(X-SCROLL-TO TIME-LINE-PANE AFTER)
(ASSOCIATED-COMBINED-TITLE-PANE TIME-LINE-PANE)
(ASSOCIATED-COMBINED-LOAD-PANE TIME-LINE-PANE)
(ASSOCIATED-TITLE-PANE SINGLE-TASK-LOAD-PANE)
(ASSOCIATED-LOAD-PANE TASK-ELEMENT-LABEL-PANE)
(INIT TASK-LOAD-WINDOW-FRAME AFTER)

NAME: *HW1*
DESCRIPTION: task-load-window-frame object initialized to nil
INITIAL-VALUE: NIL

NAME: TLM-TITLE-PANE
DESCRIPTION: Defines the top column pane containing the title "MIDAS Task Load Modelling Environment"
COMPONENT FLAVORS: DYNAMIC-WINDOW-PANE

NAME: CREW-SELECTION-MENU-PANE
DESCRIPTION: Defines the second column pane containing the mouse sensitive crew selection menu
COMPONENT FLAVORS: TOP-LABEL-MIXIN CENTERED-LABEL-MIXIN COMMAND-MENU-PANE
:COLUMNS 3
:ITEM-LIST *CREW-MENU*

NAME: TASK-TITLE-PANE
DESCRIPTION: Defines the 2nd row pane of the third column panes containing the task names
COMPONENT FLAVORS: DYNAMIC-WINDOW-PANE
INSTANCE VARIABLES: SINGLE-TASK-LOAD-PANE COMBINATION-LOAD-PANE TIME-LINE-PANE
METHODS: DISPLAY-TASKCOMBO-LABELS DISPLAY-TASK-LABELS

NAME: ELEMENT-TITLE-PANE
DESCRIPTION: defines the 1st row pane of the third column panes containing the selected crew's name
COMPONENT FLAVORS: DYNAMIC-WINDOW-PANE
METHODS: DISPLAY-CREW-LABEL

NAME: TASK-ELEMENT-LABEL-PANE
DESCRIPTION: defines the 1st row pane of the fourth column panes containing the tlm element labels
COMPONENT FLAVORS: DYNAMIC-WINDOW-PANE
INSTANCE VARIABLES: SINGLE-TASK-LOAD-PANE TASK-COMBINATION-LOAD-PANE
METHODS: DISPLAY-MOTOR-ELEMENT-LABELS DISPLAY-COGNITIVE-ELEMENT-LABELS DISPLAY-AUDITORY-ELEMENT-LABELS DISPLAY-VISUAL-ELEMENT-LABELS DISPLAY-VACM-LOAD-LABELS Y-SCROLL-TO ASSOCIATED-LOAD-PANE

NAME: TASK-LOAD-PANE-PARENT
DESCRIPTION: Parent of SINGLE-TASK-LOAD-PANE and TASK-COMBINATION-LOAD-PANE.
COMPONENT FLAVORS: DYNAMIC-WINDOW-PANE

DEPENDENT FLAVORS: TASK-COMBINATION-LOAD-PANE SINGLE-TASK-LOAD-PANE
INSTANCE VARIABLES: TICK-SCALE :initial-value 5
    TICK-SPACING :initial-value 5
    ROW-VSP :initial-value 5
    START-TIME :initial-value 9999999999
    DEFAULT-START-TIME :initial-value 0
    END-TIME :initial-value -99999999
    DEFAULT-END-TIME :initial-value 200
METHODS: DISPLAY-MOTOR-ELEMENT-VALUES DISPLAY-COGNITIVE-ELEMENT-VALUES DISPLAY-AUDITORY-ELEMENT-VALUES DISPLAY-VISUAL-ELEMENT-VALUES DISPLAY-VACM-LOAD-VALUES

NAME: SINGLE-TASK-LOAD-PANE
DESCRIPTION: displays the load and element selection values for single tasks
COMPONENT FLAVORS: TASK-LOAD-PANE-PARENT
INSTANCE VARIABLES: TASK-TITLE-PANE TASK-ELEMENT-LABEL-PANE
METHODS: X-SCROLL-TO ASSOCIATED-TITLE-PANE

NAME: TASK-COMBINATION-LOAD-PANE
DESCRIPTION: displays the load and element section values for combined tasks
COMPONENT FLAVORS: TASK-LOAD-PANE-PARENT
INSTANCE VARIABLES: TASK-TITLE-PANE TIME-LINE-PANE

NAME: TIME-LINE-PANE
DESCRIPTION: displays the ticks and times associated with each task
COMPONENT FLAVORS: DYNAMIC-WINDOW-PANE
INSTANCE VARIABLES: SINGLE-TASK-LOAD-PANE TASK-COMBINATION-LOAD-PANE TASK-TITLE-PANE
METHODS: DRAW-TIME-LINE X-SCROLL-TO ASSOCIATE-COMBINED-TITLE-PANE ASSOCIATED-COMBINED-LOAD-PANE

NAME: TIME-LINE-LABEL-PANE
DESCRIPTION: displays the label for the time-line-pane
COMPONENT FLAVORS: DYNAMIC-WINDOW-PANE

NAME: TASK-LOAD-WINDOW-FRAME
DESCRIPTION: defines the window frame that contains two configurations of all the panes: one for single tasks and one configuration for combined tasks
COMPONENT FLAVORS: BORDERED-CONSTRAINT-FRAME-WITH-SHARED-IO-BUFFER
INSTANCE VARIABLES: TLM-TITLE-PANE CREW-SELECTION-MENU-PANE TASK-TITLE-PANE ELEMENT-TITLE-PANE PILOT-SINGLE-TASK-LABEL-PANE CPG-SINGLE-TASK-LABEL-PANE PILOT-COMBINED-TASK-LABEL-PANE CPG COMBINED-TASK-LABEL-PANE TASK-ELEMENT-LABEL-PANE TASK-LOAD-PANE-PARENT SINGLE-TASK-LOAD-PANE TASK-COMBINATION-LOAD-PANE TIME-LINE-PANE TIME-LINE-LABEL-PANE PILOT-SINGLE-TASK-TITLE-PANE PILOT-COMBINED-TASK-TITLE-PANE CPG-SINGLE-TASK-TITLE-PANE CPG-COMBINED-TASK-TITLE-PANE PILOT-SINGLE-TASK-ELEMENT-LABEL-PANE PILOT-COMBINED-TASK-ELEMENT-LABEL-PANE CPG-SINGLE-TASK-ELEMENT-LABEL-PANE CPG-COMBINED-TASK-ELEMENT-LABEL-PANE PILOT-SINGLE-TASK-LOAD-PANE CPG-SINGLE-TASK-LOAD-PANE PILOT-TASK-COMBINATION-LOAD-PANE CPG-TASK-COMBINATION-LOAD-PANE PILOT-TIME-LINE-PANE CPG-TIME-LINE-PANE

INCREMENT-VALUE-COLUMN-PILOT :initial-value 4
INCREMENT-VALUE-COLUMN-CPG :initial-value 4
INCREMENT-LABEL-COLUMN-PILOT :initial-value 3
INCREMENT-LABEL-COLUMN-CPG :initial-value 3
STRING-XPOS-PILOT :initial-value 4
LINE-XPOS-PILOT :initial-value 32
STRING-XPOS-CPG :initial-value 4
LINE-XPOS-CPG :initial-value 32
TICK-HEIGHT :initial-value 53
ACTIVITIES-LIST :initial-value '()
METHODS: PRINT-THE-INITIAL-DISPLAY PRINT-TO-TASKCOMBO-ELEMENT-AND-LOAD-PANE PRINT-TO-TASK-ELEMENT-AND-LOAD-PANE INIT
:CONFIGURATIONS '((config1 config2 config3 config4))

NAME: (FLAVOR:METHOD INIT TASK-LOAD-WINDOW-FRAME)
DESCRIPTION: assigns the names of the instance variable panes their associated instances of the panes and lets instances of the panes of each instance variable know about instances of panes associated with it
INPUTS: &REST IGNORE

NAME: (FLAVOR:METHOD ASSOCIATED-LOAD-PANE TASK-ELEMENT-LABEL-PANE)
DESCRIPTION: gets the panes associated with scrolling task-element-label-pane for the appropriate configuration

NAME: (FLAVOR:METHOD ASSOCIATED-TITLE-PANE SINGLE-TASK-LOAD-PANE)
DESCRIPTION: gets the panes associated with scrolling single-task-load-pane for the appropriate configuration

NAME: (FLAVOR:METHOD ASSOCIATED-COMBINED-LOAD-PANE TIME-LINE-PANE)
DESCRIPTION: gets the combined task load panes associated with scrolling time-line-pane for the appropriate configuration

NAME: (FLAVOR:METHOD ASSOCIATED-COMBINED-TITLE-PANE TIME-LINE-PANE)
DESCRIPTION: gets the combined task title panes associated with scrolling time-line-pane for the appropriate configuration

NAME: (FLAVOR:METHOD X-SCROLL-TO TIME-LINE-PANE)
DESCRIPTION: scrolls pilot-task-combination-load-pane and pilot-combined-task-title-pane if its config3 or cpg-task-combination-load-pane and cpg-combined-task-title-pane if its config4 when time-line-pane is scrolled
INPUTS: &REST ARGS

NAME: (FLAVOR:METHOD X-SCROLL-TO SINGLE-TASK-LOAD-PANE)
DESCRIPTION: scrolls pilot-single-task-title-pane if its config1 or cpg-single-task-title-pane if its config2 when either pilot-single-task-load-pane or cpg-single-task-load-pane is scrolled
INPUTS: &REST ARGS

NAME: (FLAVOR:METHOD Y-SCROLL-TO TASK-ELEMENT-LABEL-PANE)

DESCRIPTION: scrolls pilot-single-task-load-pane if its config1 or cpg-single-task-load-pane if its config2 or pilot-task-combination-load-pane if its config3 or cpg-task-combination-load-pane if its config4 when task-element-label-pane is scrolled
INPUTS: &REST ARGS

NAME: (FLAVOR:METHOD PRINT-TO-TASK-ELEMENT-AND-LOAD-PANE TASK-LOAD-WINDOW-FRAME)
DESCRIPTION: displays task load data for single tasks (scheduler - SSCI - calls this)
INPUTS: TASK-OBJECTS TASK-NAMES LAST-LABEL-P

NAME: (FLAVOR:METHOD PRINT-TO-TASKCOMBO-ELEMENT-AND-LOAD-PANE TASK-LOAD-WINDOW-FRAME)
DESCRIPTION: displays task load data for task combinations
INPUTS: TASK-OBJECTS CURRENT-TIME TASK-NAMES

NAME: (FLAVOR:METHOD PRINT-THE-INITIAL-DISPLAY TASK-LOAD-WINDOW-FRAME)
DESCRIPTION: displays the initial frames and panes without load values displayed - used for selecting the display

NAME: (FLAVOR:METHOD DISPLAY-TASK-LABELS TASK-TITLE-PANE)
DESCRIPTION: displays the labels for each task in the task title pane
INPUTS: INCREMENT-COLUMN TASK-LABEL

NAME: (FLAVOR:METHOD DISPLAY-TASKCOMBO-LABELS TASK-TITLE-PANE)
DESCRIPTION: displays the labels for each task combintion in the task title pane
INPUTS: INCREMENT-COLUMN TASK-NAMES

NAME: (FLAVOR:METHOD DISPLAY-CREW-LABEL ELEMENT-TITLE-PANE)
DESCRIPTION: displays the crew label in the task title pane
INPUTS: CREW-LABEL

NAME: (FLAVOR:METHOD DRAW-TIME-LINE TIME-LINE-PANE)
DESCRIPTION: this queries the task index files for changes in the file upon which it reads in the time and draws the time line in the time line pane and labels each tick with the current time
INPUTS: STRING-XPOS LINE-XPOS CURRENT-TIME

NAME: (FLAVOR:METHOD DISPLAY-VACM-LOAD-LABELS TASK-ELEMENT-LABEL-PANE)
DESCRIPTION: determines the cursor position and format for each vacm component load label in the task-element-label-pane

NAME: (FLAVOR:METHOD DISPLAY-VACM-LOAD-VALUES TASK-LOAD-PANE-PARENT)
DESCRIPTION: determines the cursor position and format for each vacm component load value in the task-element-label-pane
INPUTS: OBJECT INCREMENT-COLUMN

NAME: (FLAVOR:METHOD DISPLAY-VISUAL-ELEMENT-LABELS TASK-ELEMENT-LABEL-PANE)
DESCRIPTION: determines the cursor position and format for each visual component load label in the task-element-label-pane

NAME: (FLAVOR:METHOD DISPLAY-VISUAL-ELEMENT-VALUES TASK-LOAD-PANE-PARENT)
DESCRIPTION: determines the cursor position and format for each visual component load value in the task-element-label-pane
NPUTS: OBJECT INCREMENT-COLUMN

NAME: (FLAVOR:METHOD DISPLAY-AUDITORY-ELEMENT-LABELS TASK-ELEMENT-LABEL-PANE)
DESCRIPTION: determines the cursor position and format for each auditory component load label in the task-element-label-pane

NAME: (FLAVOR:METHOD DISPLAY-AUDITORY-ELEMENT-VALUES TASK-LOAD-PANE-PARENT)
DESCRIPTION: determines the cursor position and format for each auditory component load value in the task-element-label-pane
INPUTS: OBJECT INCREMENT-COLUMN

NAME: (FLAVOR:METHOD DISPLAY-COGNITIVE-ELEMENT-LABELS TASK-ELEMENT-LABEL-PANE)
DESCRIPTION: determines the cursor position and format for each cognitive component load label in the task-element-label-pane

NAME: (FLAVOR:METHOD DISPLAY-COGNITIVE-ELEMENT-VALUES TASK-LOAD-PANE-PARENT)
DESCRIPTION: determines the cursor position and format for each cognitive component load value in the task-element-label-pane
INPUTS: OBJECT INCREMENT-COLUMN

NAME: (FLAVOR:METHOD DISPLAY-MOTOR-ELEMENT-LABELS TASK-ELEMENT-LABEL-PANE)
DESCRIPTION: determines the cursor position and format for each motor component load label in the task-element-label-pane

NAME: (FLAVOR:METHOD DISPLAY-MOTOR-ELEMENT-VALUES TASK-LOAD-PANE-PARENT)
DESCRIPTION: determines the cursor position and format for each motor component load value in the task-element-label-pane
INPUTS: OBJECT INCREMENT-COLUMN

FILE: SQUID:>stav>midas-tlm>select-task-display.lisp

DESCRIPTION: This file contains the parameters and functions that create the pop-up windows and call the functions that evaluate and display the loads and task classifications for single and combined tasks separately for the pilot and copilot-gunner
VARIABLES DEFINED: *CREW-MENU*
PARAMETERS DEFINED: SELECT-RESET-TLM-DISPLAYSELECT-CPG-DISPLAY-TYPE-COMBINED SELECT-PILOT-DISPLAY-TYPE-COMBINED SELECT-CPG-DISPLAY-TYPE-SINGLE SELECT-PILOT-DISPLAY-TYPE-SINGLE SELECT-CPG-TASK-DISPLAY
FUNCTIONS DEFINED: START-CPG-PROCESS START-PILOT-PROCESS REDISPLAY-CPG-TASK-COMBINATIONS CPG-TASK-COMBINATIONS REDISPLAY-PILOT-TASK-COMBINATIONS PILOT-TASK-COMBINATIONS REDISPLAY-CPG-SINGLE-TASKS DISPLAY-CPG-SINGLE-TASKS REDISPLAY-

PILOT-SINGLE-TASKS DISPLAY-PILOT-SINGLE-TASKS CHECK-FILE-FOR-ACTIVE-CPG-TASKS CHECK-FILE-FOR-ACTIVE-PILOT-TASKS CPG-DISPLAY-TYPE-COMBINED CPG-DISPLAY-TYPE-SINGLE CPG-TASK-DISPLAY PILOT-DISPLAY-TYPE-COMBINED
PILOT-DISPLAY-TYPE-SINGLE PILOT-TASK-DISPLAY RESET-TLM-DISPLAY CLEAR-PILOT-PANES CLEAR-CPG-PANES

NAME: *CREW-MENU*
DESCRIPTION: defines the menu for the crew-selection-menu-pane
INITIAL-VALUE: ((NO-SELECT NIL) ("CREW SELECTION MENU" :value erase :documentation "")(NO-SELECT NIL)
(Pilot VALUE PILOT DOCUMENTATION Selects the Pilot's Tasks) (NO-SELECT NIL)
(Copilot/Gunner VALUE COPILOT DOCUMENTATION Selects the Copilot's Tasks))

NAME: SELECT-RESET-TLM-DISPLAY
DESCRIPTION: creates the pop-up menu to Clear the displays and restart the MIDAS-TLM process
INITIAL-VALUE: :MAKE-WINDOW 'MOMENTARY-MENU
'LABEL How about a nice game of chess
'BORDERS 3
'ITEM-LIST '("RESET-TLM")
FUNCTIONS CALLED: RESET-TLM

NAME: SELECT-PILOT-TASK-DISPLAY
DESCRIPTION: creates the pop-up menu to select the pilot single or combined task display pop-up-menu
INITIAL-VALUE: :MAKE-WINDOW 'MOMENTARY-MENU
'LABEL Select Display
'BORDERS 3
'ITEM-LIST '("Single Tasks" "Combined Tasks" "Clear Display")
FUNCTIONS CALLED: PILOT-DISPLAY-TYPE-SINGLE PILOT-DISPLAY-TYPE-COMBINED CLEAR-PILOT-PANES

NAME: SELECT-CPG-TASK-DISPLAY
DESCRIPTION: creates the pop-up menu to select the cpg single or combined task display pop-up-menu
INITIAL-VALUE: :MAKE-WINDOW 'MOMENTARY-MENU
'LABEL Select Display
'BORDERS 3
'ITEM-LIST '("Single Tasks" "Combined Tasks" "Clear Display")
FUNCTIONS CALLED: CPG-DISPLAY-TYPE-SINGLE CPG-DISPLAY-TYPE-COMBINED

NAME: SELECT-PILOT-DISPLAY-TYPE-SINGLE
DESCRIPTION: creates the pop-up menu to select the pilot single task display
INITIAL-VALUE: :MAKE-WINDOW 'MOMENTARY-MENU
'LABEL Select Display
'BORDERS 3
'ITEM-LIST '("Calculate Loads" "Redisplay Loads")
FUNCTIONS CALLED: DISPLAY-PILOT-SINGLE-TASKS REDISPLAY-PILOT-SINGLE-TASKS

NAME: SELECT-CPG-DISPLAY-TYPE-SINGLE
DESCRIPTION: creates the pop-up menu to select the cpg single task display
INITIAL-VALUE: :MAKE-WINDOW 'MOMENTARY-MENU
'LABEL Select Display
'BORDERS 3
'ITEM-LIST '("Calculate Loads" "Redisplay Loads")
FUNCTIONS CALLED: DISPLAY-CPG-SINGLE-TASKS REDISPLAY-CPG-SINGLE-TASKS

NAME: SELECT-PILOT-DISPLAY-TYPE-COMBINED
DESCRIPTION: creates the pop-up menu to select the pilot combined task display
INITIAL-VALUE: :MAKE-WINDOW 'MOMENTARY-MENU
'LABEL Select Display
'BORDERS 3
'ITEM-LIST '("Calculate Loads" "Redisplay Loads")
FUNCTIONS CALLED: check-file-for-active-pilot-tasks redisplay-pilot-task-combinations

NAME: SELECT-CPG-DISPLAY-TYPE-COMBINED
DESCRIPTION: creates the pop-up menu to select the cpg combined task display
INITIAL-VALUE: :MAKE-WINDOW 'MOMENTARY-MENU
'LABEL Select Display
'BORDERS 3
'ITEM-LIST '("Calculate Loads" "Redisplay Loads")
FUNCTIONS CALLED: CHECK-FILE-FOR-ACTIVE-CPG-TASKS REDISPLAY-CPG-TASK-COMBINATIONS

NAME: RESET-TLM-DISPLAY
DESCRIPTION: sends a message to the defparameter select-reset-tlm-display that calls the pop-up menu that resets the *tlm-process* to the initial state
CALLING FUNCTIONS: MIDAS-TLM-DISPLAY

NAME: PILOT-TASK-DISPLAY
DESCRIPTION: sends a message to the defparameter select-pilot-task-display calls the pop-up menu that selects either the pilot's single or combined task display
CALLING FUNCTIONS: MIDAS-TLM-DISPLAY

NAME: PILOT-DISPLAY-TYPE-SINGLE
DESCRIPTION: sends a message to the defparameter select-pilot-display-type-single that calls the pop-up menu that selects pilot single task display

NAME: PILOT-DISPLAY-TYPE-COMBINED
DESCRIPTION: sends a message to the defparameter select-pilot-display-type-combined that calls the pop-up menu that selects pilot combined task display

NAME: CPG-TASK-DISPLAY
DESCRIPTION: sends a message to the defparameter select-cpg-task-display that calls the pop-up menu that selects either the cpg's single or combined task display
CALLING FUNCTIONS: MIDAS-TLM-DISPLAY

NAME: CPG-DISPLAY-TYPE-SINGLE
DESCRIPTION: sends a message to the defparameter select-cpg-display-type-single that calls the pop-up menu that selects cpg single task display

NAME: CPG-DISPLAY-TYPE-COMBINED
DESCRIPTION: sends a message to the defparameter select-cpg-display-type-combined that calls the pop-up menu that selects cpg combined task display

NAME: CHECK-FILE-FOR-ACTIVE-PILOT-TASKS
DESCRIPTION: initializes the active pilot task file and starts the process "Combined Pilot Task Loads"
FUNCTIONS CALLED: SCL:PROCESS-RUN-FUNCTION

NAME: CHECK-FILE-FOR-ACTIVE-CPG-TASKS
DESCRIPTION: initialize active cpg task file and starts the process "Combined CPG Task Loads"
FUNCTIONS CALLED: SCL:PROCESS-RUN-FUNCTION

NAME: DISPLAY-PILOT-SINGLE-TASKS
DESCRIPTION: calls all the functions necessary to display the pilot's single task load values and sets the configuration of *hwl* to 'configl
FUNCTIONS CALLED: DISPLAY-PILOT-SINGLE-TASK-LOADS

NAME: REDISPLAY-PILOT-SINGLE-TASKS
DESCRIPTION: redisplays the pilot's single task load values and sets the configuration of *hwl* to 'configl

NAME: DISPLAY-CPG-SINGLE-TASKS
DESCRIPTION: calls all the functions necessary to display the cpg's single task load values and sets the configuration of *hwl* to 'config2
FUNCTIONS CALLED: DISPLAY-CPG-SINGLE-TASK-LOADS

NAME: REDISPLAY-CPG-SINGLE-TASKS
DESCRIPTION: redisplay's the cpg's single task load valuesand sets the configuration of *hwl* to 'config2

NAME: PILOT-TASK-COMBINATIONS
DESCRIPTION: calls all the functions necessary to display the pilot's combined task load values and sets the configuration of *hwl* to 'config3
FUNCTIONS CALLED: GET-AND-DISPLAY-PILOT-COMBINED-TASK-LOADS
CALLING FUNCTIONS: START-PILOT-PROCESS

NAME: REDISPLAY-PILOT-TASK-COMBINATIONS
DESCRIPTION: redisplay's the pilot's combined task load values and sets the configuration of *hwl* to 'config3

NAME: CPG-TASK-COMBINATIONS
DESCRIPTION: calls all the functions necessary to display the cpg's combined task load values and sets the configuration of *hwl* to 'config4
FUNCTIONS CALLED: GET-AND-DISPLAY-CPG-COMBINED-TASK-LOADS
CALLING FUNCTIONS: START-CPG-PROCESS

NAME: REDISPLAY-CPG-TASK-COMBINATIONS
DESCRIPTION: redisplay's the cpg's combined task load values and sets the configuration of *hwl* to 'config4

NAME: START-PILOT-PROCESS

DESCRIPTION: starts the process to get the list of pilot active tasks from the active task list file
FUNCTIONS CALLED: PILOT-TASK-COMBINATIONS

NAME: START-CPG-PROCESS
DESCRIPTION: starts the process to get the list of cpg active tasks from the active task list file
FUNCTIONS CALLED: CPG-TASK-COMBINATIONS

NAME: CLEAR-PILOT-PANES
DESCRIPTION: this function sends a :clear-history message to clear the pilots task load and classification data from window-panes in task-load-window-frame: 'config1 and 'config3

NAME: CLEAR-CPG-PANES
DESCRIPTION: this function sends a :clear-history message to clear the cpgs task load and classification data from window-panes in task-load-window-frame: 'config2 and 'config4

5.2.4 FG4: The task-load formatting functional group

FILE: SQUID:>stav>midas-tlm>tlm-load-lists.lisp

DESCRIPTION: This file contains the functions and flavors necessary to construct the load index and element classification lists
FUNCTIONS DEFINED: GENERATE-MOTOR-ELEMENT-LIST GENERATE-COGNITIVE-ELEMENT-LIST GENERATE-AUDITORY-ELEMENT-LIST GENERATE-VISUAL-ELEMENT-LIST GENERATE-ELEMENT-LIST SUM-VALS-ACROSS-LISTS SET-INST-VARS-FROM-INDICES ASSIGN-INST-VARS-ELEMENTS-AND-LOADS SET-INST-VARS-FROM-LIST
MACROS DEFINED: PUSH-END
FLAVORS DEFINED: LOAD-VALS MOTOR-COMPONENT COGNITIVE-COMPONENT AUDITORY-COMPONENT VISUAL-COMPONENT TASK

NAME OF THE MACRO: PUSH-END
DESCRIPTION: adds list to the end of another list
INPUTS: VALUE VALUE-LIST-PLACE

NAME: SET-INST-VARS-FROM-LIST
DESCRIPTION: Utility function for setting instance variables.
INPUTS: INSTANCE LIST-OF-VALS
CALLING FUNCTIONS: ASSIGN-INST-VARS-ELEMENTS-AND-LOADS

NAME: TASK
DESCRIPTION: defines a task as an instance of each component, each with two instance variables that can be returned as two lists
INSTANCE VARIABLES: VISUAL V1 INDEX-V NOTV1 AUDITORY A1 INDEX-A NOTA1 COGNITIVE C1 INDEX-C NOTC1 MOTOR M1 INDEX-M NOTM1
METHODS: UPDATE-MOTOR UPDATE-COGNITIVE UPDATE-AUDITORY UPDATE-VISUAL DO-MTR-MAGIC DO-COG-MAGIC DO-AUD-MAGIC DO-VIS-MAGIC

NAME: VISUAL-COMPONENT
DESCRIPTION: define the elements of the visual dimension and sets them to nil
INSTANCE VARIABLES: NEAR FAR SCAN FIXATE INTEGRAL SEPARABLE OBJECTS FEATURES SALIENT MASKED STATIC DYNAMIC

NAME: AUDITORY-COMPONENT
DESCRIPTION: define the elements of the auditory dimension and sets them to nil
INSTANCE VARIABLES: ORIENT DISCRIMINATE SYGNAL SPEECH SALIENT MASKED

NAME: COGNITIVE-COMPONENT
DESCRIPTION: define the elements of the cognitive dimension and sets them to nil
INSTANCE VARIABLES: DIRECT TRANSFORM SINGLE MULTIPLE VERBAL SPATIAL PLANNED UNPLANNED

NAME: MOTOR-COMPONENT
DESCRIPTION: define the elements of the motor dimension and sets them to nil
INSTANCE VARIABLES: VERBAL SPATIAL NEAR FAR DISCRETE CONTINUOUS GROSS FINE MOUTH HEAD EYE HAND FEET FINGER RIGHT LEFT BOTH

NAME: LOAD-VALS
DESCRIPTION: defines the load valuesfor the within components and sets them to nil
INSTANCE VARIABLES: WITHIN-VISUAL WITHIN-AUDITORY WITHIN-COGNITIVE WITHIN-MOTOR VISUAL-AUDITORY AUDITORY-COGNITIVE VISUAL-COGNITIVE COGNITIVE-MOTOR VISUAL AUDITORY COGNITIVE MOTOR

NAME: ASSIGN-INST-VARS-ELEMENTS-AND-LOADS
DESCRIPTION: this function takes element list values and assigns them to their respective instance variables
INPUTS: ELEMENT-LIST LOAD-LIST
FUNCTIONS CALLED: SET-INST-VARS-FROM-LIST
CALLING FUNCTIONS: MAKE-TASK-LOAD-LIST SHOW-TASK-LOAD DISPLAY-SINGLE-TASK-LOADS DISPLAY-COMBINED-TASK-LOADS

NAME: SET-INST-VARS-FROM-INDICES
DESCRIPTION: this assigns the indices to the instance variables that the function generate load requires
INPUTS: TASK-INDEX-LIST
OUTPUTS: TASK-OBJECT
CALLING FUNCTIONS: SHOW-TASK-LOAD DISPLAY-SINGLE-TASK-LOADS DISPLAY-COMBINED-TASK-LOADS MULTIPLE-TASK-LOAD SCHEDULER-LOAD-LIST

NAME: SUM-VALS-ACROSS-LISTS
DESCRIPTION: sums across each element in each list for each task in a task combination and returns a list of the number of times each element was selected
INPUTS: LIST-OF-ELEMENT-VALUES
OUTPUTS: LIST-OF-ELEMENT-VALUES
CALLING FUNCTIONS: SHOW-TASK-LOAD DISPLAY-COMBINED-TASK-LOADS MULTIPLE-TASK-LOAD

NAME: GENERATE-ELEMENT-LIST

DESCRIPTION: calls the functions for each task that generate lists of selected elements for each component and returns a list of the component lists of selected elements
INPUTS: TASK
FUNCTIONS CALLED: (READ-INSTANCE-VARIABLE INDEX-V TASK INDEX-V) GENERATE-VISUAL-ELEMENT-LIST (READ-INSTANCE-VARIABLE INDEX-A TASK INDEX-A) GENERATE-AUDITORY-ELEMENT-LIST (READ-INSTANCE-VARIABLE INDEX-C TASK INDEX-C) GENERATE-COGNITIVE-ELEMENT-LIST (READ-INSTANCE-VARIABLE INDEX-M TASK INDEX-M) GENERATE-MOTOR-ELEMENT-LIST
CALLING FUNCTIONS: SHOW-TASK-LOAD DISPLAY-SINGLE-TASK-LOADS DISPLAY-COMBINED-TASK-LOADS MULTIPLE-TASK-LOAD

NAME: GENERATE-VISUAL-ELEMENT-LIST
DESCRIPTION: generates a list of the elements for the visual component that were selected for a task from the list of indices used to access the visual matrix
INPUTS: COMPONENT-INDICES
OUTPUTS: VISUAL-ELEMENTS
CALLING FUNCTIONS: GENERATE-ELEMENT-LIST

NAME: GENERATE-AUDITORY-ELEMENT-LIST
DESCRIPTION: generates a list of the elements for the auditory component that were selected for a task from the list of indices used to access the auditory matrix
INPUTS: COMPONENT-INDICES
OUTPUTS: AUDITORY-ELEMENTS
CALLING FUNCTIONS: GENERATE-ELEMENT-LIST

NAME: GENERATE-COGNITIVE-ELEMENT-LIST
DESCRIPTION: generates a list of the elements for the cognitive component that were selected for a task from the list of indices used to access the cognitive matrix
INPUTS: COMPONENT-INDICES
OUTPUTS: COGNITIVE-ELEMENTS
CALLING FUNCTIONS: GENERATE-ELEMENT-LIST

NAME: GENERATE-MOTOR-ELEMENT-LIST
DESCRIPTION: generates a list of the elements for the motor component that were selected for a task from the list of indices used to access the motor matrix
INPUTS: COMPONENT-INDICES
OUTPUTS: MOTOR-ELEMENTS
CALLING FUNCTIONS: GENERATE-ELEMENT-LIST

FILE: SQUID:>stav>midas-tlm>tlm-display-loads.lisp

DESCRIPTION: This file contains the functions that take lists of task element and load values and displays them in the proper windowpanes
VARIABLES DEFINED: *TASK-NUMBERS*
FUNCTIONS DEFINED: MULTIPLE-TASK-LOAD DISPLAY-COMBINED-TASK-LOADS GET-AND-DISPLAY-CPG-COMBINED-TASK-LOADS GET-AND-DISPLAY-PILOT-COMBINED-TASK-LOADS DISPLAY-CPG-SINGLE-TASK-LOADS DISPLAY-PILOT-SINGLE-TASK-LOADS DISPLAY-SINGLE-TASK-LOADS SHOW-TASK-LOAD SCHEDULER-LOAD-LIST MAKE-TASK-LOAD-LIST

NAME: *TASK-NUMBERS*
INITIAL-VALUE: nil

DESCRIPTION:

NAME: MAKE-TASK-LOAD-LIST
DESCRIPTION: This Function generates instances of tasks, puts them in a list and passes them to the functions that generate the task classifications and either evaluates each task or each task combination and then displays the load and element values
INPUTS: NUMBER-OF-TASKS &OPTIONAL LIST-OF-TASK-OBJECTS
OUTPUTS:
FUNCTIONS CALLED: MULTIPLE-TASK-LOAD ASSIGN-INST-VARS-ELEMENTS-AND-LOADS (METHOD PRINT-TO-TASK-ELEMENT-AND-LOAD-PANE TASK-LOAD-WINDOW-FRAME) (METHOD PRINT-TO-TASKCOMBO-ELEMENT-AND-LOAD-PANE TASK-LOAD-WINDOW-FRAME)

NAME: SCHEDULER-LOAD-LIST
DESCRIPTION: Renuka uses this function to generate her list of loading values for combined tasks
INPUTS: LISTS-OF-TASK-INDICES
FUNCTIONS CALLED: SET-INST-VARS-FROM-INDICES GENERATE-LOAD

NAME: SHOW-TASK-LOAD
DESCRIPTION: this is for the operator scheduler model to use to calculate and display single and combined task load data
INPUTS: LISTS-OF-TASK-INDICES
FUNCTIONS CALLED: SET-INST-VARS-FROM-INDICES GENERATE-ELEMENT-LIST GENERATE-LOAD SUM-VALS-ACROSS-LISTS ASSIGN-INST-VARS-ELEMENTS-AND-LOADS (METHOD PRINT-TO-TASK-ELEMENT-AND-LOAD-PANE TASK-LOAD-WINDOW-FRAME)

NAME: DISPLAY-SINGLE-TASK-LOADS
DESCRIPTION: takes a list of index lists, calculates their load and element values and displays the load and element values for each single task
INPUTS: LIST-OF-TASK-NAME-LISTS LISTS-OF-TASK-INDICES
FUNCTIONS CALLED: SET-INST-VARS-FROM-INDICES GENERATE-ELEMENT-LIST GENERATE-LOAD ASSIGN-INST-VARS-ELEMENTS-AND-LOADS
METHODS CALLED: (METHOD PRINT-TO-TASK-ELEMENT-AND-LOAD-PANE TASK-LOAD-WINDOW-FRAME)
CALLING FUNCTIONS: DISPLAY-PILOT-SINGLE-TASK-LOADS DISPLAY-CPG-SINGLE-TASK-LOADS

NAME: DISPLAY-PILOT-SINGLE-TASK LOADS
DESCRIPTION: gets all the pilot single task indices from the task-load-data-hash-table and calculates their load and element lists and then outputs them to the pilot single task load pane
FUNCTIONS CALLED: GET-PILOT-TASK-INDICES-FROM-TABLE DISPLAY-SINGLE-TASK-LOADS
CALLING FUNCTIONS: DISPLAY-PILOT-SINGLE-TASKS

NAME: DISPLAY-CPG-SINGLE-TASK-LOADS
DESCRIPTION: gets all the cpg single task indices from the task-load-data-hash-table and calculates their load and element lists and then outputs them to the cpg single task load pane
FUNCTIONS CALLED: GET-CPG-TASK-INDICES-FROM-TABLE DISPLAY-SINGLE-TASK-LOADS
CALLING FUNCTIONS: DISPLAY-CPG-SINGLE-TASKS

NAME: GET-AND-DISPLAY-PILOT-COMBINED-TASK-LOADS
DESCRIPTION: gets all the pilot single task indices from the task-load-data-hash-table and calculates their combined load and element lists and then outputs the pilot combined task values to the pilot task combination load pane
FUNCTIONS CALLED: UPDATE-SYM-TO-JILL GET-PILOT-ACTIVE-TASKS DISPLAY-COMBINED-TASK-LOADS
CALLING FUNCTIONS: PILOT-TASK-COMBINATIONS

NAME: GET-AND-DISPLAY-CPG-COMBINED-TASK-LOADS
DESCRIPTION: gets all the cpg single task indices from the task-load-data-hash-table and calculates their combined load and element lists and then outputs the cpg combined task values to the cpg task combination load pane
FUNCTIONS CALLED: UPDATE-SYM-TO-JILL GET-CPG-ACTIVE-TASKS DISPLAY-COMBINED-TASK-LOADS
CALLING FUNCTIONS: CPG-TASK-COMBINATIONS

NAME: DISPLAY-COMBINED-TASK-LOADS
DESCRIPTION: takes a list of index lists, calculates their load and element values and displays the load and element values for each single task
INPUTS: CURRENT-TIME LISTS-OF-TASK-INDICES LIST-OF-TASK-NAME-LISTS
FUNCTIONS CALLED: SET-INST-VARS-FROM-INDICES GENERATE-ELEMENT-LIST SUM-VALS-ACROSS-LISTS GENERATE-LOAD ASSIGN-INST-VARS-ELEMENTS-AND-LOADS
METHODS CALLED: (METHOD PRINT-TO-TASKCOMBO-ELEMENT-AND-LOAD-PANE TASK-LOAD-WINDOW-FRAME)
CALLING FUNCTIONS: GET-AND-DISPLAY-PILOT-COMBINED-TASK-LOADS GET-AND-DISPLAY-CPG-COMBINED-TASK-LOADS

NAME: MULTIPLE-TASK-LOAD
DESCRIPTION This Function generates instances of tasks, puts them in a list and passes them to the functions that generate the task classifications and evaluate each task and task combination and returns two lists: element and load values, that are used to initialize the vacm component and load instance variables that get displayed
INPUTS: NUMBER-OF-TASKS &OPTIONAL LISTS-OF-TASK-INDICES
FUNCTIONS CALLED: CREATE-TASK TASK-COMPONENTS GENERATE-ELEMENT-LIST GENERATE-LOAD SET-INST-VARS-FROM-INDICES SUM-VALS-ACROSS-LISTS
CALLING FUNCTIONS: MAKE-TASK-LOAD-LIST

5.2.5 FG5: The midas-interface functional group

FILE: SQUID:>stav>midas-tlm>test-midas-tlm.lisp

DESCRIPTION: This file contains one function used solely for testing the tlm and for hard-coding the tasks used in the simulation
FUNCTIONS DEFINED: GENERATE-TASK-LIST

NAME: GENERATE-TASK-LIST
DESCRIPTION: this function is for testing the tlm and for demos. It creates a list of active tasks that are evaluated and displayed.
OUTPUTS: TIME-AND-TASK-LIST

FILE: SQUID:>stav>midas-tlm>sym-to-jill.lisp

DESCRIPTION: This file contains the parameter whose value is the list of tasks that the TLM processes
PARAMETERS DEFINED: *TASK-COMBINATION-LIST*

NAME: *TASK-COMBINATION-LIST*
DESCRIPTION: assigns a list of tasks to *task-combination-list* that acts as input to the TLM
INITIAL-VALUE: GENERATE-TASK-LIST

FILE: SQUID:>stav>midas-tlm>current-sym-to-jill.lisp

DESCRIPTION: This file is to be used to recieve the active task list update every ten ticks from the mission decomposition module. However, it is not in use because it was never linked to up and tested.

FILE: SQUID:>stav>midas-tlm>sym-interface.lisp

Interface Files are specified in "B:>Midas>globals>interface-globals.lisp"

AUTHOR: Jerry Murray
DESCRIPTION: This file contains the functions necessary to link the task loading model and the task decomposition model so that the simulated tasks can be evaluated by the task loading model. This file was created by Jerry Murray and modified by Jerry Murray.
VARIABLES DEFINED: *INTERFACE-FILES* *TASK-COMBINATION-LIST*
FUNCTIONS DEFINED: UPDATE-Z-TO-JILL UPDATE-JILL-TO-Z UPDATE-SYM-TO-Z UPDATE-Z-TO-SYM UPDATE-SYM-TO-JILL UPDATE-JILL-TO-SYM INIT-Z-TO-JILL INIT-JILL-TO-Z INIT-SYM-TO-Z INIT-Z-TO-SYM INIT-SYM-TO-JILL INIT-JILL-TO-SYM

NAME: *INTERFACE-FILES*
DESCRIPTION: the value of this variable is set to the different files necessary to link the tlm and mission decomp model
INITIAL-VALUE:
(JILL-TO-SYM B:>Midas>interface>files>Jill-to-sym.lisp
SYM-TO-JILL squid:>stav>midas-tlm>sym-to-Jill.lisp
Z-TO-SYM B:>Midas>interface>files>Z-to-sym.lisp
SYM-TO-Z B:>Midas>interface>files>sym-to-Z.lisp
JILL-TO-Z B:>Midas>interface>files>Jill-to-Z.lisp
Z-TO-JILL B:>Midas>interface>files>Z-to-Jill.lisp)

NAME: INIT-SYM-TO-JILL
DESCRIPTION: This function stuffs the value of *Interface-Files* :current-sym-to-Jill into *Interface-Files* :sym-to-Jill
CALLING FUNCTIONS: MIDAS-TLM-DISPLAY

NAME: UPDATE-SYM-TO-JILL
DESCRIPTION: this function reads *Interface-Files* for the value of :current-sym-to-Jill and checks :sym-to-Jill to see if its the current version, if not it then stuffs *Interface-Files* :current-sym-to-Jill into *Interface-Files* :sym-to-Jill and returns *task-

combination-list* which is the value of *Interface-Files* :sym-to-Jill as a hard-coated value in place of a dynamic simulation value
CALLING FUNCTIONS: GET-AND-DISPLAY-PILOT-COMBINED-TASK-LOADS GET-AND-DISPLAY-CPG-COMBINED-TASK-LOADS

NAME: INIT-JILL-TO-SYM
DESCRIPTION: Not used by the task load model

NAME: INIT-Z-TO-SYM
DESCRIPTION: Not used by the task load model

NAME: INIT-SYM-TO-Z
DESCRIPTION: Not used by the task load model

NAME: INIT-JILL-TO-Z
DESCRIPTION: Not used by the task load model

NAME: INIT-Z-TO-JILL
DESCRIPTION: Not used by the task load model

NAME: UPDATE-JILL-TO-SYM
DESCRIPTION: Not used by the task load model

NAME: UPDATE-Z-TO-SYM
DESCRIPTION: Not used by the task load model

NAME: UPDATE-SYM-TO-Z
DESCRIPTION: Not used by the task load model

NAME: UPDATE-JILL-TO-Z
DESCRIPTION: Not used by the task load model

NAME: UPDATE-Z-TO-JILL
DESCRIPTION: Not used by the task load model

5.2.6 FG6: The system-initialization functional group

FILE: SQUID:>stav>midas-tlm>tlm-files-init.lisp

DESCRIPTION: This file initializes and opens the task load window frame object, creates the hash-tables of the task name and index associations, creates the tlm process and binds it to the H key, resets the displays
VARIABLES DEFINED: *TLM-PROCESS* *HW1*
FUNCTIONS DEFINED: MIDAS-TLM-DISPLAY RESET-TLM

NAME: *HW1*
DESCRIPTION: This function sets the task-load-window-frame object - *hw1* - to an arbitrary value
INITIAL-VALUE: tv:make-window 'task-load-window-frame
```
                              :save-bits nil
                              :expose-p nil
```

NAME: PILOT-TASK-LOAD-DATA-TABLE
DESCRIPTION: This function call creates the pilot-task-load-data-table hash table

NAME: CPG-TASK-LOAD-DATA-TABLE
DESCRIPTION: This function call creates the cpg-task-load-data-table hash table

NAME: CPG-SHORT-NAMES-TABLE
DESCRIPTION: This function call creates the init-short-names-table hash table

NAME: PILOT-SHORT-NAMES-TABLE
DESCRIPTION: This function call creates the init-short-names-table hash table

NAME: MIDAS-TLM-DISPLAY
DESCRIPTION: This function opens the midas-tlm window and starts the command menu and read file processes
FUNCTIONS CALLED: INIT-SYM-TO-JILL PILOT-TASK-DISPLAY PG-TASK-DISPLAYRESET-TLM-DISPLAY
METHODS CALLED: (METHOD PRINT-THE-INITIAL-DISPLAY TASK-LOAD-WINDOW-FRAME)

NAME: *TLM-PROCESS*
DESCRIPTION: This sets the variable *TLM PROCESS* to the midas-tlm process that runs off of the "H" key when loaded
```
INITIAL-VALUE: (UNLESS (AND (VARIABLE-BOUNDP *TLM-PROCESS*)
                             (TYPEP *TLM-PROCESS* 'PROCESS)
                             (STRING-EQUAL (PROCESS-NAME *TLM-PROCESS*)
                                           Task Load Model))
                        (PROCESS-RUN-FUNCTION Task Load Model
                                           'MIDAS-TLM-DISPLAY))
```

NAME: tv:add-select-key
ARGUMENTS: #\H 'task-load-window-frame "Task Load Window Frame"
INITIAL VALUE: nil
DESCRIPTION: This function calls the midas-tlm as a process that runs off of the "H" key when loaded

NAME: RESET-TLM
DESCRIPTION: this function resets the variable *TLM-PROCESS* to the midas-tlm process and the tlm interface to the initial configuration: no loads or tasks are displayed
FUNCTIONS CALLED: CLEAR-PILOT-PANES CLEAR-CPG-PANES

5.2.7 FG7: The tlm-system functional group

FILE: SQUID:>stav>midas-tlm>midas-tlm.lisp

DESCRIPTION: This file creates the system - MIDAS-TLM.

SYSTEM DEFINED: MIDAS-TLM
```
(defsystem MIDAS-TLM
  (:pretty-name "MIDAS Task Loading Model"
   :short-name "tlm"
   :default-pathname "midas-tlm:midas-tlm;")
 (:serial
  "task-load-interface-frame"
  "tlm-load-lists"
  (:parallel
"tlm-editor"
"tlm-calculator"
"tlm-task-load-data"
"tlm-display-loads"
"task-names"
"select-task-display"
"test-midas-tlm"
"sym-interface"
"current-sym-to-jill"
"sym-to-jill")
"tlm-files-init"))
```

6.0 USER'S GUIDE

This section describes the MIDAS Task Loading Model sufficiently to enable the designer of aerospace cockpit flight equipment to boot the MIDAS-TLM system on a Symbolics machine and provide the required input.

6.1 OVERVIEW OF PURPOSE AND FUNCTIONS

The design team members interact with the TLM via mouse sensitive items that display momentary pop-up menus on the monitor of a symbolics 3640 machine. These menus allow the user to select the task load data for each crew member, for single-tasks or combined-tasks, and whether the task load data should be re-calculated or re-displayed.

The loading values of the tasks and task combinations are displayed along with the taxonomic subset classifying the task (s), the name(s) of the task(s), and the time corresponding to when the task performance was simulated. The designer can scroll the display horizontally and vertically to display the complete timeline with the complete task classification and loading profiles. Pointing at a scroll bar in the left margin scrolls vertically through the task load and task classification data. Pointing at a scroll bar in the bottom margin displays the task load and classification data at each time or tick increment in the simulation.

The designer has the option to stop the MIDAS-TLM process at any time, clear the displays and restart the process.

The designer is currently restricted to evaluating tasks that have been manually classified and coded as a hash table. This is a pre-simulation requirement. All the tasks generated by the Symbolic Operator Model must be manually classified and coded before a simulation, but once the task names and indexes are coded as hash tables, only the names of the tasks need to be passed to the TLM for task evaluation.

6.2 INSTALLATION AND INITIALIZATION

To run the MIDAS-TLM on a Symbolics machine, the user boots the system by typing the command **Boot** on the command line if the machine has not already been booted. The MIDAS-TLM system is now ready to be loaded into the machine.

The MIDAS-TLM is installed as a system in the directory **sys:site;** on the file server host **Barracuda**. The system's files reside in the directory **stav;midas-tlm** on the host **Squid**. To install the system, the user logs on to squid and compiles the midas-tlm system from the dynamic Lisp listener window by typing the command

Compile System Midas-Tlm.

This command compiles and loads the system. To select the MIDAS-TLM display, the user types the keys **Select-H.**

The TLM currently accesses hash tables that provide the task data to calculate the task loads. These hash tables must be created before a simulation can occur. Currently, the only means to create these tables is to edit two files, enter the necessary data into the tables and then compile the files. To edit the files, type the command

Edit System Midas-Tlm

This command loads all of the MIDAS-TLM system's files into the edit buffer, from which the appropriate files can be selected for editing.

6.3 STARTUP AND TERMINATION

The MIDAS-TLM system operates in two modes: pre-simulation and runtime. During pre-simulation, the user selects the crew-member to be analyzed by clicking on either the "**Pilot**" or "**Copilot/Gunner**" mouse-sensitive label on the display. This brings up a momentary menu with the labels "**Single**", "**Combined**" and "**Clear Display**". The user clicks the mouse on the "**Single**" label which brings up a second momentary menu with the labels "**Calculate**" and "**Redisplay**". Clicking on these labels calculates the loads for the active, single tasks, or redisplays them if they have already been calculated. The user can scroll through the task loading values and task classifications via scroll bars on the bottom and left-hand sides of the display.

During runtime simulation, the list of active tasks are written to a file at each tick-increment of the simulation. Once the TLM-process starts, the TLM accesses the file at each tick and automatically calculates and displays the loads and classifications for the combined tasks in the active task list.

The user starts the TLM-process by clicking the mouse on either the "**Pilot**" or "**Copilot/Gunner**" mouse-sensitive label on the display. This brings up a momentary menu with the labels "**Single**", "**Combined**" and "**Clear Display**". The user clicks the mouse on the "**Combined**" label which brings up a second momentary menu with the labels "**Calculate**"and "**Redisplay**". Clicking on these labels calculates the loads for the active, combined tasks, or redisplays them if they have already been calculated. The user can scroll through the task loading values and task classifications via scroll bars on the bottom and left-hand sides of the display.

Once the TLM-process has begun, it continues until the simulation stops, providing a complete history of the task loads and task classifications for the single and combined tasks. These can be sequentially displayed and scrolled through for post-simulation analyses.

The user can terminate the TLM-process at anytime by clicking the mouse on the "**Crew Selection Menu**" label on the display. This stops the TLM-process and re-initializes the TLM, at which point the user can begin again.

The user follows the same procedures to run the TLM during each simulation.

6.4 FUNCTIONS AND THEIR OPERATION

There are four files containing the functions that control the interface and contain the task hash-tables. **Squid:>stav>midas-tlm>select-task-display** and **Squid:>stav>midas-tlm>tlm-files-init** control the interface. These two files do not need to be edited to run the TLM during simulations. **Squid:>stav>midas-tlm>tlm-task-load-data** and **Squid:>stav>midas-tlm>task-names** contain the task hash tables. These two files need to be edited to run the TLM during simulations.

Squid:>stav>midas-tlm>tlm-files-init contains two variables and two functions. The values of the two variables are initialized to an instance of the task-load-window-frame and the process **"midas-tlm"**. The two functions start the command menu and read file processes, and call functions to clear or reset the displays.

Squid:>stav>midas-tlm>select-task-display contains six defparameters set to instances of the pop-up menus, and eighteen functions that call the functions that evaluate and display the loads and task classifications for single and combined tasks for both the pilot and copilot-gunner. Two separate functions clear the task and load classification data from the window-panes.

Squid:>stav>midas-tlm>tlm-task-load-data contains two variables set to instances of the pilot and copilot/gunner task name and index hash tables. Six functions get the pilot and copilot single and combined active task lists from the task decomposition model's interface file **Squid:>stav>midas-tlm>sym-interface**, and access' the task-name tables, and the task-name-and-index tables. Two functions **"pilot-task-load-data-table"** and **"cpg-task-load-data-table"** create the task-long-name-and-index tables for the pilot and copilot respectively. These two functions must be edited before a simulation. Using the syntax provide by the current tables, the user must enter the new task name and classification data into these tables and compile the buffer.

Squid:>stav>midas-tlm>task-names creates the tables of task-long-names that are used to access the index lists in the task-name-and-index tables, and access the task-short-names in the task-short-name table that are used in the display (The long names are too long for the display window). Two variables are set to the pilot and cpg long names with their respective short names. Two defparameters are set to the pilot and cpg long names used to access the short names stored in the two variables, and access the index lists in the task-name-and-index tables. Two functions retrieve the short names stored in the two variables, and two functions create the instances of the long-name/short-name hash table for the pilot and cpg task names and store them in their respective variables. These last two functions, **"pilot-short-names-table"** and **"cpg-short-names-table"** must be edited, replacing the current names with the ones that will be used in the simulation. The two defparameters must also be edited, replacing the current names with the ones that will be used in the simulation. This buffer must also be compiled.

Once the two files have been edited, the file **Squid:>stav>midas-tlm>tlm-files-init** must be recompiled to create instances of the new hash tables. The midas-tlm process is now ready for simulation.

6.5 ERROR AND WARNING MESSAGES

The possible error messages that may occur would probably be associated with either output holds on the display window-panes, or associated with task names and classifications that are referenced incorrectly as a result of errors in the task hash tables or that aren't in the hash tables.

6.6 RECOVERY STEPS

Recovery from any error states is accomplished by clicking on the **"RESET-TLM"** menu item in the pop-up display associated with the **"Crew Selection Menu"** display label.

7.0 ABBREVIATIONS AND ACRONYMS

AC_{bw}	Auditory-Cognitive between matrix
A_{cw}	Auditory within matrix
CAD	Computer Aided Design
C_b	Conflict Value for between matrices
CCSCI	CalculatorComponent Software Configuration Item
C_{cw}	Cognitive within matrix
CM_{bw}	Cognitive-Motor between matrix
CPU	Central Processing Unit
CSCI	Component Software Configuration Item
CTA	Cognitive Task Analysis
C_w	Conflict Value for within matrices
FG1:	Functional Group 1: The task editor and data-input
FG2:	Functional Group 2: task-load calculator
FG3:	Functional Group 3: display windows and menus
FG4:	Functional Group 4: task-load formatting
FG5:	Functional Group 5: midas-interface
FG6:	Functional Group 6: system-initialization
FG7:	Functional Group 7: tlm-system
I/O	Input/Output
L_a	Auditory load value
L_c	Cognitive load value
Lisp	List processor (lots of silly irritating parentheses)
L_m	Motor load value
LMFS	Lisp Machine File System
L_v	Visual load value
M_{cw}	Motor load value
MIDAS	Man-Machine Design and Analysis System
SCI	Software Configuration Item
SSCI	Scheduler Software Configuration Item
TACSCI	Task Adjustor Component Software Configuration Item
TDSCI	Task Decomposition Software Configuration Item
TECSCI	Task Editor Component Software Configuration Item
TLM	Task Load Model
TLMSC	Task Load Model Software Component
TLMSCI	Task Load Model Software Configuration Item
UICSCI	User Interface Component Software Configuration Item
VA_{bw}	Visual-Auditory between-matrix
VACM	Visual Auditory Cognitive Motor
VACP	Visual Auditory Cognitive PsychoMotor
VC_{bw}	Visual-Cognitive between-matrix
V_{cw}	Visual within-matrix

8.0 GLOSSARY

Abstraction Hierarchy A hierarchical set of invariant qualities that are common to the the human information processing systems capability to represent and manipulate information symbolically.
Additional-Cost An unknown function or constant that accounts for the obtained increases in the magnitude of workload predicted by an averaging model.
Attention A mental control mechanism that guides, focuses, or elaborates the acquisition and processing of information.
Auditory A modality of perception involving aural stimuli.
Auditory-Cognitive The interaction of aural perceiving and central processing mechanisms.
Auditory-Visual The interaction of aural perceiving and visual perceiving.
Averaging Model A model that predicts retrospective workload ratings by averaging the difficulty of the events or stimuli that are experienced.
Behavioral State A description of the attributes and values that constitute a specific pattern of human performance.
Best-Fitting Model A model that best describes the obtained patterns of human performance or behavioral states.
Between-Matrix A two dimensional matrix of values that represents the demands incurred on the human information processing system from the interaction of information processing mechanisms in different stages.
Bottom-Up Model A human performance model that is dictated by a detailed set of primitive elements of human behavior.
Cognitive The central processing stage composed of at least the attentional, transformational and memory mechanisms and processes that act on perceived information, and which prepare the information processing system to generate a response to stimuli if necessary.
Cognitive Task Analysis An analytic technique to explicate the interactions among the current and desired states of the world and agent, and the representations available to the agent that are necessary to model human-system interactions.
Cognitive-Motor The interaction of central processing mechanisms and motor response mechanisms.
Component Software Configuration Item A component of a software architectural unit/item used to implement a specific model or tool within MIDAS.
Conflict Matrix A two dimensional matrix of values that represents the demands and conflicts among the structural and procedural psychological attributes that describe a task.
Conflict Value A specific value that represents the interaction between two structural or procedural psychological attributes classifying a task.
Dimension A top level set of structural or procedural psychological attributes that corresponds to a specific stage of information processing. This is the level that load values are assigned to a task.
Element A bottom level structural or procedural psychological attribute that is used to classify a task.
Event Based A simulation that progresses according to changes in state variables that represent specific events that occur in the environment or behavior of the operator.
Genera 7.2 The symbolics Lisp operating system.
Human Information Processing The human mental activities and structures that represent and manipulate information symbolically, and which enable humans to perceive and respond to changes in external (environmental) or internal (mental) states.
Human Performance Model A quantitative (analytic or computer-based) representation or description of all or parts of human operators or maintainers of complex, dynamic systems.

Invariant Property A characteristic of the information processing system that does not change as a function of the information extracted from the perceptual array.
Load Balancing Strategy A behavioral change in an operator as a function of the imposed visual, auditory, cognitive and motor loads imposed by performing a task. The effect of the behavioral change is a regression to the mean workload (reduction of peak loads) of the pertinent tasks with regard to alloted time.
Load Value A value that represents the amount of psychological resources required to perform a task relative to performing another task or set of tasks.
Memory The psychological mechanisms that maintain information over time.
Mental Workload An evaluation about the difficulty of ongoing experiences and the impact of those experiences on the physical and mental states of an operator. The evaluation is a function of the collection of attributes that may or may not be relevant in controlling the evaluations or behavior that depend on the circumstances and design of a given task(s), and the a priori bias of the operator.
Motor The effector mechanisms of the human body.
Multi-Task Model A comprehensive, quantitative model that represents the combined effects of a variety of tasks on human performance.
Object-Oriented Programming Programming languages that represent data structures as objects with attributes and values.
Output Model A model that focuses on the result of human performance.
Normative Model A model that predicts how an operator should perform by assuming rational operator behavior.
Phase IV The period of research and development on MIDAS from the small offsite in March 1989 to the end of demos in July 1990.
Phase V The period of research and development on MIDAS from the large offsite in November 1990 to the end of demos sometime in late 1991 or early 1992.
Physical System The design of the system hardware and software.
Process A specific information processing mechanism that manipulates information symbolically.
Resource The attentional, physical and memory capabilities of an operator.
Scheduling Model The software configuration item that sequences the order of the simulated tasks.
Serial Constraints The imposed order of interactions between the dimensions used in the TLM.
Simulation Executive The software configuration item used to control the simulation by controlling the flow of execution of the rest of MIDAS' software configuration items.
Software Component A component of a software architectural unit/item used to implement a specific model or tool within MIDAS.
Software Configuration Item An architectural unit of software used to implement a specific model or tool within MIDAS.
Structure A specific information processing mechanism that represents information symbolically.
Summing Model A model that predicts retrospective workload ratings by summing the difficulty of the events or stimuli that are experienced.
Task Decomposition Model The software configuration item that decomposes high level goals into the simulated tasks.
Task Loading Model The software configuration item that predicts the visual, auditory, cognitive and motor loads that the simulated tasks impose on the operator of the system.
Tick Based A simulation that progresses according to changes in state variables that represent specific events that occur in the environment or behavior of the operator.
User A good question and currently ill-defined, but usually meant to be an operator of the MIDAS workstation.
Variant Property A characteristic of the information processing system that changes as a function of the information extracted from the perceptual array.

Visual A modality of perception involving visual stimuli.
Visual-Cognitive The interaction of visual perceiving and central processing mechanisms.
Within-Matrix A two dimensional matrix of values that represents the demands incurred on the human information processing system from the interaction of information processing mechanisms within the same stage.

9.0 NOTES

The software contained in this documentation is completely new. The TLM did not exist before phase IV, and was completed four months prior to phase IV demos. The software of the TLM described in this document was developed in that four month period, and was completed a few days prior to the phase IV demos. This software will provide the core for future TLM software development efforts.

9.1 LIMITATIONS

The limitations of the TLMSCI are primarily related to its new inception as a computational model. The model is untested and so valididy can not be established. The computational structure of the model may not be the most optimal as a result of the lack of understanding of the models strengths and weaknesses, as well as the model's structure and its algorithms. This lack of understanding leads to vague notions of the most appropriate future computational efforts. Additionally, the programmer was inexperienced, and even with superb tutoring and guidance from a Lisp guru, the efficiency and efficacy of the current computational efforts may be questioned, and may lead to problems in future development efforts.

9.2 LESSONS LEARNED

The author learned far too many lessons to state them all. Suffice it to say that aside from learning how to program, how to use the Symbolics machine, learning Lisp, and the meaning of AI, the author learned far more than he intended.

9.3 FUTURE DIRECTIONS

The future software development efforts for the TLM will consist of fully implementing the Task Adjustor TLM component. This component will add the "real" power to the TLM by providig the mechanisms to adjust the task classifications in order to tailor the tasks to the the current operating environment. This capability will allow the tasks to be automatically classified after they have been generated by the Symbolic Operator Model. This capability will also preclude the necessity for the user to edit any files to prepare and initialize the MIDAS-TLM system for use in simulations.

The specific nature of the software development efforts has currently not been detailed. However, the intial ideas suggest that the individual elements used in the task classifications will be coded as agents, some independent and others mutually dependent on each other. Each agent will contain the necessary slots, rules and functions that enable the agent to search relevant data pools that exist elsewhere in the simulation environment for the appropriate state variables values. Specific search algorithms, state variables and data pools can not be specified at the time of writing this document, because the level of detail used to represent the simulation tasks has not been decided. The level of detail will determine the state values available for the TLM to classify tasks.

10.0 APPENDIX A

APPENDIX A

CONFLICT MATRICES

AUDITORY DIMENSION	OR	DI	SI	SP	SA	MA
ORIENT	1					
DISCRIMINATE	2	3				
SIGNAL	1	2	2			
SPEECH	1	3	3	999		
SALIENT	1	2	1	1	1	
MASKED	2	3	2	3	3	5

FIGURE A-1 CONFLICT MATRIX FOR THE AUDITORY DIMENSION

(OR=orient, DI=discriminate, SI=signal. SP=spatial, SA=salient, MA=masked).

COGNITIVE DIMENSION	DI	TR	SC	MC	VE	SP	PL	UN
DIRECT	1							
TRANSFORMATION	2	3						
SINGLE CHOICE	1	2	1					
MULTIPLE CHOICE	1	3	2	3				
VERBAL	2	3	2	3	4			
SPATIAL	1	2	1	2	2	3		
PLANNED	1	3	1	2	2	1	1	
UNPLANNED	2	4	2	3	4	3	2	3

FIGURE A-2 CONFLICT MATRIX FOR THE COGNITIVE DIMENSION

(DI=direct TR=transformation, SC=single choice, MC=multiple choice, VE=verbal, SP=spatial, PL=planned)

MOTOR DIMENSION	VE	SP	NE	FA	DI	CO	GR	FI	MO	HE	EY	HA	FE	FI	RI	LE	BO
VERBAL	999																
SPATIAL	2	3															
NEAR	1	1	1														
FAR	1	2	2	3													
DISCRETE	2	1	2	1	3												
CONTINUOUS	3	2	3	2	2	1											
GROSS	1	1	1	2	1	2	1										
FINE	1	2	2	3	3	4	2	3									
MOUTH	1	1	1	1	1	2	1	1	999								
HEAD	1	1	1	1	1	2	1	2	1	999							
EYE	1	1	1	2	1	2	2	1	1	5	999						
HAND	1	2	1	2	1	2	1	2	1	1	2	3					
FEET	1	2	1	1	1	2	1	2	1	1	1	2	3				
FINGER	1	3	2	3	2	3	2	3	1	2	2	3	2	4			
RIGHT	1	2	1	1	1	1	1	1	1	1	2	1	2	2	999		
LEFT	1	2	1	1	1	1	1	1	1	1	2	1	2	2	3	999	
BOTH	1	3	1	1	1	1	1	1	1	1	1	2	3	4	999	999	999

FIGURE A-3 CONFLICT MATRIX FOR THE MOTOR DIMENSION

(VE=verbal, SP=spatial, NE=near, FA=far, DI=discrete, CO=continuous, GR=gross, FI=fine, MO=mouth, HE=head, EY=eye, HA=hand, FE=feet, FI=finger, RI=right, LE=left, BO=both).

AUDITORY / VISUAL	OR	DI	SI	SP	SA	MA
NEAR	1	2	2	2	1	2
FAR	2	3	2	2	1	2
SCAN	1	2	2	2	1	2
FIXATE	2	3	3	3	2	3
INTEGRAL	3	4	3	3	3	4
SEPARABLE	1	2	1	1	1	2
OBJECTS	1	2	1	1	1	2
FEATURES	2	3	2	2	2	3
SALIENT	1	2	1	1	1	2
MASKED	2	3	2	2	2	3
STATIC	1	2	1	1	1	2
DYNAMIC	3	4	2	2	2	3

FIGURE A-4 CONFLICT MATRIX FOR THE VISUAL-AUDITORY DIMENSION

(OR=orient, DI=discriminate, SI=signal. SP=spatial, SA=salient, MA=masked).

COGNITIVE VISUAL	DI	TR	SC	MC	VE	SP	PL	UN
NEAR	1	2	1	2	1	2	1	2
FAR	1	3	2	3	1	3	1	2
SCAN	1	2	2	4	3	4	2	3
FIXATE	1	3	1	3	1	2	1	2
INTEGRAL	1	3	3	4	3	4	3	4
SEPARABLE	1	2	1	2	2	3	1	2
OBJECTS	1	2	1	2	2	3	1	2
FEATURES	1	3	2	3	1	3	2	3
SALIENT	1	2	1	2	1	2	1	2
MASKED	2	3	2	3	4	3	2	3
STATIC	1	2	1	2	2	1	1	2
DYNAMIC	1	3	2	3	3	2	2	3

FIGURE A-5 CONFLICT MATRIX FOR THE VISUAL COGNITIVE DIMENSION

(DI=direct TR=transformation, SC=single choice, MC=multiple choice, VE=verbal, SP=spatial, PL=planned)

COGNITIVE AUDITORY	DI	TR	SC	MC	VE	SP	PL	UN
ORIENT	1	2	1	2	1	1	1	2
DISCRIMINATE	2	3	2	3	3	2	2	3
SIGNAL	1	2	2	3	1	1	1	2
SPEECH	1	1	1	2	1	3	2	2
SALIENT	1	1	1	2	1	1	1	2
MASKED	2	2	2	3	3	2	2	3

FIGURE A-6 CONFLICT MATRIX FOR THE AUDITORY-COGNITIVE DIMENSION

(DI=direct TR=transformation, SC=single choice, MC=multiple choice, VE=verbal, SP=spatial, PL=planned).

COGNITIVE / MOTOR	DI	TR	SC	MC	VE	SP	PL	UN
VERBAL	1	2	1	2	1	4	1	2
SPATIAL	2	3	2	3	4	1	2	3
NEAR	0	0	0	0	0	0	0	0
FAR	0	0	0	0	0	0	0	0
DISCRETE	1	2	1	2	1	1	1	2
CONTINUOUS	2	3	2	3	2	2	2	3
GROSS	1	2	1	2	1	1	1	2
FINE	2	3	2	3	1	1	2	3
MOUTH	1	2	1	2	2	1	1	2
HEAD	1	2	1	2	1	2	1	2
EYE	1	2	1	2	1	1	1	2
HAND	2	3	1	2	3	2	1	2
FEET	2	3	1	2	3	2	1	2
FINGER	3	4	2	3	4	3	2	3
RIGHT	1	2	1	2	3	2	1	2
LEFT	1	2	1	2	3	2	1	2
BOTH	2	3	2	3	4	3	2	3

FIGURE A-7 CONFLICT MATRIX FOR THE MOTOR-COGNITIVE DIMENSION

(DI=direct ,TR=transformation, SC=single choice, MC=multiple choice, VE=verbal, SP=spatial, PL=planned).

Annex D

Army-NASA Aircrew/Aircraft Integration Program: Phase IV

A^3I

Man-Machine Integration Design and Analysis System (MIDAS)
Software Detailed Design Document

Symbolic Equipment Model

prepared by

David Bushnell

Table of Contents

Table of Contents

MAN-MACHINE INTEGRATION DESIGN & ANALYSIS SYSTEM (MIDAS) SOFTWARE DETAILED DESIGN DOCUMENT PHASE IV:

SYMBOLIC EQUIPMENT MODEL

1.0 INTRODUCTION

1.1 IDENTIFICATION OF DOCUMENT

This document gives the requirements and design of the Equipment Model Software Component for MIDAS Phase IV.

1.2 PURPOSE OF DOCUMENT

This document is for programmers and other technical specialists working on the development of the Equipment Model Component for the prototype MIDAS computer-aided engineering workstation. It describes the model's requirements and design. Familiarity with object-oriented systems is assumed.

2.0 RELATED DOCUMENTATION

2.1 APPLICABLE DOCUMENTS

Symbolics Genera 7.2 Documentation, Symbolics Publication Number 999079, Symbolics, Inc., Cambridge, Massachusetts, 1988.

3.0 CONCEPT

3.1 DEFINITION OF THE EQUIPMENT MODEL

3.1.1 Purpose and Scope

The Equipment Model lets its users define equipment components to be used in a MIDAS simulation. It is designed to allow equipment models that reflect the functional and physical properties of the equipment being simulated. It is also designed to make equipment definition tractable by allowing the reuse of previously defined components in new equipment, by separating physical and functional aspects of equipment so that each can be specified independently, and by enforcing a separation between the Symbolic Operator Model and the Equipment Model so that the applicability of the functional components to particular tasks can be determined dynamically at runtime. Finally, the Equipment Model includes the Longbow equipment required to support the Phase IV demonstrations.

3.2 USER DEFINITION

People directly use the Equipment Model only for defining equipment for a simulation. Under normal operations the equipment is controlled by the various parts of the Symbolic Operator Model. These parts of Symbolic Operator Model can request components that perform some specified function, request components by name, perform specified functions, or tell components to advance one time tick.

3.3 CAPABILITIES AND CHARACTERISTICS

The Equipment Model uses the physical and functional component types to separate the physical and functional aspects of a component. Every component in a piece of equipment is represented by both a physical component type and a functional component type. The physical component type contains information about the object's physical characteristics. For example, if a switch is being modeled then its physical model would specify whether it was a toggle switch or a push button and, if it was a toggle switch, how much force would be required to flip it from one state to another. The functional component type contains information about the object's functionality. In the previous example, the switch's functional model would specify that the switch was used for turning on and off a radio.

When executing a simulation, the user (which is normally the Symbolic Operator Model) typically accesses only the functional components and accesses them not by name but by function. More specifically, a user will typically issue a command to the Equipment Model of the form: "Execute the functional component that performs the following function." The functional component will determine what physical actions are required to execute the function and tell the associated physical component to perform those actions.

3.4 SAMPLE OPERATIONAL SCENARIOS

Typical usage of the Equipment Model has two parts: definition and execution. In the definition phase, a person describes a model of some equipment of interest. In the Execution phase, the Symbolic Operator Model selects functions and tells the Equipment Model to execute them.

A simple component definition in the Longbow Equipment component is

```
(1)    (def-equip-comp-type \ D,#TD1Ps-T [Begin using 006 escapes](1 0 (NIL 0) (NIL
       :BOLD NIL) "CPTFONTCB")rts-switch
       (2 0 (NIL 0) (NIL NIL NIL) "CPTFONT")
(2)             ()
(3)      (normally-off-momentary-switch)
(4)      :referenced-equipment (func-cpu)
         :specializations
         (:self-specializations
(5)        ((switch-state-output  (rts-switch-state func-cpu))
(5)         (activate-switch  (activate-rts ufd))
(5)         (deactivate-switch  (deactivate-rts ufd)))))
```

(The numbered lines in this definition are described below.)

1) The name of the component. This component's name is RTS-SWITCH.

2) Internal state variables of the component. RTS-SWITCH has none of its own, but may inherit some. See (3) below.

3) More general components on which this one is built. RTS-SWITCH depends on the component NORMALLY-OFF-MOMENTARY-SWITCH.

4) The names of other components in the equipment model. These either supply inputs to or accept outputs from this component. RTS-SWITCH refers to the component named FUNC-CPU.

5) What the inputs, outputs, and supported functions (in the more general components named in (3) above) really refer to. In this case, the generic NORMALLY-OFF-MOMENTARY-SWITCH component has an output called SWITCH-STATE-OUTPUT. For an RTS-SWITCH, this really refers to the RTS-SWITCH-STATE of the FUNC-CPU object. Also, the functions "Activate Switch" and "Deactivate Switch" supported by the generic NORMALLY-OFF MOMENTARY-SWITCH component become the more specific "Activate the RTS Switch of the UFD" and "Deactivate the RTS Switch of the UFD" functions.

This definition creates the RTS-SWITCH component type. To actually create an RTS-SWITCH component object, the user executes:

```
(make-instance 'RTS-SWITCH
               <initializations for RTS-SWITCH instance variables>)
```

Note that the new RTS-SWITCH object automatically gets inserted into the equipment model that is the value of the special variable *EQUIPMENT-MODEL*. It is stored both by name, RTS-SWITCH, and by function, (ACTIVATE-RTS UFD) and (DEACTIVATE-RTS UFD).

At runtime, the Symbolic Operator Model may find that it needs to activate the UFD's RTS switch. It retrieves all components of the equipment model that support that function (there is only one in this case), selects the one best suited to its needs, and executes it.

4.0 REQUIREMENTS

4.1 REQUIREMENTS SPECIFICATION

There are five major requirements levied on the Equipment Model:

1) to describe the functional and physical characteristics of the equipment used in the MIDAS simulation with enough detail for the Symbolic Operator Model to be able to interact with a simulation's equipment.

2) to allow a simulation's equipment to be built from libraries of existing generic components, rather than having to be built from scratch each time they are needed.

3) to allow the functional and physical characteristics of the equipment to be specified independently, so that physical components can be replaced by other physical components (with the same functionality) without requiring modifications to the rest of a simulation.

4) to allow the Symbolic Operator Model to dynamically choose from the applicable functional components of a simulation which should be executed.

5) to represent the Longbow equipment needed to support the Phase IV demo.

4.2 EXTERNAL INTERFACE REQUIREMENTS

The external interface to people is only required to support equipment component definition, not execution. The users shall be able to define equipment components by typing the appropriate definitions into text files and executing them.

The external interface to the Symbolic Operator Model shall allow it to retrieve and execute components by both name and functionality. The Symbolic Operator Model can do this by calling the appropriate methods of the Equipment Database.

4.3 IMPLEMENTATION CONSTRAINTS

The Equipment Model requires a Symbolics computer with Genera 7.2 system software.

5.0 DESIGN

5.1 ARCHITECTURAL DESIGN

5.1.1 Architectural Design Description

The Equipment Model comes in three parts: one is a way of defining new equipment components and specifying their physical and functional properties (the Equipment Definition module); another part is a database indexing the components by both their names and their functional properties (the Equipment Database module); the final part is the actual Longbow equipment model (the Longbow Equipment module). The Equipment Definition module allows for the definition of generic components that can be easily specialized into the particular components needed in a simulation. The Equipment Database module is implemented as an object with two components: the Equipment Name Database and the Equipment Model Database. The Equipment Name Database stores the equipment components indexed by their names, while the Equipment Model Database stores them indexed by their function. These two components are described in more detail below. The Longbow Equipment module contains the definitions of the Longbow helicopter's equipment components.

5.2 DETAILED DESIGN

5.2.1 The Equipment Model

This section describes each of the components of the Equipment Model.

5.2.1.1 Equipment Definition module

The Equipment Definition module contains the fundamental object types which underlie all equipment components in a model, as well as the functions and macros needed to define the pieces of equipment in a given model. There are two object types that are used by all equipment components: a physical component type (called Physical-Component) and a functional component type (called Equipment-Component). Associated with these are two macros used to define components: one macro for defining physical components (Def-Phys-Comp-Type) and one for defining functional components (Def-Equip-Comp-Type).

5.2.1.1.1 Physical-Component Type

The Physical-Component type is a component flavor of all physical components in an equipment model. As outlined in section 3.3 above, it interacts with the user mainly through the functional model. Since the physical components are generic, there must be an abstract interface between them and the functional components. This interface consists of specified sets of inputs and outputs, the allowed values for them, and the methods for mapping between the functional and physical states. In addition, the user needs to know what physical actions (e.g. reaches) are required to achieve physical states. The Physical-Component type therefore supports a mapping from physical states to physical actions.

5.2.1.1.2 Physical-Component Instance Variables

The Physical-Component type contains the following instance variables:

1) The associated functional component.

2) The set of physical inputs.

3) The set of physical outputs.

4) The mapping from functional inputs and outputs to physical inputs and outputs.

5.2.1.1.3 Physical-Component Methods

Physical-Component type supports the following major methods:

1) Map an abstract state to a physical state.

2) Map a physical state to an abstract state.

3) Map a physical state to a physical action.

4) Execute the current clock tick's actions.

5.2.1.1.4 Equipment-Component Type

The Equipment-Component type is a component flavor of all functional components. It supplies the main execution interface between a user and an equipment model. As such its principle interaction with a user is the execution of its individual functions.

5.2.1.1.5 Equipment-Component Instance Variables

The Equipment-Component type contains the following instance variables:

1) The component's name.

2) The associated physical component.

3) The functions that the component supports.

4) The set of functional inputs.

5) The set of functional outputs.

6) The mappings from functional inputs and outputs to abstract inputs and outputs.

5.2.1.1.6 Equipment-Component Methods

Equipment-Component type supports the following major methods:

1) Map a functional input to an abstract input.

2) Map an abstract output to a functional output.

3) Execute a specified function.

4) Execute the current clock tick's actions.

5.2.1.2 Equipment Database module

The Equipment Database module contains the object type that is used to build the equipment databases for a simulation. It also contains the functions and methods needed to update, query, and execute the contents of the databases.

5.2.1.2.1 Equipment Database Type

Each Equipment Database type is used to represent equipment models. It is possible to have many equipment models in a single simulation, with each representing a single major piece of equipment.

5.2.1.2.1.1 Equipment Database Instance Variables

The Equipment Database type contains the following instance variables:

1) The Equipment Name Database, which contains a database that indexes the equipment components by their names.

2) The Equipment Model Database, which contains a database that indexes the equipment components by their functions.

5.2.1.2.1.2 Equipment Database Methods

Equipment Database type supports the following major methods:

1) Graph the contents of the Equipment Database.

2) Retrieve all equipment components with a function that matches a pattern.

3) Retrieve the equipment component with a given name.

4) Create an equipment component of a given type and add it to the Equipment Database.

5) Execute the function of the first equipment component that matches a given pattern.

5.2.1.2.2 Equipment Name Database Type

The Equipment Name Database indexes the equipment components by their names. It is used when the Symbolic Operator Model needs access to specific pieces of equipment. It is implemented as a standard system hash table.

5.2.1.2.2.1 Equipment Name Database Instance Variables

The Equipment Name Database type contains no instance variables other than those of a standard system hash table.

5.2.1.2.2.2 Equipment Name Database Methods

The Equipment Name Database type supports no methods other than those of the standard system hash tables.

5.2.1.2.3 Equipment Model Database Type

The Equipment Model Database indexes the equipment components by their functionality. It is used when the Symbolic Operator Model needs access to pieces of equipment that perform specific functions. The Equipment Model Database is implemented as a discrimination network, each of whose nodes is a Database Node object.

5.2.1.2.3.1 Database Node Type

A Database Node is a single node in the discrimination network that makes up the Equipment Model Database. It contains key, data, and network information. The key stored in a single Database Node object represents only the terminal element of the pattern that accesses the data. The entire key for a node's data is represented by the list of keys of all the nodes between the root node of the Equipment Model Database and the node in question.

5.2.1.2.3.1.1 Database Node Instance Variables

The Database Node type contains the following instance variables:

1) The terminal component of the key for the data stored at this node.

2) The data corresponding to the pattern formed by the list of all keys between this node and the Equipment Model Database's root node.

3) The children of this node.

5.2.1.2.3.1.2 Database Node Methods

The Database Node type supports the following major methods:

1) Match a pattern with the key of a single database node.

2) Insert data corresponding to a pattern.

3) Insert data corresponding to a linearized pattern.

4) Push data onto a list of existing data corresponding to a pattern.

5) Push data onto a list of existing data corresponding to a linearized pattern.

6) Retrieve the data corresponding to a pattern.

7) Search a network of database nodes starting from a root to find the database node corresponding to a pattern.

8) Apply a function to every node in a database.

9) Apply a function to every node in a database that satisfies a predicate and collect the results of the function application.

10) Draw a graph of the database network.

5.2.1.3 Longbow Equipment module

The Longbow Equipment module contains the definitions of the Longbow helicopter equipment simulated in the Phase IV demo. This equipment consists of the subset of cockpit displays and controls that was required to run the Symbolic Operator Model:

1) The Central Processor

2) The Communication Control Panels

3) The Keyboards

4) The Up-Front Displays

5) The Multi-Function Displays

5.2.1.3.1 The Functional CPU Type

Much of the equipment in the Longbow cockpit is controlled by or supplies inputs to a computer called the Central Processor. In the Longbow Equipment module this computer is modeled by the Function CPU type. Most of the other equipment modeled in the Longbow Equipment module makes some reference to the Functional CPU, some for input, some for output.

5.2.1.3.1.1 Functional CPU Instance Variables

The Functional CPU type has the following instance variables:

1) The current state of the pilot's RTS switch

2) The previous state of the pilot's RTS switch

3) The current state of the pilot's LAST switch

4) The previous state of the pilot's LAST switch

5) The currently pressed key on the pilot's keyboard

6) The previous state of the pilot's keyboard

7) The contents of the pilot's keyboard buffer

8) Whether the pilot's keyboard buffer is complete

9) The current state of the copilot-gunner's RTS switch

10) The previous state of the copilot-gunner's RTS switch

11) The current state of the copilot-gunner's LAST switch

12) The previous state of the copilot-gunner's LAST switch

13) The currently pressed key on the copilot-gunner's keyboard

14) The previous state of the copilot-gunner's keyboard

15) The contents of the copilot-gunner's keyboard buffer

16) Whether the copilot-gunner's keyboard buffer is complete

17) The current state of the copilot-gunner's mfd-button

18) The previous state of the copilot-gunner's mfd-button-last

19) The current page on the copilot-gunner's mfd

20) The current vhf frequency.

21) The previous vhf frequency.

22) The current uhf frequency.

23) The previous uhf frequency.

24) The current fm1 frequency.

25) The previous fm1 frequency.

26) The current fm2 frequency.

27) The previous fm2 frequency.

28) The current vhf call sign.

29) The previous vhf call sign.

30) The current uhf call sign.

31) The previous uhf call sign.

32) The current fm1 call sign.

33) The previous fm1 call sign.

34) The current fm2 call sign.

35) The previous fm2 call sign.

36) The current transmitter selected by the pilot.

37) Whether the pilot has selected the vhf receiver.

38) Whether the pilot has selected the uhf receiver.

39) Whether the pilot has selected the fm1 receiver.

40) Whether the pilot has selected the fm2 receiver.

41) The current transmitter selected by the copilot-gunner.

42) Whether the copilot-gunner has selected the vhf receiver.

43) Whether the copilot-gunner has selected the uhf receiver.

44) Whether the copilot-gunner has selected the fm1 receiver.

45) Whether the copilot-gunner has selected the fm2 receiver.

5.2.1.3.1.2 Functional CPU Methods

The Functional CPU type has the following methods:

1) Execute the current time step's activities. This consists of:

- Processing new inputs from the pilot's keyboard.
- Processing new inputs from the copilot-gunner's keyboard.
- Processing new inputs from the pilot's RTS button.
- Processing new inputs from the copilot-gunner's RTS button.
- Processing new inputs from the pilot's LAST button.
- Processing new inputs from the copilot-gunner's LAST button.

5.2.1.3.3 The CCP Type

The Communication Control Panel (CCP) in the Longbow cockpit controls what the crew members hear from the helicopter's radios. There is one Communication Control Panel at each crew station. The CCP type in the Longbow Equipment module only models the selection of which radios the crew members listen to.

5.2.1.3.3.1 CCP Instance Variables

The CCP type has the following instance variables:

1) The VHF receiver control.

2) The UHF receiver control.

3) The FM1 receiver control.

4) The FM2 receiver control.

5.2.1.3.3.2 CCP Methods

The CCP type does not support any methods beyond those of the Equipment Component type.

5.2.1.3.4 The CCP VHF Receiver Control Type

The Communication Control Panel's VHF Receiver Control selects the VHF radio for listening and controls its volume. Only the selection function is implemented in the Longbow Equipment module.

5.2.1.3.4.1 CCP VHF Receiver Control Instance Variables

The CCP VHF Receiver Control type has the following instance variables:

1) The state of the control.

2) The Functional CPU object.

5.2.1.3.4.2 CCP VHF Receiver Control Methods

The CCP VHF Receiver Control type has the following methods:

1) Select the VHF radio for listening.

2) Unselect the VHF radio for listening.

5.2.1.3.5 The CCP UHF Receiver Control Type

The Communication Control Panel's UHF Receiver Control selects the UHF radio for listening and controls its volume. Only the selection function is implemented in the Longbow Equipment CSC.

5.2.1.3.5.1 CCP UHF Receiver Control Instance Variables

The CCP UHF Receiver Control type has the following instance variables:

1) The state of the control.

2) The Functional CPU object.

5.2.1.3.5.2 CCP UHF Receiver Control Methods

The CCP UHF Receiver Control type has the following methods:

1) Select the UHF radio for listening.

2) Unselect the UHF radio for listening.

5.2.1.3.6 The CCP FM2 Receiver Control Type

The Communication Control Panel's FM2 Receiver Control selects the FM2 radio for listening and controls its volume. Only the selection function is implemented in the Longbow Equipment module.

5.2.1.3.6.1 CCP FM2 Receiver Control Instance Variables

The CCP FM2 Receiver Control type has the following instance variables:

1) The state of the control.

2) The Functional CPU object.

5.2.1.3.6.2 CCP FM2 Receiver Control Methods

The CCP FM2 Receiver Control type has the following methods:

1) Select the FM2 radio for listening.

2) Unselect the FM2 radio for listening.

5.2.1.3.7 The CCP FM2 Receiver Control Type

The Communication Control Panel's FM2 Receiver Control selects the FM2 radio for listening and controls its volume. Only the selection function is implemented in the Longbow Equipment module.

5.2.1.3.7.1 CCP FM2 Receiver Control Instance Variables

The CCP FM2 Receiver Control type has the following instance variables:

1) The state of the control.

2) The Functional CPU object.

5.2.1.3.7.2 CCP FM2 Receiver Control Methods

The CCP FM2 Receiver Control type has the following methods:

1) Select the FM2 radio for listening

2) Unselect the FM2 radio for listening.

5.2.1.3.8 The Keyboard Type

Both crew stations have an alphanumeric keyboard in the Longbow cockpit. This keyboard has 50 keys, a 44 character buffer with a 22 character display, and a display brightness control. The keyboard model has 40 keys (A-Z, 0-9, period, ESC, CLR, and ENTER) and no brightness control. The character buffer and display are modeled in the Central Processor.

5.2.1.3.8.1 Keyboard Instance Variables

The Keyboard type has the following instance variables.

1) The keys on the keyboard (each is a separate instance variable).

5.2.1.3.8.2 Keyboard Methods

The Keyboard type does not support any methods beyond those of the Equipment Component type.

5.2.1.3.9 The Key Types

The various key types (one for each of the keys on a keyboard) support the entry of alphanumeric data.

5.2.1.3.9.1 Key Instance Variables

Each Key type has the following instance variables:

1) An instance of the Central Processor object.

2) The state of the key (ON or OFF).

5.2.1.3.9.2 Key Methods

Each Key type has the following methods:

1) Activate (i.e. press) the key.

2) Deactivate (i.e. release) the key.

5.2.1.3.10 The UFD Type

The Up-Front Display (UFD) in the Longbow cockpit shows important communications status information and the current Caution/Warning/Advisory messages. There is one UFD at each crew station. The UFD type in the Longbow Equipment module only displays the statuses of the VHF, UHF, FM1, and FM2 radios and allows these to be changed with the RTS and LAST buttons.

5.2.1.3.10.1 UFD Instance Variables

The UFD type has the following instance variables:

1) The RTS switch, which selects the radio to be used as the crew member's transmitter.

2) The LAST switch, which swaps the previous frequency with the current frequency on the radio being used as a transmitter.

5.2.1.3.10.2 UFD Methods

The UFD type does not support any methods beyond those of the Equipment Component type.

5.2.1.3.11 The RTS Switch Type

The RTS switch cycles through the radios (VHF, UHF, FM1, and FM2), selecting each in turn as the crew member's transmitter. The RTS Switch type implements this behavior by updating the crew member's currently selected transmitter in the Functional CPU.

5.2.1.3.11.1 RTS Switch Instance Variables

The RTS Switch type has the following instance variables:

1) The Functional CPU.

2) The switch's current state.

5.2.1.3.11.2 RTS Switch Methods

The RTS Switch type has the following methods:

1) Activate (i.e. push) the RTS switch.

2) Deactivate (i.e. release) the RTS switch.

5.2.1.3.12 LAST Switch Type

The LAST switch swaps the currently selected frequency and call sign on the crew member's transmitter with the previously selected frequency and call sign. The LAST Switch type implements this behavior by updating the appropriate instance variables in the Functional CPU.

5.2.1.3.12.1 LAST Switch Instance Variables

The LAST Switch type has the following instance variables:

1) The Functional CPU.

2) The switch's current state.

5.2.1.3.12.2 LAST Switch Methods

The LAST Switch type has the following methods:

1) Activate (i.e. push) the LAST switch.

2) Deactivate (i.e. release) the LAST switch.

5.2.1.3.13 The MFD Type

The Multi-Function Display (MFD) in the Longbow Helicopter is a touch-sensitive CRT display that shows the status of and controls nearly every other system in the helicopter. There is one MFD at each crew station. The information and control functions are broken down into "pages" and only one page can be displayed on an MFD at a time. The pages are broken down into five groups: communications, navigation, weapons, aircraft, and fire control radar.

The MFD type implements subsets of the communications and navigation pages. The MFD type contains a finite state machine (FSM) to control the complex relations among the MFD pages. This FSM makes it easier to specify the legal transitions between pages and the effects of actions within a page.

5.2.1.3.13.1 MFD Instance Variables

The MFD type has the following instance variables:

1) The current state of the MFD's FSM.

2) The method used by the FSM's current state for processing its tick-based activities.

3) The set of legal FSM states and their legal transitions to new states.

4) The currently selected preset frequency.

5) The buffer for the manually keyed-in frequency.

6) The manually selected radio.

7) The currently pressed MFD soft button (or :OFF if no button was pressed during the last time step).

8) The previously pressed MFD soft button (or :OFF if no button is pressed during the current time step).

9) Buffer for holding things typed for the MFD.

10) The MFD's preset frequencies.

11) The number of pixels horizontally and vertically in the MFD screen.

12) The proportion of the MFD screen that corresponds to the diameter of the outer range arc on the navigation page.

13) The waypoint selected by the operator.

14) The transform between the MFD navigation page screen coords and real world coords.

15) The translation from the MFD world coordinate system to the screen coordinate system.

16) The inverse of the MFD Nav page map scale (only the part of the scale between the MFD world coordinate system and the screen coordinate system).

17) The distance to the outer arc on the MFD's navigation page. This is computed whenever the MFD's range scale or internal scale is set.

18) The scale between the MFD's navigation page range scale and the MFD's screen coordinate system.

19) The ownship's present position.

20) The ownship's present heading.

21) The list of all waypoints.

22) The list of waypoints currently visible in the MFD.

23) The list of all waypoints on the ownship's route.

24) The list of waypoints on the ownship's route that have not yet been passed.

25) A waypoint that has been selected to be added to or deleted from the ownship's route.

26) The number of pounds of fuel per nautical mile being used at the present time.

27) The number of minutes of fuel remaining at the current burn rate.

5.2.1.3.13.2 MFD Methods

The MFD type has the following methods:

1) Execute the current time step's activities. This consists of executing the FSM's current processing method.

2) The processing method for entering a waypoint's altitude.

3) The processing method for entering a waypoint's utm coordinates.

4) The processing method for entering a waypoint description.

5) The processing method for entering a waypoint's id number.

6) Find the next unused waypoint id number.

7) The processing method for pressing the "add waypoint" soft button.

8) The processing method the waypoint subpage.

9) The processing method for the route subpage.

10) The processing method for the present position subpage.

11) The processing method for the navigation top page.

12) The processing method for when one of the radio buttons (VHF, UHF, on the manual communications subpage is selected.

13) The processing method for when the ENTER button on the manual communications subpage is selected.

14) The processing method for when the CLEAR button on the manual communications subpage is selected.

15) The processing method for when one of the digit buttons on the manual communications subpage is selected.

16) The processing method for the manual communications subpage.

17) Update the mfd's graphics transform from its relevant instance variables.

18) The processing method for when either the TUNE FM1 or TUNE FM2 button on the presets communication page is pressed.

19) The processing method for when the TUNE button on the presets communication page is pressed.

20) The processing method for when the PRESET button on the communications top page is pressed.

21) The processing method for the communications top page.

22) The initial processing method for the mfd.

23) Update the distance to the outer arc on the mfd's display.

24) Determine whether it is legal to make a given transition from the current state.

25) Update the mfd's FSM state information for a new state.

26) Convert world coordinates to screen coordinates

27) Update the list waypoints on the current route that haven't been passed yet.

28) Decide whether a waypoint is visible on the mfd.

29) Update the list of visible waypoints.

6.0 USER'S GUIDE

6.1 INSTALLATION AND INITIALIZATION

The Equipment Model is maintained as a set of "systems" on Symbolics computers. It can therefore be compiled, loaded, and so on by using Symbolics standard system utilities. There are two systems in the Equipment Model, one named EQUIPMENT-MODEL and one named LONGBOW-MODEL. All files should be referenced with logical pathnames. The logical hosts for these systems are EQM-MDL for the EQUIPMENT-MODEL system and LONGBOW-MDL for the LONGBOW-MODEL system.

6.1.1 Installing the EQUIPMENT-MODEL System

The logical host for the EQUIPMENT-MODEL system is EQM-MDL. The host is defined in the file EQUIPMENT-MODEL.SYSTEM, which must be loaded before this system can be used. It is currently in the directory B:>MIDAS>BASIC>EQUIP>. This file also contains the definition of the EQUIPMENT-MODEL system. The files for the EQUIPMENT-MODEL system are stored under the directory tree "EQM-MDL:EQUIP;**;".

When installing the EQUIPMENT-MODEL System, the EQUIPMENT-MODEL.SYSTEM file can be restored to any directory, but for consistency it is recommended that it be put in the directory >MIDAS>BASIC>EQUIP>. The file should then be edited so that the system's files are put in the desired directories.

6.1.2 Installing the LONGBOW-MODEL System

The logical host for the LONGBOW-MODEL system is LONGBOW-MDL. The host is defined in the file LONGBOW-MODEL.SYSTEM, which must be loaded before this system can be used. It is currently in the directory B:>MIDAS>EQUIP>LONGBOW>. This file also contains the definition of the LONGBOW-MODEL system. The files for the LONGBOW-MODEL L system are stored under the directory tree "LONGBOW-MDL:LONGBOW;**;".

When installing the LONGBOW-MODEL System, the LONGBOW-MODEL.SYSTEM file can be restored to any directory, but for consistency it is recommended that it be put in the directory >MIDAS>EQUIP>LONGBOW>. The file should then be edited so that the system's files are put in the desired directories.

6.1.3 Initializing the EQUIPMENT-MODEL System

The EQUIPMENT-MODEL System is compiled with the standard system compilation commands. For example, suppose the system is loaded on Barracuda. At a "Command:" prompt, load the file EQUIPMENT-MODEL.SYSTEM and then compile the EQUIPMENT-MODEL system:

```
        Command: Load File B:>MIDAS>BASIC>EQUIP>EQUIPMENT-
MODEL.SYSTEM

        Command: Compile System EQUIPMENT MODEL
```

If the system is already compiled, then you can load it as in:

```
        Command: Load File B:>MIDAS>BASIC>EQUIP>EQUIPMENT-
MODEL.SYSTEM

        Command: Load System EQUIPMENT MODEL :Version Newest
```

The keyword argument ":Version Newest" is required.

6.1.4 Initializing the LONGBOW-MODEL System

The LONGBOW-MODEL System is compiled with the standard system compilation commands. For example, suppose the system is loaded on Barracuda. At a "Command:" prompt, load the file LONGBOW-MODEL.SYSTEM and then compile the LONGBOW-MODEL system:

```
        Command: Load File B:>MIDAS>EQUIP>LONGBOW>LONGBOW-
MODEL.SYSTEM

        Command: Compile System LONGBOW MODEL
```

If the system is already compiled, then you can load it as in:

```
        Command: Load File B:>MIDAS>EQUIP>LONGBOW>LONGBOW-
MODEL.SYSTEM
```

```
Command: Load System LONGBOW MODEL :Version Newest
```

The keyword argument ":Version Newest" is required.

6.2 STARTUP AND TERMINATION

The Equipment Model does not require any startup or termination beyond that described in section 6.1 above.

Annex E

Army-NASA Aircrew/Aircraft Integration Program: Phase IV

A³I

Man-Machine Integration Design and Analysis System (MIDAS) Software Detailed Design Document

Visual Editor and Simulation Tool (VEST)

prepared by

Scott Chen

Table of Contents

Table of Contents

Table of Contents

MAN-MACHINE INTEGRATION DESIGN & ANALYSIS SYSTEM (MIDAS) SOFTWARE DETAILED DESIGN DOCUMENT PHASE IV:

VISUAL EDITOR AND SIMULATION TOOL

1.0 INTRODUCTION

1.1 IDENTIFICATION OF DOCUMENT

This is the Software Product Specification for the VEST (Visual Editor and Simulation Tool) module of the MIDAS Software System. Descriptions of the detailed processing requirements, structure, I/O, and control are provided for each lower level Computer Software Component (CSC), units, or functions contained within this module.

1.2 SCOPE OF DOCUMENT

This document describes the framework, functions and use of the VEST software developed during Phase IV of the A^3I program. This document assumes that the reader is familiar with computer-graphics concepts, 3D geometrical modeling techniques, window oriented user interface, C programming language, UNIX operating system, and Silicon Graphics Inc.'s Graphics Library. It is also assumed that the reader has some experience with MultiGen® CAD.

1.3 PURPOSE AND OBJECTIVES OF DOCUMENT

VEST is developed on top of commercial CAD package MultiGen. The basic operations and programming techniques of MultiGen are well documented in the Software Systems MultiGen manuals. The purpose of this document is to show the results of VEST accomplished during Phase IV of the A^3I program. Its objectives are to present module/component layout, animation interface, and internal structures as the need to modify the source code may arise.

2.0 RELATED DOCUMENTATION

2.1 APPLICABLE DOCUMENTS

The following documents are referenced herein and are directly applicable to this volume:

Software Systems MultiGen® - Modeler's Guide, Interface Manager, MultiGen Kernel, Writing MultiGen DBL, Release 3.0, Software Systems, San Jose, September 1988.

Data Format Specification for Software Systems Flight Data Bases, Format Release 4, July 28, 1988.

Software Systems MultiGen®, DMA Terrain Conversion Option User's Guide, July, 1988.

A^3I Phase III Views CSCI, June 1990.

A^3I Phase III Cockpit Design Editor CSCI, June 1990.

A^3I Phase III JACK CSCI, June 1990.

APACHE longbow documentation.

2.2 INFORMATION DOCUMENTS

The following documents amplify or clarify the information presented in this volume:

Brian W. Kernighan and Dennis M. Ritchie, *The C Programming Language*, Prentice-Hall, Englewood Cliffs, N.J., 1978.

J. D. Foley and A. Van Dam, *Fundamentals of Interactive Computer Graphics*, Addison-Wesley, Reading, Massachusetts, 1982.

Silicon Graphics Inc., IRIS User's Guide, Volume I and II, Version 3.0, Mountain View, California, 1986.

Silicon Graphics Inc., IRIS Programmer's Manual, Volume IB, System Calls and Subroutines, Version 2.1, Mountain View, California, 1986.

3.0 CONCEPT

3.1 DEFINITION OF SOFTWARE

3.1.1 Purpose and Scope

The purpose of VEST is to develop a set of visual tools to support the A^3I simulation world. It provides an environment for flight simulation, instrument animation, and Jack movement. To rapidly create 3D graphical objects for simulation, MultiGen® modeling system is adopted for its ease of use and VEST is implemented as a subsystem of it. The software described here contains MultiGen® modules, DMA, VIEWS, CDE, MFD and Jack. This document emphasizes Phase IV effort on VIEWS, MFD and Jack-Interface. CDE extensions will also be discussed. DMA is well documented in Phase III and is now part of the MultiGen® package.

3.1.2 Goals and Objectives

The goals of VEST are:

(i) to create animation databases for Multifunction Displays (MFD) and traditional gauges. The values of their dynamic parameters can be assigned so that their behaviors can be monitored during simulation.

(ii) to display the simulation in different viewing perspectives with the progression of the Mission Simulation.

(iii) to provide an interface with Jack software to animate aviator's reaches.

(iv) to provide a convenient interface for integrating the graphics system with the simulation network.

3.1.3 Description

VEST attempts to provide a user tools to create 3D graphical objects and to display them based on data received from the simulation models. It runs on SGI IRIS workstations at a cost well below those of visual simulation systems and incorporates interactive graphic tools for constructing and displaying the Mission Simulation at an acceptable speed. In the animation mode, it communicates, through the network, with the Executive which contains a datapool available for all processes.

3.2 USER DEFINITION

VEST is developed for cockpit designers. However, any users with some experience with SGI IRIS workstations can use VEST to create and edit 3D graphical objects. The use of the DMA component needs knowledge of DMA tape, UNIX and MultiGen®. To effectively use CDE and MFD modules , users need to know instrument characteristics, network interface with the communication manager, and some data structure of the datapool. Jack interface requires experience with Jack and MultiGen®. The use of VIEWS is based on the organization of the MultiGen® database tree-structure and the orientation of graphical objects.

3.3 CAPABILITIES AND CHARACTERISTICS

The following are VEST modules with their capabilities:

1. MultiGen® modeling system: a CAD package developed by Software Systems to create and edit 3D geometrical objects.

2. DMA: a subsystem to generate terrain surfaces from any standard Level 1 DMA DTED tape. It was completed in Phase III.

3. CDE: a subsystem to animate traditional instruments. Most of its components were completed in Phase III. Extensions are made to display character strings, to correct the z-buffer problem, and to link instruments to dynamic data pool.

4. MFD: a subsystem to create MFD pages and simulate them. Most components are based on APACHE longbow configuration.

5. Jack Interface: a component to display and animate Jack images/objects in MultiGen® window environment.

6. VIEWS: a module to change camera views and control data flow between VEST and network simulation manager.

VEST operates within MultiGen® and thus uses MultiGen® windows for display and user interface. To run dynamic animation, communication network must be established between VEST and Simulation Executive. However, with some modification to the datapool interface, it can be used to animate any graphic images. In that regard, VEST can also run standalone animations.

3.4 SAMPLE OPERATIONAL SCENARIOS

VEST is a Mac-like software on top of MultiGen® user interface and is thus user friendly and mouse-driven . Using the menu bar and icons, the user specifies commands for VEST. Through dialog boxes, the user is prompted for inputs. The user first creates graphical objects in an organized tree structure with the animation in mind. Then, the designer uses CDE or MFD tools to link the graphical objects to datapool variables and starts animation through VIEWS functions.

4.0 REQUIREMENTS

4.1 REQUIREMENTS APPROACH AND TRADEOFFS

VEST is developed to provide an intuitive look at the progression of the Mission Simulation for cockpit designers and mission analysts. As a result, its requirements come from the needs of displaying the A^3I simulation at an acceptable speed. As a prototyping tool, MultiGen® is adopted as the underlying CAD software for rapidly creating 3D graphical objects. The tradeoff is that a license fee is needed for use of MultiGen® software. To model APACHE longbow MFDs which are not in production yet, updated documention is needed from McDonnell Douglas Helicopter Company.

4.2 EXTERNAL INTERFACE REQUIREMENTS

VEST is developed to run on SGI IRIS graphics workstations and Mission Simulation to run on Symbolics machines. To support the A^3I integration world, VEST is required to communicate with the Simulation Executive. This network interface is derived from the message formats and commands defined by the Executive.

VEST needs to be linked to part of Jack software. An interface is required to resolve the difference in orientation and scale. A solution to this is to keep Jack drawing routines from MultiGen® and define an interface matrix. Before any Jack drawing or reach, the MultiGen® world is modified by this interface matrix to let Jack perform it in its own environment. Since MultiGen® has its own color setup, the attributes of Jack sites need to be disabled.

For better performance, VEST is implemented on IRIS 4D/GT workstations.

4.3 REQUIREMENTS SPECIFICATION

4.3.1 Process and Data Requirements

VEST is required to be a process driven by the mouse and network communication. It runs in a loop waiting for mouse events or network messages. When a mouse is pressed, VEST operates within MultiGen® and takes user's commands through the pull-down menu or icons. It opens dialog boxes to prompt the user for more inputs when necessary. When a network message arrives, it follows the Executive's protocol and performs necessary actions. For each tick, VEST informs the Executive of Jack status.

MultiGen® provides a set of functions to a user to generate 3D graphical objects. As a subsystem, VEST provides the following major functions, through the pull-down menu, to meet Phase IV requirements:

CDE:
- Gauge Library: to create 3D graphical objects of gauges from library tools.
- Animation Linking: to define animation methods and link animation to the datapool variables.
- Color Table: to animate changes in color intensity of gauges.

VIEWS:
- Communication: to open a network socket to the Executive.
- Camera Views: to attach cameras to moving objects.
- Setup Script: to set up camera views from a script file.

MFD:
- Editing: to open the library tools to build MFD pages.
- Structure: to modify the MFD tree and pages.
- Animation: to page through the MFD tree.

JACK:
- Interface : to interactively define VEST-Jack interface matrix.
- Reach: to send a command to Jack to reach a MultiGen® polygon.

It is noted that DMA and most of CDE were completed in Phase III. VIEWS was also developed in Phase III but is restructured in Phase IV to support flexible and interactive views including the attach of a camera to the aviator's eye.

4.3.2 Performance and Quality Engineering Requirements

VEST provides a designer an interactive and 3D visual environment. Its performance is limited mainly by the graphics hardwares, MultiGen® Kernel and simulation network. Because of the enormously large database in the A^{3}I simulation world, the speed of IRIS 4D/GT is sometimes intolerable. The solution to this is to reduce the size and details. The MultiGen® Kernel converts floating numbers into integral numbers. During the conversion, some degree of accuracy is lost and it causes graphics problems in z-buffer and shading. The version of MultiGen® being adopted was implemented on old IRIS platforms and thus limits the quest for realistic graphical images. The animation of VEST is also tied to the performance of the Simulation models.

VEST is written following MultiGen® conventions which are well documented in Software Systems' manuals. It is portable on most SGI IRIS platforms.

4.3.3 Implementation Constraints

VEST is written in C and uses IRIS Graphics Library. It also opens a TCP/IP socket for network communication. It is implemented under the UNIX V operating system integrated with Berkeley (4.3 BSD) extensions and C compiler.

5.0 DESIGN

5.1 ARCHITECTURAL DESIGN

5.1.1 Design Approach and Tradeoffs

As mentioned before, the goal of VEST is to support A^{3}I simulation. For rapid development as a prototyping package, a set of library tools are built to simulate MultiFunction Displays and traditional gauges. The result is that VEST is not as flexible as it should be and its animation capabilities are limited.

VEST is developed in parallel to MultiGen® Kernel. Modules are designed to be independent as much as possible. However, to avoid high cost of development, CDE closely interacts with MultiGen® Kernel.

5.1.2 Architectural Design Description

VEST is designed on top of MultiGen® which is a window-oriented modelling system for generating 3D graphical objects. The structure of MultiGen® is well documented in Software Systems' manuals. The following figure illustrates the modules of VEST:

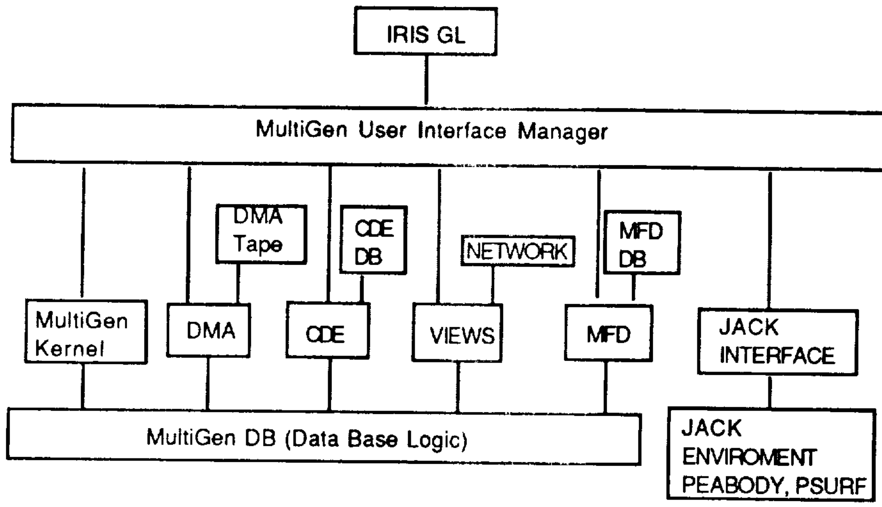

IRIS GL
MultiGen User Interface Manager
DMA Tape
CDE DB
NETWORK
MFD DB
MultiGen Kernel
DMA
CDE
VIEWS
MFD
JACK INTERFACE
MultiGen DB (Data Base Logic)
JACK ENVIROMENT PEABODY, PSURF

Figure 1. Modules of VEST

Under IRIS MEX window manager, the User Interface Manager accepts all the commands from the user and passes them to the appropriate subsystem. MultiGen® stores graphical objects in the Data Base Logic (DBL) format. CDE and MFD generate hierarchical descriptive database for the designer to evaluate cockpit prototypes. The Jack-VEST interface simplifies many Jack features and only supports FLAT-shaded images.

5.1.3 External Interface Design

When VEST is connected to the simulation network, it assumes no knowledge of the history and actions of the Executive. It is completely passive and takes mouse commands at the same time.

5.2 DETAILED DESIGN

5.2.1 Detailed Design Approach and Tradeoffs

VEST is designed top-down. Each module has its own mouse-driven event handler to handle events passed by the MultiGen® Interface manager. Under the handler, a set of tools are developed.

5.2.2 Detailed Design Description

5.2.2.1 Compilation Unit

The source files are organized in the home directory of VEST as follows:

MultiGen User Interface Manager (under Im/):

imcont.c	imde.c	imlib.c	immem.c	immenu.c
imtext.c	imwind.c			

MultiGen Kernel (under Mg/Ker):

mgcom.c	mgcont.c	mgdbvw.c	mgdesk.c	mgdev2d.c
mgfile.c	mgicnstr.c	mgiconst.c	mgicre.c	mgidisp.c
mgidl.c	mgiedit.c	mgifun.c	mgilib.c	mgiman.c
mginfo.c	mgipick.c	mgistruc.c	mgiutl.c	mgixform.c
mgmain.c	mgmath.c	mgpage.c	mgstd.c	mgtab.c

MultiGen DBL (under Mg/Flt):

fltcolor.c	fltdbl.c	fltid.c	fltlink.c	fltpage.c

DMA (under Mg/Dma):

terdma.c	terfont7.c	terpolt.c	terrain.c

CDE (under a3i/):

cdecolor.c	cdedefault.c	cdedisp.c	cdefile.c	cdefmter.c
cdefunc.c	cdegauge.c	cdelink.c	cdeneig.c	cdepage.c

VIEWS (under a3i/):

a3istamp.c	a3iutil.c	animate.c	aplselect.c	simdata.c

MFD (under MFD/):

mfdani.c	mfdbut.c	mfdcompass.c		mfdcont.c
mfddefault.c	mfddesk.c	mfddisp.c	mfdfield.c	mfdflpath.c
mfdhsd.c	mfdio.c	mfdlift.c	mfdmain.c	mfdmisc.c
mfdmod.c	mfdpage.c	mfdsprintf.c	mfdstruc.c	mfdtool.c
mfdway.c				

JACK Interface (under a3i/):
a3ijcl.c

JACK (under JACK/):

assign.c	delete.c	draw.c	jclinit.c	jclreach.c
msg.c	peaparse.y	psurfparse.y	stamp.c	
peabody/:	dof.c	error.c	expr.c	func.c
	globals.c	joint.c	keyword.c	metric.c

	name.c	new.c	reach.c	reachsite.c
	reachsites.c	scale.c	segment.c	symtab.c
	tree.c	update.c	value.c	verify.c
psurf/:				
	bin.c	edge.c	get.c	globals.c
	normal.c	read.c	reference.c	spec.c
	util.c			
vec/:				
	chunk.c	findfile.c	lexi.c	list.c
	matrix.c	mattorot.c	metric.c	pseud.c
	quaternoin.c	string.c	tomatrix.x	vector.c

VEST is implemented on IRIS workstations and linked to the Graphics Library.

5.2.3 Detailed Design of Compilation Units

5.2.3.1 MultiGen® User Interface Manager

This subsystem is a collection of procedures that support a mouse- and window-oriented user interface. It provides the interface between the user and application program. For details, refer to Software Systems' MultiGen®, Programmer's Guide to the Interface Manager, Release 3.0.

Four flags are added to the window structure in the file "imstrucs.h":

windowtype	a flag to indicate whether the window is created by VIEWS
mfdcontrol	a flag to indicate whether the window takes user's input
showmfd	a flag to indicate whether the window should display MFD pages
showjack	a flag to indicate whether the window should display Jack figures

5.2.3.2 MultiGen® Kernel

This subsystem is the core of the MultiGen® CAD package and contains all the editing functions for graphical objects. Detailed information can be found in Software Systems' MultiGen®, MultiGen® Kernel, Release 3.0.

The display component in the file "mgidisp.c" is modified to provide an interface for VEST. This unit controls the view of the window, CDE animation of graphical objects, MFD displays and Jack drawing. This display unit first sets up the camera view. When a window is marked by VIEWS, its camera view is calculated from the position and orientation of the moving object. The drawing component then recursively scans all graphical objects. If an object is marked by VIEWS, it is regarded as a world object. If an object is marked by CDE, it is regarded as an instrument. After all graphical objects are drawn, this display unit invokes the drawing routines of MFD and Jack.

The CDE module closely interacts with the MultiGen® Kernel. Refer to A^{3}I Phase III Cockpit Design Editor CSCI for the changes.

5.2.3.3 MultiGen® Data Base Logic

This subsystem handles MultiGen® database. Detailed information can be found in Software Systems' MultiGen®, Programmer's Guide to Writing MultiGen® DBL, Release 3.0.

5.2.3.4 DMA

This module is an option developed by A3I staff as an add-on to MultiGen® and later on modified by Software Systems to make it more user friendly. Its design can be found in A^{3}I Phase III Views CSCI and also in Software Systems' MultiGen®, DMA Terrain Data Conversion Option User's Guide.

5.2.3.5 CDE

The following figure shows the design structure of CDE-user interactions:

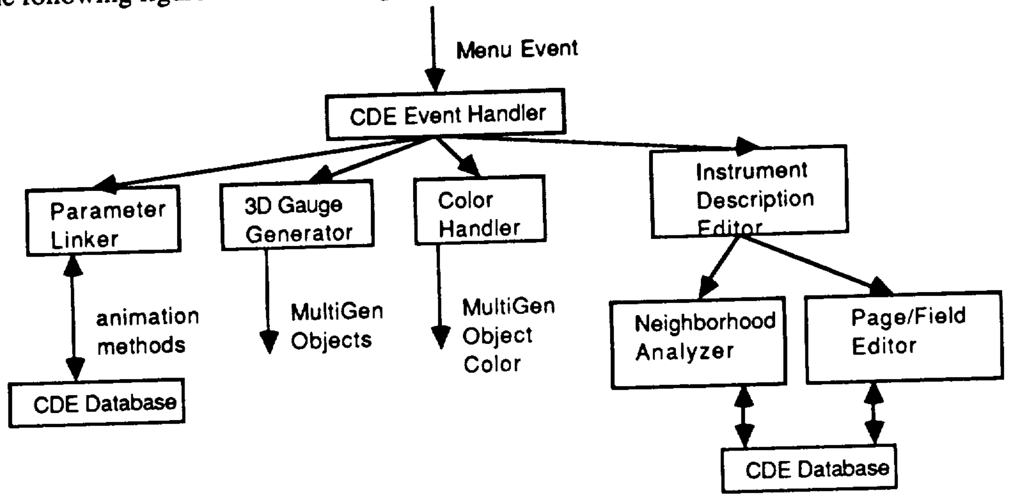


Figure 2. CDE Design Structure

The CDE animation is called by the MultiGen® display component, as showing in the following figure.

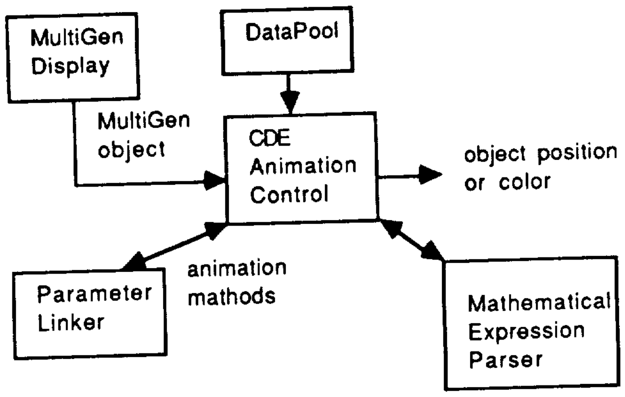


Figure 3. CDE Animation Structure

If the CDE animation involves any movement, it first modifies the matrix stack of IRIS GL to redefine the object position before drawing it and then restores the matrix stack after the drawing is finished. If the animation involves a color change, it redefines the RGB color of the object.

Most of CDE functions were completed in Phase III and documented in A^{3}I phase III Cockpit Design Editor CSCI. CDE module has an Animation Controller in "cdefile.c" and Parameter Linker in "cdelink.c". New animations can be easily added to these two components by following the data flow and MultiGen/CDE convention. As required, CDE is enhanced capable of displaying character strings. In Phase IV, CDE animation updates graphical displays by taking values from the datapool. The file "cdedefault.c" contains the addresses and types of datapool variables. When requested, this unit returns the value and/or type of a datapool variable.

Note that because of the enhancements made, the CDE cannot read old-version databases. The following sections discuss the extenstions to the CDE.

5.2.3.5.1 File cdefile.c

This file handles the CDE animation. Changes are made to the following routines.

Cdemenu(): This routine handles the menu event. If the event is for VIEWS, it is passed to the VIEWS event handler. If the event is due to the menu function **Link Animate**, it modifies the values of datapool variables by alternately opening three files and loading them into the datapool. This standalone animation continues until the mouse is pressed.

CDE_transformation(): This function is called by the MultiGen® display unit. It decides which animation method should be invoked. If the graphical object is for string display, the following steps are taken to ensure the z-buffer accuracy:

```
zwritemask(0x0);
draw_polygonal_surface(); /* within MultiGen */
CDE_show_str_disp();      /* draw the string */
zwritemask(0xffffff);
writemask(0x0);
draw_polygonal_surface(); /* draw the polygon again */
writemask(0xfff); /* depends on the MultiGen color map */
```

Exec_select(): This function controls most of the CDE animation methods. For a string display, it performs the necessary format conversion and then calls the string display function.

CDE_search_up(): This function is used by Jack to determine if an object is animated by CDE (the control sticks can be animated by the CDE tools). When an object changes the position and Jack's hand is on it, the new position must be obtained before Jack does the reach.

5.2.3.5.2 File cdelink.c

This file controls the user interface for defining the linking parameters and animation method. Extensions are listed as below:

Str_disp_setup(): This function opens the dialog box to take user's inputs for the parameters needed to display a character string. It allows a user to change the font size interactively.

CDE_show_str_disp(): This function displays a character string on the screen.

5.2.3.5.3 File cdedefault.c

This file defines datapool variables. There are three global variables available for all VEST modules:

datapool	a data structure same as the Simulation Executive.
cdedefvals	a reference id array used by the user to access the datapool.
comm_inmsg	the address of the datapool to store the network message.

CDE_open_default_db(): This function takes one argument as the filename. It opens the file and loads its content into the datapool. It then appends a file extension ".text" to the filename. If the file exists, its content is copied to the Text Clipboard of the MultiGen. A user can then open the Text Clipboard window to browse through the datapool variables. The first field in the window is the reference id, the second the current value, and the third the variable or instrument.

5.2.3.6 VIEWS

VIEWS was developed in Phase III and restructured in Phase IV to support dynamic camera views and datapool structure. The following shows the flow chart of VIEWS:

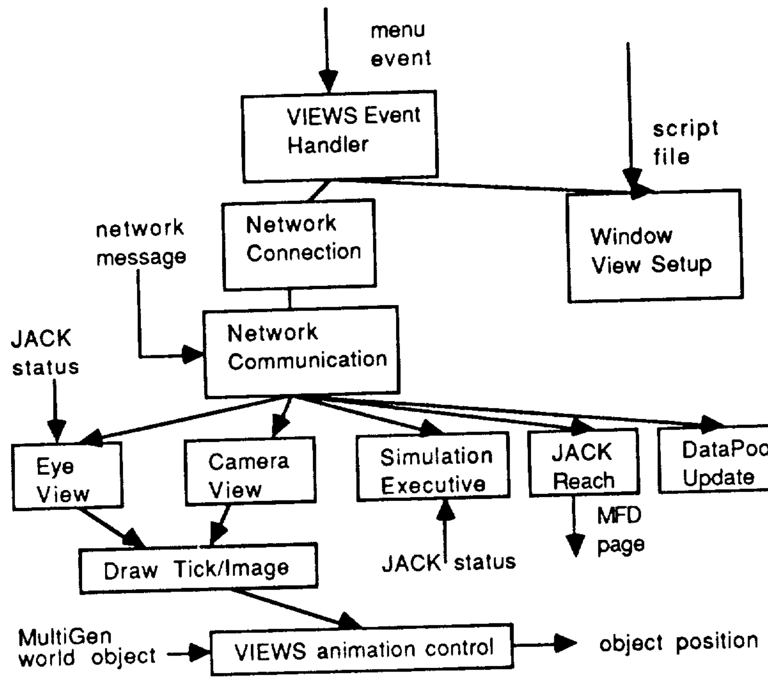


Figure 4. VIEWS Design Structure

In addition to the view setups, the VIEWS module also positions the world objects such as helicopters and tanks. The world object is referred to by an ID following the order defined by the Executive for the datapool structure:

1	lead helicopter
2	wing helicopter
3	tank
4	truck
5	jeep
6	launcher

In order to do the animation for the world objects, the graphical objects for a world object should be organized to be under the same parent node in the MultiGen® tree.

VIEWS assumes that the nose of the MultiGen® helicopter object points to the positive-x direction. It also assumes that the north of the terrain is in the positive-y direction, which is the case if the terrain is extracted by the DMA module. It is noted that the orientation of a static world object must be specified through the menu function "Setup" so that VIEWS knows how to rotate the object to the assumed heading direction.

5.2.3.6.1 File animate.c

VIEWS opens MultiGen windows but assigns a different set of functions for control windows to interactively adjust camera views. The world objects and window parameters can be modified in a setup window. For each tick, the moving camera view is re-calculated from the current position and orientation of the attached world object. The animation concept for world objects is same as that in CDE. In addition to animation, VIEWS contains a constrain table for Jack to perform unfinished reaches when a new tick is sent from the Executive.

Anamenu(): This function handles the menu event.

ANA_hide_window(): This function toggles between activating and deactivating a window. It can be used to deactivate a window when too many windows are opened on the screen. Its main use is to hide the database window of the MultiGen when VIEWS animation is in progress so that the drawing speed can be increased. Note that if the main window is closed, MultiGen frees the memory space and other windows should not try to refer to the database deallocated.

ANA_process(): This function processes the script file to set up VIEWS windows.

ANA_transformation(): This function first modifies the stack matrix for the world object, draws it, and then restores the stack matrix. The position and orientation of the object are taken from the datapool. This function also takes account of the scale and offset of the object as specified in the content of the setup window. Note that due to the assumed heading direction, there is a 90-degree difference in the yawing angle from the Aero Model.

ANA_pilotview(): The function is called by the display unit of MultiGen® to set up the perspective view of a window. From the world object to which the camera is attached, VIEWS performs the following steps to determine the viewing matrix:

> i) Determine the matrix so that after the world object is drawn, it remains at the same position and orientation in the IRIS world. This is the inverse of the function "ANA_transformation". The effect is that the rest of MultiGen objects are drawn in the local coordinates of the moving world object. At this step, the camera is fixed with the helicopter.

ii) The matrix is modified so that the local coordinates of the camera agree with the IRIS world.

iii) The above matrix is further modified so that the camera's view (local x axis) points to the minus-z direction and the camera's top (local z axis) to the positive-y direction of the IRIS world. The purpose of this transformation is to show the helicopter upward on the screen.

ANA_ethernet_ready(): This function checks if the content in the setup window has been processed and then tries to connect VEST to the Executive through a network socket.

ANA_communicate(): This function is called by the function IWM_waitforevent() inside the event loop of the MultiGen Interface Manager. It always calls SIM_read_network(). When a new-tick message arrives, it calls SIM_tickdone(), invokes a Jack routine to perform those reaches unfinished, redraws all the windows to reflect the current values in the datapool, and displays the tick number on the top window.

ANA_ibwindow(): This function follows the steps taken by the MultiGen function DBVW_ibwindow() to create a new display window. However, the viewing portion of the new window deals with moving camera views. The icons in the viewing portion can be used to adjust the camera position. Note that the camera position and orientation are defined in the local coordinate system of the moving helicopter. It means that to the helicopter, it is static; but, to the world, it is moving. The first dial of the viewing icon indicates the rotation in x, the second in y, and the third in z.

ANA_search_up(): The function is used by Jack and MFD to determine if a graphical object moves with the helicopter. If it does, its world position must be calculated before a reach or drawing is performed.

ANA_menu_focus_view(): This function calculates the camera position to highlight the selected MultiGen polygon in a blown-up view. It assumes that the window shows the perspective view.

ANA_draw_route(): This function draws the dynamic route defined and updated in the datapool. It draws a straight line between two waypoints.

ANA_terrain_coord(): This function calculates the relative location of a MultiGen point (external to the moving helicopter) in the local coordinate system of the flying helicopter.

ANA_get_win_fig(): This function returns the Jack figure to whose eye the camera is attached. It is used when the eye-view flag is set for a window.

5.2.3.7 MFD

The following figure shows the design of MFD:

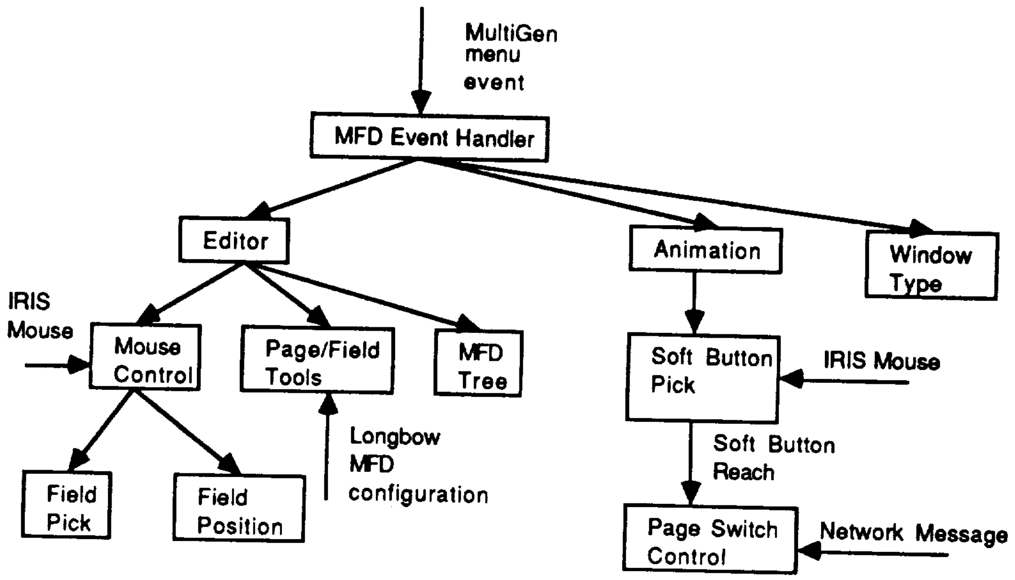


Figure 5. MFD Design Structure

The display of MFD is invoked by the MultiGen display component:

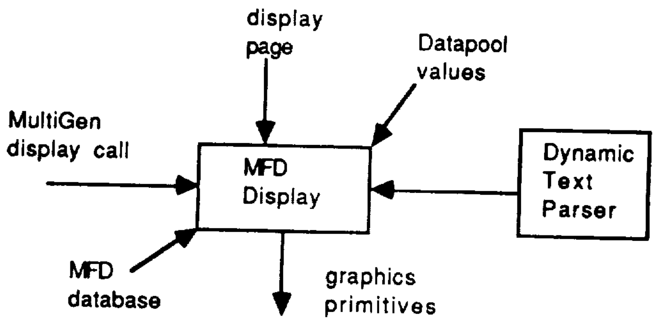


Figure 6. MFD Display Structure

5.2.3.7.1 File mfdani.c

This file contains functions to page through the MFD tree. It issues a Jack reach command for a MFD button and performs the page switch when the reach is done.

MFD_proc_bid_map(): This function processes the content in the page-map window. It assigns an initial display page to a MFD.

MFD_animate(): This function pops up a dialog box for the standalone page-switch animation. The reach for a soft button is achieved by clicking the mouse on it. The control procedure for mouse input is "MFD_animate_func()".

MFD_set_anipages(): After a reach is finished, this function is called by Jack module to assign a page to a MFD.

MFD_set_treenode(): This function performs the search for a page in the MFD tree when a button is pressed. If the "jump" flag of the button is set, it returns the page as specified. Otherwise, it searches the child pages of the current displaying page to find one that matches the button id.

MFD_net_reach(): This function is called by the network-interface procedures when a message is a reach for a MFD button. The steps it performs are similar to those taken by the function "ANA_animate_func()" when a button is pressed.

5.2.3.7.2 File mfdbut.c

This file contains functions to create soft buttons (see Section 5.2.3.7.14) and to calculate their 3-D locations for Jack reaches.

MFD_reach_sbutton(): This function first calculates the 3-D location of a button from the 2-D MFD database and then issues a Jack reach command.

MFD_reach_field(): This function determines the 3-D location of a field in the displaying MFD.

5.2.3.7.3 File mfdcont.c

This file contains functions to identify MFD screens from MultiGen objects.

MFD_base(): This function pops up a dialog box to identify a MultiGen object as a MFD (through the picking function) and its integral id.

MFD_face_matrix(): This function determines the relation between a 3-D MultiGen surface and 2-D MFD page. Before the MFD page is drawn, this matrix is loaded so that the page appears on the MFD screen.

MFD_refangle(): This function calculates the angles that can be used to rotate a vector about x and y axes to align it with the z axis.

MFD_adjustx(): Given a surface and the results for its normal vector from the function "MFD_refangle()", this function calculates the z-axis rotation angle to align the x axis.

MFD_mfdwindow(): to open a MultiGen window as the working MFD for editing.

5.2.3.7.4 File mfddefault.c

This file contains several routines to access the datapool variables and return their values.

MFD_mod_ctrl_by_page(): This function is called by the page editor to change the values of those variables controllable by VEST such as the scale and display format of NAV pages. The values of these variables may be also changed by the simulation models.

5.2.3.7.5 File mfddesk.c

This file contains functions to open a desktop window which shows a set of library tools and allows a user to select a MFD field (tool) to be added to a MFD page.

MFD_build_tool_window(): This function specifies the number of tools in the library. Whenever a new tool is added to the library, this number should be corrected accordingly. The tool name should also be added to the file "mfd.msg".

5.2.3.7.6 File mfddisp.c

This file contains functions to invoke drawing routines of MFD fields.

MFD_draw_picture(): This function controls the display of a MFD page. To draw a field, it first checks its type and then invokes the corresponding drawing procedure for the field. If an underlying page is specified, it draws it first by calling itself recursively.

MFD_pickfield(): The function is similar to the function "MFD_draw_picture()" except that the graphics state is set in the picking mode. It is called to determine the field which the mouse is pointed at.

5.2.3.7.7 File mfdfield.c

This file contains generic functions to add a field to a MFD page.

MFD_field_funcs(): This function provides a way to allow MFD tools to create their dialog boxes and set up those control functions when "OK", "CANCEL", and "TRY" buttons are pressed.

MFD_edit_field(): This function is called by MFD tools to allocate enough memory to store the attributes of a field and insert it into the linking list of the editing page.

MFD_field_func(): This function controls the mouse event when a field is being edited. If the left mouse is pressed, it positions the field to the cursor point. If the middle mouse is pressed, it puts the graphics state in the picking mode and the attributes of the picked field are copied to those of the working field.

5.2.3.7.8 File mfdio.c

This file contains functions to save/open MFD database. From the "filename" given by the user, the I/O routines open two files "filename.button" and "filename.tree". The file "filename.button" is a text file containing information about initial pages of MFDs. The file "filename.tree" is a binary file storing the MFD page structure and its fields.

MFD_save_db(): This function saves the MFD database in memory into disk files.

MFD_open_db(): This function reads the MFD database files into the memory. The reading procedure recursively calls itself to read in the MFD tree.

5.2.3.7.9 File mfdmain.c

This file contains functions to interact with MultiGen® Interface Manager and allocate/free memory space for a MFD tree.

MFD_init(): This function is called by MultiGen® to set up the menu functions of MFD. It also sets up a text window for page mappings.

MFD_menu(): This function handles the menu events.

MFD_mkbutton(): This function allows other functions to initialize the values of radio buttons in a MultiGen dialog box.

MFD_mkcheckbox(): This function allows other functions to initialize the values of check boxes in a MultiGen® dialog box.

5.2.3.7.10 File mfdmod.c

This file contains functions to select a field in the editing page.

MFD_mod_field(): This function is called by the page editor. It sets up a dialog box and puts the system in the picking mode. It controls the functions to delete a field, to change the display priority (order), and to invoke the corresponding MFD tool to modify its attributes.

5.2.3.7.11 File mfdpage.c

This file contains functions to create/edit MFD pages.

MFD_page_window(): This function sets up a dialog box and defines its control functions for editing. The control function for creating fields checks the MFD tool selected in the desktop window to invoke the right tool functions.

MFD_new_page(): This function creates a page structure, inserts it in the tree under the parent node, defines the default attributes for the new page, and then calls the function "MFD_page_window()".

MFD_mod_page(): This function loads the attributes of the selected MFD page in the editing buffer and then calls the function "MFD_page_window()" to modify the content of the page.

MFD_duplicate_page(): This function creates a page structure which copies the content and attributes of the selected page, inserts it under the parent node, and then calls the function "MFD_page_window()".

MFD_delete_page(): This function tries to remove a MFD page from the tree. If there is any child page under it, it does nothing and reports an error.

MFD_tool_button(): When a mouse is pressed in the MFD desktop tool window, this function is called to determine which library tool is to be invoked.

5.2.3.7.12 File mfdsprintf.c

This file contains functions to parse user's inputs for dynamic strings.

MFD_sprintf(): This function parses a character string from user input. If there is any dynamic portion of the input, it is converted into the string format. The dynamic field is a datapool-variable id in the input string which is enclosed by '<' and '>'.

5.2.3.7.13 File mfdstruc.c

This file contains functions to draw the MFD tree and to modify its structure.

MFD_redraw_struc(): This function draws the MFD tree in a MultiGen window (tree window). This window is created with scrollable bars and its use is similar to the MultiGen "Structure" window. The drawing steps are also similar to the MultiGen function "ISD_drawstruc()".

MFD_struc_pick(): This function is called when a mouse is pressed inside the MFD "tree" window. To save memory, the picking method is different from MultiGen. It sets the graphics state in the picking mode to load the selected page box into a picking buffer. After the picking process is finished, it scans the MFD tree again to find the page loaded in the picking buffer.

MFD_find_treenode(): This function scans the MFD tree to find the page whose id matches the input string.

5.2.3.7.14 Other Files

Most of these files contain MFD library tools used to add fields to MFD pages. A new tool can be easily added to the library by the following template:

MFD_ToolName_window(): This function is to open a dialog box to specify the attributes of the field.

MFD_new_ToolName(): This function is to specify the default attributes for the field.

MFD_set_ToolName_params(): This function is to process user inputs for the attributes.

MFD_select_ToolName(): This function is to set the attributes when the field is picked.

MFD_mod_ToolName(): This function is to set the attributes of the field when it is selected for modification.

MFD_dup_ToolName(): This function is o duplicate attributes of the field.

MFD_draw_ToolName(): This function is to draw the image of the field.

5.2.3.8 Jack Interface

Jack is a software package to observe how a human mannequin performs a task. The usage and detailed design of Jack is described in UPENN's manuals, Jack User's Guide (Version 4, 1989) and Programming with Jack (Dec. 21, 1988). In this document, only the Jack-VEST interface is discussed.

The figure below shows the design structure of VEST-Jack interface:

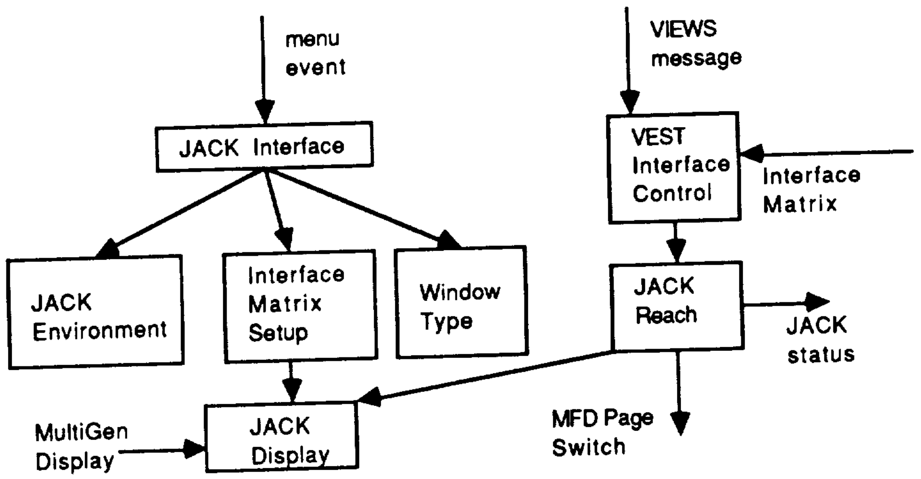


Figure 7. VEST-Jack Interface

VEST-Jack interface opens only a Jack environment file. Since MultiGen® does not support lighting models, the attributes of Jack defined in the file are ignored. To resolve the difference in the scale and orientation, an interface matrix must be defined before attempting to draw the Jack figure. The display of Jack is called from the MultiGen display function. Since VEST does not interact with Jack except the reach movement, it depends on a flag to determine if the Jack display function should be invoked. Before this function is called, VEST modifies the GL matrix stack by the interface matrix and then restores the matrix stack after the drawing.

When VEST receives a reach command, it puts it in a constrain table. For each tick, VEST follows Fitt's law and calculates the location that the movement can reach at. It then directs Jack to reach the point. Before a reach can be performed, the MultiGen® coordinates must be converted into Jack systems through the use of the interface matrix. When a reach is issued at a MFD button, this information is also stored. When the reach is completed, it invokes the page-switch function of MFD.

5.2.3.8.1 File a3i/a3ijcl.c

This file contains most of the procedures for VEST-Jack interface.

JCL_draw_picture(): This function controls the drawing of Jack figures and eye tracers. Before any drawing, the interface matrix is loaded. If the window is to show the eye view of a Jack figure, it is not drawn. If the figure is drawn, the graphical objects for the eyes may obscure the cockpit.

JCL_Mg2point(): Given a MultiGen point, this function calculates the corresponding location in Jack coordinate systems.

JCL_setup(): This function is called by MultiGen® Interface Manager to pop up a dialog box to interactively scale and move a Jack figure so that it looks correct in the MultiGen window.

JCL_process_param1(): This function processes the content in the parameter window to calculate the VEST-Jack interface matrix. See the Appendix for the data format.

JCL_menu(): The function handles the menu events.

JCL_init(): This function is called by MultiGen to initialize the VEST-Jack interface.

JCL_update_eyeview(): This function is called by the VIEWS module when a MultiGen window is set to the eye view of a Jack figure. This function assigns the middle point of the Jack figure as the camera's position and the vector from this point to the focus point as the camera's orientation.

JCL_do_ani_move(): This function goes through the constrain table and performs all unfinished reaches (for one-tick period). If a reach is marked as done but its target is marked as attachable (such as control sticks), the reach is always issued to Jack in case that the object changes its position and/or orientation as the simulation proceeds.

JCL_fishish_allmove(): This function is called in the standalone animation to finish all reaches in the constraint table.

JCL_reset_ani_move(): Any reach request first calls this function. The request is put in the constraint table and overrides the old content which has the same reach id. If the target is a MFD soft button, a flag is set so that at the end of the reach, the page-switch function of MFD can be invoked.

JCL_MgReach(): This function controls static reaches for MultiGen surfaces by popping up a dialog box.

JCL_new_eyetrace(): This function stores the positions of the eyes for a Jack figure into a record. This record can be used to draw the eye tracers and to determine the viewing direction of the Jack figure.

JCL_pilotstatus(): This function loads the status of both hands and viewing direction into the datapool buffer so that the network procedures can send them to the simulation models.

JCL_jpoint2MG(): Given a Jack point, this function calculates the corresponding location in MultiGen coordinate systems.

5.2.3.8.2 File JACK/jclreach.c

This file controls Jack reach and head-turning movements. Using Fitt's law, it calculates the point to be reached at for the tick. It then issues the reach command to Jack. The functions are taken from Jack source files with modifications.

5.2.3.8.3 File JACK/jclinit.c

This file opens the environment file of Jack if the environment variables "PEALIB" and "PSURFLIB" are defined. Since VEST does not load Jack environment, its purpose is to avoid the segmentation-fault error caused by the peabody-parser when these two variables are not defined.

5.2.3.8.4 File JACK/stamp.c

This file contains those dummy routines called by the peabody-parser and psurf-parser, but useless in VEST.

5.2.3.8.5 File JACK/assign.c

This file is taken from Jack source file peabody/assign.c. It contains those dummy functions for Jack attributes which are disabled in VEST.

5.2.3.8.6 File JACK/delete.c

This file is taken from Jack source file peabody/delete.c. A function is added to it to remove an old environment when VEST opens a new Jack environment.

5.2.3.8.7 File JACK/draw.c

This file controls the drawing functions of Jack images. It disables all Jack attributes. Most drawing routines are taken from Jack source file psurf/draw.c.

5.2.3.8.8 File JACK/peaparse.y

This file contains the peabody-parser. It is modified to ignore Jack attributes.

5.2.3.8.9 File JACK/psurfparse.y

This file contains the psurf-parser and is a copy of Jack source file psurf/rp.y.

5.2.3.8.10 File JACK/msg.c

This file contains the methods to display Jack messages. Since VEST has its own user interface, all Jack messages are either ignored or redirected to the standard output.

5.2.4 External Interface Detailed Design

As mentioned before, VEST is driven by the Executive. During the simulation, the "redraw" event of each window is mostly determined by the VIEWS module. The decision is based on the network messages from the Executive.

5.2.4.1 File simdata.c

This file precesses network messages. When a message is received, it checks the message type and performs data conversion. The messages can be a reach for a MultiGen object, a reach for a MFD soft button, or updated values for datapool variables.

SIM_connect_to_host(): This function opens a TCP/IP network socket for read and write. VEST processes the user's setup and determines the location of the Executive.

SIM_read_network(): This function uses the UNIX system call "select()" to determine if a message has arrived. If no message is in the queue, it returns immediately so that VEST can process any mouse event.

SIM_handle_msg(): This function checks the message type and determines what action to perform. If the message is to update datapool variables, its body content is copied to the datapool buffer.

SIM_do_reach(): The reach message has two fields: reach id and target site. The reach id is an index into the reach table specified by the user in the parameter file for Jack interface. If the first character of the site name is 'L', the rest is the button name for the left MFD. If the first character is 'R', it is for the right MFD. If the first character is 'E', it is a MultiGen object outside the helicopter. Otherwise, it is a MultiGen object moving with the helicopter.

SIM_tickdone(): This function sends the aviator's status if it is available and a "tickdone" message to the Executive.

SIM_proc_script(): This function reads a script file and processes it to set up MultiGen databases, window positions, VIEWS setups, CDE databases, MFD databases and Jack databases. Each line in the script file has two fields. The first field is the command and the second the value.

5.2.5 Coding and Implementation Notes

The executable file of VEST created from one IRIS 4D/GT machine may not run well on other 4D/GT series. In order to be portable, the shared libraries should be used in the linkage. Internally, MultiGen® converts user input into integral numbers and may overflow the graphics engine, especially on the personal IRIS machines. If the screen shows weird images, try to change the resolution (COM_internal_resol) in the file mgcom.c to a lower number. However, to make the conversion process accurate, this number should be as large as possible.

6.0 USER'S GUIDE

6.1 OVERVIEW OF PURPOSE AND FUNCTIONS

This section describes how to use VEST software and its installation procedures. Before attempting to use it, the user should have a basic understanding of MultiGen®. VEST/MultiGen is a program for creating and editing 3D graphical objects and providing tools for viewing the simulation sequence. It is controlled by using the mouse and pointing the cursor at icons and menus on the graphics workstation display.

6.2 INSTALLATION AND INITIALIZATION

The following procedure shows how to install the VEST software:

1. Put the source tape of VEST in the tape driver.
2. Change the directory to the destination that will be the home directory of the package. Assume it is "/usr/VEST".
3. Type "tar xvo" and wait for downloading the files. A directory structure will be created as follows:

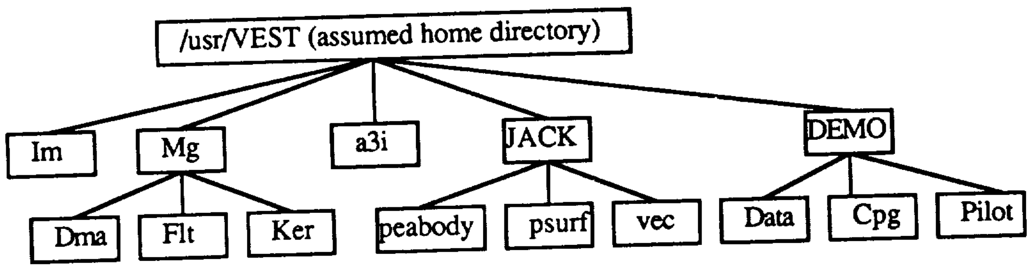


Figure 8. VEST Source files

4. To create a new executable VEST, type "make" at the home directory of VEST (in this example, the home directory is /usr/VEST). It is noted that the workstation is assumed to be IRIS 4D/GT. If it is an IRIS 2500T, get rid of the "-DOPT_GT" from the CFLAGS in the Makefile.

6.3 STARTUP AND TERMINATION

To bring up VEST, change the directory to the home directory of VEST and type "mgvest". After the desktop of MultiGen is set up, the user may choose the options under the "I/O" pull-down menu to open an existing database, create a new database, or quit.

6.4 FUNCTIONS AND THEIR OPERATION

6.4.1 Phase IV Demo

The following is a summary of step-by-step procedures for running VEST standalone demonstration through a script file.

1. Change the directory to the home directory of VEST.

2. Issue the command "mgvest -DEMO/demosim" and wait until the automatic setup is done.
3. Close the top window which shows a moving camera view for integration demo.
4. Resize the instrument window to expose the covered dialog boxes. This window demonstrate the CDE capabilities.
5. Click on the Comm Flag command of the CDE menu.
6. Click on the Link Animate command of the CDE menu to start the CDE animation.
7. Press the mouse three times to stop CDE animation.
8. Click on the Demo Window of the MFD menu to bring up MFD windows.
9. Click on the Link Animate command of the CDE menu to simulate MFD dynamic fields. Press the mouse once to stop the animation.
10. Click on the Hide Window command of the MFD menu to bring up a MFD tree window.
11. Click on the Animate command of the MFD menu to page through the MFD.
12. Bring the tree window and then MFD window to the top.
13. Point the cursor at MFD soft buttons and press the mouse.
14. Close the MFD window and click on the DONE button of the dialog box (animation window).
15. Quit from VEST.

6.4.2 CDE Menu

Open Coord. File opens an external file and converts its data into MultiGen format. The format is contained in Appendix.

New Gauge pops up a menu that contains a list of predefined instruments. Select the desired instrument and click on the "OK" button. A pop-up dialog box will appear for input. Click on the "SHOW" button and then follow the instructions on a control window to define the location of the instrument.

Process Link computes the 3D animation data for each instrument in the link list which is created by the **Load Link File** and/or **Link Parameters** commands. Before any CDE animation, this command should be issued.

Load Link File opens a binary link file and appends it to the CDE parameter link list.

Save Link File saves the current CDE parameter link list into a binary file.

Link Parameters pops up dialog boxes to define the animation method on MultiGen objects and linking parameters for animation. The parameter ID is the variable ID in the first field of the Text Clipboard window after the **Datapool Default** is loaded. For the animation method, please refer to Phase III CDE CSCI. The string display is an interactive tool. It needs inputs for the starting point (a MultiGen vertex), local x vector (a MultiGen edge), and local z vector (a MultiGen polygon).

Color Palette brings up the CDE color-palette window. The paired colors are designed to animate the on/off switch of warning lights and numerical leds. When the writemask flag of a CDE instrument is set TRUE in the **Link Parameters**, the unerasable colors can cover-up the erasable colors. This property is to provide some "windowing"-effect animation for instruments such as ADI and Drum Dial gauges.

Insert Color modifies the color of the selected MultiGen object.

Modify Color modifies the RGB of the selected color from the CDE color palette.

Mod Attributes allows the user to browse through the CDE link list and opens the **Link Parameters** window to modify the linking parameters of the selected instrument.

Data Format O/P allows a user to convert MultiGen database to other formats.

Link Animate opens some external files to update current values in the VEST datapool and thus shows CDE standalone simulation. This command can also be issued for standalone animation of MFD dynamic fields.

Unlink All removes the CDE link list.

Open Setup loads the named file into the setup window for VIEWS animation.

Save Setup saves the content of the setup window into a file.

Process Setup processes the content of the setup window and then opens MultiGen windows for VIEWS animation.

Setup brings up a text window to define VIEWS setups. The format is contained in Appendix.

Ethernet opens a network socket to the Simulation Executive. It also sets the **Comm Flag**.

Reset suspends the communication process and also resets the **Comm Flag**.

Hide Window toggles between showing and hiding a MultiGen window.

Focus View shows a blown-up view of the selected MultiGen polygonal surface in VIEWS animation.

Comm Flag sets the animation flag to tell the display module of MultiGen to call CDE or VIEWS functions for animated objects.

DataPool Default loads the named file into the VEST datapool. The file "default" in the directory "DEMO/DATA" is supplied. Once it is loaded, the user may refer to it for the variable id through the Clipboard window of the MultiGen menu function "TEXT".

Simulation Script open a script file to set up MultiGen windows and animations for CDE, VIEWS, MFD and JACK. This is designed for Phase IV demo.

Track Window toggles between showing and hiding the MultiGen track window.

Show/Hide Path toggles between showing and hiding the flight path.

6.4.3 MFD Menu

It is noted that most of the time, when a MFD dialog box pops up, it disables MultiGen menu and dialog-box functions. The user must take action against the dialog box before issues any other commands.

MFD Base pops up dialog boxes to define MultiGen objects as MFDs. Each MFD should have a unique ID between 1 and 10.

MFD ID defines the editing MFD for New Page and Mod Page. This allows a way to switch MFDs with different physical sizes such as Longbow UFDs.

MFD Show/Hide toggles between showing and hiding MFD display in the top MultiGen window.

New Page pops up library-tool windows to create a MFD page which is inserted as a child of the parent page. However, if no working MFD is defined, this function does nothing. A MFD can be defined by the command **MFD Base** or **Open/New**. If no parent page is specified, the selected page, if it exists, is taken as the parent page.

Mod Page allows a user to modify the selected MFD page.

Dup Page duplicates a selected MFD page which is inserted as a child of the current parent page. The command also invokes **Mod Page** for the duplicated page.

Delete Page removes the displaying page of the editing MFD from the tree.

Select Page searches the MFD tree and tries to find the page with the name matching the user's input. The user may also opens the tree window and uses the mouse to pick the page.

Attach moves the selected page to be a child of the current parent page.

Detach removes the selected MFD page from the tree.

Set Parent sets the selected page as the parent page.

Page Map opens a text window for user's input to set up MFD animation. The format is contained in Appendix.

Structure opens a MultiGen window to show the MFD tree. A user can use the mouse to select a MFD page.

Animate allows a user to page through the MFD tree. The dialog box allows the user to enter the reach id for Jack and a reach for a waypoint.

Open/New opens the MFD database files created from the same MultiGen objects.

Save saves the MFD database into a file.

Close removes the MFD database from the memory.

Lift Page allows a user to draw the MFD pages above the MultiGen object in order to solve the z-buffer problem.

Open/Old loads the MFD database created from another set of MultiGen objects. Before this command is called, the MFD IDs should have been defined by **MFD Base** or **Open/New.**

6.4.4 JACK Menu

New Env opens a Jack environment file which defines names of Jack figures and their configurations. To locate Jack image files, the two environment variables, "PEALIB" and "PSURFLIB", must be defined before VEST is run. If VEST is running, these two can be defined in a script file through the command **Simulation Script**.

Param Window brings up a text window for defining parameters of the VEST/Jack interface. The format is contained in Appendix.

Setup pops up a dialog box to interactively refine the position and scale of a Jack figure.

Reach Face performs Jack reaches at MultiGen polygonal surface. The user may do a single reach or multiple (two) reaches.

Jack Show/Hide toggles between showing and hiding displays of Jack figures in the top window.

Open Param loads a file into the parameter window.

Save Param saves the parameters of the VEST/Jack interface into a file.

Process computes the 3D relations between Jack and VEST from the content of the parameter window.

Tracer Show/Hide toggles between showing and hiding eye tracers.

6.5 ERROR AND WARNING MESSAGES

MultiGen communicates with the user through dialog boxes. When an error occurs, the bell rings once and the user may see the error message using the "Last Error Msg" option under the "INFO" pull-down menu. Press the mouse once and the error -message window goes away.

6.6 RECOVERY STEPS

A core-dump file is produced when VEST aborts abnormally.

7.0 ABBREVIATION AND ACRONYMS

A^3I	Army-NASA Aircrew/Aircraft Integration
DMA	Defense Mapping Agency
CDE	Cockpit Display Editor
MFD	Multi Function Display
UFD	Upper Front Display
VEST	Visual Editor and Simulation Tool

8.0 GLOSSARY

9.0 NOTES

9.1 MISCELLANEOUS

9.2 LIMITATIONS

The limitations for using VEST are:

1. All graphical objects have to be in a single MultiGen database file.

2. The network interface must follow the data structure of the Executive. As a result, only the instruments of the lead helicopter can be animated.

3. The library tools of MFD are not "soft" enough and are limited to the APACHE longbow configuration.

4. All MFDs share the same database and global variables and cannot display their own setups such scales on NAV pages.

5. VEST Jack can only perform the reach movement and head turning.

9.3 LESSONS LEARNED

VEST is a mouse-driven process. When an event arrives, it goes through almost the same checking steps. Therefore, when a new tool is added, it is possible to duplicate previous effort by adding new codes but performing similar functions. Before tools are added to MultiGen/VEST, one should try to figure out what functions are currently available. In doing this, one should also try to make the tools as independent as possible. If not, the code may become untraceable.

9.4 FUTURE DIRECTIONS

Future enhancements should consider the following aspects:

1. Performance

The performance of VEST is affected by MultiGen displaying module for extra checkings. The animation component of VEST should be separated from MultiGen to by-pass all those unnecessary checkings. Computations for data conversion from network messages into CDE/MFD internal format should be reduced as much as possible.

2. Portability

The animation part of VEST is too closely tied to the data structure of the simulation Executive. A set of internal data structures should be used and some external functions are provided for data conversion.

3. Flexibility

The capabilities of VEST are limited by the set of built-in tools. It will be better if external libraries are provided. A user can then edit the objects in the libraries or add new objects to the libraries.

10.0 APPENDICES

APPENDIX A. — VEST FILE FORMATS

```
/**************CDE COORD FILE FORMAT ************/
Getnconvert (fpath)
FILE *fpath;
{
int i, j, k,l, ncluster, nobject, nface, nvertex, color;
float x, y, z;
char dummy[80];
        fscanf ( fpath, "%d", &ncluster);
        for (l=0; l<ncluster; l++)
        {
                fscanf ( fpath, "%d", &nobject);
                for (i=0; i<nobject; i++)
                {
                fscanf ( fpath, "%d %s", &nface, dummy);
                for (j=0; j<nface; j++)
                {
                        fscanf (fpath, "%d %d %s", &nvertex, &color, dummy);
                        for (k=0; k<nvertex; k++)
                          fscanf (fpath, "%f%f%f", &x, &y, &z);
                }
                }
        }
}
```

```
/****** VIEWS setup format ****************/

        !OBJECT SECTION
        1 LEAD 16.0 16.0 16.0 -12.3 0 0 x 0 y 0 z 180
        2 WING 16.0 16.0 16.0 -12.3 0 0 x 0 y 0 z 180
        3 TRUCK 2.0 2.0 2.0 0 0 0 x 0 y 0 z 0
        #starfish 1039 1041
        1 1 -14.5 5.5 15.0 x 0 y 6 z -45 7 1 pilot
        2 1 -14.5 5.5 15.0 x 0 y 6 z -45 7 1 pilot
```

WORLD OBJECT SECTION
It starts with "!" and is followed by descriptions of the MultiGen objects. These objects are animated by their locations and orientations as sent by the Executive. Note that the rotation angles must be specified in such a way that the heading points to the positive x direction and the top points to the positive z direction.

Filed 1 = Object ID as is defined by the Executive.
Filed 2 = MultiGen Name for the Object.
Filed 3 = X scale.
Filed 4 = Y scale.
Filed 5 = Z scale.
Filed 6 = X offset from the terrain origin.
Filed 7 = Y offset from the terrain origin.
Filed 8 = Z offset from the terrain origin.
Filed 9 = axis of rotation.
Filed 10 = degree of rotation.
Filed 11 = axis of rotation.
Filed 12 = degree of rotation.
Filed 13 = axis of rotation.
Filed 14 = degree of rotation.

WINDOW SECTION
It starts with "#" followed by the host-name of the Executive, socket port of the Executive and own socket port. The section body has 14 fields:

Field 1 = Window ID.
Field 2 = Object ID to which the camera is attached.
Field 3 = X offset of the camera from the object origin.
Field 4 = Y offset of the camera from the object origin.
Field 5 = Z offset of the camera from the object origin.
Field 6 = rotation axis for the next field.
Field 7 = rotation degree of the camera.
Field 8 = rotation axis for the next field.
Field 9 = rotation degree of the camera.
Field 10 = rotation axis for the next field.
Field 11 = rotation degree of the camera.
Field 12 = an ORed flag for the window type.
    JACK_FLAG (=1) displays JACK figures.
    MFD_FLAG (=2) displays MFD pages.
    TRACER_FLAG (=4) displays JACK eye tracers.
    EYEVIEW_FLAG (=8) attaches the camera to a JACK figure.
Field 13 = flying speed for interactively adjusting camera positions
    in a way similar to change of viewer-position in MultiGen.
Field 14 = name of JACK figure needed when EYEVIEW_FLAG is set.

```
/***** MFD Mapping  format ******************/

        !
        1 COMMTOP
        2 NAVTOP
        3 UFD
        4 CKYBD
```

INITIAL PAGE SECTION
It starts with "!" and is followed by descriptions of the page names.

Filed 1 = MFD ID.
Filed 2 = name of the Initial Displaying Page on the MFD.

```
/***** JACK Parameters format ******************/
        !MOVE
        1 pilot.left_fingers.distal pilot.waist 2
        2 pilot.right_fingers.distal pilot.waist 2
        +ATTACH
        ccoll
        ccyclic
        #pilot 1 1 2
        C 133
        F 0.076068
        S 20.000000 20.000000 10.000000 50.000000
        R z 1800 x -900 y 0
        P PCHAIR1
        M -0.823516 -2.331641 -2.268802
        I 1 MFD
        I 2 ccoll
```

MOVE SECTION
This section starts with "!" and contains the lookup table for a reach ID sent by the Symbolics models.

Field 1 = reach ID. Note that a reach ID between 20 and 29 is reserved for head/eye turning. The symbolics models uses this range for CPG. The VIEWS module which controls the network communication takes of the difference.
Field 2 = end effect of the reach.
Field 3 = end joint of the reach.
Field 4 = a flag to determine whether to turn head as the reach is performed. When the value is 0, there is no change in eye focus. When the value is 1, the focus of the eyes is set to the target. When the value is 2, the head is also turned to the target.

ATTACH SECTION

This section starts with "+" and takes only one field. The field is the MultiGen object which the end effect of a JACK reach is constrained and attached to after the reach is done.

INTERFACE SECTION

This section starts with "#" and defines the VEST/JACK interface and others. Following "#", it takes 4 fields as follows:

Field 1 = a string to identify a JACK figure in the environment file.
Field 2 = a flag to identify the JACK figure as the pilot or CPG.
    When the flag is 0, no network status is sent to the Symbolics for this figure. When the value is 1, a CPG status is sent out. When the flag is 2, a pilot status is sent to the Symbolics.
Field 3 = any reach id associated with the left hand.
Field 4 = any reach id associated with the right hand.

The rest of this section defines the interface, color, and initial positions:

'C' defines the color index for the figure.
'F' defines the scale for the figure. For a one-to-one mapping, this value is 1/12 since one interface unit is 12 JACK units.
'I' defines the initial position of the figure. Field 1 is the reach id and Field 2 is the target (MultiGen object).
'M' defines the offset of the JACK figure from the seat. Note that one interface unit is 12 JACK units.
'P' defines the seat for the JACK figure, which is a MultiGen object.
'R' defines the orientation of JACK. Its format is "axis degree axis degree axis degree". Note that the degree is expressed in tenths.
'S' defines the incremental speeds in three axes for interactively adjusting the JACK position when Setup Menu function is invoked. Note that one interface unit is 12 JACK units.

Annex F

Army-NASA Aircrew/Aircraft Integration Program: Phase IV

A^3I

Man-Machine Integration Design and Analysis System (MIDAS)
Software Detailed Design Document

Display Layout Analysis

prepared by

Carolyn Banda and Michael Prevost

Table of Contents

Table of Contents

MAN-MACHINE INTEGRATION DESIGN & ANALYSIS SYSTEM (MIDAS) SOFTWARE DETAILED DESIGN DOCUMENT PHASE IV:

DISPLAY LAYOUT ANALYSIS

1.0 INTRODUCTION

1.1 IDENTIFICATION OF DOCUMENT

This is the Preliminary Design Document for the Display Layout Analysis module of the MIDAS Software System for Phase IV. MIDAS stands for Man-machine Interface Design and Analysis System.

1.2 SCOPE OF DOCUMENT

The Display Layout Analysis module was developed rapidly to demonstrate proof of concept at the Phase IV A^3I demonstrations. Accordingly, the emphasis in this document is on the purposes and concepts of this tool, methods and approaches, and a preliminary design. Broadly stated, the Display Layout Analysis module explores the idea of assisting human-machine interface designers with a computer-based tool which incorporates human factors design knowledge concerning spatial location of information to be displayed to human operators of the machine. This document is directed toward three categories of readers: 1) readers with an interest in this subject and issues involved in developing such a tool, 2) readers who wish to learn what the Display Layout Analysis tool's current and future capabilities are, and 3) those who wish to use the tool in its preliminary form and explore its existing features. Familiarity with C and CLIPS, an expert system building tool written in C at Johnson Space Center, is helpful but not necessary.

1.3 PURPOSE AND OBJECTIVES OF DOCUMENT

This document provides a description of the purpose and approach of the Display Layout Analysis module for the reader interested in an overview as well as a description of the module's preliminary design. Also included is a discussion of issues and questions that have come up thus far. A user's guide is available to assist in use and exploration of the tool.

2.0 RELATED DOCUMENTATION

2.1 APPLICABLE DOCUMENTS

CLIPS Reference Manual, Artificial Intelligence Section, Johnson Space Center, May 1989.

Giarrantano, Joseph C., *CLIPS User's Guide*, Artificial Intelligence Section, Johnson Space Center, June 1989.

Hartley, Craig S. and Rice, John R., "A Desktop Expert System as a Human Factors Work Aid", *Proceedings of the Human Factors Society,* 31st Annual Meeting, 1987.

Haskell, Ian D., Wickens, Christopher D. and Sarno, Kenneth, "Quantifying Stimulus-Response Compatibility for the Army/NASA A^{3}I Display Layout Analysis Tool", *Proceedings of the 5th Mid-Central Human Factors/Ergonomics Conference*, 1990.

Human Engineering Guidelines for Management Information Systems, DOD-HDBK-761, 1985.

Kirlik, Alex, notes from "Display Layout Analysis", presented at A^{3}I Offsite Planning Conference, Asilomar, CA, May 1989.

Wickens, Christopher D., Section on Display Layout Analysis in "Use and Integration of Models", in *Human Performance Models for Computer-Aided Engineering*, ed. by Elkind, Card, Hochberg, and Huey, National Academy Press, Washington, D.C., 1989.

Witken, Andrew, and Kass, Michael, "Spacetime Constraints", *Computer Graphics*, 22:159-168, 1988.

Woods, David D., and Eastman, Mary Claire, "Integrating Principles for Human-Computer Interaction into the Design Process: Heterarchically Organized Principles", in *Proceedings for 1989 IEEE International Conference on Systems, Man, and Cybernetics*, November 14-17, 1989, Cambridge, Mass.

2.2 INFORMATION DOCUMENTS

The following references were not used directly in the Phase IV version of the Display Layout Analysis tool but are related to this general problem area and may be useful in future work on the tool.

Beshers, Clifford M., and Feiner, Steven, "Scope: Automated Generation of Graphical Interfaces", *Proceedings of the ACM SIGGRAPH Symposium on User Interface Software and Technology*, Williamsburg, Virginia, Nov. 13-15, 1989.

Hix, Deborah, "A Procedure for Evaluating Human-Computer Interface Development Tools", *Proceedings of the ACM SIGGRAPH Symposium on User Interface Software and Technology*, Williamsburg, Virginia, Nov. 13-15, 1989.

Joy, Kenneth, "A Model for Graphics Interface Tool Development", *Proceedings of Graphics Interface* 1985.

Olsen Jr., Dan R., "An Editing Model for Generating Graphical User Interfaces", *Proceedings of Graphics Interface/Vision Interface* 1986.

Palmiter, Susan L. and Elkerton, Jay, "Evaluation Metrics and a Tool for Control Panel Design", *Proceedings of the Human Factors Society*, 31st Annual Meeting, 1987.

Perlman, Gary, "An Axiomatic Model of Information Presentation", *Proceedings of the Human Factors Society*, 31st Annual Meeting, 1987.

Singh, Gurminder and Green, Mark, "Automatic Generation of Graphical User Interfaces", *Proceedings of Graphics Interface/Vision Interface* 1986.

Singh, Gurminder and Green, Mark, "Chisel: A System for Creating Highly Interactive Screen layouts", *Proceedings of the ACM SIGGRAPH Symposium on User Interface Software and Technology*, Williamsburg, Virginia, Nov. 13-15, 1989.

Tommelein, Iris D., Johnson Jr, M. Vaughan, Hayes-Roth, Barbara and Levitt, Raymond E., "SIGHTPLAN: A blackboard expert system for construction site layout", in *Expert Systems in Computer-Aided Design*, Gero, John S., ed., North-Holland, New York, 1987.

Tullis, Thomas S., "A system for evaluating screen formats", *Proceedings of the Human Factors Society*, 30th Annual Meeting, 1986.

3.0 CONCEPT

3.1 DEFINITION OF DISPLAY LAYOUT ANALYSIS TOOL

3.1.1 Purpose and Scope

Design of the human-machine interface for use in complex, dynamic environments is a formidable task. A good design will assist the human operator's performance and a poor design will hinder it. A prototypical version of a Display Layout Analysis tool was developed to explore the concept of using a computer-based tool with human factors design knowledge to guide the process of designing the human-machine interface. Knowledge in human-machine interface design can range from detailed, low-level guidelines such as display luminance and font sizes to higher level issues such as placement of displays in the overall design configuration. As its name suggests, the Display Layout Analysis tool is focussed on assisting the designer in the placement of displays which provide information to the operator from various information sources.

This tool currently addresses the domain of aircraft crewstation display panel design. However, it is expected that the methodology and indeed many of the same design guidelines and constraints could also be applied to human-machine interface design in other domains, such as wind tunnel display and control panels.

In designing crewstation display panels, a number of issues arise:

1) What information does the pilot need for effective job performance? (e.g., altitude, airspeed)
2) What are the sources for this information? (e.g., barometric altimeter and radar altimeter, airspeed indicator)
3) If we consider the crewstation display area to be the design space, what display media (which may use different modalities) are available to be placed in this space?
4) How can displays for information sources be grouped or used singly and allocated to the various available display media?
5) How can these media, which provide different types of display surfaces, be located in the design space?

The emphasis in the Phase IV demonstration tool is on assisting the designer with the simplest case--that is, placing dedicated displays on the control panel. However, a conceptual framework has been developed to address the more general cases, which use advanced display media such as Video Display Units (VDUs), Multi-Function Displays (MFDs), Heads-Up Displays (HUDs), Helmet Mounted Displays (HMDs), Audio Displays (of type tone or message), Tactile Displays (e.g., switch), and others. Windows for these advanced display types (media) appear on the screen for the DLA tool to indicate they will be available in the future.

3.1.2 Goals and Objectives

Evidence supports the validity of the human factors design metrics described in the NRC report (see the Section on Display Layout Analysis). Our goal was to test the feasibility of using these human factors design metrics to guide the designer in the area of display layout. A short development period of about 6 months led to rapid prototyping of a demonstration tool to investigate the following questions:

1) Can we build a design-aiding tool which uses the proposed human factors design metrics to guide the process of designing display layout?
2) Can data for these metrics be reliably gathered?
3) What type of graphical user interface to the tool would best assist the designers?
4) How could we structure the tool so that its architecture will support design assistance in definition and placement of advanced display media, such as MFDs, HUDs, HMDs, as well as traditional display layouts with dedicated displays?

Another issue was to investigate using two approaches in combination, a pro-active analytic design-aiding mode and an evaluative rule-based advisory mode.

For Phase IV, the emphasis has been on concept exploration. The Display Layout Analysis tool is in the beginning stages of development; a prototype was developed quickly to test out the general feasibility and utility of such a tool and to obtain feedback from attendees of the A^{3}I Phase IV demonstrations.

3.1.3 Approach

After Woods and Eastman (1989), we consider displays to be viewing windows for the operator through which he/she obtains information about machine and environment status. The Display Layout Analysis tool assists the crewstation designer in laying out the displays which provide the pilot with windows to the information sources which he/she needs to successfully operate the aircraft. Display layout assistance is guided by a set of design metrics incorporated into the tool, which can be operated both in an analytic, design-aiding mode and in an evaluative mode. Just as the spatial locations of objects in the physical world are completely determined by the forces acting upon them, the locations of the information sources being placed by designer are determined by the human factors-related forces acting on them. In the physical world, the forces are entities such as gravity and table tops; in the Display Layout Analysis tool environment, the forces which influence the spatial locations of displays are engineering psychology principles. Based on the cumulative forces of the relevant relations, the displays can be relocated to optimize the overall design with respect to human factors. Used in the evaluative mode, the tool employs human engineering design guidelines to examine an existing design and issue warnings for violations of the guidelines. A verbose warning mode is available in which a rationale behind the warning is also displayed as well as applicable literature references.

The human factors guidelines currently available in the Display Layout Assistant were obtained from the section on Display Layout Analysis by Christopher Wickens in *Human Performance Models for Computer-Aided Engineering* (1989) by the National Research Council. These guidelines are described below. Estimates for how strongly display pairs are related to each other in various dimensions were obtained from pilots using questionnaires as was other display-related information.

1) Displays for information sources which are highly related should be located close together; "highly related" is measured by a "task proximity" metric which is a weighted combination of the functional proximity, physical proximity, and correlational proximity for the information sources. Definitions for these measures are:

    - *Functional proximity* is based on an estimate of the extent to which two indicators must be integrated or compared in performing a task. For example, all indicators of system flight dynamics (airspeed, power setting, bank, pitch, etc.) will have close functional proximity.
    - *Physical proximity* is based on the similarity between the physical sources of the two indicators of each pair of displayed sources. For example, two indicators of rotor functioning are more similar than one of rotor functioning and one of navigational functioning.
    - *Correlational proximity* is the degree to which the values shown by two indicators are correlated over time. For example, airspeed and power would be highly correlated but bank and altitude might have low correlation of values; that is, the aircraft might be just as likely to be level as banked, in low and high flight.

2) According to symmetrical location compatibility (one form of stimulus-response compatibility), displays should generally be located on the same side as the hand controlling the parameter being shown. That is, displays that provide information relevant to left-handed controls should be positioned to the left of those providing information relevant to right-handed controls and vice versa.

3) Displays which have high frequency of use are strongly attracted to the prime area on the display panel, which is a "T" shaped area covering the top of the panel next to the windscreen and the panel center.

With advisory assistance from Christopher Wickens and from Jerry Murray and Loran Haworth (helicopter pilots), a questionnaire was designed to gather data from pilots for the human factors metrics: functional, physical, and correlational proximities, location of control action, frequency of use, and criticality. (Criticality information was not used in Phase IV, but will likely be used in the future.) An unexpected finding was that, of the three proximity measures, physical proximity appeared to be the most ambiguous, perhaps due to the interrelated nature of some aircraft subsystems. Two pilots completed the questionnaire; of course, a larger sample size will be required for future data gathering from pilots.

3.1.4 Description of Component Architecture

At an overview level of description, the Display Layout Analysis tool consists of:

- functions to provide a direct manipulation graphics interface and visualization methods for assessing design merit; also, menus to provide access to the tool's capabilities (user interface module)
- functions which allow the user to create icons and edit and optimize (manually and automatically) the existing design (design module)
- functions for displaying human factors relationships (analysis module)
- an embedded rule-based advisor to check an existing configuration of displays for constraint violations (advisor module)
- a module which contains data for the human factors design guidelines (human factors data module)

- a module which contains geometric data and equipment models (models module)
- functions to save and recall designs in progress (housekeeping module)

All modules except the Advisor's rules are written in C; the advisor rules are written in CLIPS format. (CLIPS itself is written in C.) Figure 1 shows the relationship of these components to each other and to the user/designer.

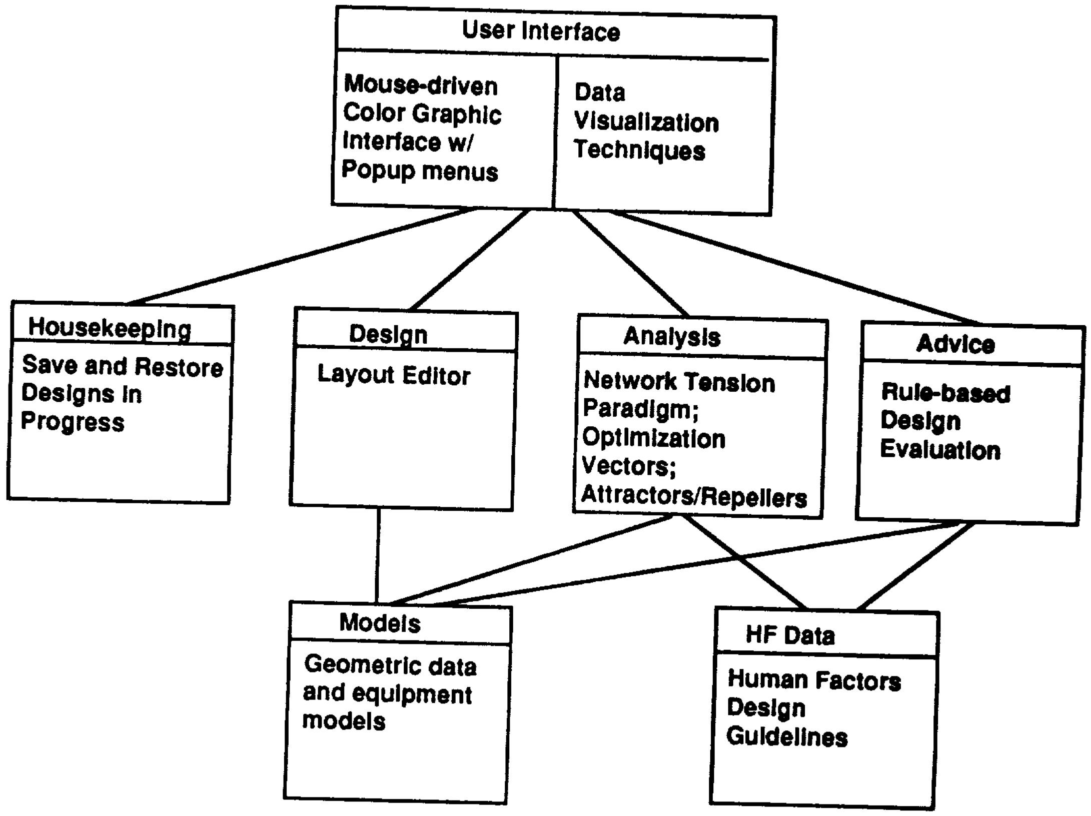

User Interface
Mouse-driven Color Graphic Interface w/ Popup menus
Data Visualization Techniques
Housekeeping
Save and Restore Designs In Progress
Design
Layout Editor
Analysis
Network Tension Paradigm; Optimization Vectors; Attractors/Repellers
Advice
Rule-based Design Evaluation
Models
Geometric data and equipment models
HF Data
Human Factors Design Guidelines

Figure 1. Overview of DLA Tool Structure (Preliminary Design)

The Display Layout Analysis tool is currently a stand-alone module constructed to explore the ideas of using human factors metrics expressed both in mathematical and heuristic forms to guide the design process. Since it deals with crewstation displays, it is likely that it will be integrated with the Crewstation Design Editor (CDE) in the future.

3.2 USER DEFINITION

Users of the Display Layout Analysis tool are expected to be designers of human-machine interfaces who want to take into account human factors design guidelines 1) when grouping and placing displays for information sources on the various types of display surfaces and 2) when locating these display media in the design space (i.e., crewstation display areas). The designer uses the Display Layout Analysis tool to directly manipulate the placement of displays through which the pilot is to obtain information about environment and aircraft status.

3.3 CAPABILITIES AND CHARACTERISTICS

The Display Layout Analysis tool provides a color graphics interface with a variety of features to assist in placement of displays, given their relationships to each other and to various areas of the panel. The panel contains areas of attraction for frequently used displays and can also contain areas of repulsion to indicate panel space which should be used sparingly or not at all. In addition, an Advisor can be invoked to examine the current design configuration and issue warnings for any violations of the human factors design guidelines.

A description of using the Display Layout Analysis tool follows. Using the Edit capability of the tool, the designer builds a set of pages by selecting the primitive information sources from a library of sources; a page can contain one or more information sources. Pages can be collected in groups or used singly; a Multi-Function Display would use a set of pages. The set of pages is mapped to a display surface (medium), such as dedicated display, Video Display Unit (VDU), Multi-Function Display (MFD), Heads-Up Display (HUD), Helmet Mounted Display (HMD), Audio Display (of type tone or message), Tactile Display (e.g., switch), and others. The resulting display surface is then mapped to a physical coordinate system, which can be fixed, as with a Heads-Up Display, or moving, as with a Helmet Mounted Display. This process is diagrammed in Figure 2.

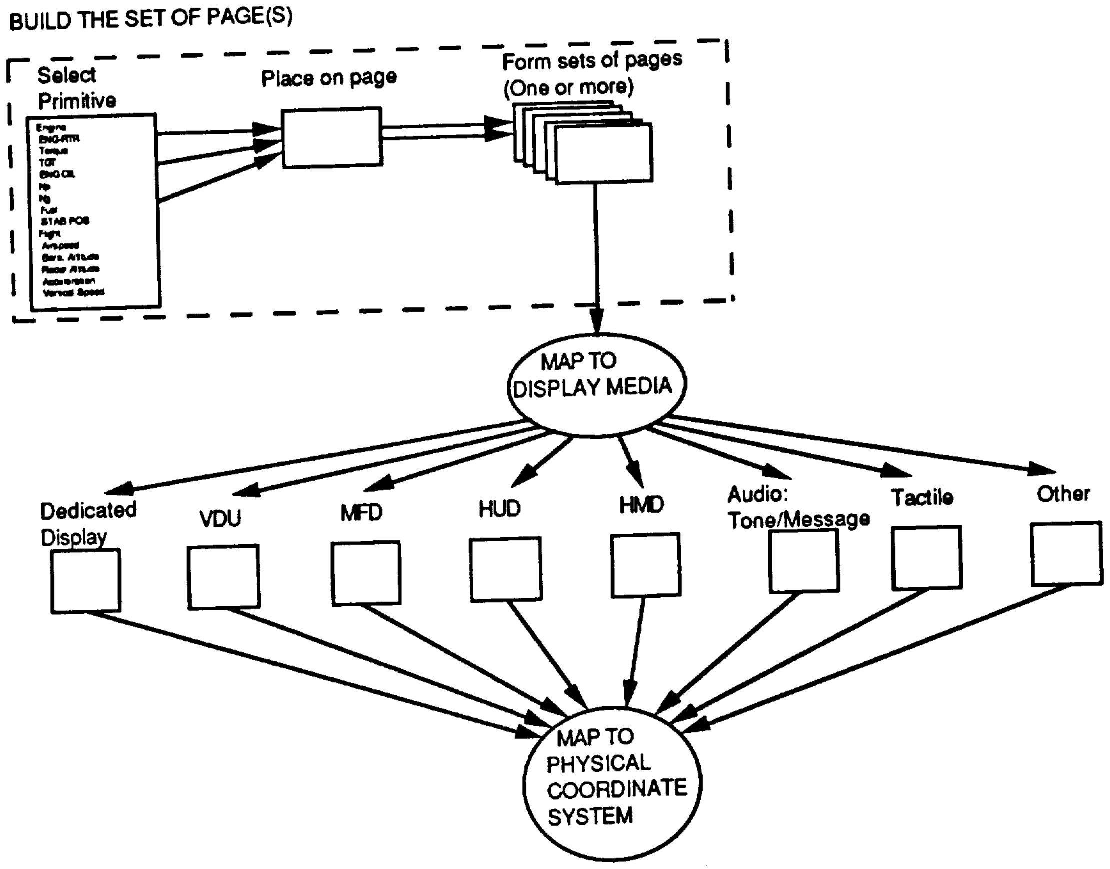


Figure 2. Building the database of information sources

The DLA tool has two modes: math-based design and analyze mode (continuous) and rule-based evaluate mode (discrete). Typically the user would build and analyze a design, then evaluate it by running the rule-based Advisor. These modes are described in more detail below.

Analytic Mode: The previously described human factors design guidelines are embedded in the Display Layout Assistant. They form networks of relations between information sources with other information sources, between information sources with controls, and between information sources with regions of the display surfaces, such as attractor and repeller areas of the display panel. (Hereafter, "source" in this paper refers to information source.) In the task proximity network, for example, network nodes represent information sources and connecting arcs represent the task proximity relation, with arc thickness indicating strength of the relation. After Alex Kirlik (1989), the forces in the arcs act like springs. The designer can show relations for a single information source, a set of sources or all sources.

Both manual and automatic optimization modes are available. The user can optimize the design manually by showing the optimization vector for a display icon, which points in the direction that reduces tension from other related sources, and moving the icons toward the location indicated by the optimization vector. The user can also select a group optimization mode which automatically moves displays to reduce network tension for the group. In both cases, the effects of moving the display icons are seen graphically. A global network tension measure indicates overall layout optimization according to the task proximity metric.

Rule-based Evaluation Mode: An Advisor can be invoked which issues warnings for detected violations of display layout guidelines. When operated in "verbose" mode, the Advisor lists a rationale for each warning as well as any associated references in the human factors literature.

3.4 SAMPLE OPERATIONAL SCENARIOS

For the Phase IV prototype DLA tool, thirteen representative information sources were selected from various areas of functionality, including flight, navigation, and communications functions. These information sources appear as prepositioned dedicated display icons on the the crewstation panel. This set of information sources represents less than one quarter of the total number of sources which would be eventually necessary in typical rotorcraft.

In a typical use of the prototype DLA tool, the designer would perform the following activities:

- show the various relations available (source-source, source-panel, source-associated control),
- observe and possibly move the attractor and repeller regions on the display panel,
- show optimization vectors for selected sources,
- interactively move display icons which represent information sources to new positions in the display space, while observing optimization vectors and relation network,
- invoke automatic optimization for a selected set of sources, and
- invoke the Advisor to check the design and issue warnings (if applicable) for deviations from the human factors design guidelines incorporated into the tool.

These actions are described in more detail in Section 6.0, Users Guide.

4.0 REQUIREMENTS

4.1 REQUIREMENTS APPROACH AND TRADEOFFS

With respect to requirements, the tool should:

- incorporate valid human factors design guidelines
- be easy to understand and easy to learn
- be responsive since it is being used to assist the design process
- automatically assist the designer where appropriate
- provide intuitive feedback on the goodness of the design and how to change it in a way that will improve the design

4.2 HARDWARE ENVIRONMENT

The Display Layout Analysis tool runs in a stand-alone mode on the the SGI-4D family of workstations. At this point, there are no special memory and disk space requirements. The DLA tool fits within the minimum configuration; however, this will likely change in the future as the tool is expanded.

4.3 SOFTWARE ENVIRONMENT

The analytic portion of the tool and the graphics interface are written in C and depend on the SGI (Silicon Graphics Iris) GL library for graphics and window functions. The rule-based Advisor is written in CLIPS, an expert system building tool written in C by the Artificial Intelligence Section at Johnson Space Center. Government agencies and their contractors can obtain CLIPS free of charge by calling the CLIPS Help Desk between the hours of 9:00 AM to 4:00 PM (CST) Monday through Friday at (713) 280-2233. Others can obtain CLIPS for about $350 from COSMIC; 382 E Broad St.; Athens, GA 30602; phone: (404) 542-3265.

4.4 EXTERNAL INTERFACE REQUIREMENTS

During Phase IV, the DLA tool was constructed as a stand-alone program and therefore has no interface requirements with other programs. Eventually it is anticipated that it will be integrated with the Cockpit Display Editor (CDE) and other equipment models.

The DLA tool was required to be easy to use; the purpose of the tool's interface is to give the designer an iconic representation of the display/control panel and easy access to the tool's capabilities. The tool should also be very responsive to the user since a slow response time would greatly diminish the tool's usefulness.

4.5 REQUIREMENTS SPECIFICATION

4.5.1 Process and Data Requirements

In the future, equipment data may require large amounts of memory and disk space; in fact, the DLA tool itself will require more CPU processing as new capabilities are added and the number of information sources is expanded.

4.5.2 Performance and Quality Engineering Requirements

Since the DLA tool is to be used in the design process for manipulating icons in the design space, it is required to be highly interactive. The designer needs almost instantaneous response time.

4.5.3 Implementation Constraints

Since MIDAS is intended to be distributed in whole, or in parts, to the user community at a low cost, use of low cost or free-of-charge tools is encouraged to avoid large license fees. This requirement had an impact on choice of the expert system building tool.

5.0 DESIGN

5.1 ARCHITECTURAL DESIGN

5.1.1 Design Approach and Tradeoffs

To collect data for the human factors design metrics, a questionnaire (shown in Appendix A) was designed. The data gathered from administering the questionnaire formed the relations between displays with each other and with areas of the design space and with associated controls. For a start, the questionnaire was given to two AH-64 pilots whose responses agreed fairly well in some areas and differed widely in others. To be valid, a larger sample will be needed in future work if data is to be gathered from pilots; however, the data gathered was sufficient to build a prototype tool for demonstration purposes. Eventually some of the necessary data may be obtained from MIDAS simulation results. Currently, each element in the task proximity matrix is computed by weighted average of functional: physical: correlational relations data using a ratio of 2:1:1. This weighting is subject to future change.

Under implementation issues, it was decided that the DLA tool would run on the SGI machines to take advantage of their color graphics capabilities and to be compatible with the Cockpit Display Editor (CDE), with which the DLA tool will eventually integrated. Color is used to assist the design and evaluation process; for example, borders of display icons for which warnings are issued are colored red to indicate a warning message applies to them.

After consideration of several expert system building tools, CLIPS was chosen since it is written in C and will run on the SGI machines and because it met our need for a proof of concept prototype. It was decided that the DLA tool should reside on a single machine to avoid heavy network penalties during run time; this ruled out use of ART. Also, CLIPS source code is available to government users at nominal cost and distribution costs are lower than with a commercial expert system shell.

Good program design (ease of maintenance) dictated the need for a single copy of the human factors design data; this copy is used by both the analytic and the rule-based portions of the tool.

5.1.2 Architectural Design Description

The data containing the human factors guidelines resides in matrices and vectors and is accessed by both the analytic and rule-based portions of the tool. (CLIPS rules call C functions to access the data.) Functional, physical, and correlational proximity relations between display pairs are stored in matrices, while frequency and location of control action are located in vectors. For each display pair, a weighted average of functional, physical, and correlational proximity relations was computed to form the task proximity relation, which forms one of the networks used in design analysis.

Four housekeeping functions include file operations which allow the user to save designs in progress and recall them later. (These are currently inactive.)

The design functions, which are to be used in close conjunction with the analysis functions, allow the user to move icons around in the design space; the icons represent information sources required by the pilot. The analyze functions allow the user to view relations between displays relative to the current design space; these relations appear as colored networks with color indicating the type of relation. The user can also show optimization vectors for display icons which indicate the direction and force with which an information source is pulled. There is an "automatic optimization" feature, which repositions all selected display icons into their natural "resting" place, given the attractor and repeller forces acting on them and tension from related information sources.

5.1.3 External Interface Design

The user interface uses color graphics to show its components: the design space, display icons, and windows for display media. This interface is highly mouse-driven in that the user accesses the tool's options by mousing on popup menus and moves display elements around in the display space by dragging them with the mouse.

5.2 DETAILED DESIGN

5.2.1 Detailed Design Approach and Tradeoffs

Since DLA was intended only as a prototype many of the design decisions were made based on what was most expedient in both time and money and availability. The optimizer for DLA is one area that needs to investigated further.

DLA uses an iterative solver for a damped oscillator to optimize the IS locations. This solver is not guaranteed to converge on the best solution because it can get stuck in local minima. This attribute does not appear to be a problem in practice because the theoretical maximum may not be the best but only another good solution, due to the heuristic nature of the metrics. The pseudo-code for this algorithm follows:

```
for all j do
        relation_vector = $\sum_{i=1}^{n}$ relation[j,i] * Distance[j,i])
        force_vector = $\sum_{i=1}^{q}$ force[i] * Frequency[j] *cofactor[i,j]
        tension = force_vector + relation_vector
        step_dir = compute_step_dir ( j, tension )
        if unoccupied( j, step_dir) then move ( j,step_dir)
end for all j
```

Here j is the index to the information source that is to be moved, n is the number of ISs, q is the number of forces, and cofactor[i,j] is the percentage of force[i]'s attraction/repulsion that is applied to this information source. Relation[j,i] then is the numerical strength of the relation between IS[j] and IS[i]. Distance[j,i] is the distance between the centers of IS[j] and IS[i]. Frequency [j] is the relative frequency with which IS[j] is accessed by the pilot during task performance. Each IS can only move one step per iteration and only if there is a vacant cell at that location.

The big advantage of this simple solver is that it works fast and provides solutions not easily found by the designer. As the optimization is being performed, the intermediate solutions are displayed, providing the designer with an animation that shows the general trend of the solution.

The optimizer does not work well for multimode solutions spaces and will get trapped in local maximums. For this reason it is suggested that a second, more cpu intensive, algorithm is implemented that could be called when the first method is unsatisfactory.

5.2.2 Detailed Design Description

5.2.2.1 Compilation Unit

The following source files comprise the DLA tool:

- ece.h
- metrics.h
- main.c
- windows.c
- menus.c
- hmd_win.c
- main_win.c
- mfd_win.c
- clips_win.c
- draw_area.c
- icon_win.c
- create.c
- metrics.c
- task_prox.c
- dla-fns.c
- select.c
- optimize.c

5.2.2.2 Detailed Design of Compilation Units

Not done since current version of DLA tool is a prototype.

5.2.3 External Interface Detailed Design

Not done since current version of DLA tool is a prototype.

5.2.4 Coding and Implementation Notes

The human factors design data describing information sources and their relations to each other is stored in matrix and vector form:

- 3 2-D arrays for functional, physical, and correlational relationships
- 1 vector for frequency of use
- 1 vector for location of control action
- 1 vector for criticality (not used yet)

The contents of the DLA tool Makefile appear below:

```
#
#       makefile ece
#

EXE = ece
#INCLUDEDIR = /usr2/prevost2/panel/include
#LIBDIR= /usr2/prevost2/panel/lib

#INCLUDES= -I${INCLUDEDIR}
#HEADER= ${INCLUDEDIR}/panel.h
```

```
LIB= ../libclips.a ../mfd/libmfd.a

OBJ=   main.o  windows.o menus.o hmd_win.o main_win.o mfd_win.o\
     clips_win.o draw_area.o icon_win.o create.o metrics.o task_prox.o\
     dla-fns.o select.o optimize.o

SRC = $(OBJ:.o=.c) ece.h Makefile

#$(OBJ) : $(SRC)

ece : $(OBJ) Makefile ece.h metrics.h
        $(CC) -o $(EXE) $(LDFLAGS) $(CFLAGS) $(OBJ) $(LIB) -g -lgl_s -lm
```

6.0 USERS GUIDE

6.1 OVERVIEW OF PURPOSE AND FUNCTION

The Display Layout Analysis program is a computer-aided design tool with built-in human factors knowledge which can be invoked to guide the design process and evaluate designs in progress with respect to layout of displays. The tool addresses design of display panels for man-machine systems; the current focus is on rotorcraft crewstation display panel design. The tool was constructed as a demonstration version using rapid prototyping in order to test the feasibility of this type of aid for designers. The tool is in the beginning stage of development. Currently included are three human factors design guidelines (suggested by Christopher Wickens in the NRC report, 1989) which pertain to display layout; more are envisioned for the future. A number of capabilities are also envisioned, some of which are indicated in the menu selections and display media windows. Currently dedicated displays are emphasized; in the future, more complex display media and surfaces will be included, such as VDUs (Video Display Units), MFDs (Multi-Function Displays), HUDs (Heads-Up Displays), and HMDs (Helmet Mounted Displays). The menu structure is not in its final form; however, enough features exist to give the user the look and feel of what a human factors based design-aiding tool might provide. The planned DLA design process is shown in Figure 3. The user first selects one or more displays and places them on a set of pages; dedicated displays have one page, while MFDs have multiple pages. The set of pages is then mapped to a display medium, such as panel (for dedicated display), MFD, HUD, or HMD; finally this medium is mapped to a physical coordinate system. Currently dedicated displays are emphasized although other display surface types such as VDU, MFD, HUD, HMD, and audio are shown. The current (demonstration) version of the tool has thirteen pre-selected information sources on the panel. The user can move these information sources around, show relationships between them and the design space, and show optimization vectors using design-aiding features of the DLA tool. The user can also call on the rule-based Advisor to evaluate the design at any time. This process is described in more detail in Figure 2 in Section 3.3 (Capabilities and Characteristics).

The domain currently addressed is rotorcraft crewstation panel design; on the main screen of the Display Layout Analysis tool is a panel outlined in red. Thirteen display icons representing information sources are available for manipulation and testing; eventually many more information sources will be available in a library.

For the three human factors design metrics currently in use, data includes functional, physical, and correlational proximities (2-D matrices), frequency of use (vector), and location of control action (vector). Definitions for these metrics are given in Sec. 3.1.3 (Approach). A description of the tool's capabilities appears in Section 3.3 (Capabilities and Characteristics); instructions for using these capabilities follow.

6.2 INSTALLATION AND INITIALIZATION

The required DLA program files are ece (for ergonomic crewstation editor), which is the executable file for the DLA tool, and two data files, **dla.clp** (contains Advisor facts and rules in CLIPS format) and **dla-data.clp** (contains Advisor facts in CLIPS format).

6.3 STARTUP AND TERMINATION

Once CLIPS and the program files for the DLA tool are installed, go into the ../clips/ece directory and enter:

```
% ece
```

This loads and initializes the DLA tool, bringing up a color screen with a crewstation panel outlined in red and pre-selected information sources which appear as display icons on the panel. A number of display media appear as windows; these indicate possible display surfaces which will be available in a future version of the DLA tool. Click the left mouse button once on the small box in the upper left corner to "park" these windows out of the way; click left twice on a window icon to re-open it. Hint: if display icons or media flicker, multiple left clicks will refresh the screen and stop the flicker. A text window appears below the DLA panel for displaying messages from the DLA tool and communicating with the Advisor.

To gain access to the DLA tool's features, press down on the right mouse button to make the main menu appear. Menu items with subsidiary menu items have a right arrow next to them. When the desired menu item is under the cursor, release the mouse button, and the menu item will be activated. To exit from the DLA tool, select "Quit", a top-level menu item.

6.4 FUNCTIONS AND THEIR OPERATION

First, a word about strategy and the assumptions underlying the DLA tool. Information sources are related to each other and to areas on the panel by a set of relations which are among the human factors design metrics encoded in the DLA tool. These relationships form a network with nodes representing the information sources and arcs representing the relationships. Arcs have varying strengths, indicated graphically by line thickness. These strengths are based on the strength of the relationship, which can range from "not related" to "strongly related". The network can be envisioned as a network of springs which has a tension or desire to contract by pulling related nodes together. A global network tension is expressed on the screen by a number, which is based on the total lengths and strengths of all the arcs in the network. The goal of the designer is to lessen the network tension (and lower the tension number) by moving related displays together and by moving frequently used displays toward attractor areas of the design space. The attractor areas appear as green bars and can be placed wherever the designer feels is an optimum place for the pilot/operator to view displays. Likewise there is a moveable repeller region which will repel displays. This might be placed in the lower right corner if the pilot is expected to be wearing a monocle over the right eye, for example.

The current menu structure is preliminary and is meant to show the possibilities; it will undoubtedly change in the future. Menu options are described below, including those which are not yet active. Note that menu selections are made with the right mouse button, while display icons are manipulated with the left button. Certain menu options put you into a new mode; for example, "edit move" (described below) allows you to move display icons; you leave the mode by entering "escape". Many options function in a toggle mode; that is, the first invocation activates some display option, while the second one deactivates that option. For example, selecting the option "show all relations" displays the relevant networks and selecting this option again turns this display off. (Again, sometimes you have to do a multiple left click anywhere on the screen to have the action take effect.)

* File options: These include Load, Save, Relations, and Others. They are not yet active but are included to show a method of saving and recalling designs in progress.

* Edit options:
    - Source: Source here means information source and its 3 sub-options are active:
        - Move: Enter "move mode" to move display icons around the design space; enter escape to leave move mode.
        - Nail: Enter "nail mode" to fix the location of designated displays; enter escape to leave nail mode. The center of the display icon turns red to indicate it is nailed down.
        - Revert: Returns all displays to their initial location.

- Region: Edit region is inactive.
- Relation: The edit relation items (functional, correlational, and physical) are inactive.

* Create options:
- Add information source is inactive, but will be used to bring new information sources into the design space.
- Source items are inactive, but show the categories of information sources which will be available in a library (Communication, Navigation, Engine, Flight, Hydraulic, Environmental Control System (ECS), Electrical, Auxiliary Power Unit (APU), Lighting, Target, Other)
- Region: Add region is inactive.
- Relation: Add relation is inactive.

* Show options are active:
- All relations: shows task proximity network and frequency network for all information sources. Task proximity connections are shown in purple; attraction forces of sources to selected spots on the attractor bars are shown in green. In both cases, the strength of the connection is indicated by line thickness. Use Revert to turn these networks off.
- Local relations: enter "show local" mode; clicking left on a display icon causes the task proximity network for that information source to be shown in purple to indicate graphically how the selected information source is related to other sources in the design space. Enter "escape" to leave the "show local" mode.
- Optimization Vector: enter "show optimization vector" mode; to show a vector for any information source, simply move the cursor over it and click left mouse button. Leave this mode by entering escape.

* Tools option is active and currently contains the Advisor and optimize options:
- Advisor capabilities include:
  - Evaluate: run the Advisor on the current design configuration. The advisor looks at the locations of all displays, checks for violations of the human factors design metrics encoded in its rules, and issues applicable warnings in the text window. When the Advisor is invoked, it asks "Print rationale and references for warnings? (y/n)"; enter "y" to obtain verbose printout mode.
  - Reset: reset the advisor. Note: the Evaluate option also resets CLIPS before performing the evaluation. Having a separate Reset option allows the user to view the agenda before evaluation and to step through the agenda, one rule at a time (see Step).
  - Show Facts: list all facts currently in the Advisor's fact base in the text window.
  - Show Agenda: list contents of the agenda in the text window.
  - Step: Fire one rule from the top of the agenda; used in debugging.
  - Watch: turns on the "watch" mode for facts, activations, compilations (currently not used), or all of these, which causes the relevant information to be printed to the text window when the Advisor is invoked.
  - Unwatch: turns off "watch" mode.
  - Dribble On: start a dribble file for everything printed to the text window. Output to the text window is also written to the dribble file, dla•clips-run.log.
  - Dribble Off: turn off dribble file.
  - Print Rule: display a sample rule; demo use only.
- Optimize: active; provides for automatic optimization of locations of selected information sources.
  - Local: enter the "optimize" mode; clicking left on a display icon while in this mode will optimize locations of all icons in its group. Enter "escape" to leave this mode.
  - Group: same as local.

    - Global: inactive.

* Quit: exit from the DLA tool.

6.5 RECOVERY STEPS

If the DLA tool or CLIPS crashes, start over by entering "ece" into the text window.

7.0 ABBREVIATIONS AND ACRONYMS

APU	- Auxiliary Power Unit
DLA	- Display Layout Analysis
ECE	- Ergonomic Crewstation Editor
ECS	- Environmental Control System
GL	- Graphics Library
HMD	- Helmet Mounted Display
HUD	- Heads Up Display
MFD	- Multi-Function Display
MIDAS	- Man-machine Interface Design and Analysis System
SGI	- Silicon Graphics Inc.
VDU	- Video Display Unit

8.0 NOTES

8.1 ISSUES AND IDEAS

A number of interesting issues and questions have arisen in the course of developing this tool:

- DLA uses both mathematical and heuristic forms to give feedback to the designer. Which types of guidelines and constraints are best expressed with mathematical formulas (which are mostly continuous in nature) and which are best incorporated as rules (which tend to be discrete)? How can these two modes best work together? To what extent should they be integrated (if at all)?

- Currently we use 3 main guidelines; what happens when we add more? How will we handle interacting, and possibly contradictory, constraints? Can we use rules for this and mimic a human expert? Can math formulas be applied in some of these cases, or would this be too difficult?

- Inter- and intra-media information transfers and relations need to be investigated.

- What about broadening the usefulness of the tool by adding guidelines and constraints from other areas of design consideration, such as maintainability, physical limitations (e.g., varying available depth behind the panel at various locations).

- Which human factors design metrics are domain dependent? Which are domain independent? What must be done to adapt the DLA tool to another domain, such as design of fixed wing crewstation panels or power plant display/control panels?

8.2 LIMITATIONS

The DLA tool is limited to 2-D.

Currently we use only 13 displays which is less than one fourth of the displays used in the Apache cockpit.

Domain information for the task proximity measures and other relations must be acquired and placed in matrices and vectors. There are no metrics currently for menu spaces in multi-page devices.

8.3 FUTURE DIRECTIONS

Future directions:

- Include a more comprehensive set of information sources in a library (currently 13 are used). Allow user to define new information sources.
- Incorporate a richer set of human factors principles for display layout; one such principle is local stimulus-response compatibility.
- Further develop both analytic mode and evaluation mode to handle more display media. Current focus is on dedicated displays and MFDs.
- Enable designer to take into account more contexts. The current context includes a mix of flight and communication tasks.
- Gather data from more pilots. Current task proximity measures are computed using estimates from two pilots.
- Expand capabilities for design aiding in the analytic mode.
- Investigate possibility of incorporating a prototype Hypertext capability in the evaluation mode.
- Investigate virtual reality technology. Designers can work in a 3-D environment and, for example, can place information sources in HMD (Helmet Mounted Display) coordinates and visualize the effects.
- Add more visualization tools.
- Add more forces.
- Continue literature search to become more aware of related work.
- Cooperate with other researchers involved in complementary areas.
- Allow user to define relations and behaviors and change weightings for existing relations
- Allow user to change and add rules

9.0 APPENDIX

PILOT QUESTIONNAIRE

Questionnaire on Crewstation Displays

We seek to build a tool to help the A/C designer lay out displays on the crewstation panel. A good layout will be determined by how the displays relate to each other and their relative position on the panel. Aircraft displays can be related in several ways: they may be used together in performing a task (functional proximity); their information can come from the same or similar physical subsystems, e.g., the engine (physical proximity); or their values can change together over time as the flight progresses (correlational proximity). Together these measures make up what we call "task proximity". We seek information from experts such as yourself on how (and whether) various A/C displays are related.

All together, there are four matrices, one for each of the three measures of task proximity and one for location of control action (i.e., which hand do you use to control the subsystem whose values are being displayed by a given instrument). This questionnaire uses only a subset of all the possible displays.

Please fill out the following four forms as best you are able. We realize that these are subjective values and will vary from pilot to pilot. We ask that you place one of the following letters: **N** = None, **L** = Low, **M** = Medium, **H** = High, **V** = Very high, in each square in the top portion of the matrix that best describes how strong you feel the relationship holds for the particular proximity measure. The proximity measure for each metric is explained at the bottom of each page. There is also a sample matrix provided.

For the "location of control action" matrix, we ask that you answer: **L** = Left hand, **R** = Right hand, **E** = Either hand, or **NA** = Not Applicable. Please mark **L** or **R** if that hand is primarily used to perform the task related to the display in question; mark **E** if either hand could be used.

Please make use of the following additional information to guide your answers.

Context #1: You are flying contour in an AH-64 Apache (version ?) helicopter. You are VFR in daylight and traveling at 80 knots???? You are also concerned with determining general navigation information (little help ?)

Context #2: COMM

Context #3: Emergency procedure (?)

SAMPLE PROXIMITY MATRIX

	torque	rotor speed	air speed	attitude	radar alti-meter	vert. speed	accel-eration	standby compass	com-pass	clock	
torque	XXXX	H	L	L	N	N	L	N	N	N	
rotor speed	XXXX	XXXX	V	L	N	L	L	N	N	N	
airspeed	XXXX	XXXX	XXXX	M	L	M	H	N	N	N	
attitude	XXXX	XXXX	XXXX	XXXX	H	H	L	N	N	N	
radar altimeter	XXX	XXXX	XXXX	XXXX	XXXX	H	M	L	N	N	
vertical airspeed	XXXX	XXXX	XXXX	XXXX	XXXX	XXX	H	L	N	L	
acceleration	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX	XXX	N	N	N	
standby compass	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX	XXX	H	N	
compass	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX	XXX	L	
clock	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX	
	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX

N = None L = Low M = Medium H = High V = Very high

SAMPLE proximity

FUNCTIONAL PROXIMITY MATRIX

	torque	rotor speed	air speed	attitude	radar alti-meter	vert. speed	accel-eration	standby compass	com-pass	clock	
torque	XXXX										
rotor speed	XXXX	XXXX									
airspeed	XXXX	XXXX	XXXX								
attitude	XXXX	XXXX	XXXX	XXXX							
radar altimeter	XXX	XXXX	XXXX	XXXX	XXXX						
vertical airspeed	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX					
acceleration	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX				
standby compass	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX			
compass	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX		
clock	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX	
	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX

N = None L = Low M = Medium H = High V = Very high

Functional proximity is based on an estimate of the extent to which two indicators must be integrated or compared in performing a task. That is, what set of display elements do you consult jointly to perform tasks in the given context? For example, all indicators of system flight dynamics (airspeed, power setting, bank, pitch, etc.) will probably have close (high or very high) functional proximity because the pilot is consulting these displays continuously in order to fly the A/C. Suggestion: Mark V (Very high) if 2 displays are always used together and M (Medium) or H (High) if they are often but not always used together.

PHYSICAL PROXIMITY MATRIX

	torque	rotor speed	air speed	attitude	radar alti- meter	vert. speed	accel- eration	standby compass	com- pass	clock	
torque	XXXX										
rotor speed	XXXX	XXXX									
airspeed	XXXX	XXXX	XXXX								
attitude	XXXX	XXXX	XXXX	XXXX							
radar altimeter	XXX	XXXX	XXXX	XXXX	XXXX						
vertical airspeed	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX					
acceleration	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX				
standby compass	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX			
compass	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX		
clock	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX	
	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX

N = None L = Low M = Medium H = High V = Very high

Physical proximity is based on the similarity between the physical sources that generate the information that each pair of displays presents to the pilot. Thus two indicators of rotor functioning are more similar than one of rotor functioning and one of navigational functioning.

CORRELATIONAL PROXIMITY MATRIX

	torque	rotor speed	air speed	attitude	radar alti-meter	vert. speed	accel-eration	standby compass	com-pass	clock	
torque	XXXX										
rotor speed	XXXX	XXXX									
airspeed	XXXX	XXXX	XXXX								
attitude	XXXX	XXXX	XXXX	XXXX							
radar altimeter	XXX	XXXX	XXXX	XXXX	XXXX						
vertical airspeed	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX					
acceleration	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX				
standby compass	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX			
compass	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX		
clock	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX	
	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX

N = None L = Low M = Medium H = High V = Very high

Correlational proximity is the degree to which the values shown by two indicators are correlated over a four second time window. For example, airspeed and power would be highly correlated but bank and altitude might have low correlation of values; that is, bank might be just as likely to be level as banked, in low and high flight.

LOCATION OF CONTROL ACTION

torque	
rotor speed	
airspeed	
attitude	
radar altimeter	
vertical airspeed	
acceleration	
standby compass	
compass	
clock	

Location of control action associated with the display:

NA = Not Applicable **L** = Left hand **R** = Right hand **E** = Either hand

Left:	control action is normally performed with the left hand
Right:	control action is normally performed with the right hand
Either:	control action can be performed with either hand

Annex G

Army-NASA Aircrew/Aircraft Integration Program: Phase IV

A^{3}I

Man-Machine Integration Design and Analysis System (MIDAS)
Software Detailed Design Document

Anthropometric Model (JACK)

prepared by

Christian Neukom

Table of Contents

Table of Contents

MAN-MACHINE INTEGRATION DESIGN & ANALYSIS SYSTEM (MIDAS) SOFTWARE DETAILED DESIGN DOCUMENT PHASE IV:

ANTHROPOMETRIC MODEL (JACK)

1.0 INTRODUCTION

1.1 IDENTIFICATION OF DOCUMENT

This is the software Product Specification for the Anthropometric Model (Jack) component of the MIDAS Software System. Description of the detailed processing requirements, structure, I/O, and control are provided for each lower level Computer Software Component (CSC), unit, or function contained within this module.

1.2 SCOPE OF DOCUMENT

This document describes the function and use of the Jack software used in Phase IV of the A3I simulation. Since Jack has its own set of detailed user's and programmer's manual published by UPenn, this document will complement such data and describe the files created by A^{3}I for use in Jack. The reader of this document is expected to know basic graphics programming and should be familiar with a high level computer language.

1.3 PURPOSE AND OBJECTIVES OF DOCUMENT

A fundamental requirement of MIDAS is a physical representation of the human figure. For this purpose we have Jack, a dynamic anthropometric model, which has been developed through a grant to Dr. Norm Badler at the University of Pennsylvania. Jack is an interactive graphic package that allows the creation and manipulation of 3-D human figures, in a 3-D object space. Different sized human figures or graphic mannequins can be selected which include the 5th through 95th percentile male and female, based on NASA astronaut demographics. Joint limits have been installed to eliminate unreasonable movements. The figure representation is not limited to the statistical data — data of individuals can be entered in the Spreadsheet Anthropometry Scaling System (SASS) and a corresponding figure can be displayed in Jack. Kinematic and inverse kinematic algorithms are employed to generate realistic looking movements. These features allow a designer to explore human figure interaction in a 3-D environment and make predictions about reach, fit, and visibility.

2.0 RELATED DOCUMENTATION

2.1 APPLICABLE DOCUMENTS

The following documents are referenced herein and are directly applicable to this volume:

Carry B. Phillips, *Jack User's Guide*, Jack Version 4, Computer graphics Laboratory, Department of Computer and Information Sciences, University of Pennsylvania, Philadelphia, Pennsylvania 19104-6389, October 20, 1989.

Carry B. Phillips, *Programming with Jack*, Computer Graphics Laboratory, Department of Computer and Information Sciences, University of Pennsylvania, Philadelphia, Pennsylvania 19104-6389, December 21, 1988.

Norman I. Badler, *Anthropometry for Computer Graphics Human Figures*, Department of Computer and Information Science, University of Pennsylvania, Philadelphia PA 19104-6389.

Silicon Graphics .Inc., *IRIS-4D Users Guide*, Volume I and II, Version 1.1, Mountain View, CA, 1987

2.2 INFORMATION DOCUMENTS

The following documents amplify or clarify the information presented in this volume:

Norman I. Badler, *Simulating Personnel and Tasks in a 3-D environment*, Progress *Report No. 33*, Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA 19104-6389, July 13, 1989.

Norman I. Badler, *Computer Graphics Research Laboratory Quarterly Progress Report No. 34*, Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA 19104-6389, February 9, 1990.

Norman I. Badler, *Computer Graphics Research Laboratory Quarterly Progress Report No. 35*, Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA 19104-6389, First Quarter, 1990, May 18, 1990.

3.0 CONCEPT

3.1 USER DEFINITION

The Jack module is used by A^{3}I either interactively by a human operator or controlled through an integrated simulation as demonstrated during our phase IV demo. In the interactive mode Jack accepts commands via menu selection, keyboard entry and animation script files. In the integrated mode, Jack receives instructions from the Symbolic Operator Model via the Communication manager. Figure 1 displays the various interactions of the Jack software with the A^{3}I's computers, files, and system programs, as well as the human operator.

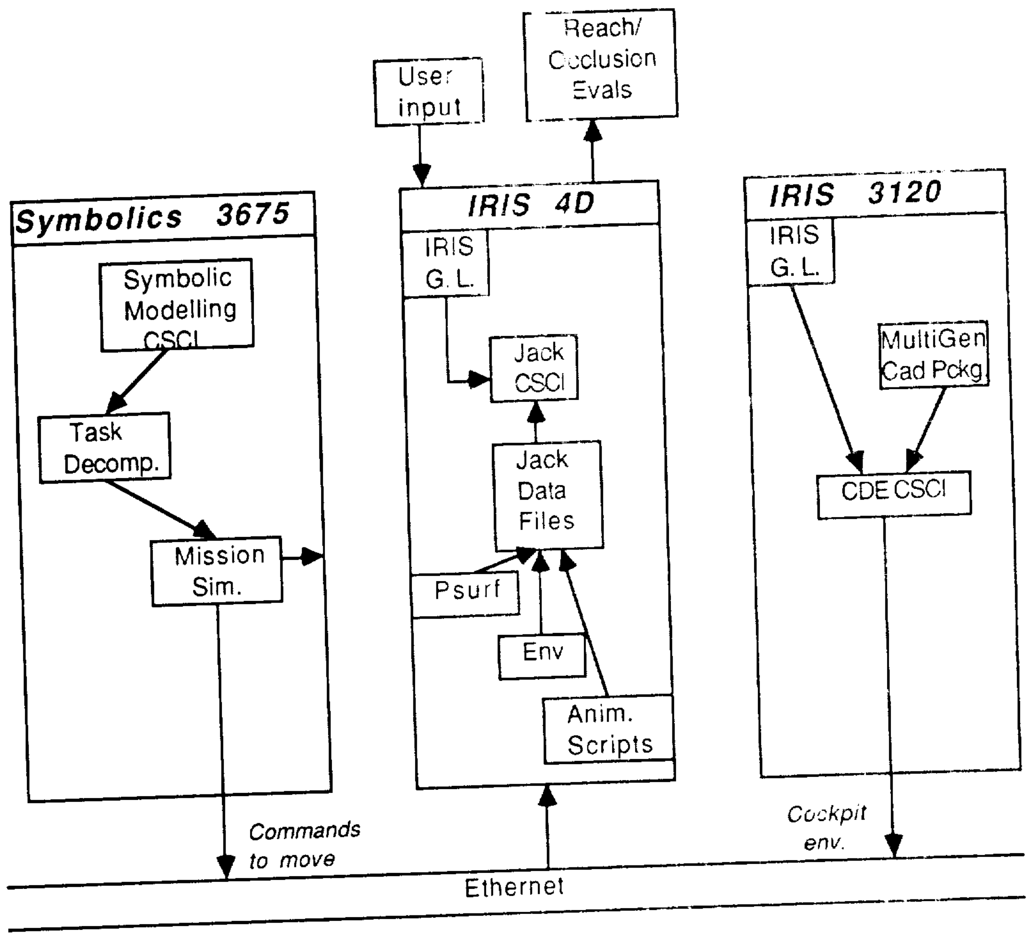


Figure 1. Jack/A^{3}I Workstation Interaction.

4.0 REQUIREMENTS

4.1 HARDWARE ENVIRONMENT

The Jack software runs on various IRIS 4D workstations. However, some functions do not work properly on the Personal IRIS and large environment files may not load at all. For this reason, an IRIS 4D70GT or better is required to run all the available functions and to obtain a satisfactory response time.

The machines available at A^{3}I are the IRIS 4D120GTX and the IRIS 4D220GTXB. The technical specification for these machines are listed below.

W-4D120GTX Power Series Workstation (Coral) consisting of:

- 32MB CPU memory
- 380MB Winchester disk drive

- SCSI 1/2" tape drive, built in
- image memory
    - 48 1280 * 1024 image bit-planes (8 bits for each red, green, and blue; double-buffered.
    - 16 1280 * 1024 image bit-planes for double buffered alpha
    - 24 bit Z-buffer (1280 * 1024)
    - 4 1280 * 1024 overlay or underlay bit planes
    - 4 window ID bit-planes (total 96 bits/pixel)
- Ethernet interface card
- 19" high resolution monitor
- keyboard & mouse

W-4D220GTXB Power Series Workstation (Starfish) consisting of:

- 32MB ECC CPU memory
- 380MB, and 780MB ESDI Winchester disk drives
- SCSI 1/2" tape drive, built in
- image memory
    - 48 1280 * 1024 image bit-planes (8 bits for each red, green, and blue; double-buffered.
    - 16 1280 * 1024 image bit-planes for double buffered alpha
    - 24 bit Z-buffer (1280 * 1024)
    - 4 1280 * 1024 overlay or underlay bit planes
    - 4 window ID bit-planes (total 96 bits/pixel)
- Ethernet interface card
- 19" high resolution monitor
- keyboard & mouse

4.2 SOFTWARE ENVIRONMENT

- Operating system: IRIX V system release 4D1-3.2
- Network File System Software
- C programming language
- C++ translator, release 1.0 source code
- IRIS graphics library
- 4 sight window manager

5.0 DESIGN

5.1 ARCHITECTURAL DESIGN

The foundation of the Jack software is Peabody, which is a graph-structured representation for articulated geometric objects. It represents figures composed of segments connected by joints, also under the influence of constraints. The peabody library provides routines for accessing and manipulating the figures in a global environment. The geometry of each segment in the environment is represented by a polyhedron surface (psurf), which is a boundary representation using tables of nodes, edges, and faces.

Jack is a facility for displaying and manipulating objects represented by peabody. It provides a standard user and programmer interface to routines which operate on the environment. Jack by itself is primarily a user interface, and its principal function is maintaining the windows and control input from the user in a consistent way.

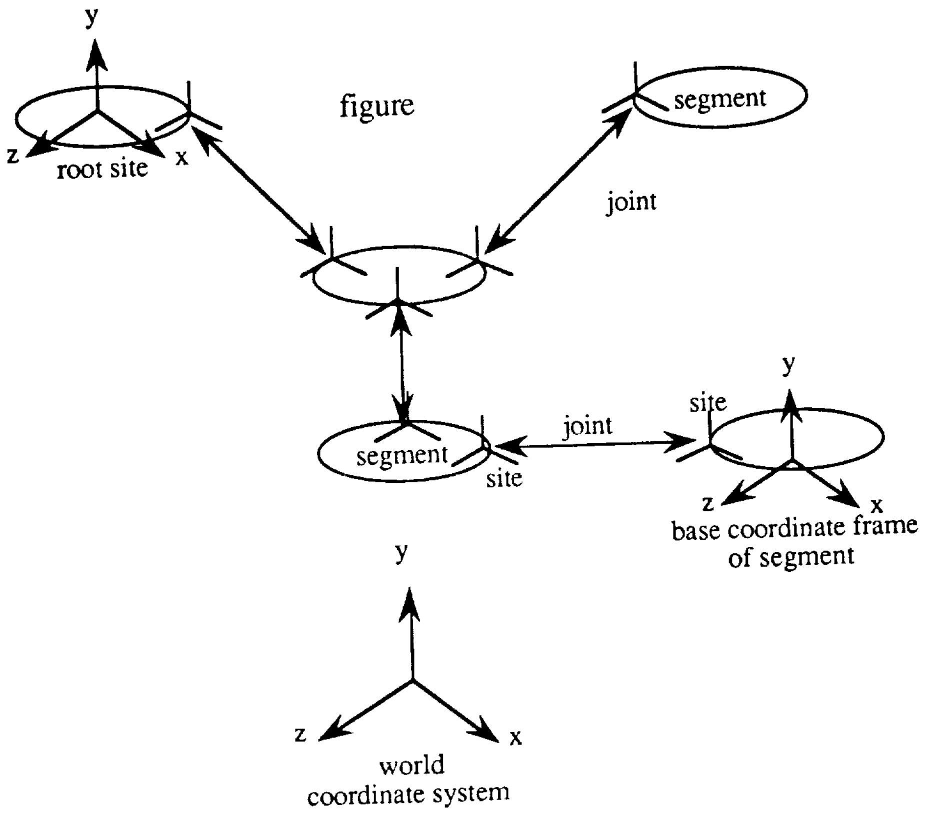


Figure 2. A graph of a peabody environment showing the connectivity of objects.

Figure 2 above shows a graphic representation of a peabody environment. A figure is a collection of segments which are connected to each other by means of joints through sites. The location of an articulated figure is specified through a site designated as the root. The root site roughly corresponds to the origin of the figure, and it provides a handle by which the location of the figure can be specified. The geometry of the psurf is specified relative to the coordinate frame of the segment not the world coordinate frame. That means that each psurf is designed in its own coordinate system.

6.0 USER'S GUIDE

A complete user's guide to Jack is supplied by Upenn as referenced in section 2.1 of this document. Information of the basic design and structure of Jack can be found in the "Programming with Jack" manual also referenced in section 2.1. Instruction for running the A^{3}I demo files can be found in "Anthropometry Demo 1990" located in Appendix D. Appendix A, labeled "Menu Road Map," shows all of the menu selectable commands presently implemented in the A^{3}I version of Jack. Appendix B, named "Description of Menu Selectable Commands," is in addition to the description in the user's manual and can be used as a quick reference. Appendix C shows the Jack directory structure as it appears on the tape. Appendix E, the Readme File, gives instructions for installing Jack from tape.

6.1 DESCRIPTION OF JACK FILES CREATED IN PHASE IV

6.1.1 Psurf Files (.pss)

lb_fwd01a.pss .. .. lb_fwd13.pss	This collection of 22 files make up the copilot-gunner cockpit of a Longbow helicopter. Included in these files is part of the fuselage surrounding the cockpit.
lb_aft01.pss .. .. lb_aft13.pss	These 21 files make up the cockpit of the pilot of a Longbow helicopter. Included in these files is part of the fuselage surrounding the cockpit.
lb_ext01a.pss .. .. lb_ext10c.pss	These 21 files make up the exterior of a Longbow helicopter.
ah64_hta.pss .. .. ah64_htd.pss ht_visor.pss	These 5 files collectively make up an Apache IHADDS helmet includi[illegible] visor and monacle.
lfviewcone.pss lviewcone.pss	Static viewcone for vision analysis.
rfviewcone.pss rviewcone.pss sfrviewcone.pss slviewcone.pss srviewcone.pss	Field of view (FOV) view cones for vision analysis.
ssphere.pss	This file is used in the vision demo.

6.1.2 Environment Files (.env)

05pilot.env	Fifth percentile polybody figure in pilot posture.
bio_pilot_helmet.env	Biostereometric figure with IHADDS helmet in pilot posture.

bio_pilot_helmet_lb_aft.env	Biostereometric figure with IHADDS helmet in pilot posture and in longbow pilot cockpit environment.
bio_pilot_lb_aft.env	Biostereometric figure in pilot posture and in Longbow pilot cockpit environment.
d3.env	Longbow copilot-gunner cockpit with polybody figure seated in crew-position. No fuselage and back of seat.
d5.env	Longbow helicopter with two biostereometric figures seated in crew position.
d6.env	Two polybody figures. One carrying box (reach constraints between hands and box).
demo1.env	Line-up of male and female polybody figures (5th, 50th, 95th percentile) and 95th percentile by height biostereometric male figure.
demo2.env	Biostereometric figure wearing IHADDS helmet.
demo3.env	Biostereometric figure wearing helmet seated in copilot-gunner cockpit.
fwd_bio_pilot.env	Biostereometric figure in pilot posture.
helmet80.env	IHADDS helmet. Requires ah64_htx.pss and ht_visor.pss files.
lb_aft_bio_pilot.env	Longbow pilot cockpit with biostereometric figure seated in crew position.
lb_aft_pilot1.env	Longbow pilot cockpit with polybody figure seated in crew position.
lb_ext_1bio_pilot.env	Longbow helicopter with biostereometric figure seated in copilot cockpit.
lb_ext_2bio_pilot.env	Longbow helicopter with two biostereometric figures seated in crew position.
lb_ext_2pilot.env	Longbow helicopter with two polybody figures seated in crew position.
lb_fwd_bio_pilot.env	Longbow CPG cockpit with biostereometric figure wearing IHADDS helmet.
lb_fwd_pilot.env	Longbow CPG cockpit with 95th percentile polybody figure.

lb_fwd_pilot05_helmet.env	Longbow CPG cockpit with 5th percentile polybody figure wearing helmet.
p31.env	Longbow CPG cockpit with
p32.env	Longbow CPG cockpit with 95th percentile biostereometric figure and IHADDS helmet.
p33a.env	Longbow pilot cockpit with 95th percentile polybody figure seated in crew position.
pilot05_helmet_lb_fwd.env	5th percentile polybody figure wearing IHADDS helmet in seating posture in copilot cockpit environment.
pilot05_lb_fwd.env	95th percentile polybody figure in seated posture in copilot environment (Can be imported into copilot cockpit).
pilot_helmet.env	5th percentile polybody figure in seating posture in copilot cockpit environment.
pilot_lb_aft.env	95th percentile polybody figure in seated posture in longbow pilot environment (Can be imported into longbow pilot cockpit).
pilot_lb_fwd.env	95th percentile polybody figure in seated posture in copilot environment (Can be imported into longbow copilot-gunner cockpit).

6.1.3 Jack Command Language Files (.jcl)

d1.jcl	Loads demo1.env file and makes it shaded.
d2.jcl	Loads demo2.env file. Turns projections off, displays 4-panel screen, makes window shaded, and makes visor transparent.
d3.jcl	Loads d3.env, and sets up animation. In the animation, the copilot-gunner seated in the cockpit reaches for the left MFD with his right hand. This animation also requires the files d3.scr and d3.act.
d5.jcl	Loads d5.env, turns on 6 lights, and makes scene shaded.
d6.jcl	Loads demo6.env, makes it shaded, and sets up interactive balance reach. Creates reach constraint between figure and box for constraint demo. Requires box.env file.

p31.jcl	Setup for p31.env (shading, transparency, etc.).
p32.jcl	Setup file for p32.jcl
p33a.jcl	Setup file for p33a.env
vis.jcl	Setup file for vision demo.

6.1.4 Action Files (.act)

d3.act	This file is used by d3.jcl. It contains the action script.

6.1.5 Script Files (.scr)

d3.scr	This file is used by d3.jcl.

6.1.6 Data Files

QO_i1_d1.cons QO_i1_l1_d1.dat QO_i2_l1_d1.cons QO_i2_l1_d1.dat QO_i3_l1_d1.cons QO_i3_l1_d1.dat QO_i4_l1.5_d1.cons QO_i4_l1.5_d1.dat QO_i4_l1_d1.cons QO_i4_l1_d1.dat QO_i4_l2.54_d1.cons QO_i4_l2.54_d1.dat QO_i4_l2_d1.cons QO_i4_l2_d1.dat	Vision confusion data and their contour data for large font size.
cone_den.left cone_den.right	Cone density data and contours.
contour.left contour.right	Iso contour data.
frviewcone lviewcone	Static viewcones.
qo_i0_l1_d1.cons qo_i1_l1_d1.cons qo_i2_l1_d1.cons qo_i3_l1_d1.cons qo_i4_l1.5_d1.cons qo_i4_l1_d1.cons qo_i4_l2.54_d1.cons	Confusion data and contours for small font size.

qo_i4_l2__d1.cons	
rfviewcone	Field of view (FOV) viewcones

6.2 DOWNLOADING PSURF FILES FROM CDE

Once a CSE environment has been loaded into the CDE, it can then be downloaded to Jack. Simply start the CDE with the environment that will be downloaded on Jack. Then go to the CDE menu and choose "Data Format". A window will appear at the bottom that asks for one of several types of file formats. Select the Psurf format and click on OK. This window will go away and another window will appear in the center of the screen that will ask for a scaling factor. A 1 and a return should be typed in, unless the graphic is to be scaled to a different size.

This window will be replaced by another window asking for a filename to be typed in. This is the filename that the psurf will be saved in. This window will go away and another will appear in the upper left corner that has the legend "Select Bead Then Hit OK or Done" and three buttons...OK, Done, and Undo. Now, to choose a section of the environment that you want to save, simply click on that section so that a white line appears around the edge, then click on "OK". Each section will be saved in the order it is selected. When all the sections have been selected, clicking on "Done" will save all those section in a psurf file.

The psurf thus produced contains color information which Jack is not able to read directly - Jack keeps the color information in an environment file. To fix this problem, a short awk program was written to extract the color information from the psurf files and to write them to an environment file with the same name and the suffix .env. To run this program, use the following command:

```
awk -f pss.awk  psurf.pss > psurf.env
```

7.0 ABBREVIATIONS AND ACRONYMS

A^3I	Army-NASA Aircrew/Aircraft Integration
CDE	Cockpit Design Editor
CSC	Computer Software Component
CSCI	Computer Software Configuration Item
Psurf	Polyhedron surface
UPenn	University of Pennsylvania
SASS	Spreadsheet Anthropometric Scaling System

8.0 GLOSSARY

9.0 NOTES

9.1 LIMITATIONS

Jack is limited by the graphics and computation power of the machine it runs on. Environments containing the highly detailed Longbow cockpits or biostereometric figures slow down the graphics processing enormously, sometimes to a point, where the operator looses the feeling of interactive control. We have worked around this problem by simplifying the environments we work with to a bare minimum. However, to take full advantage of our detailed environments and all the capabilities of Jack, a high end model Iris workstation, such as the IRIS-4D-220GTX, is required.

9.2 FUTURE DIRECTIONS

One important aspect of Jack is the anthropometry data that Jack uses to create and display figures. At the present, statistical data for the 5th, 50th, and 95th percentile male and female are employed and interpolated to produce the whole range of figures (1st through 99th percentile). To get a more realistic representation of figure data, it will be necessary to have a database of anthropometric measurements from a large group of individuals. This data base can then be searched either on a statistical or individual basis and a corresponding figure can be represented in Jack.

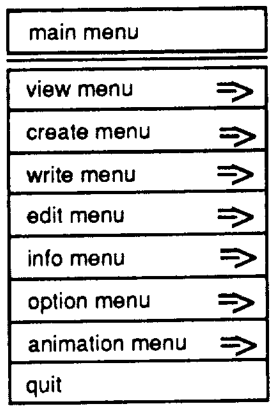
It will also be necessary to do some validation work along the line. In the present phase, the mannequin is placed into the cockpit solely on the ground of visual inspection by the operator. The placement of the figure affects the reach analysis and the view from the mannequins eyes. There are many other factors and parameters affecting the results obtained by the Jack model. It is therefore important to have some validation for the most basic conditions and functions, such as the placement of the mannequin into the cockpit and the reach analysis.

10.0 APPENDICES

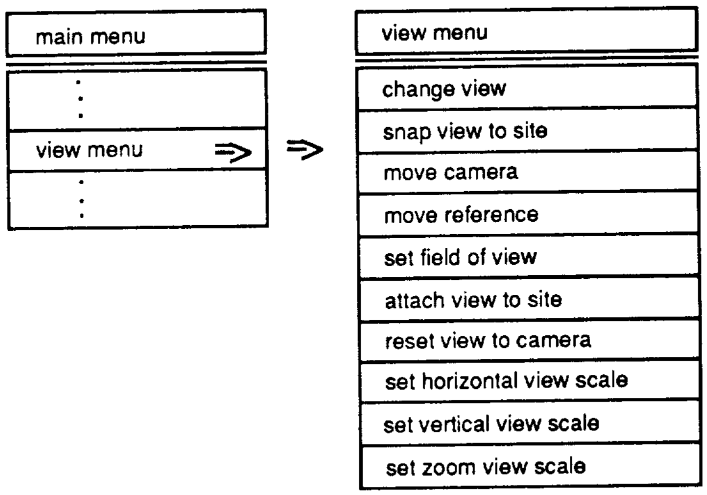
APPENDIX A — MENU ROAD MAP

Menu Roadmap

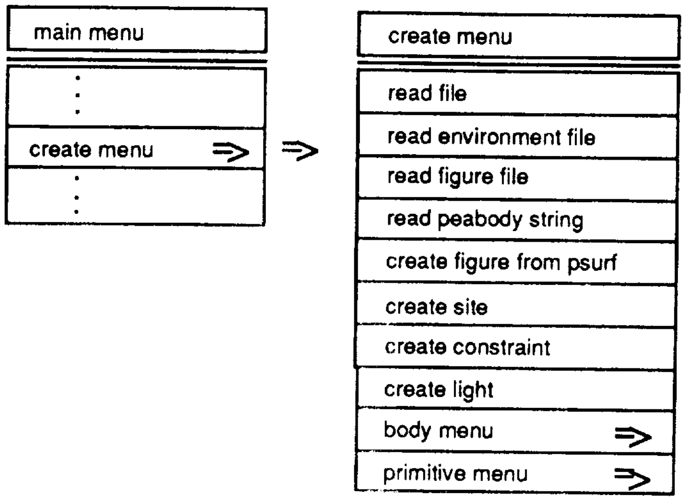
Main Menu



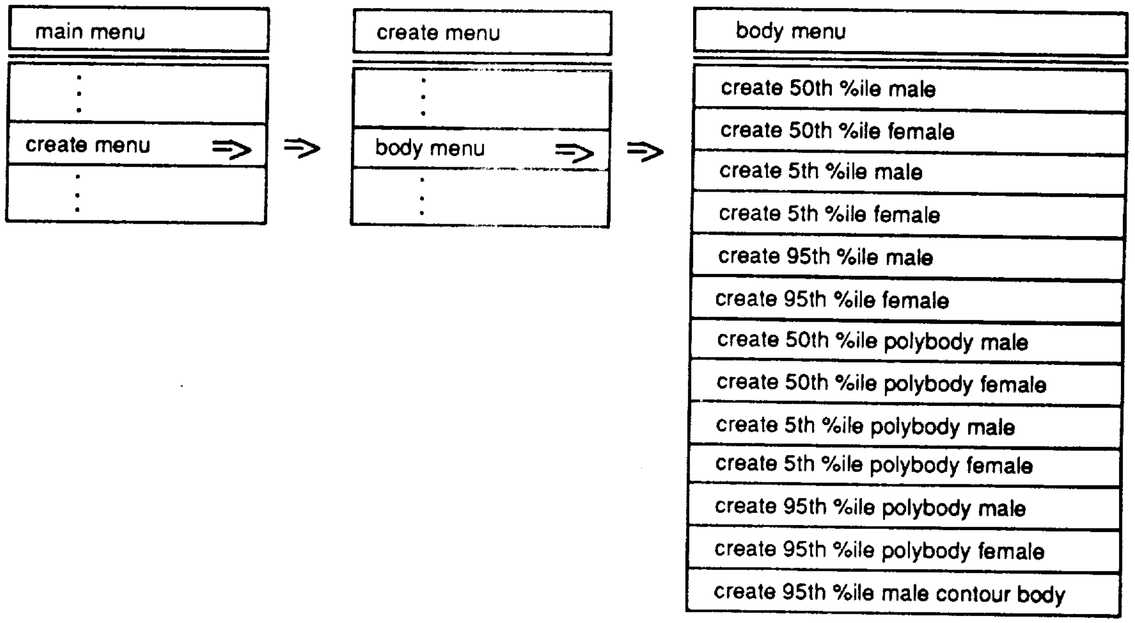
View Menu



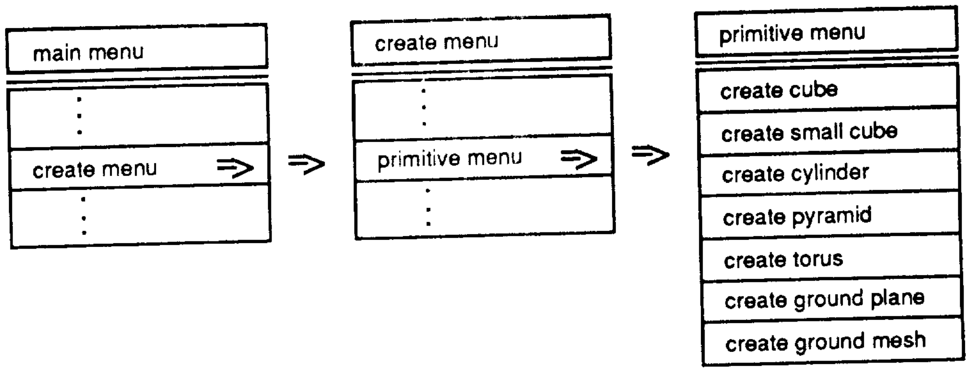
Create Menu



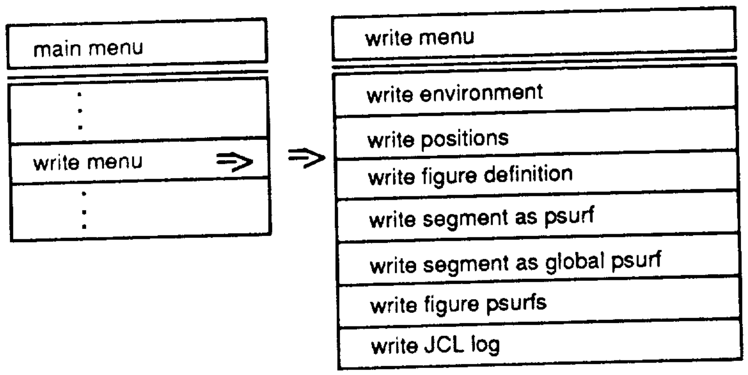
Body Menu


main menu
create menu
create menu
body menu
body menu
create 50th %ile male
create 50th %ile female
create 5th %ile male
create 5th %ile female
create 95th %ile male
create 95th %ile female
create 50th %ile polybody male
create 50th %ile polybody female
create 5th %ile polybody male
create 5th %ile polybody female
create 95th %ile polybody male
create 95th %ile polybody female
create 95th %ile male contour body

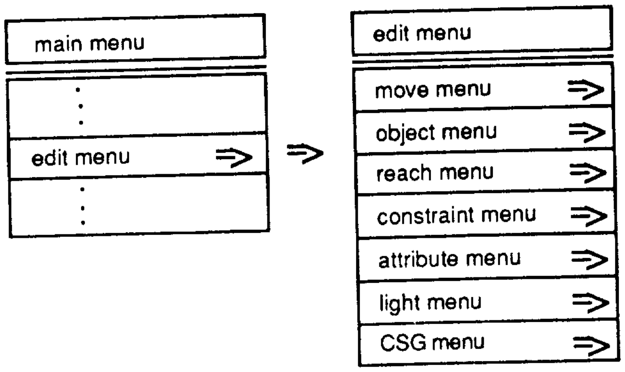
Primitive Menu



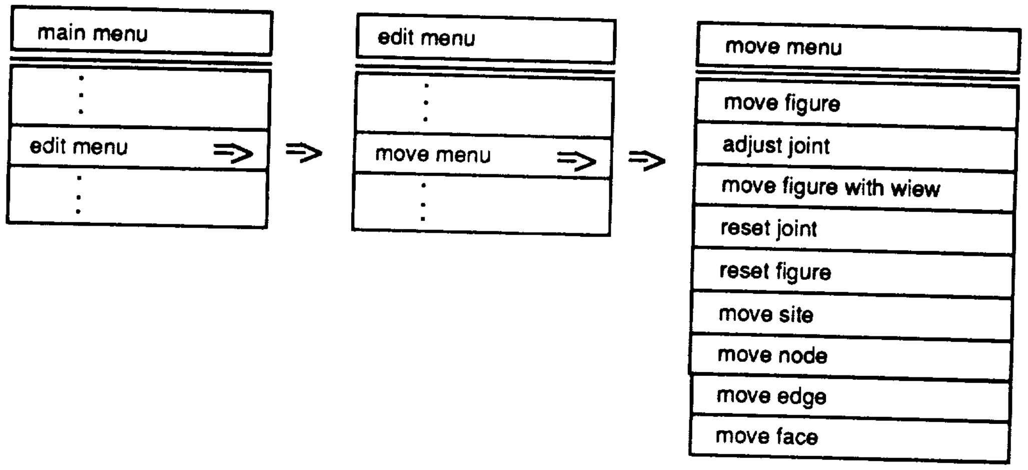
Write Menu



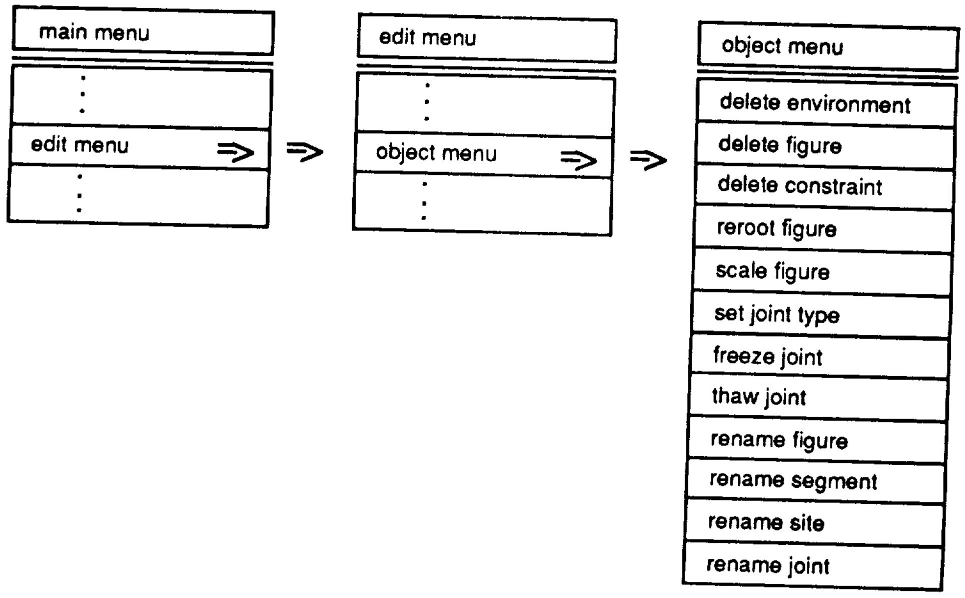
Edit Menu



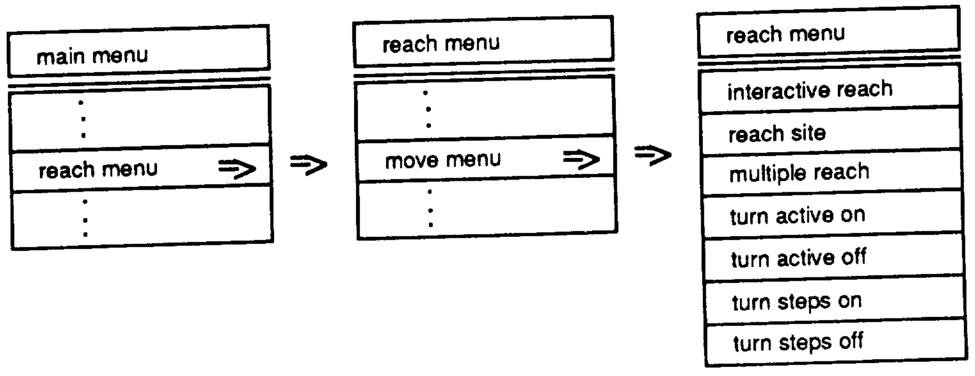
Move Menu



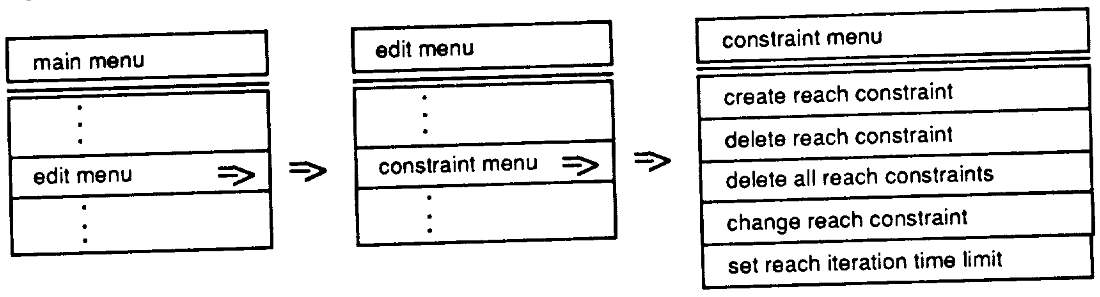
Object Menu



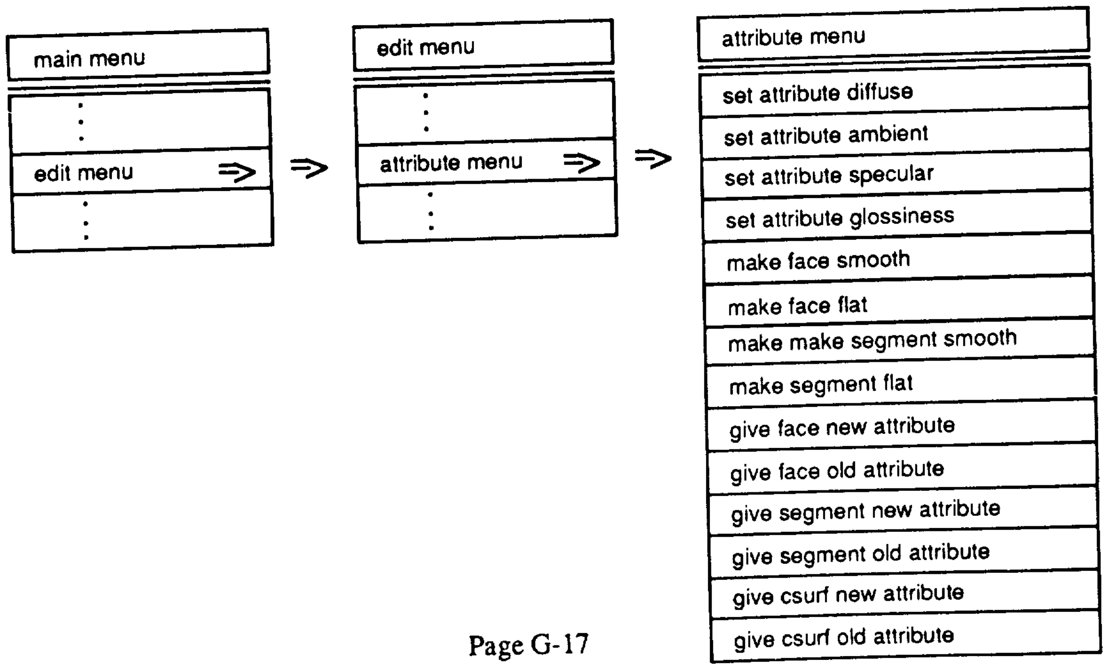
Reach Menu



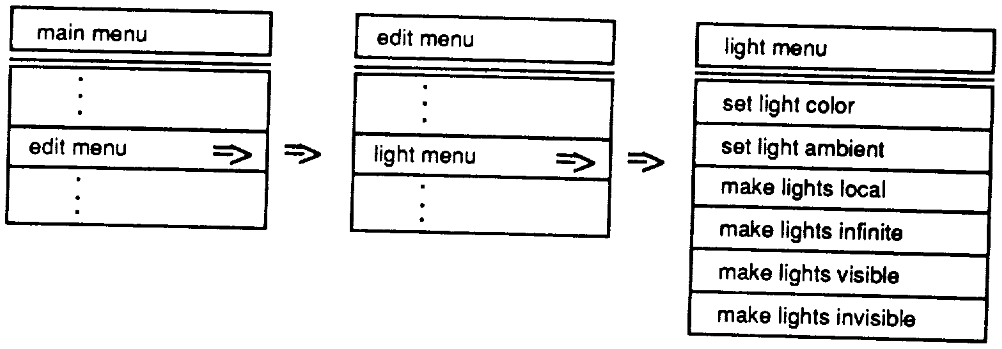
Constraint Menu



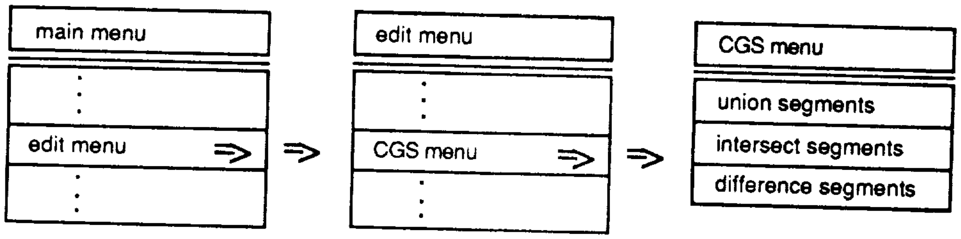
Attribute Menu



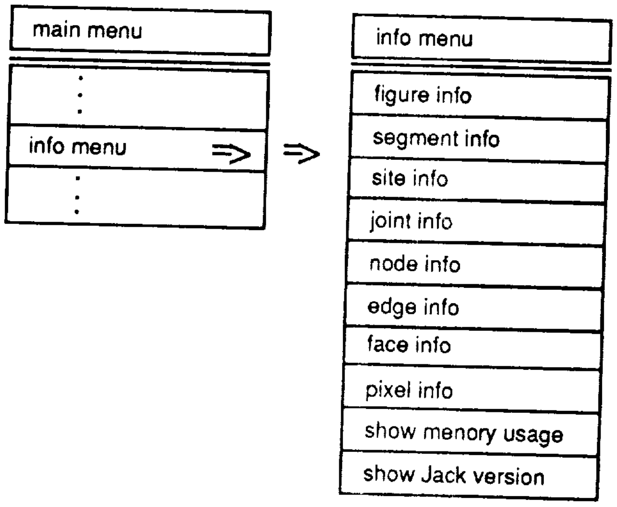
Light Menu



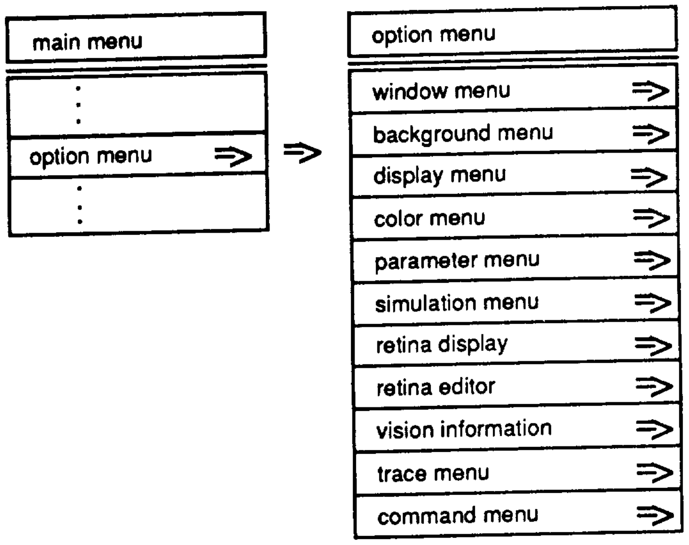
CSG Menu



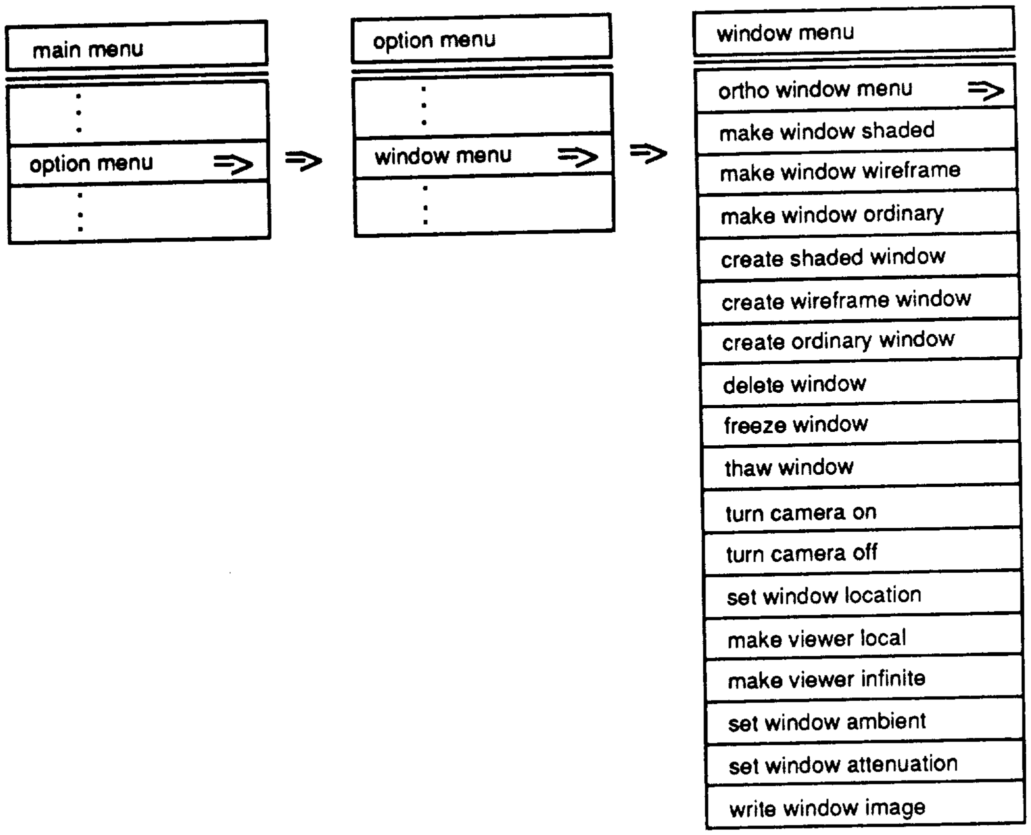
Info Menu



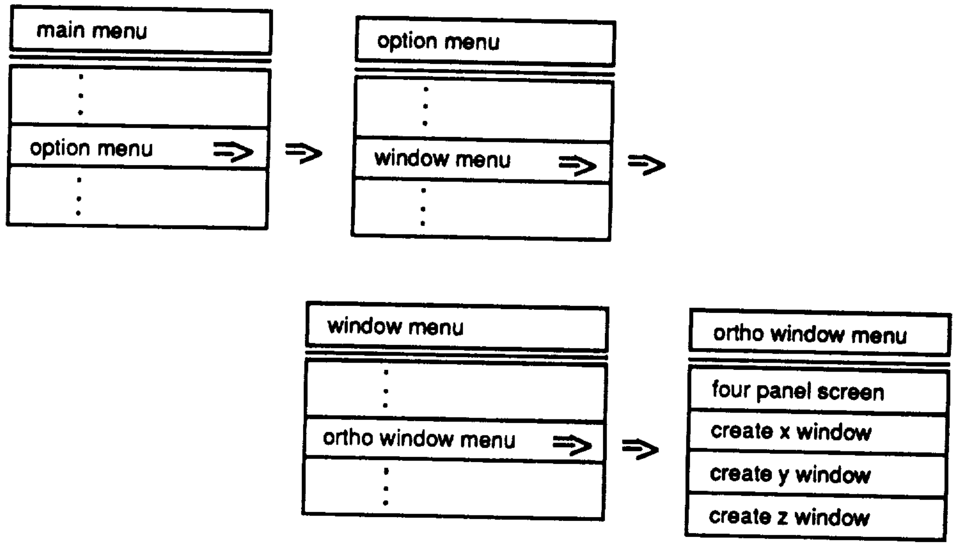
Option Menu



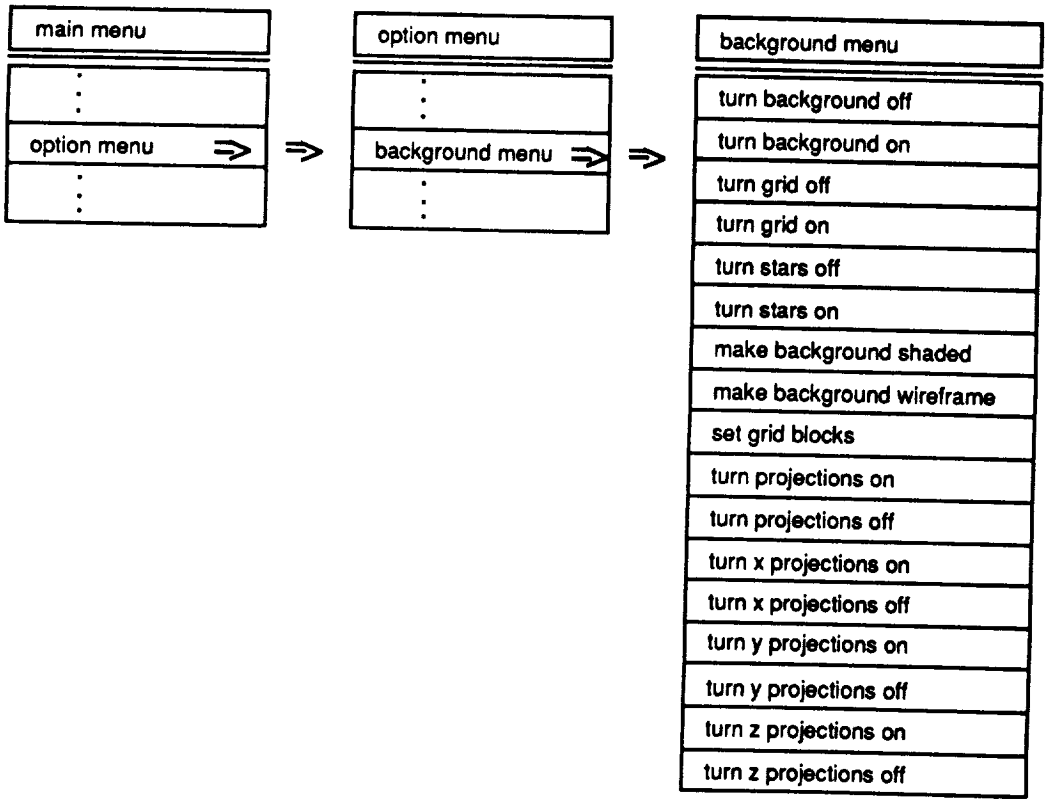
Window meu



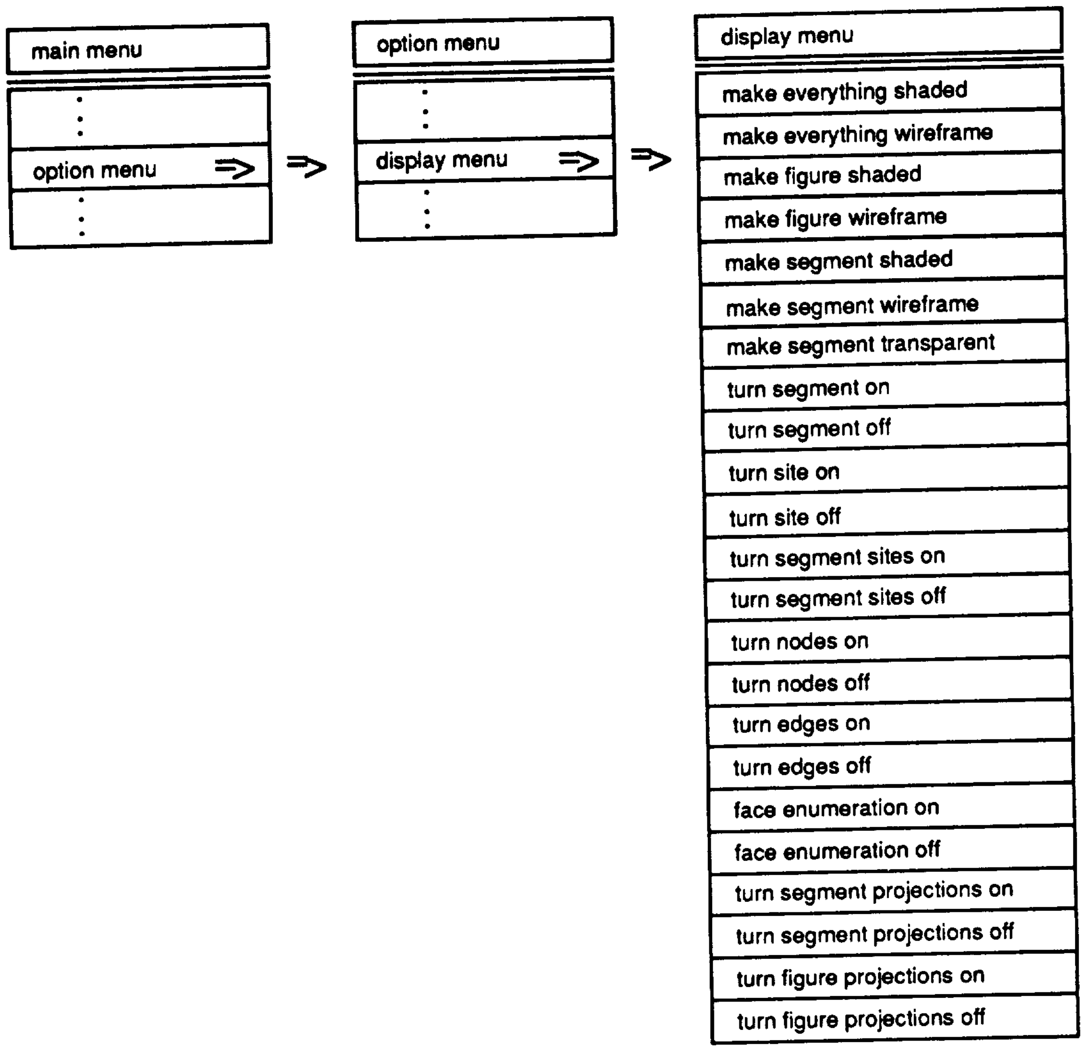
Ortho window menu



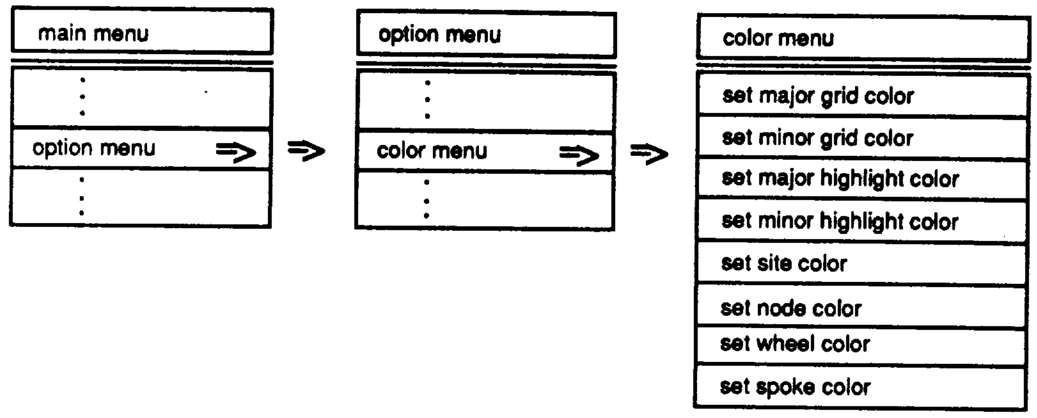
Background Menu



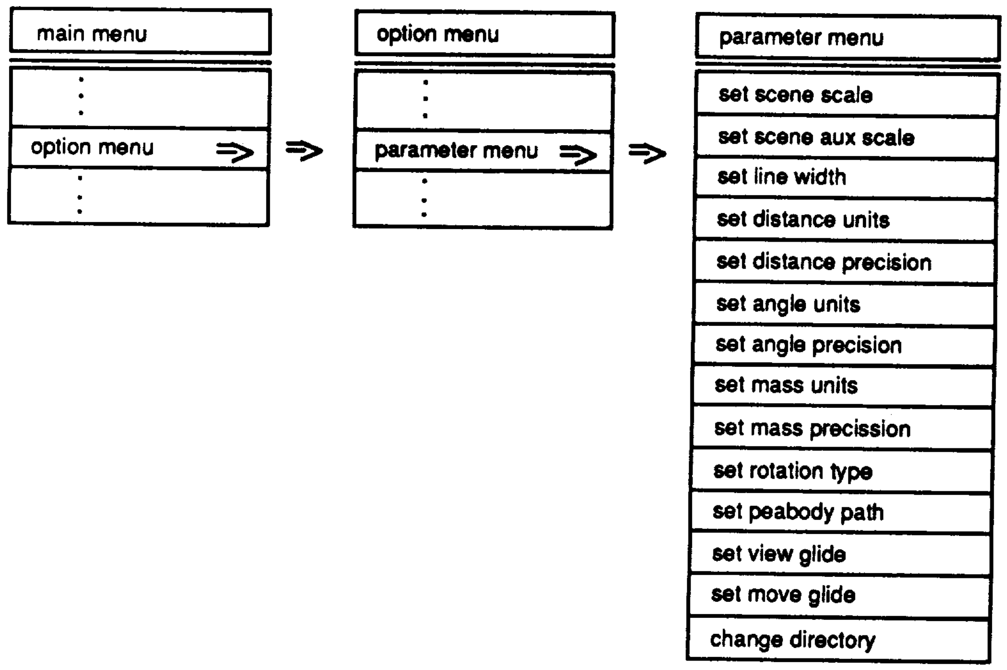
Display Menu



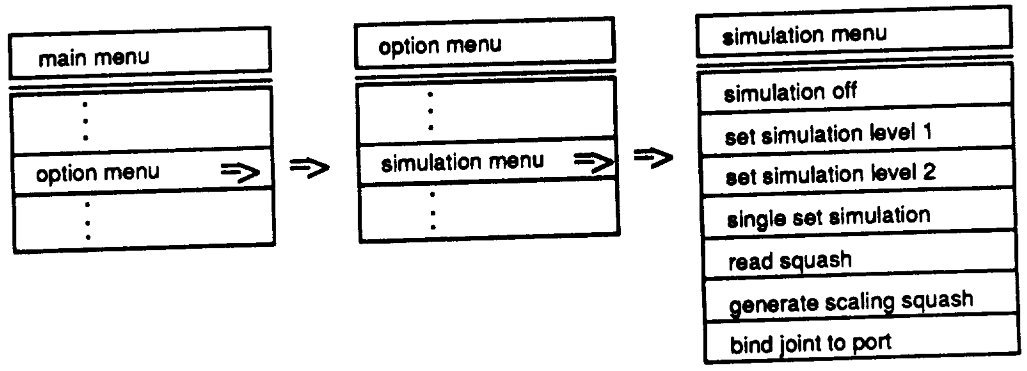
Color Menu



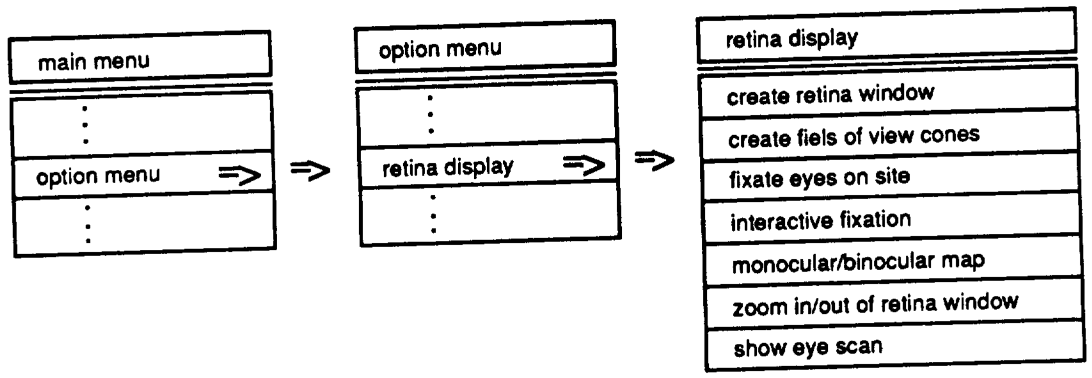
Parameter Menu



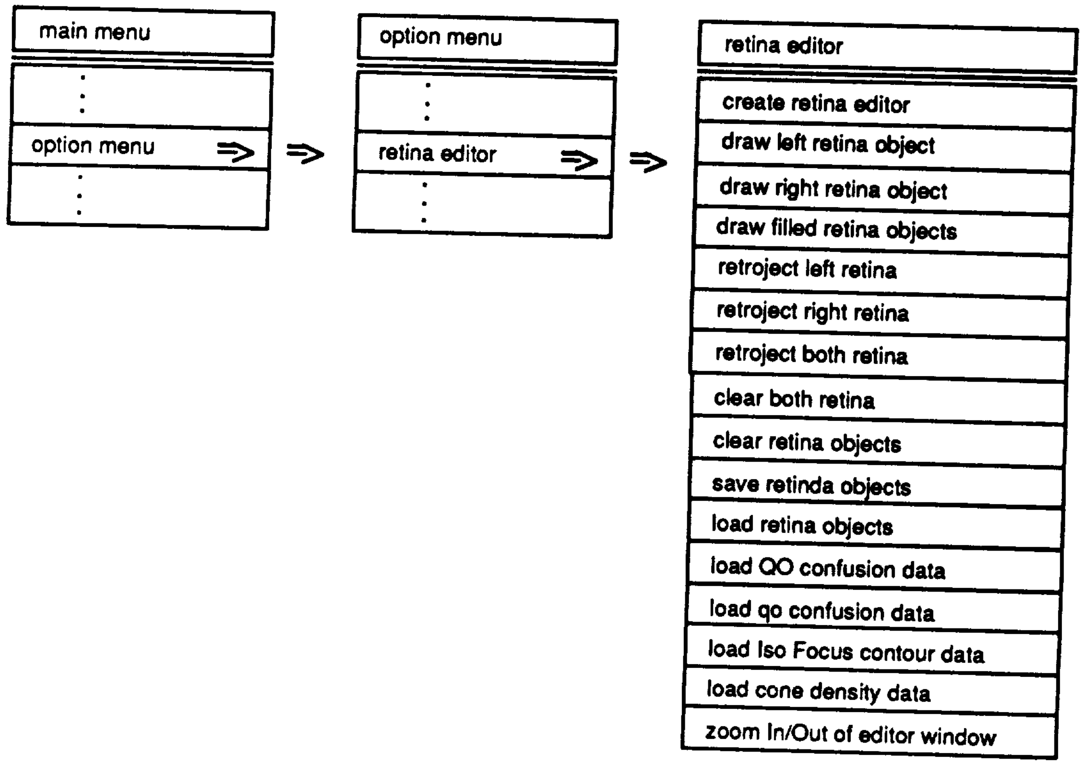
Simulation Menu



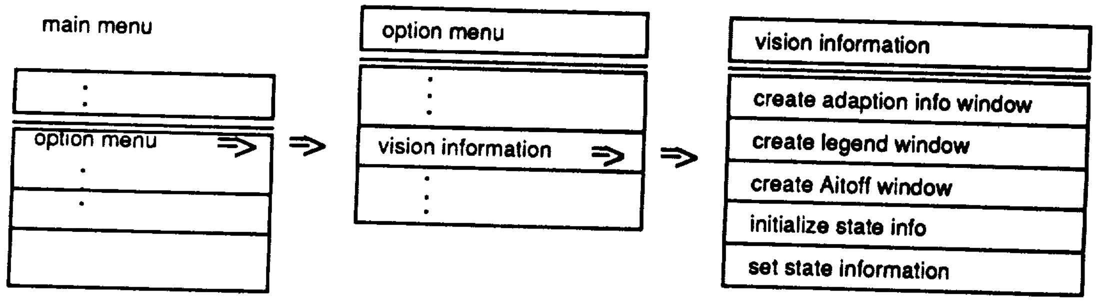
Retina Display



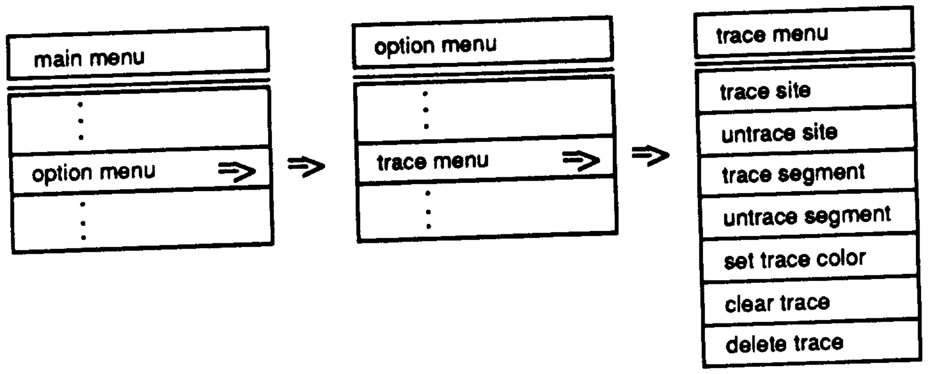
Retina Editor



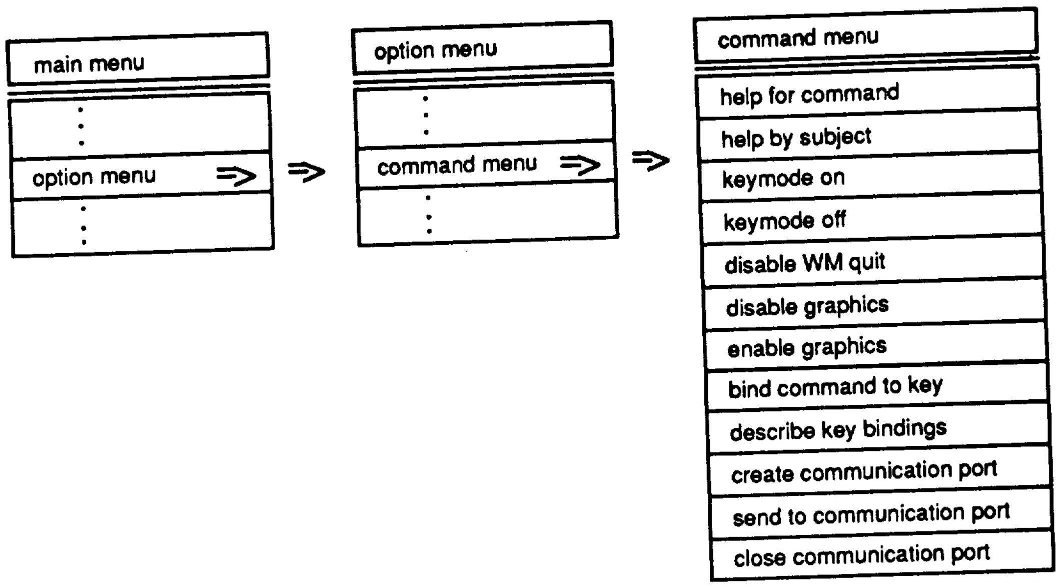
Vision Information



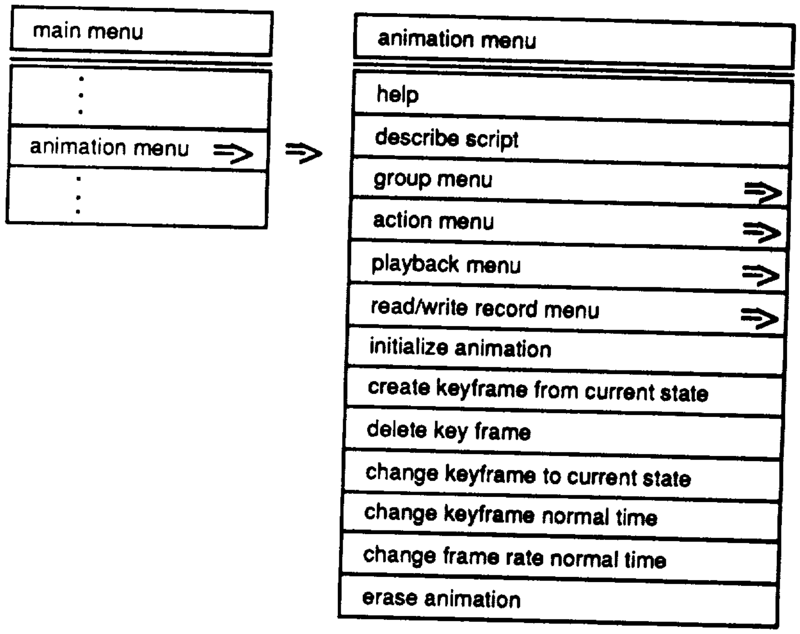
Trace Menu



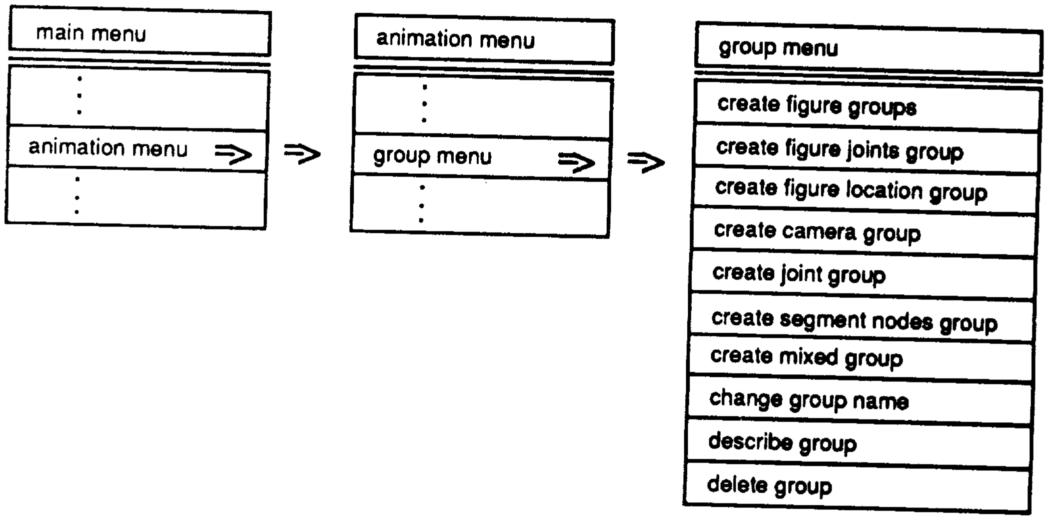
Command Menu



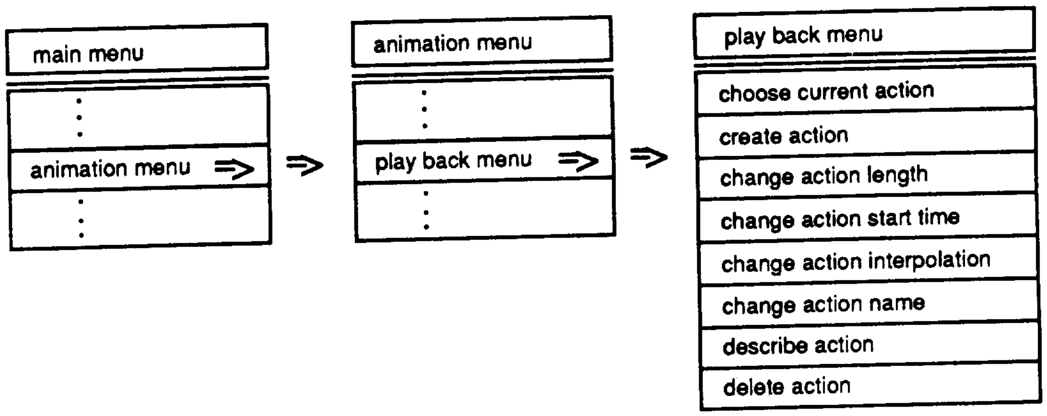
Animation menu



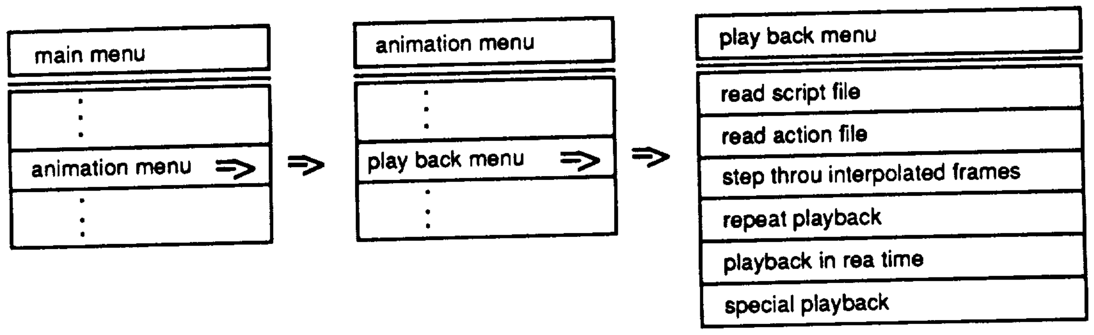
Group Menu



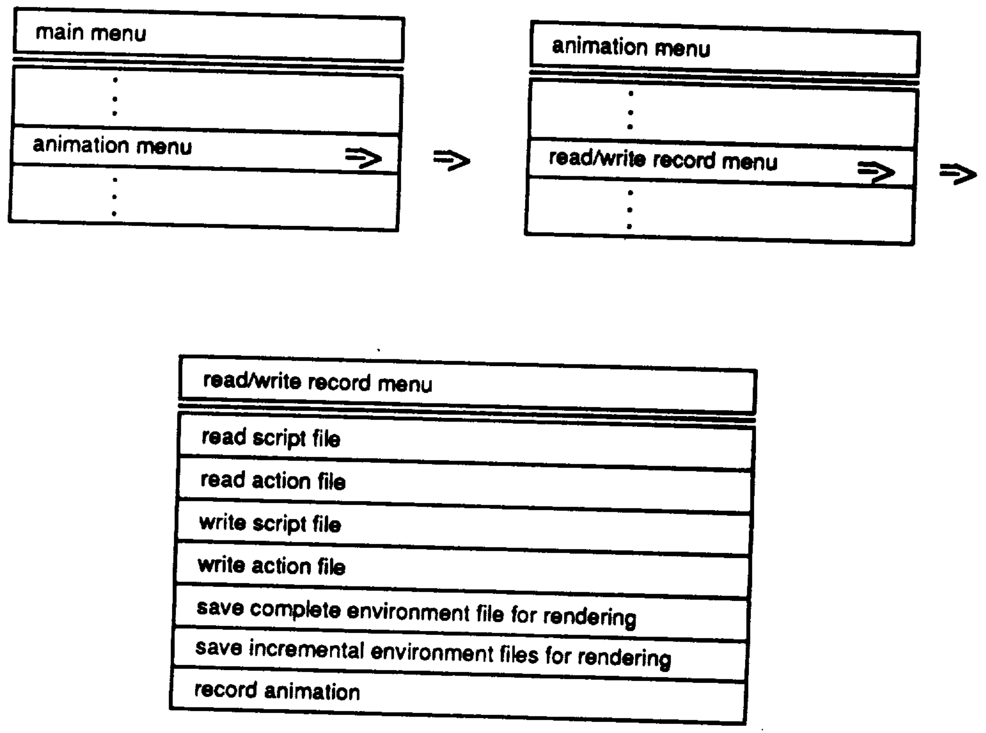
Action Menu



Playback Menu



Read/Write Record Menu



APPENDIX B — DESCRIPTION OF MENU SELECTABLE COMMANDS

Description of Menu Selectable Commands

VIEW MENU

change view	Interactively changes view angle, direction, and zoom. The left mouse button controls the horizontal swing, the middle mouse button the vertical swing, and the right mouse button the zoom. With the control key pressed, change view executes the same function but in the "panning" mode. ESC terminates the process or alternatively, ^C terminates and resets the view to its original position.
snap view to site	Changes the view reference point to a new location.
move camera	Changes view "non-interactively"
move reference	Changes the view reference point.
set field of view	The default field of view is 40^o. Angles of more than 70^o give distorted images.
attach view to site	Moves view to a specified site.
reset view to camera	Resets view after it has been changed by some of the above commands.
set horizontal view scale	Changes the sensitivity of the left mouse button for viewing.
set vertical view scale	Changes the sensitivity of the middle mouse button for viewing.
set zoom view scale	Changes the sensitivity of the right mouse button for viewing.

CREATE MENU

read file	Files with the following extensions can be read with this command: .env, .fig, .pss, .bps, .jcl.
read environment file	Reads an environment file if the suffix is not .env.
read figure file	Reads a figure file if the suffix is not .fig.
read peabody string	Reads a string of peabody syntax.
create figure from psurf	Reads a psurf file if suffix is not .pss.
create site	Prompts for segment on which to create site.
create constraint	Do not confuse this with the reach constraints.

create light	Creates a light source.

BODY MENU

(The following commands create the specified figures.)

create 50th percentile male

create 50th percentile female

create 5th percentile male

create 5th percentile female

create 95th percentile male

create 95th percentile female

create 50th percentile polybody male

create 50th percentile polybody female

create 5th percentile polybody male

create 5th percentile polybody female

create 95th percentile polybody male

create 95th percentile polybody female

create 95th percentile by height
biostereometric male

PRIMITIVE MENU

(The following commands create the specified objects.)

create cube

create sphere

create cylinder

create pyramid

create torus

create ground plane

create ground mesh

WRITE MENU

write environment	Writes the entire environment to a file which should have the suffix .env.
write positions	Writes the positions of all the figures in the environment.
write figure definition	Writes figure file of a specifies figure.
write segment as psurf	Writes segment psurf to a file as is.
write segment as global psurf	Writes segment as global psurf.
write figure psurfs	Writes figure as a global psurf.
write JCL log	Writes script file of interactive session.

MOVE MENU

move figure	Move figure to new position.
adjust joint	Joint angles may be adjusted, one degree at a time.
move figure with view	Attaches the view to a figure and allows moving the view. When done, it reattaches the view to the the previous site.
reset joint	Resets joint to its default position.
reset figure	Resets figure to its default position.
move site	Move a site to a new location
move node	Moves node. This command alters the coordinates of the psurf node but it does not alter the file from which it was originally read. To save the changed geometry, write the psurf back to file.
move edge	Moves a pair of nodes. Same rules as for save node.
move face	Moves all nodes in a selected face.

OBJECT MENU

delete environment	Deletes the environment in the current window.
delete window	Deletes a selected figure from an environment. If the figure has a constraint attached to it, it will be deleted too.
delete constraint	Removes a constraint.

reroot figure	This command changes the root site of a figure. In the process, the figure "location" is also changed since the location is the global placement of its root site.
scale figure	Each psurf is scaled in its own coordinate system.
set joint type	Sets the degree of freedom in a joint.
freeze joint	A frozen joint may not be manipulated and will not be changed by the inverse kinematics algorithm.
Thaw joint	Unfreezes a joint.
rename figure	Prompts for new figure name.
rename segment	Prompts for new segment name.
rename site	Prompts for new site name.
rename joint	Prompts for new joint name.

REACH MENU

interactive reach	Allows you to interactively drag an end effector. Prompts for goal type, end effector site, and a base joint.
reach site	Allows to specify single reach goal. Prompts for goal type, goal site, end effector, and base joint.
multiple reach	Allows to specify several reach goals. Prompts for goal types, goal sites, end effectors, base joints, and weight factor.
turn active reach on	Adds more active segments to reach chain if goal cannot be reached with the specified joint chain.
turn active off	Default setting. Only specified joint chain is used for reaches.
turn steps on	Default setting. Intermediate position of the figure are displayed as reach algorithm is executed.
turn steps off	Intermediate positions are not displayed as algorithm is executed.

CONSTRAINT MENU

create reach constraint	A restriction to the reach commands is created. Prompts for goal type, goal site, end effector, base joint, and weight factor.
delete reach constraint	Removes constraint imposed by "create constraint".
delete all reach constraints	Removes all constraints.

change reach constraint	Uses default parameters of the previous reach constraint.
set reach iteration time limit	Puts a time limit on each iteration step for the reach algorithm.

ATTRIBUTE MENU

set attribute diffuse	Sets surface attribute "diffuse" of an object. Describes the color of an object when it is illuminated.
set attribute ambient	Sets surface attribute "ambient" of an object. Describes the color of an object when it is not illuminated.
set attribute specular	Sets surface attribute "specular" of an object. Describes the color of the specular highlights of an object.
set attribute glossiness	Sets surface attribute "glossiness" of an object. Describes the specular scattering of the surface.
make face smooth	Applies Phong shading to specified face.
make face flat	Default. The surface of an object is modeled as discrete polyhedron.
make segment smooth	Applies Phong shading to specified segment.
make segment flat	No Phong shading.
give face new attribute	Creates a new attribute and associates it with the specified face.
give face old attribute	Allows to select a previously defined attribute and associate with specified face.
give segment new attribute	Creates a new attribute and associates it with the specified segment.
give segment old attribute	Allows to select a previously defined attribute and associate with specified segment.
give csurf new attribute	Sets the attribute association for all faces which are connected to a selected face.
give csurf old attribute	Returns old attribute to csurf.

LIGHT MENU

set light color	By default, lights are white, but their color may be changed by this command.
set light ambient	Sets the ambient light parameters.
make lights local	The light direction vector used in the light model is different for each point in the scene.
make lights infinite	The light direction vector used in the light model is the same for all points in the scene.
make lights visible	Turns on the lights and and makes visible the figures associated with the lights.
make lights invisible	Makes the figures associated with the lights invisible. Does not turn off the lights.

CSG MENMU

(These commands don't work. They are supposed to perform constructive solid geometry operations.)

union segment

intersect segments

difference segments

INFO MENU

(The following commands give information about the specified object).

figure info

segment info

site info

joint info

node info

edge info

face info

pixel info

show memory usage

show Jack version

WINDOW MENU

make window shaded	Once a window is made shaded, the attributes of individual figures and segments cannot be displayed (i.e. make segment wireframe).
make window wireframe	Once a window is made wireframe, the attributes of individual figures and segments cannot be displayed (i.e. make segment shaded).
make window ordinary	In an ordinary window, the attributes of individual figures and segments can be displayed.
create shaded window	Creates a shaded window with a slightly different view.
create wireframe window	Creates a wireframe window with a slightly different view.
create ordinary window	Creates an ordinary window with a lightly different view.
delete window	Deletes a window without quitting Jack.
freeze window	A frozen window is not redrawn while executing the "change view" command until ESC is pressed.
thaw window	Unfreezes a frozen window.
turn camera on	Displays figure associated with camera.
turn camera off	Hides figure associated with camera.
set window location	Prompts user for window coordinates and redraws window in specified location.
make viewer local	Controls the lighting properties.
make viewer infinite	Controls the lighting properties.
set window ambient	The default ambient lighting can be changed by this command.
set window attenuation	Attenuates lighting values.
write window image	Dumps window as a .rle file for printing.

ORTHO WINDOW MENU

four panel screen	Creates four equal sized windows, one ordinary and three orthogonal.
create x window	Creates a x orthogonal window.

create y window	Creates a y orthogonal window.
create z window	Creates a z orthogonal window.

BACKGROUND MENU

turn background off	Removes the grid that represents the ground plane of the scene and the stars in the sky.
turn background on	Creates a grid which represents the ground plane of the scene and stars in the sky.
turn grid off	Turns ground plane grid off but leaves stars.
turn grid on	Creates a grid which represents the ground plane of the scene.
turn stars off	Removes the stars from the sky.
turn stars on	Places stars in the sky.
make background shaded	Makes grid shaded.
make background wireframe	Displays grid in wireframe.
set grid blocks	This controls the size of the ground plane. Prompts user to input major and minor grid size.
turn projections on	Turns all orthogonal projections on.
turn projections off	Turns all orthogonal projections off.
turn x projection on	Turns on x orthogonal projections.
turn x projection off	Turns off x orthogonal projections.
turn y projection on	Turns on y orthogonal projections.
turn y projection off	Turns off y orthogonal projections.
turn z projection on	Turns on z orthogonal projections.
turn z projection off	Turns off z orthogonal projections.

DISPLAY MENU

make everything shaded	Display window in shaded mode
make everything wireframe	Display window in wireframe mode
make figure shaded	Displays specified figure in shaded mode.

make figure wireframe	Displays specified figure in wireframe.
make segment shaded	Displays specified segment shaded.
make segment wireframe	Displays specified segment wireframe.
make segment transparent	Displays specified segment as transparent.
turn segment on	A turned-off segment can be redisplayed by this command.
turn segment off	A turned off-segment is not displayed.
turn site on	Displays site as a read axes, labeled with x, y, and z.
turn site off	Turns site marker of specified site off.
turn segment sites on	Display all sites as crosshairs.
turn segment sites off	Don't display segment's sites.
turn nodes on	Displays nodes as crosshair.
turn nodes off	Cancels display of nodes.
turn edges on	Error: This command turns edges off.
turn edges off	Error: This command is not functioning correctly.
face enumeration on	The index of each face will be displayed in the center.
face enumeration off	Turns off display of face indices.
turn segment projection on	Turns on x, y, and z projections of segment.
turn segment projection off	Turns off x, y, and z projections of segment.
turn figure projections on	Turns on x, y, and z projections of figure (default).
turn figure projections off	Turns off x, y, and z projections of figure.

COLOR MENU

set major grid color	Sets color of major grid of ground plane.
set minor grid color	Sets color of minor grid of ground plane.
set major highlight color	Changes color Jack uses to indicate picked figures.
set minor highlight color	Changes color Jack uses to indicate picked figures.
set site color	Sets color of site marker.

set node color	Sets color of node markers.
set wheel color	Sets color of rotation indicator.
set spoke color	Sets color of rotation wheel spoke.

PARAMETER MENU

set scene scale	Default is 100, which means that a human figure fills most of the screen. The scale can be changed by this command.
set scene aux scale	Changes the auxiliary scene scale which includes sites, nodes, and rotation wheel. By default, the setting is 1/4 of the scene scale.
set line width	Default is 1. May be increased.
set distance units	Legal units are: mm, cm (default), m, in, ft,yd, mi.
set distance precision	By default, the precision is set to two decimal places.
set angle units	Legal units are: deg (default) and radians.
set angle precision	Change value from default 2 decimal.
set mass units	Legal units are: g, kg, and lb.
set mass precision	Change value from default 2 decimal.
set rotation type	Prints the rotation part of the homogeneous transformation either in terms of xyz (default) or the quat operator.
set peabody path	Sets the path to the peabody library.
set view glide	Select mouse sensitivity for view command.
set move glide	Select mouse sensitivity for move command.
change directory	Lets you change directory from within Jack.

SIMULATION MENU

simulation off	Simulation turned off completely.
set simulation level 1	Default. Jack evaluates the simulation function only during object manipulation: moving figures, adjusting joints, etc.
set simulation level 2	Continuous evaluation of the simulation function.
single set simulation	Simulation stops when minimum of function is found.

read squash	Define deformation matrix.
generate scaling squash	Lets you deform figures.
bind joint to port	Waits for a specific connection on a specific port.

RETINA DISPLAY MENU

create retina window	Creates a window displaying a retina, with the view projected onto it in retinal coordinates.
create field of view cones	Prompts for figure selection. Draws 60^o field of view cones from each eye of the selected figure.
fixate eyes on site	Fixates the view to a specified site. Draws lines from each eye to the site.
interactive fixation	The fixation point can be moved interactively.
monocular/binocular map	By default, the retina window shows the images perceived by both eyes. To simplify, a monocular view can be displayed.
zoom in/out of retina window	Zoom toggle.
show eye scan	Trace of eye scan.

RETINA EDITOR MENU

create retina editor	Creates a retina window into which user may draw.
draw left retina object	Accepts user input for right retina.
draw right retina object	Accepts user input for left retina.
draw filled retina object	Displays retina object in shaded mode.
retroject left retina	Object of left retina is projected into 3-dimensinal space.
retroject right retina	Object of right retina is projected into 3-dimensional space.
retroject both retina	Objects of both retina is projected into 3-dimensional space.
clear retina objects	Clears retina window
save retina objects	Writes coordinates of retina objects to file.
load retina objects	Reads objects into retina editor window from file.
zoom In/Out of editor window	Zoom toggle.

VISION INFORMATION MENU

create adaptation info window	Creates a window that gives information about: retinal map orientation, field of view objects, and fixation distance.
create legend window	Creates a legend window that gives information about color codes etc.
create Aitoff window	Creates an Aitoff window with the view from a specified figure.
initialize state info	Initializes the vision parameters to default values.
set state information	Creates a meter window that allows to adjust clipping planes, lumination and illumination parameters.

TRACE MENU

trace site	Creates a trace of the site if the segment, containing the site, is moved.
untrace site	Turns off tracing of site but leaves trace intact.
trace segment	Creates a trace of the segment if the segment is moved.
untrace segment	Turns off tracing of the segment, but leaves trace intact.
set trace color	The color of an existing trace can be changed.
clear trace	Clears trace without disabling it.
delete trace	Deletes trace and turns off tracing.

COMMAND MENU

help for command	We don't have on-line help.
help by subject	We don't have on-line help.
keymode on	When turned on, expects keyboard input.
keymode off	Returns mode to default.
disable WM quit	Prevents user from accidentally quitting jack.
disable graphics	Window will not be drawn until executing "enable graphics".
enable graphics	Redraws window after it has been disabled.
bind command to key	After a command is bound to a key, the command can be executed by pressing the specified key.

describe key bindings	Displays a list of all the key bindings.
create communication port	Waits for a connection by a client program to be established.
send to communication port	Lets user send message to communication port.
close communication port	Closes a communication port.

ANIMATION MENU

help	Prints instructions for using the animation menu.
describe script	Prints script on screen
initialize animation	This command needs to be called once before the animation process can be started. For simple animation, the default values may be accepted.
create key frame from current state	A key frame is generated as part of the animation script. As a keyframe normal time, choose 0 for first and 1 for last keyframe and values such as 0.2 , 0.6 in between.
delete keyframe	Removes a keyframe from the script.
change key frame to current state.	Changes a keyframe from the script.
change keyframe normal time	Lets you changes the normal time of a keyframe
change frame rate	The default is 30 frames per second.

GROUP MENU

create figure group	Uses all joints of a figure plus its location (root).
create figure joints group	Uses all joints of a figure.
create figure location group	Uses a figure's location (root).
create camera group	Uses the camera's location.
create joint group	Uses a single joint.
create segment nodes group	Uses nodes of a segment. (Used for deformation)
create mixed group	May use any number of members of the above type.
change group name	Replaces old group name.
describe group	Prints all group members on screen.

delete group	A group may be deleted.

ACTION MENU

choose current action	Current action is the most recently created action.
create action	Create a new action.
change action length	The action time, which was set by "initialize animation, may be changed by this command.
change action start time	With multiple actions, the start times can be changed.
change action interpolation	Choose between spline (default) and linear.
change action name	This does not create a file.
delete action	An action can be deleted.

PLAYBACK MENU

step through key frames	Lets you look at the key frames you have created.
playback	Single playback of animation.
step through interpolated frames	Steps through the in-between frames the program has generated.
repeat playback	Continuous playback of the animation.
playback in real time	Playback speed may not be consistent.
special playback	Use for squashed objects.

READ/WRITE RECORD MENU

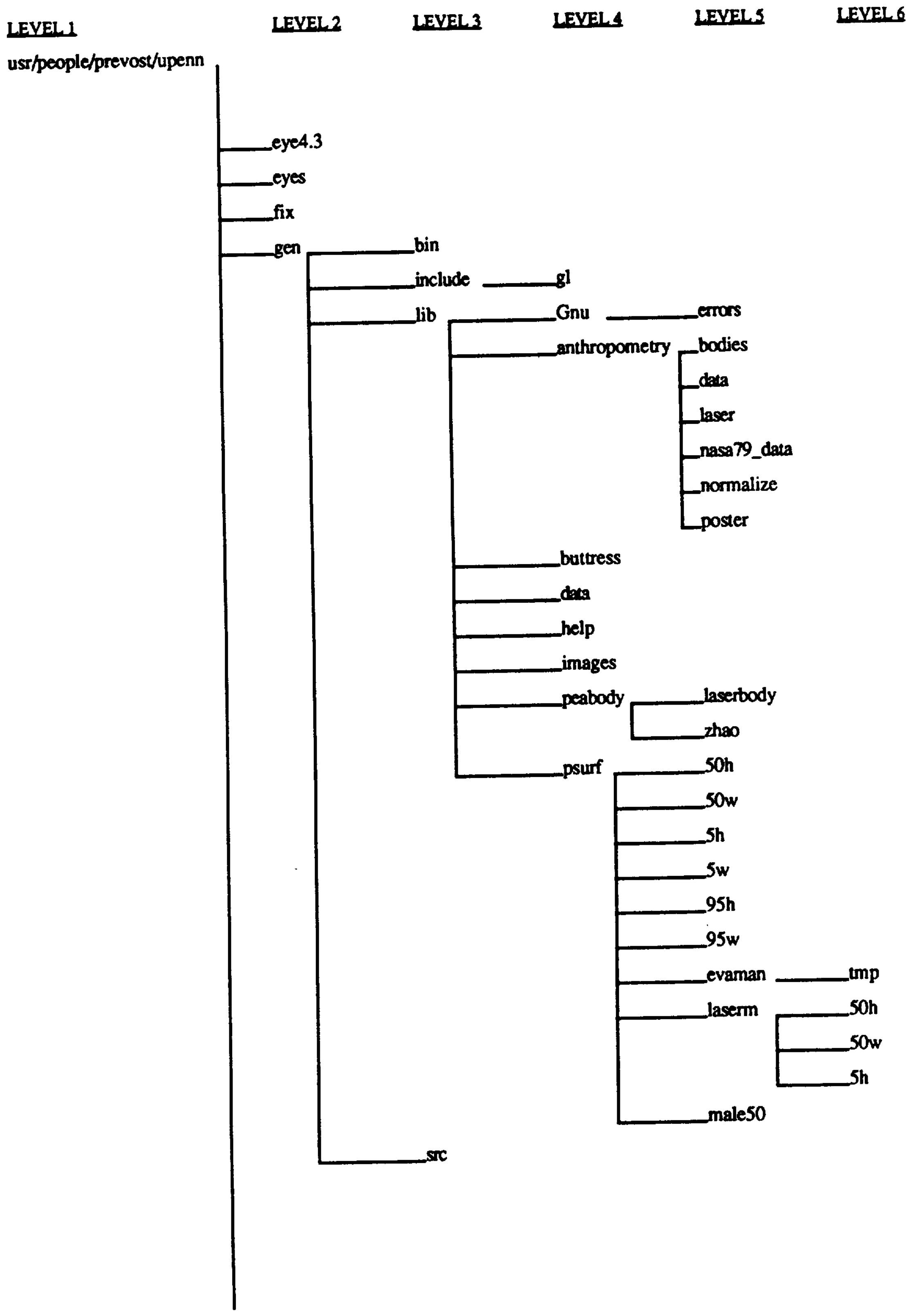
read script file	Reads a file with a .scr suffix.
read action file	Reads a file with a ACT
write script file	Generates environment, script, and action files.
write action file	Generates action file only.
save complete environment files for rendering	Generates environment files of key frames.
save incremental environment	Generates environment files of incremental frames.

files for rendering	
record animation	Recording animation on video.

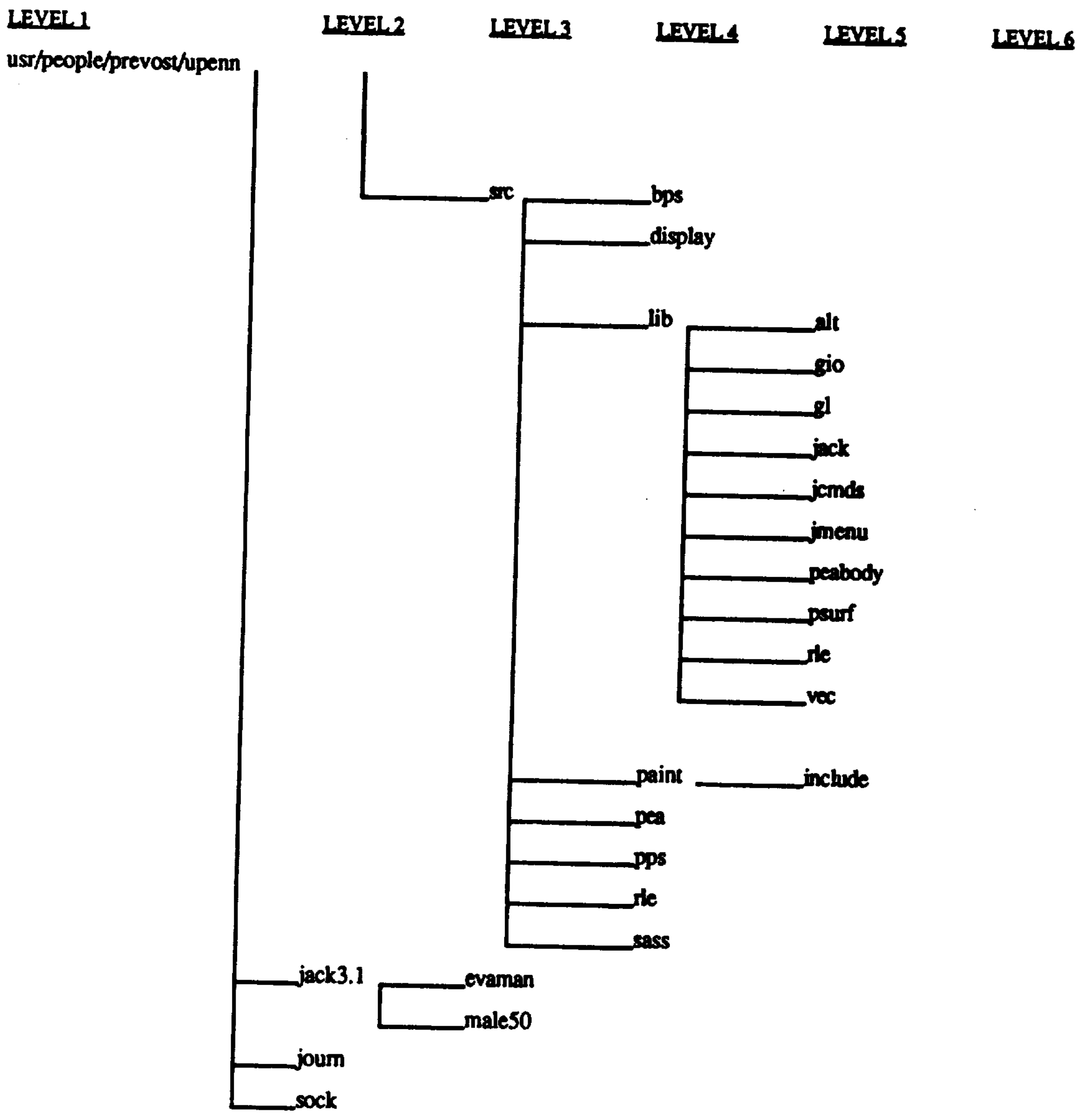
APPENDIX C — JACK DIRECTORY STRUCTURE

JACK DIRECTORY

```
LEVEL 1                    LEVEL 2   LEVEL 3    LEVEL 4    LEVEL 5    LEVEL 6
usr/people/prevost/upenn
    4.4
        4D
            bin
        eye
        gen
            include
                gl2
            src
                jack
                lib
                    gio
                    alt
                    gl
                    jack
                    jcmds
                    jmenu
                    peabody
                    psurf
                    rle
                    vec
        lib
    4.5
        4D
            lib
        gen
            include
                gl2
                    ipII
            src
                4.5orig
                jack
                    vp_src
                lib
                    grace
                    jack
                    jcmds
                    jmenu
                    peabody
                    psurf
                    vec
                xtend
        longbow2
        mdhc
        wave
    4D
        bin
        lib
    demo
        apache
        tank
        test
```



```
LEVEL 1                   LEVEL 2   LEVEL 3     LEVEL 4          LEVEL 5        LEVEL 6
usr/people/prevost/upenn
    |__eye4.3
    |__eyes
    |__fix
    |__gen
         |__bin
         |__include
         |      |__gl
         |__lib
         |      |__Gnu
         |      |     |__errors
         |      |__anthropometry
         |      |     |__bodies
         |      |     |__data
         |      |     |__laser
         |      |     |__nasa79_data
         |      |     |__normalize
         |      |     |__poster
         |      |__buttress
         |      |__data
         |      |__help
         |      |__images
         |      |__peabody
         |      |     |__laserbody
         |      |     |__zhao
         |      |__psurf
         |            |__50h
         |            |__50w
         |            |__5h
         |            |__5w
         |            |__95h
         |            |__95w
         |            |__evaman
         |            |     |__tmp
         |            |__laserm
         |            |     |__50h
         |            |     |__50w
         |            |     |__5h
         |            |__male50
         |__src
```


LEVEL 1
LEVEL 2
LEVEL 3
LEVEL 4
LEVEL 5
LEVEL 6
usr/people/prevost/upenn
src
bps
display
lib
alt
gio
gl
jack
jcmds
jmenu
peabody
psurf
rle
vec
paint
include
pea
pps
rle
sass
jack3.1
evaman
male50
journ
sock

APPENDIX D — ANTHROPOMETRY DEMO 1990

ANTHROPOMETRY DEMO 1990

INTRODUCTION OF ANTHROPOMETRY ANALYSIS BY B. SMITH:

One of the most fundamental requirements for MIDAS is a physical representation of the human operator to explore their interaction within the geometry of a crew station or aircraft. To meet this need we have Jack, a dynamic anthropometric model developed through a grant with Dr. Norm Badler at the University of Pennsylvania. Jack is an interactive graphic package that allows the creation and manipulation of 3-D human figures, both statistical and individual, in a 3-D object space, allowing a designer to explore reach, fit, and human motion. Christian Neukom will demonstrate several of the features of this model and following him will be a discussion by Dr. Norm Badler from UPenn, to describe a little bit about where he intends to take Jack in the Future.

INITIAL SETUP:

```
open window; god1; stow
open window; god2; stow
open window; god3; stow
open window; god5; stow
```

START DEMO:

DEMO1 (Display of a selection of polybody and biostereometric figures)

pop first window

Jack is capable of displaying and manipulating a range of either individually defined or statistical body figures. Here I'm showing you a selection of polybody figures including the 5th, 50th, and 95th percentile male and female based on the NASA Aircrew Demographics for the Year 2000 contained in NASA std 3000. Also shown is a 95th percentile by height biostereometric male figure. Associated with these figures is a great deal of anthropometry data regarding segment size, strength, center of mass, and joint limits, which I will go into more detail on later. What I want to show you now is a couple of the new features of Jack.

quit jack window

DEMO2 (Female polybody figure is set-up to do balanced reach male polybody figure has two hands constrained to a box)

pop window2

do balanced reach (ready to go, just move mouse)

One of the new features of Jack is balanced reach. I'm demonstrating it with an interactive reach toward the floor. As you can see, the mannequin uses the opposite leg to balance its body.

hit Esc to quit reach

move figure (box) (up/down and forward/backward only)

Another addition is the facility of the various reach constraints. The mannequin's hands are constrained to the sides of the box. As I move the box, the hands follow simulating handling of the box. This feature is useful when the mannequin interacts with a maintenance environment for example. I have turned on the site tracing on the mannequin's left hand. The trace, shown as a yellow line, shows us the exact path and indicates if the hand went through any objects since there is no built-in collision detection and avoidance.

quit jack

DEMO3 (Co-pilot Gunner crewstation with polybody figure doing a reach with the right hand to the left Multi Functional Display in an animation script. A "figure Info" command has been inserted into the script to prevent the animation from executing - an Esc will start the animation)

Jack can be used to explore human figure interactions in a 3-D space environment either interactively as I am doing now, through animation script files, or controlled through our integrated simulation as you will see later. What I'll do now is import one of our Longbow crewstation files created by S Systems from Apache source data and execute a few movements.

pop window3 (reach animation)

I've loaded the Co-pilot Gunner crewstation, with several extraneous portions deleted to speed up graphics processing. With the new Longbow cockpit changes, one of the questions a designer may want to answer is if the two Multi Functional Displays can comfortably be used with either hand, since a great many cockpit functions are enabled by activating soft-switches on the MFD touch screens. I've placed a 95th percentile figure into the seat and as you can see, a rather large Optical Relay Tube protrudes significantly from the front panel in between the left and right MFDs.

hit Esc to start animation

I will now run an animation script that I previously set up to see if the Co-pilot Gunner can reach the left MFD with his right hand. I have constrained his left hand to the collective and I'm allowing him to move through his waist--so his seatbelt inertial reel is not locked. Now generally, the left MFD is operated with the left hand and the right MFD with the right hand but there may be situations when this is not possible - for example if one of the hands is busy with another task or one of the MFDs is out of order. We can conclude from watching this animation that it is fairly cramped quarters and the Co-Pilot Gunner has to contort a bit to perform this task.

quit jack

DEMO4 (Biostereometric figure wearing IHADDS helmet, visor and monacle. Figure is displayed in 3-D window and in 3 orthogonal windows)

god4

Another one of the new features of Jack concerns body figure data. While the polybody figures are quick to generate and manipulate, Dr. Badler has incorporated some biosteriometric scanned body images from the Airforce which provide a more pleasing graphical image together with more details in certain segments of the body such as the torso and head. These figures, one of which is shown here, contain over 6000 data points each and are based on actual subjects from three different somatotypes in each sex. I am also showing an Apache IHADDS helmet, visor, and monacle with this figure since we've received a number of questions about wether the effects of protective clothing or flight gear can be represented in Jack. The three orthogonal windows are also a new feature, and are extremely handy to line up objects such as this helmet for proper placement.

move helmet

Jack also allows you to turn individual portions of figures into wireframes or transparent as shown in the visor.

quit Jack

DEMO5 (Apache Longbow helicopter with two biostereometric figures seated in crew position)

pop window 3

Before I leave Jack, I'll bring up a window showing you the complete Apache Helicopter with two Biostereometric figures seated in the crew positions. While impressive to look at, this figure has about 10'000 polygons, and it unfortunately slows down the graphic processing enormously, especially in shaded mode. As these hardware platforms evolve, perhaps we will be able to easily manipulate environments as complex as this one.

DEMO6 (Spreadsheet Anthropometry Scaling System, SASS)

go through poster first

god6

Displayed here on the screen is the anthropometry spreadsheet displaying the segment dimensions of the 50th percentile. Other anthropometric groups can easily be selected to view or modify the data. Below is a summary of the data being displayed to allow the user to have a "global" view of the figure he is working on. Presently there are eleven labels defined: population, gender, mass, stature, group percentile, strength type, motion, speed, handedness, training, and fatigue level.

select Bottom Head and change y to 3.25"

select Group %ile and change to new %ile

Let's look at an example. Let's say we want to find out what percentile figure can wear a certain sized helmet. We simply input the dimension (3.25") in the respective cell and read the corresponding percentile. We can then change the group percentile to display the other dimension of that group.

click on Query --> Database Query Screen

click on Input Data; from keybord enter: people.db

click on Query Database and build query:

male, stature > 160 and stature < 170, done

Let us now turn our attention to the database query spreadsheet. First I'm going to load the person database. Let's say we want to know, if there are any individuals in this database who are male and have a height of 160 - 170 cm. The query is conveniently built through menu selection. We come up with 3 individuals who satisfy these requirements. The data of these individuals could now be displayed in the anthropometry spreadsheet and their figures displayed in Jack.

Summary

This concludes the anthropometry presentation. I had only time to show you a small selection of the features of Jack and SASS. If you like to know any other features or more details about the ones I presented, please come to me in the individual discussion session after the formal demo.

APPENDIX E — README FILE

Instructions for Installing Jack from tape.

INSTALLING THE JACK SOFTWARE

Create a directory where you want to copy your files to, i.e.

```
            mkdir /usr/upenn    (if you are super user)
      or  mkdir upenn          (if you are in your home dir)
```

Remember that you need 41 Mb of disk space.

Read the tape into this directory, i.e.

```
      tar xvo
```

When the software is extracted from the tape, it will create the following directories:

4.5	contains source files
4D	contains executable files
gen	contains figure and other data files
sass	contains files used by the Spreadsheet Anthropometric scaling system
Jackshell	Source this shell to set up environmetal variables used by Jack and sass.
Readme	This file

Edit the Jackshell and change the UPENN environmental variable. UPENN contains the absolute path from the root to the directory into which you copy the files from the tape. Find the line "setenv UPENN /usr/upenn" and change the path.

After making the changes source the Jackshell to set all the paths:

```
            source Jackshell
```

To run Jack, change to the 4.5/longbow directory and run Jack:

```
            cd 4.5/longbow
            jack vis.jcl       (vis.jcl is an environment example)
```

If Jack runs but cannot find the files it needs to load, check the path set in UPENN. If it runs and the crashes, it might be because of a different version of the operating system. In this case, try to relink Jack by executing:

```
cd $UPENN/4.5/gen/src/jack
make
```

if there are still problems, try to recompile the complete program by executing the following commands:

```
cd $UPENN/4.5/gen/src/lib
make
```

This command will execute a makefile in each of its subdirectories. After the make completes run the command

```
$UPENN/4.5/gen/src/jack/make.
```

If, after undertaking all of these steps, you have still problems, please contact Mike Prevost at E-mail address:

prevost@eos.arc.nasa.gov

or Christian Neukom at:

neukom@eos.arc.nasa.gov

To run sass, change to the sass directory and type sass. If it doesn't run, relink it by executing make in the sass directory.

Annex H

Army-NASA Aircrew/Aircraft Integration Program: Phase IV

A^{3}I

Man-Machine Integration Design and Analysis System (MIDAS)
Software Detailed Design Document

Vision Models

prepared by

Michael Prevost

Table of Contents

Table of Contents

Table of Contents

Table of Contents

1.0 INTRODUCTION

1.1 IDENTIFICATION OF DOCUMENT

This is the Software Product Specification for the Vision analysis modules of the Man-machine Integration Design and Analysis System (MIDAS). Included is documentation for the David Sarnoff Legibility Model (LM), the Lighthouse's Volume Perimetry Model (VP) and the Vision Enhanced Jack Interface (VEJI). These three components combine to form the MIDAS vision model. Documentation for the standard Jack software is documented in Annex G. The vision model (inside the circle area) is shown in the context of the entire MIDAS system in Figure 1.

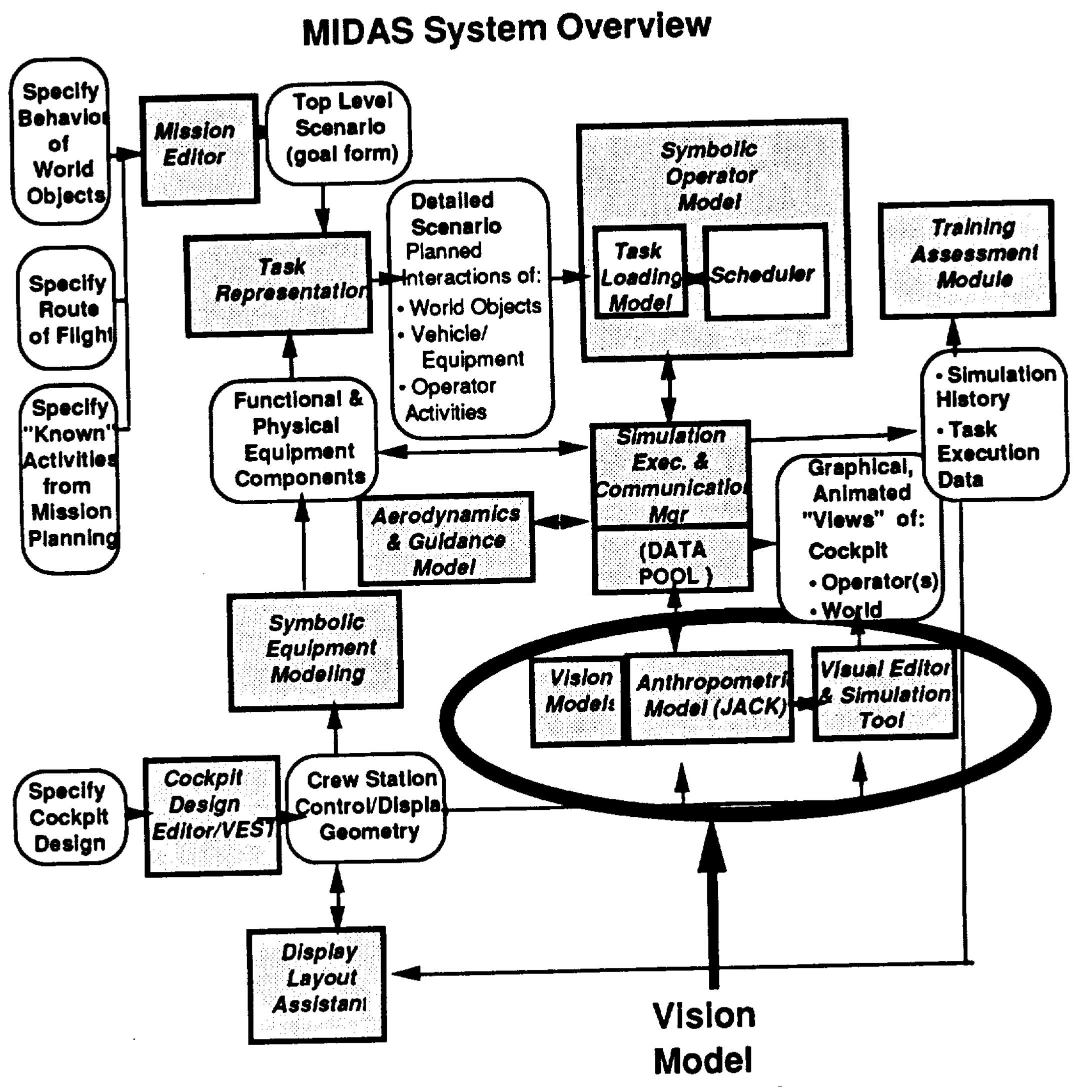

MIDAS System Overview
Specify Behavior of World Objects
Mission Editor
Top Level Scenario (goal form)
Symbolic Operator Model
Task Loading Model
Scheduler
Training Assessment Module
Specify Route of Flight
Task Representation
Detailed Scenario Planned Interactions of: • World Objects • Vehicle/ Equipment • Operator Activities
Specify "Known" Activities from Mission Planning
Functional & Physical Equipment Components
Simulation Exec. & Communication Mgr
(DATA POOL)
• Simulation History • Task Execution Data
Aerodynamics & Guidance Model
Graphical, Animated "Views" of: Cockpit • Operator(s) • World
Symbolic Equipment Modeling
Vision Models
Anthropometric Model (JACK)
Visual Editor & Simulation Tool
Specify Cockpit Design
Cockpit Design Editor/VEST
Crew Station Control/Display Geometry
Display Layout Assistant
Vision Model

Figure 1. MIDAS Vision Model Overview

1.2 SCOPE OF DOCUMENT

The high rate of change of features and the exploratory nature of the vision model makes documentation a difficult task. It should be emphasized that the software that this document pertains to is still under development. What this document gives the reader is a snapshot of the current state of a continuously evolving software package.

Because the software is under development there has been little effort made to bullet proof the interface and no attempt to optimize the code. There are functions that are incomplete and may not work well due to their early phase of development. There are ways to view and plot data that may lead the user to erroneous conclusions. Most of the data presented by the vision model has not been validated. There are known inaccuracies in the mapping functions.There will be no attempt made to stay upwardly compatible, consistent with, or many of the other conventions that the user may expect from a more mature commercial software product. The ability for me to support, answer questions or help users may be very limited.The next phase of development will involve substantial changes to the software architecture of the model.

In short this is developmental software and not production software. The following standard disclaimer applies: there are no warranties, either expressed or implied, regarding the enclosed computer software package, its merchantability, or its fitness for any particular purpose.

1.3 PURPOSE AND OBJECTIVES OF DOCUMENT

This document is intended to convey to the user the class of vision phenomena supported by the model, and how he/she can vary the parameters of the vision model in order to answer crewstation design related questions. The software requirements and rationale are explained and the limitation and tradeoffs of the models enumerated.

2.0 RELATED DOCUMENTATION

2.1 APPLICABLE DOCUMENTS

The following documents are referenced herein and are directly applicable to this volume:

A. Arditi, The Volume Visual Field: A Basis for functional perimetry, *Clinical Vision Sciences,* 3 (1988).

A.Arditi,. Alternate Representation of Visual Space, SPIE Vol. 1083 Three dimensional Visualization and Display Technologies., 242-245,1989.

C.A. Curcio, et al., Human Photoreceptor Topography, *The Journal of Comparative Neurology* 292:497-523 1990.

C.R.Carlson and R.W. Cohen, A simple Psychophysical Model for Predicting the Visibility of Displayed Information, *Proceedings of the SID*, 21, 229-246 (1980).

J. Larimer, et al, A Computer-Aided Tool for Assessing the Visibility of Cockpit Displays, *Society For Information Display 90 Digest* 133-135, 1990.

E Adelson, C. Carlson, A. Pica, Modelling the Human Visual System, *RCA Engineer* 27-6 Nov./Dec. 1982.

E Adelson, et al, Pyramidal Methods in Image Processing, *RCA Engineer* 29-6 Nov./Dec. 1984.

C. Hall, E. Hall, A Nonlinear Model for the Spatial Characteristics of the Human Visual System, *IEEE Transactions on Systems, Man, and Cybernectics,* Vol. SMC-7, No. 3, March 1977.

R. Hall, *Illumination and Color in Computer Generated Imagery,* Springer-Verlag, New York, New York, 1989.

IES Lighting Handbook , 1984 Reference Volume

2.2 INFORMATION DOCUMENTS

The following documents amplify or clarify the information presented in this volume:

R.Blake and R. Fox, The psychophysical inquiry into binocular summation, *Perception and Psychophisics,* 14, 161-185 (1973).

Bouma, H. Visual recognition of isolated lower-case letters. *Vision Research 11,* 459-474 (1971).

Coffin, S. Spatial frequency analysis of block letters does not predict experimental confusions. *Perception and Psychophysics* 23(1), 69-74 (1981).

Egeth, H.E., and Santee.J.L. Conceptual and perceptual components of interletter inhibition. *Journal of Experimental Psychology* 7(3), 506-517 1981.

B. Esterman, Functional scoring of the binocular field, *Opthalmology,* 89, 1226-1234.

Gervia,M.J., Lewis O. Harvey, and Roberts, J.O. Idntification confusions among letters of the alphabet. *Journal of Experimental Psychology* 10(5) 655-666 1984.

Geyer,L.H., and DeWald, C.G. Feature list and confusion matrices . *Perception and Psychophysics* 14(3), 471-482 1973.

Geyer, L.H. and Gupta, S.M. Recognition/confusion of dot matrix vs. conventional font capital letters. *Perception and Psychophysics 29,* 280-282, 1981

Gilmore,G.C.,Hersh,H.,Caramazza,A., and Griffin, J. Multidimensional letter similarity derived from recognition errors. *Perception and Psychopysics* 25, 425-431, 1979.

Heijden, A.H.C., Malhas, M.S., and Roovaart, B.P.. An empirical interletter confusion matrix for continuous-line capitals. *Perception and Psychophysics* 35(1), 85-88, 1984.

Mewhort, D.J.K., and Dow,M.L.. Multidimensional letter similarity: A confound with brightness? *Perception and Phychophysics* 26(4), 325-326, 1979.

Morrison, I.A LISP program to determine similarity relations in letter displays. *Behavior Reseach Methods and Instrumentation* 15(1), 69-71, 1983.

Townsend, J.T., Theretical analysis of an alphabetic confusion matrix. *Perception and Phychophysics* 9(1A), 40-50, 1971.

Watson, A.B., The ideal observer as a modelling tool. Frontiers of visual science: Proceedings of the 1985 Symposium (pp. 32-37). Washington D.C.: National Academy Press.

Watson, A.B., and Fitzhugh, A.E., A new look at letter recognition. *Perception* 15(1), A31, 1986.

Watson, A.B., and Fitzhouh,A.E., Modelling character legibility, *Society for Information Display Digest of technical Papers* 20, 360-363, 1989.

3.0 CONCEPT

3.1 DEFINITION OF VISION MODELS

3.1.1 PURPOSE AND SCOPE

In a crewstation (e.g. an aircraft cockpit) the ability of the operator (pilot) to unambiguously perceive rapidly changing information both internal and external to the crewstation is critical. To assess the impact of crewstation design decisions on the pilot's ability to perceive information, the designer needs a means of evaluating the trade-offs that result from different designs. There exist many CAD tools for creating and manipulating geometrical objects but few contain any vision capabilities. Of the ones that do have the ability to address some visual aspects of design, only the geometrical questions eg. vision plots, field of view, etc. are answered. Others (Watson, Bergen,Hall) have developed tools to predict visual performance but they have never been integrated with a CAD-like tool. Because of the interactive nature of design it is necessary to provide feedback on visual performance as a continuum of design parameters are varied. Without integration the usefulness of vision models to the design community is greatly limited.

The process of crewstation design involves a large number of tradeoffs. The purpose of the vision model is to provide an easy to use interactive interface to vision relevant parameters of crewstation design, and through data visualization techniques provide feedback to the designer on the effects of varying these parameters to the appropriateness of the design.

3.1.2 GOALS AND OBJECTIVES

The primary goal of the vision model is simply to explore ways to better communicate vision related design issues and provide information for design changes that would result in a better overall design. It is beyond the capabilities of the current vision model and the scope of our simulation, to predict when **all** of the necessary conditions are present for proper perception to occur. The goal of the vision model is to show only when it is at least possible to perceive the stimuli and in the case of MFD characters their probability of correct discrimination. Some primary areas of focus will be on character legibility as displayed on the Longbow MFD, field of view volumes and intersections, performance data linking to the CAD representation of the design and a polar foveal centric representation of objects in the design.

3.2 USER DEFINITION

The vision model assumes that the user is an experienced crewstation designer but not necessarily adept at applying human factors principles to all aspect of the design. It is expected that the designer is using this tool in an iterative fashion to alter the design to meet requirements and still be consistent with human factors guidelines.

3.3 CAPABILITIES AND CHARACTERISTICS

The vision enhancements to the Jack software provide the designer, for the first time, an integrated environment featuring CAD like capabilities, an anthropometric model and a visual performance model. Inside this environment an entire spectrum of design trade offs can easily be explored. Through a simple interactive interface, a designer can manipulate design parameters such as cockpit layout geometry, environmental factors such as ambient lighting, pilot parameters such as point of regard, and equipment parameters such as display size and contrast of displayed symbology. These visibility options provide an end to end analysis that answers questions such as "Is it possible for the pilot to read the display?".

Visual performance data can be projected, in the form of 3D contours, into the crewstation graphic model providing the designer with a footprint of the operator's visual characteristics given the current parameter settings.

In Appendix A the figured labeled Binocular Vision Model provides a good overview of the of the complete vision model.

3.4 SAMPLE OPERATIONAL SCENARIOS

The following is a sample scenario intended to demonstrate how one might typically use the vision model. The scenario has three parts: First, an example of how to use the designer's interface, second, the retinal window projections, and thirdly, the retinal editor, including the use of discrimination data.

You must first follow the standard startup procedure for Jack. The exact commands used will vary according to the particulars at your installation site but would look something like the following:

1) **Initial setup:**

```
cd /usr2/neukom2/4.5
source sshell ( cshell if on coral)
cd longbow2
djack vis.jcl
```

This should start up the proper version of Jack and bring up a subset of the pilot crew station. It will put Jack where and how he belongs. The first example shows how to vary the fixation point. This must be done in order for other options to make sense. An example of view cones that are drawn down the newly changed visual axes is given for further clarity.

2) **Demonstrate the User interface:**

```
select -> options menu:retinal display:        fixate eyes on site
select -> jack
select -> site on left MFD
```

click left mouse button again to show the visual axis
press esc key

select -> options menu:retinal display: **field of view cones**
select -> jack
enter -> fov for cone type (message window or mouse)
WARNING: These cones can not be deleted after you create them without crashing the system.

select -> options menu:retinal display: **interactive fixation**
select -> jack
move fixation point via mouse
WARNING: eyes are joint limited. You will see the lines representing the visual axes stop moving when you reach a joint limit. Also an out-of-limit message above the fixation distance and angle information (lower left corner) will be displayed.
press esc key.

Next an example of how to use the retinal window is given.

3) Demonstration of retinal window:

select-> options menu:retinal display: **create retinal window**
select-> jack
open window in upper left corner.
WARNING: This window only redraws when information is changed in the window. If you move the window it will not redraw until you make a menu selection that changes what is displayed in it. I usually overlay it on top of the retinal window to conserve screen space.

select->options menu:retinal display: **Monocular/binocular**
select->option:vision information: **set pilot state open window**
set near clipping to a small value ~10 by clicking right mouse button near bottom of the "near" slide bar.
set far clipping to ~300 by clicking right mouse button near top of the "far" slide bar.
press esc key.

select->options menu:retinal display: **interactive fixation**
select -> Jack
move fixation point around via mouse
press esc key.

The retinal editor is the primary means by which performance data is introduced in Jack. Known data can be drawn directly on the editor and then saved to a file. The file can then be loaded in at a later time when that data is desired. For ease of demonstration, some menu selections have been created so that the user can load a file directly, without having to type the name of the file. In general, this is not the case. When the user desires to load a file that he/she has created it will be necessary to type the name of the file before loading it into the system.

4) Demonstration of retinal editor:

Drawing of an arbitrary object:

select->options menus:retinal editor: **create retinal editor**
open window in upper right part of screen

select->options:retinal editor: **draw right retina**
draw in retinal editor by moving the mouse and
pressing the left mouse button, end drawing by
pressing the middle button.
WARNING: nothing shows in the retina editor as you draw in it on any of the
GTX machines.
select->options:retinal editor: **clear retinal objects Loading and retrojecting published data:**

select->options menus:retinal editor: **load cone density data**
select->options menus:retinal editor: **retroject both**
select->options menus:retinal display: **interactive fixation**
select->Jack
move fixation point via mouse
select->options menus:retinal editor: **clear retinal objects**

Sarnoff Confusion data:

select->options:retinal editor: **load OQ confusion data**
select->options menus:retinal editor: **zoom editor window**
select->options menus:retinal editor: **draw filled retinal objects**

select->options menus:vision information: **set pilot state open window**
set ill to high value (> 4) by clicking near the top of the slide bar.
press esc key
select->options menus:retinal editor: **clear retinal objects**
select->options menus:retinal editor: **load OQ confusion data**

4.0 REQUIREMENTS

4.1 REQUIREMENTS APPROACH AND TRADEOFFS

Geometrical data such as the volume field of view, occlusions, facial geometry and helmet margins needs to be projected into the cockpit with respect to the coordinates of the aviator's eyes and fixation point. The intersections of the projections with objects in the crewstation will delineate the area of coverage, masking, or occlusion associated with the objects. This must be done at updates speed of at least 1HZ to make interaction with the model acceptable.

Objects in the crewstation space shall be projected onto models of the operator's retinas. These projections can be used to provide the designer with the retinal coordinates and the visual angles subtended by objects in the crewstation space. Both the right and left eye retinal projections shall be mapped. The retinal map shall be yoked to the fixation point and change as the fixation point is interactively manipulated. Performance contours on the retinas can also be indicated thus aiding the designer in understanding the limitations to visibility imposed by retinotopic processing.

The performance contours shall be generated based on a discrimination model that takes as input the aviator's adaptive state, environmental factors and stimulus description and generates iso-discrimination contours. These performance data must be mapped back into the environment and linked to their stimuli.

4.2 HARDWARE ENVIRONMENT

The three software components, Volume Perimetry (VP), Legibility Model (LM) and Vision Enhanced Jack Interface (VEJI) can be run as an integrated program on the Silicon Graphic 4D series of workstations. These workstations have double buffered, 24 bit color, 8 bits alpha and Z buffered frame buffers. The workstations have hardware support for light models and viewing transformations. The 4D 220 GTX (MIDAS' fastest workstation) has two RS2000 25MHZ CPUs for a total of 40 MIPS. The software makes use of only one CPU at this time although this will change next phase. The recommended memory is 32MB, but this is strongly dependent on the details (number and size of polygons) of the environment. We have 32MB on the 220GTX. For medium size environment (~ 3000 polygons) this program is graphic bound, even though the 4D 220 GTX is capable of 100,000 (10 pixel, lighted, Gouraud shaded triangles) polygons/sec. The problem is exacerbated as more windows are opened because the entire scene must be drawn multiple times.

While these workstations provides the performance we require for the integrated VEJI, VP, JACK and LM can be run independently on less costly workstations. An earlier version of VP ran on an Amiga. A newer version is under development for the SGI Personal Iris. A version of LM runs on the Sun family of workstation and has no graphic requirements.

4.3 SOFTWARE ENVIRONMENT

The entire visibility software modules have about 100,000 lines of code and are completely written in the C programming language. It is comprised of three major software components.

The first, VP, which stands for Volume Perimetry, is the portion of the software that represents the Volume Visual Field (VVF) and retinal images as distinct graphical constructs in separate windows that are yoked to the fixation point. VP illustrates the effects of changes in the fixation point with respect to the VVF or object image projections onto the observer's retina. This software is located in the directory of $UPENN/gen/src/jack/vp_src and was supplied under a contract from NASA-Ames A^3I by Aries Arditi and Steve Azueta from the Lighthouse Inc. Research Labs.

The second software component, the Legibility Model (LM), computes the probability for correct discrimination between two symbols based on stimulus characteristics, environmental factors and observer state. This software was produced at SRI/David Sarnoff Research Center by Jeff Lubin under contract from NASA-Ames A^3I. Due to the intensive computational requirements of the model, and the desirability of building an interactive visibility tool, legibility data is precomputed and stored in files. LM is a completely stand-alone program and only the data files are access during the running of VEJI. During operation of the VEJI, based on the current vision relevant parameters the appropriate data files are read in and displayed. Currently the data files must be located in the same directory that the executable called from. This restriction will be changed in a future release.

The third software component is Jack, the anthropometric modeling tool. This software provides the kinematically correct models of stereotypical male/female body types. The software was produced at the University of Pennsylvania by Norman Badler and Cary Phillips under a partial grant from NASA-Ames A^{3}I.

4.4 EXTERNAL INTERFACE REQUIREMENTS

The standard Jack menu and mouse interface was extended to allow the user access to the vision model and control of the vision model's parameters. The general interface to Jack is described in the standard Jack documentation.

There are no network interfaces for the vision model. However, future plans may incorporate a network connection. Jack has an existing network connection capability (Badler 1989) that will allow controlling the movement of the manikin. This interface was found inadequate for control in the A^{3}I environment and a secondary network connection function was created. See AnnexJ, the documentation of the A^{3}I Communication module, for further details.

5.0 DESIGN

5.1 ARCHITECTURAL DESIGN

5.1.1 Design Approach and Tradeoffs

The design of a vision model that could predict the sufficient conditions for the observer to perceive the a stimulus is beyond the current capabilities of the MIDAS simulation. There are many ways in which perception can be influenced by factors such as cognitive loading and inattention to tasks that make it difficult to say when a human observer would perceive a given stimulus with certainty. At this point the best that can be done is to say that it is possible/impossible to perceive the stimulus within a given probability. The first design tradeoff was to settle on a rather modest set of visual properties to model in order to produce performance probability contours. Still, many design factors can be evaluated using just what we know about these early vision properties.

A second tradeoff was to keep the system interactive. Design is such an iterative process that non-interactive design tools can increase design time significantly.

5.1.1.1 Legibility Model

It has been shown [Ginsburg 84] that contrast sensitivity is a more accurate predictor of visual performance than the standard Snellen type visual acuity test. For this reason the legibility model is sensitive to changes in contrast and is similar to Watson cortex model [Watson 85] with some notable extensions.

Due to the computational demands of the model, the time required to compute the results (discrimination values sets) takes between 1 and 2 minutes on our fastest machine, a 4D 220 GTX. Since it was important to keep the system responsive to the user these data sets needed to be precomputed. At run time the state parameters are constrained so they may only be set to values that have a data set associated with them. The legibility model is still under development and has not been optimized. An initial look at the legibility model suggests that it could be optimized to compute a data set in under 20 seconds on a 4D 220 GTX.

The legibility data is projected with the same functions that are used to project retina objects into Jack's environment. This was the simplest way to present the data but leads to a problem. The data is only valid for the MFD at a fixed distance from the observer, using MFD character descriptions and contrast information. It therefore makes no sense to place these contours anywhere but on the surface of the MFD. Unfortunately, there is nothing to stop the user from projecting the contours anywhere in the crewstation, perhaps investigating the legibility of labels on the panel. This just isn't correct and further it not even possible to predict how incorrect it maybe. The user should never try to use these contours on anything besides the MFD! It is my contention that the tool should not allow this data to be placed where it is not valid but at this time there are no restrictions.

5.1.1.2 View Cones

View cones were selected as a natural way to show volumes with a given angular size. The first approach involved creating a psurf formatted viewcone for a specific environment that was the desired size in both steradians and length. They were made with a specific color according to which eye they would be projected from. They were also made semi-transparent so that the user could see the intersections of the cones and the surface of the object of interest. It was thought that a library of these cones could be made and the user would just have to select the desired one.

There were two problems with this approach. The first was that the drawing routine for the retina window had to "know" that the view cones were special psurf objects (they are on the object link list) and not to draw them in the retina window. This required having a special check every time it was to draw an object in the retina window. What was worse was that every time a new cone was added, the draw routine had to be modified to look out for that cone. The second problem was in the number of different cones that would be needed in a real application. There are an infinite number of different size cones that might be needed given the specifics of a task domain. Even with a large library of cones it was likely that the user would not find the one he really wanted.

As a solution to the problem pseudo-objects were introduced in Jack. These are dynamic objects that are not "seen" (they are not on the object link list) by the normal Jack window draw routines. They are not psurf, but are vertices that are computed, in real time, when a new object (cone in this case) is created or moved. Since the Jack draw routines don't know about these objects it is not necessary to explicitly check for them in retina window draw routines. Further, it now possible at run time to set the angular size of the cones. The length of the cone is set to the fixation distance with the apex origin at the nodal point of the eye. It is dynamically sized and positioned as the fixation point is varied. This is consistent with the way in which the other vision features work and also allows for arbitrary sized cones.

These cones could be projected from anywhere not just the figure's eyes. At this point the that is the only choice implemented, but it maybe desired to project them from a define site such as the design eye or sensor location. This would change the implementation of how cones are done now because a fixation point associated with the cones could not be assumed.

5.1.1.3 Color

Throughout the vision model the color red is associated with the right eye and the color green with the left eye. Where they both mix (binocular) the color yellow is used. It is important to maintain color consistency through the model but there have been some undesirable effects caused by trying to maintain this approach. First, the technique of using alpha blending to achieve transparency and color blending allows the model to support these features in real-time. In general it produces good results, but the view angle can be placed in such a way that the correct intersection of the left and right view cones is not presented correctly. Specifically, alpha blending occurs anytime two transparent polygons are drawn that over each other on the screen but intersection real only occurs if the depth of the polytopes overlap also. Since the distance between the cones is never very large relative to their size the error is never very large. In addition it is only erroneous at certain viewing angles and therefore is not a big issue at this time. However, the user should be aware of this inaccuracy.

A second design issue that involves color occurs when retina objects are presented on the retina window as solid polygons. If completely solid they would obscure the data underneath them. For this reason alpha blending is used on this display to allow distinction between data. If too many polygons, greater than 6, are overlaid you can no longer distinguish all of the data. The blending of so many overlapping polygons saturates the intensity values for the display. You can still go back to the line mode and see all of the data contours.

5.1.1.4 Mapping Inaccuracies

The retina window and the Aitoff charts both plot the vertices of the data correctly but then draw straight lines to connect these vertices. In Jack's environment this is correct but on polar maps straight lines don't always map to straight lines. This results in an inaccuracy that is proportional to the length and orientation of the line. The lines could be interpolated, something like how circles are represented, but this would push response time for all but the simplest environments out of the interactive range. It seems like a good tradeoff could be made by running with the mapping function the way it is now but when accuracy is important the more time consuming but accurate mapping function could be used. This function has not been implemented at this time. The new 4D VGX machine can now do texture mapping in real-time and would be the most accurate and fastest way do address this problem.

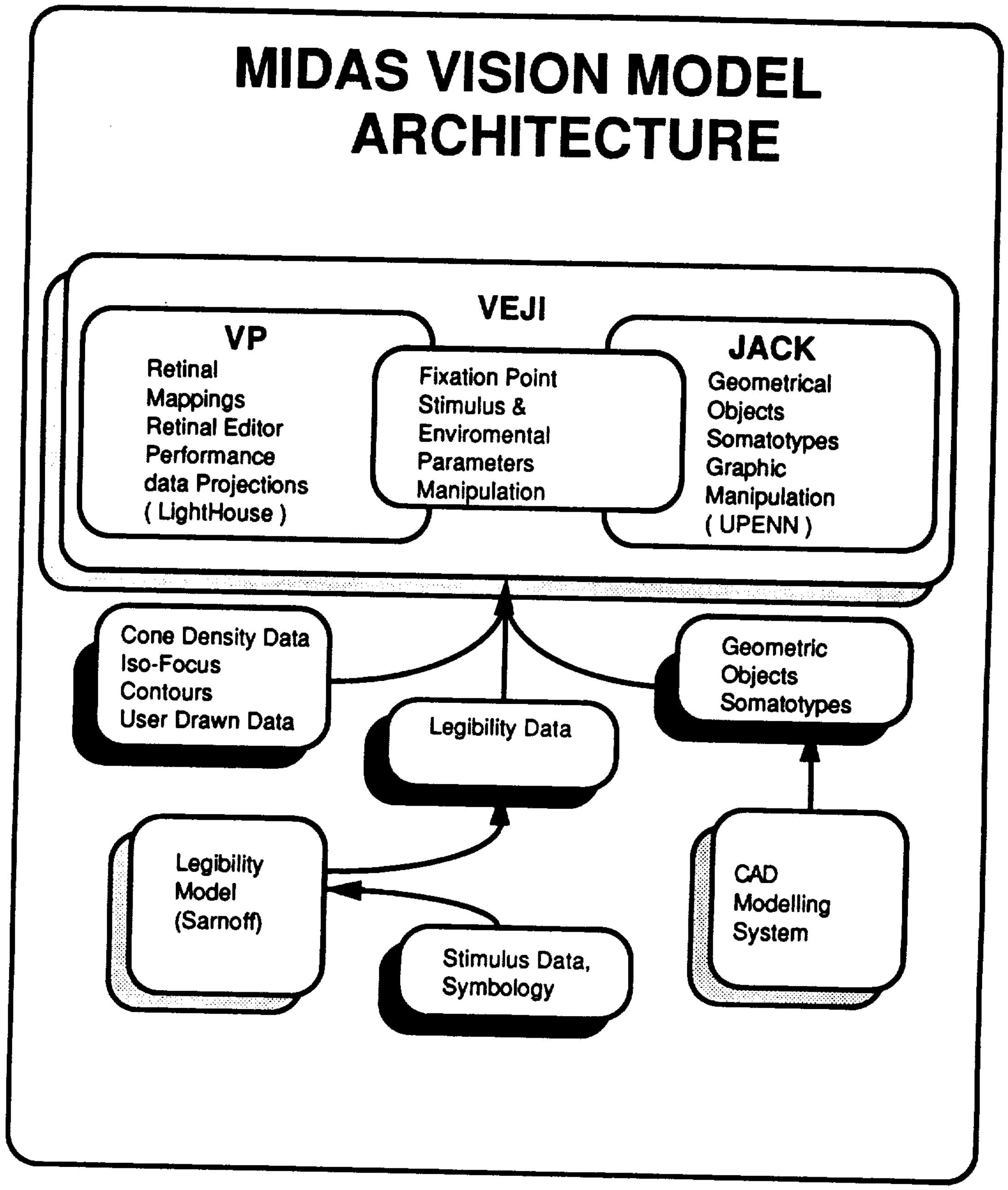


Figure 2. MIDAS Vision Model Architecture

5.1.2 Architectural Design Description

Figure 2 shows a high level diagram of the vision model's architecture. First note that the boxes with solid dark drop shadows represents data, boxes with light drop shadows represent processes that generate or use the data. Starting with the Legibility model, note that it takes as input, stimulus data and a pixel description of the MFD characters or symbology and produces legibility data. There are no direct links with the rest of the vision model. This is the most portable module of the vision model. The rest of the vision model then makes use of the geometrical data produce by a CAD system, in this case MultiGen®, to display an environment. VP is integrated to Jack via VEJI. VP makes use of the data sets by reading them into the retina editor. The data is displayed in the editor window and if desired then projected back out into environment.

5.1.3 External Interface Design

VEJI is an easy to use, mouse driven, multi-windowed program. The anthropometric model in Jack was extended to include eye coordinates and fixation point. The normal jack interface (Badler 1989) was extended to include the ability to manipulate vision relevant parameters. The vision options are included in the normal Jack menu space, and are reached via the options portion of the menu. See section 6.1.4.2.1 of the user guide documentation for a complete explanation of the extended menu space. Under the "options" menu pullout, the vision relevant selections are: Retina Display, Retina Editor, and Vision Information. The designer can interactively set the fixation point, via the mouse, anywhere in 3D space as long as it is within the normal eye joint limits. The fixation point can also be set to predefined points (Jack sites). As the fixation point is varied the designer can watch the vision retina plots and the field of view cones reflect those changes in real time. The user can set model parameters, such as ambient light, font size, etc., via slider bars.

The user must be able to relate dynamic vision characteristics to the objects within the design environment and make design decisions based on those relationships. Data visualization techniques are employed to illuminate these relations and are discussed further in the following sections.

5.2 DETAILED DESIGN

5.2.1 Detailed Design Approach and Tradeoffs

5.2.1.1 Volume Perimetry Considerations

Of primary concern to the designer is that portion of the environment that falls within the aviator's visual field. Volume perimetry (a generalization of visual fields) can be viewed as a way of studying specific geometric relationships between objects in the world and objects on the retinas. Several key concepts enhance this goal.

5.2.1.1.1 Three Dimensionality of Retinal Images

While each of the retinal images is considered to be two-dimensional with, for example, horizontal (x) and vertical (y) coordinates, the composite of the two retinal images is viewed as three dimensional, with an additional coordinate denoting the horizontal difference or retinal disparity between the image position of a world object point in the two eyes, or equivalently, the amount of rotation of one or both eyes required to bring the images of a world object point into registration . This, of course, is one way the visual system extracts depth coordinates in biological vision.

5.2.1.1.2 Concurrent Retinal Images

Retinal information and information about the visual world that is imaged on the retinas should be viewed concurrently. In this way, the impact of transformations in one domain can be appreciated in the other. Motion of world objects, for example, will result in retinal image motion, but often in non-intuitive directions and speeds---viewing a yoked display of visual space and retinal space side by side can be a powerful aid to the understanding of such motion. Similarly, changes in global and local retinal sensitivity over time can drastically alter the visibility of objects in space. The ability to view a rendition of visual space that indicates visibility of objects would be of obvious value.

5.2.1.1.3 Retinal Projections

Retinal images are projections of visual world objects onto the retinas. A kind of converse operation that we call here retrojection is the construction of the locus of possible points in the visual world which may give rise, through projection, to a specified point on the retina(s). Projection and retrojection are the operations which relate visual world geometry to retinal geometry.

5.2.1.1.4 Volume Visual Field

The volume visual field (VVF) is the locus of points in the visual world which fall on sensitive retina. Since retinal sensitivity varies over space and time, and since the size and shape of the VVF changes with eye position, it should be viewed as a dynamic construct.

5.2.1.2 Legibility Model Considerations

Sarnoff's task in the A^{3}I project is to develop a quantitative, easily computable model of instrument visibility within an aircraft cockpit environment. The purpose of the model is to provide visibility data to display designers, who need quantitative information on the effect that their design choices will have on crew performance over a broad range of mission scenarios. Given this range, the visibility of alphanumeric and other information depends not only on the spatial configuration of the display, but also importantly on lighting parameters such as light level in the cockpit and the observer's state of adaptation. The model must therefore accurately assess the effect of such parameters, and convey this information to the display designer in an easily understandable form.

Legibility Model Overview

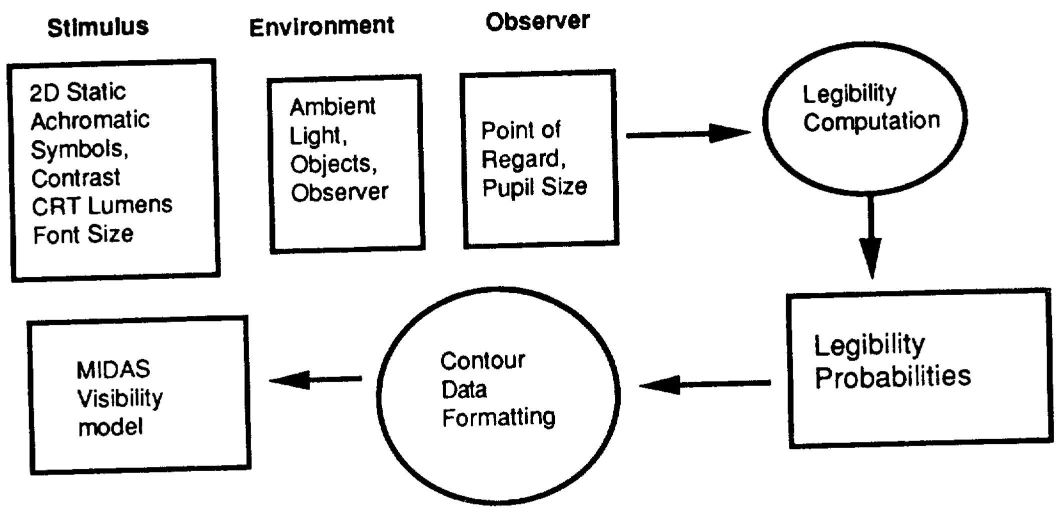

Stimulus
Environment
Observer
2D Static Achromatic Symbols, Contrast CRT Lumens Font Size
Ambient Light, Objects, Observer
Point of Regard, Pupil Size
Legibility Computation
Legibility Probabilities
Contour Data Formatting
MIDAS Visibility model

Figure 3. Legibility Model Overview

A conceptual diagram of the legibility model is shown in Figure 3. The inputs to the model are a set of stimulus, environmental, and observer variables. The output of the model are a parametric data set that spans the parameter ranges of interest. A more detail description will be given in section 5.2.2.2 under detailed design of the Legibility model. For a more complete description of the legibility model see appendix B.

5.2.1.2.1 Basic strategy

Sarnoff's approach to the NASA legibility modeling task has been to augment the one-dimensional discrimination models of Carlson, Barten, and the basic psychophysics community to include the two-dimensional spatial processing used in the Watson detection model. In addition, because the NASA task requires performance prediction over a wide range of lighting conditions and observer perceptual states, the Sarnoff model includes a front end that pre-processes the input images to model the effects of changes in illuminance, screen luminance, and observer fixation location in three-dimensional space.

5.2.1.2.2 Choice of performance measure

The performance measure chosen for the Sarnoff model is probability of a correct discrimination between two input images; e.g., between two alpha-numeric characters. Other performance measures are of course possible; one common measure from the applied psychophysics community is image quality, expressed in units of just-noticeable differences (JNDs) between the two images. In fact, the probability measure used in the Sarnoff model is derived from a JND-like measure, as will be described below. Another potentially useful measure is the reaction time required for correct discrimination.

However, this measure has received little attention to date in psychophysics research, and its use here would thus require a large amount of additional data collection and modeling. Another important consideration in the formulation of the Sarnoff model is that the performance measure be conservative. That is, display design engineers need a measure that would allow them to rule out very bad designs, but would let marginal designs pass for additional testing. In other words, it is better to err on the side of overestimating rather than underestimating the probability of successful discrimination.

5.2.1.2.3 Incremental improvement

Finally, the model has been constructed in such a way as to allow incremental improvement in its ability to predict performance among complex stimuli. For example, the model as currently formulated can accurately predict performance only among static, monochromatic images. However, by replacing the model's spatial filters with a set of spatio-temporal filters, performance measurement among moving stimuli would be possible. The strategy in model development is therefore to validate the simple model on simple stimuli, and then incrementally build in model complexity to handle the more complex stimuli.

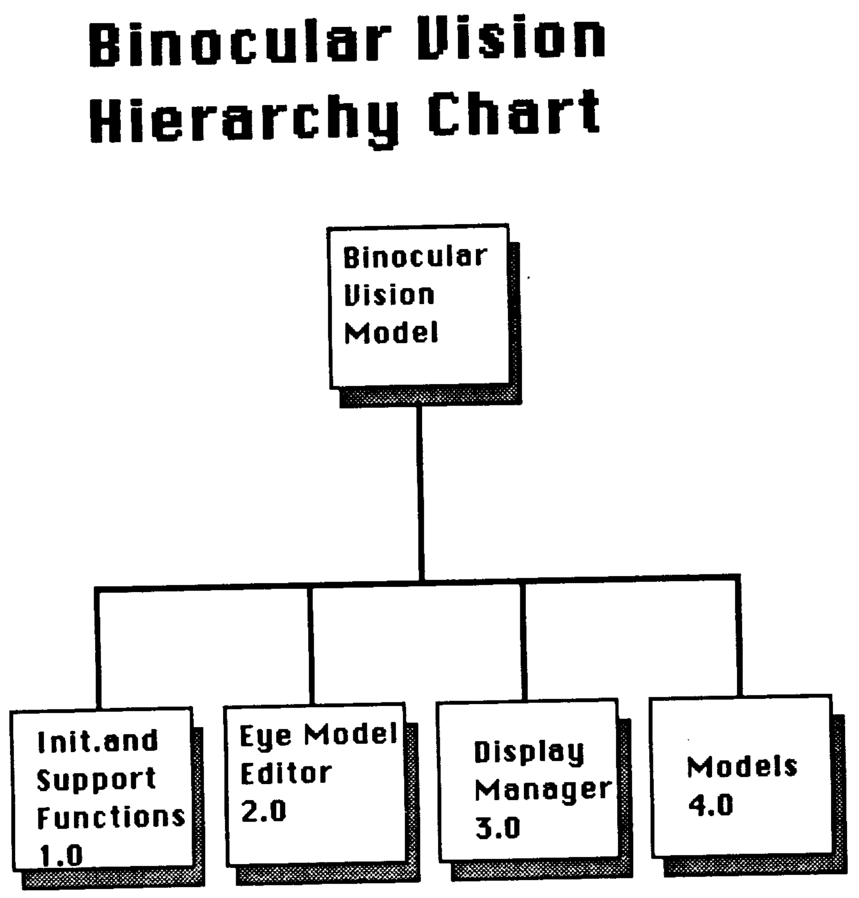


Figure 4. Binocular Vision Hierarchy Chart

5.2.2 Detailed Design Description

5.2.2.1 Compilation Unit

The software implementation of the vision model can be thought of as being comprised of four main groups (see figure 4):

1. The initialization and support functions; which is the group of functions that perform general house keeping activities.

2. The Eye model editor; the group of functions that allow editing the definition of eye and environmental parameters.

3. The Display Manager; the group of functions that maintain the various window displays.

4. The Models; the group of functions that model stimuli, the environment, and the observer.

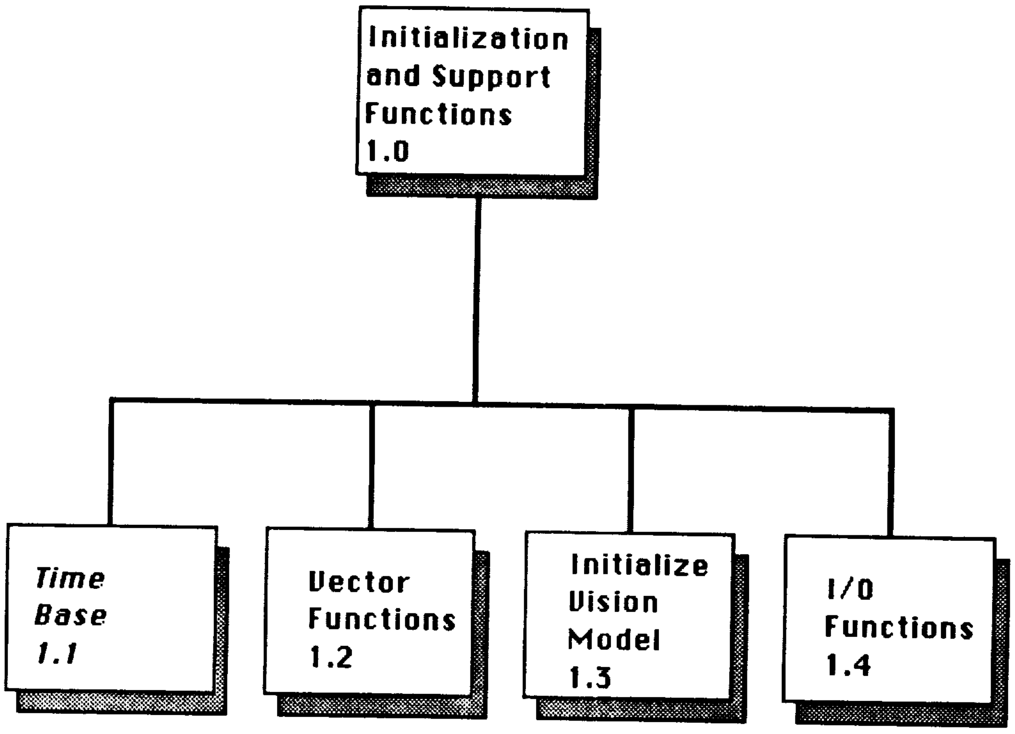

Initialization and Support Functions 1.0
Time Base 1.1
Uector Functions 1.2
Initialize Uision Model 1.3
I/O Functions 1.4

Figure 5. Initialization and Support Functions

5.2.2.1.1 Initialization and Support Functions.

As shown in Figure 5, the initialization and support module is comprised of submodules used to initialize the vision system and by modules to perform mundane activities such as user input and vector arithmetic.

5.2.2.1.1.1 Time Base (not implemented)

This module provides a tick-based approach to time simulation. Many vision responses are time variant and require the maintenances of a time function. These functions are not implemented at this time.

5.2.2.1.1.2 Vector Functions

This module provides some basic vector arithmetic functions. There is a common need to define vector and perform arithmetic on them. This module defines the data structure for a vector and provides the functions such as dot and cross products, vector adds and subtract, etc. These function can be found in the file *vector.c*

5.2.2.1.1.3 Initialize Vision Model

This module provides the basic functions to read and initialize the environmental database. Also a configuration file must be read to set the environment to a user defined state. These function can be found in the file *setstate.c*

5.2.2.1.1.4 I/O Functions

This module provides the functions necessary to input and output information to the vision model. These functions can be found in the files in the *jack directory for Jack I/O and in the file retina_light for data files for the retina editor.*

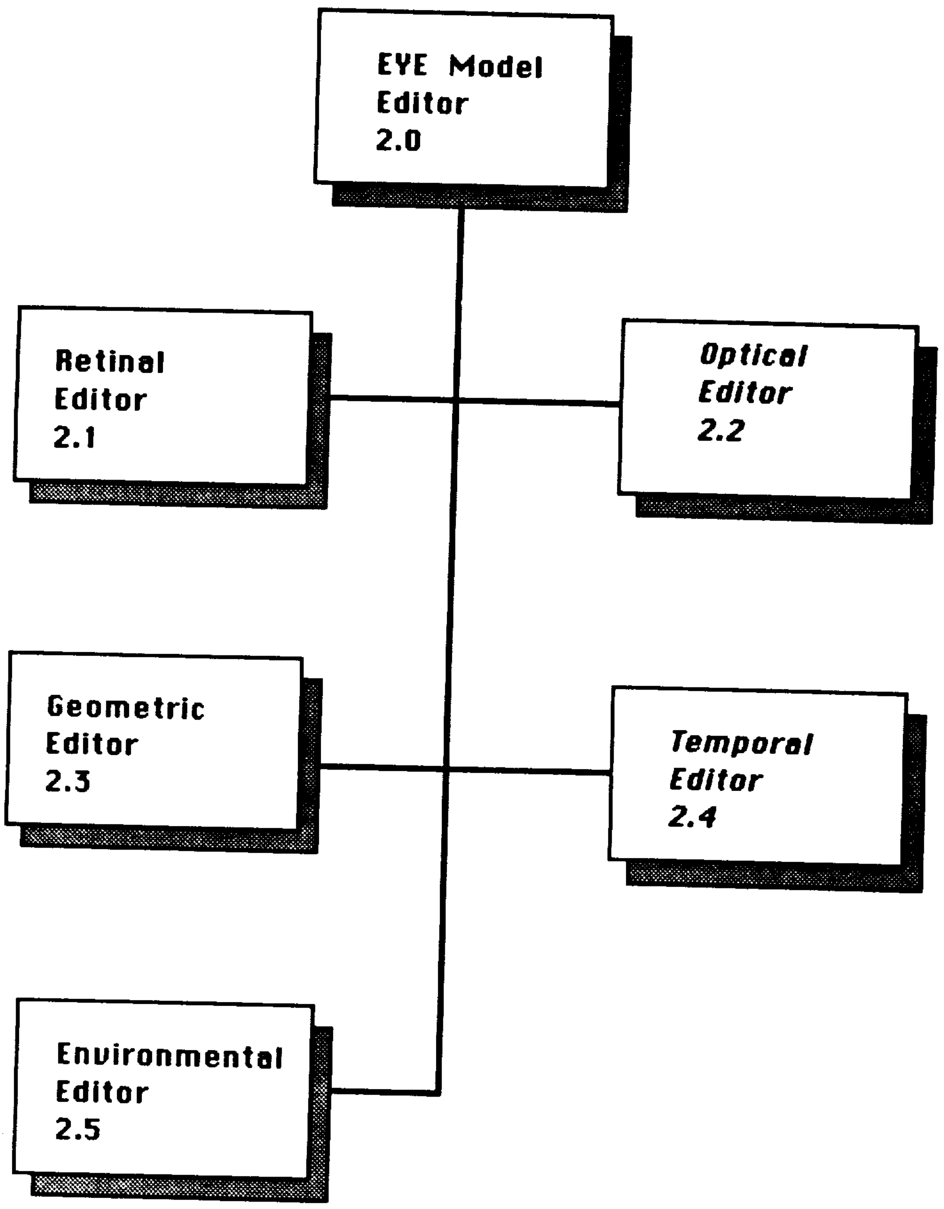


Figure 6. Eye Model Editor

5.2.2.1.2 Eye Model Editor

The philosophy of the vision model design is to allow the user to modify any portion of the vision model. It is the nature of design to consider "what if" type questions and the vision model must be sufficiently flexible to allow a large variety of questions. Some of the editing could be performed by a designer. Other parts of the editing would be used by human factor engineers to tune the model to their application. A set of 5 editors will be provided to assist the user in entering data. See Figure 6 for an overview.

5.2.2.1.2.1 Retinal Model Editor

The retinal model is intended to provide the user with an easy method to describe spatial relationships between the retinal map and areas typically found on the retina. These function can be found in the file *retina_light.c* The software has not progress to the point where the distinction between the objects effects how they are projected. All data is entered pretty much the same, either manually drawn in the editor or read from a data file.

5.2.2.1.2.2 Perimeter Objects

This class of objects define the periphery for the left and right orbs.

5.2.2.1.2.3 Default Template

This object defines the default (generic) retinal objects. Objects such as fovea, macula are predefined for the X percentile person.

5.2.2.1.2.4 Pathological Objects

This class of objects defines areas on the retina that are dysfunctional. The user will enter coordinates via a mouse or data file that describes the geometry of these objects.

5.2.2.1.2.5 Static After Image

This class of objects defines a static area on the retina that will be associated with a intense light source. A more robust model is defined in the temporal eye model.

5.2.2.1.2.6 Concentric Objects

This class of objects define the contours that share a common attribute. The user creates the objects by the same method described for Pathological object.

5.2.2.1.2.7 Performance Objects

This class is the same as Concentric of object except that the geometry is not concentric.

5.2.2.1.2.8 Optics Model Editor (Not Implemented)

This editor has the facilities to store and modify optical attributes, such as myopia, of the eye model.

5.2.2.1.2.9 Temporal Model Editor (Not Implemented)

This editor has the facilities to store and modify temporal attributes, such as photoreceptor light sensitivity, of the eye model. This model would be necessary to track the adaptive state of a pilot over the course of the simulation.

5.2.2.1.2.10 Geometric Model Editor (Not Implemented)

This editor has the facilities to store and modify geometric attributes, such as nodal points location, of the eye model.

5.2.2.1.2.11 Environmental Factors Editor (Not Implemented)

This editor has the facilities to store and modify environmental factor that effect visual performance, such as fog attenuation, of the eye model.

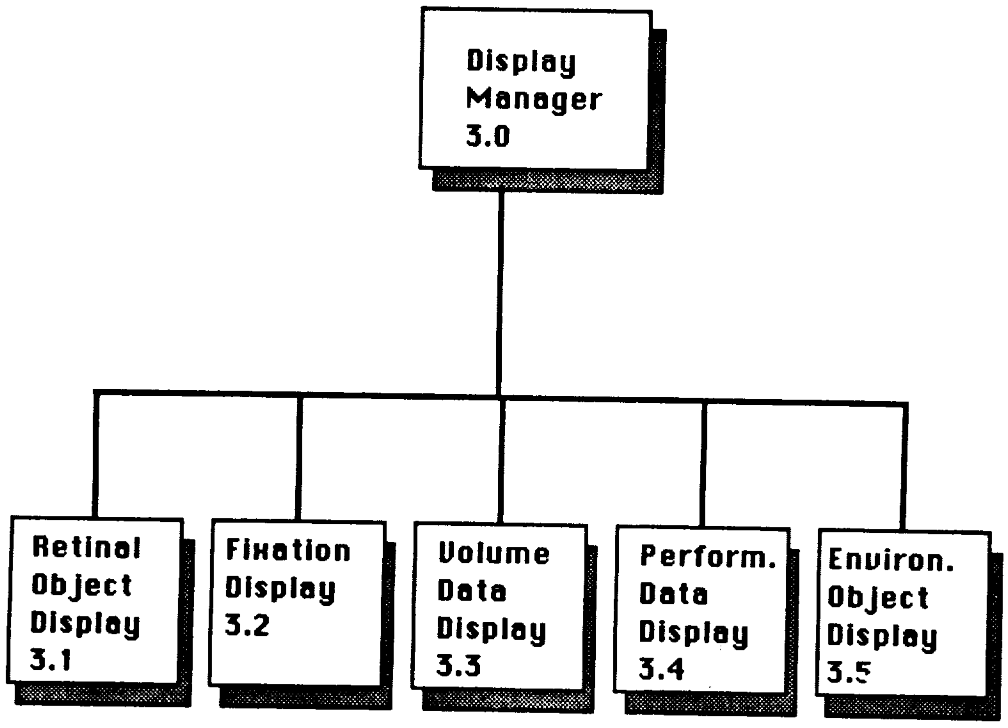

Display Manager 3.0
Retinal Object Display 3.1
Fixation Display 3.2
Volume Data Display 3.3
Perform. Data Display 3.4
Environ. Object Display 3.5

Figure 7. Display Manager

5.2.2.1.3 Display Manager

The Display Manager coordinates the presentation of data with the in place windowing system. System dependent windowing calls are used by this module to display the requested data. See Figure 7 for an overview of t his module. If this software is ported to other systems not running the same windowing system, this module will require special attention. Most display functions can be found in *retina.c and window_light.c*

5.2.2.1.3.1 Retinal Model Display

The Retinal Model Display is a graphical representation of retinal features drawn in fovea centric polar coordinates. The retinal map along with the retinal objects defined in the retinal editor, are in a window of its own. These objects can then be projected out into the environment. In addition, environmental objects can be projected onto the retinal map to provide retinal position information to the user. In appendix A the illustration entitled Fovea Centric Projection shows an example of this type of display.

5.2.2.1.3.2 Fixation Display

The fixation display overlays an environment display with two lines that represent the visual axes. The intersection of the lines is the current fixation point. These functions can be found in *retina.c and vp_src/retina_light..*

5.2.2.1.3.3 Volumes Display

The Volumes Displays can be an independent window or and overlay on an existing environment window. Some visual performance data is best visualized as volume data. The area coverage of the field of view is one such data set. In appendix A the figure titled Jack Environment Window shows an example of FOV information overlaid on the environment.

5.2.2.1.3.4 Function Display (not implemented)

This display is used for visualizing the behavior of the functions used in the simulations given a particular adaptive state. There are a large number of competing vision theories. The vision model is being designed to facilitate the application of new functions characterizing vision systems. It will be informative to plot the behavior of many parametric situations without having to simulate them.

5.2.2.1.3.5 Environmental Display

The Environmental Display is used to display a graphical view into the environmental database. One or more windows may be opened into the database. A six degree of freedom viewing angle may be selected. Objects may be defined and repositioned in these windows. These objects are one way to provide the stimuli to the vision model. The environmental display is also where the visual performance data is projected. This allows registration between performance data and the object that they were based on. In Appendix A the figure titled Projection of Legibility Contours shows an example of an environmental display with FOV (cone projecting from the eyes) information overlaid.

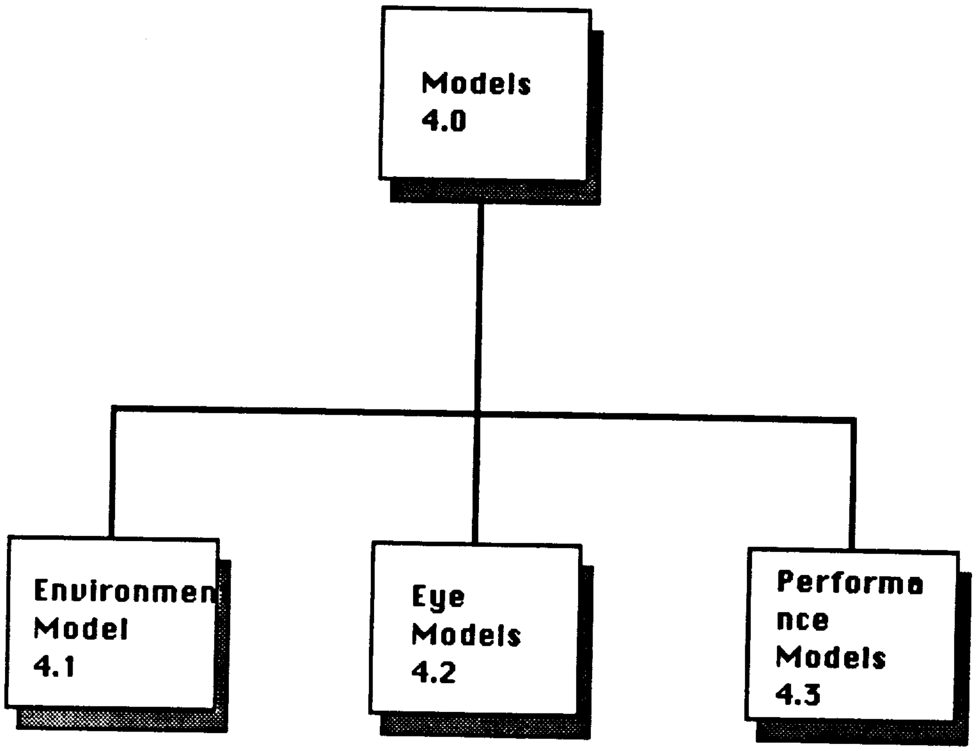
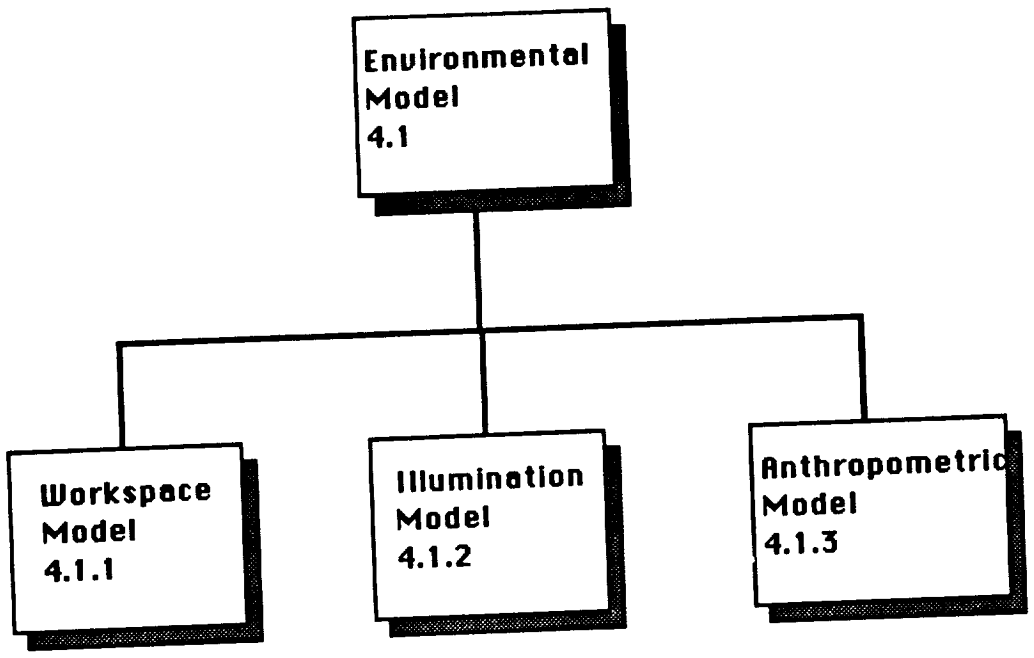


Figure 8. Models

Figure 8 shows the further break down of the models group. The degree to which the models are implemented varies widely. As the simulation requirements are defined for the next phase the priorities of particular models will become known and particular models can be targeted for expansion.



4.1.1.1 Objects
Orientation
Material Description
Geometry
etc.
4.1.1.2 Environmental Factors
Precipitaon
Clouds
Fog, Mist, Smog
Darkness
4.1.1.3 Texture Maps

4.1.2.1 Light
Direction Vector
Energy Spectrum
Intensity
Diffuse, Specular, Emissive
Attenuation
Position
4.1.2.2 Sun
4.1.2.3 Flares, Lasers
Direction Vector
Energy Spectrum
Position
Duration & Start Time
Intensity Curve.
4.1.2.4 Shadows
Umbras
Penumbras
Geometric Attenuation

4.1.3.1 Human Figure
Geometry
State
Age
Fatigue
Cognitive Loads
4.1.3.2 Kinematics
Joint Space and Limits
Waypoints
4.1.3.3 Head
Orientation and Position
Scan History
Tracking Function

mp 10-89 I
4.1

Figure 9. Environmental Model

5.2.2.1.4 Models

5.2.2.1.4.1 Environment Model

The environment model encompasses both the graphical and non-graphical information about the environment to supply the necessary data to the simulation functions characterizing the vision model. Figure 9 shows some of the major subcategories that could be addressed. Only the anthropometric model is generally complete.

5.2.2.1.4.1.1 Workspace Model (Not Implemented see future directions)

5.2.2.1.4.1.2 Illumination Model (Not Implemented see future directions)

5.2.2.1.4.1.3 Anthropometric Model

An anthropometric model is only needed for vision issues if self occlusion issues are of interest. Otherwise a simpler eye coordinate model could be used. If an anthropomorphic model is used then the user/simulation can manipulate this model and the data used as input to the eye model.

Dr. Norman Badler from the University of Pennsylvania provides such capabilities in his program called Jack. Some of the more important features are:

Human Figure Geometry
Adaptive State of the Pilot (not implemented)
Head Specific Attributes
Orientation and Position
Scan History (not implemented)
Tracking Function
Kinematics
Joint Space and Limits
Paths
Inverse Kinematics
Etc.

See Jack User Guide for a detailed description.

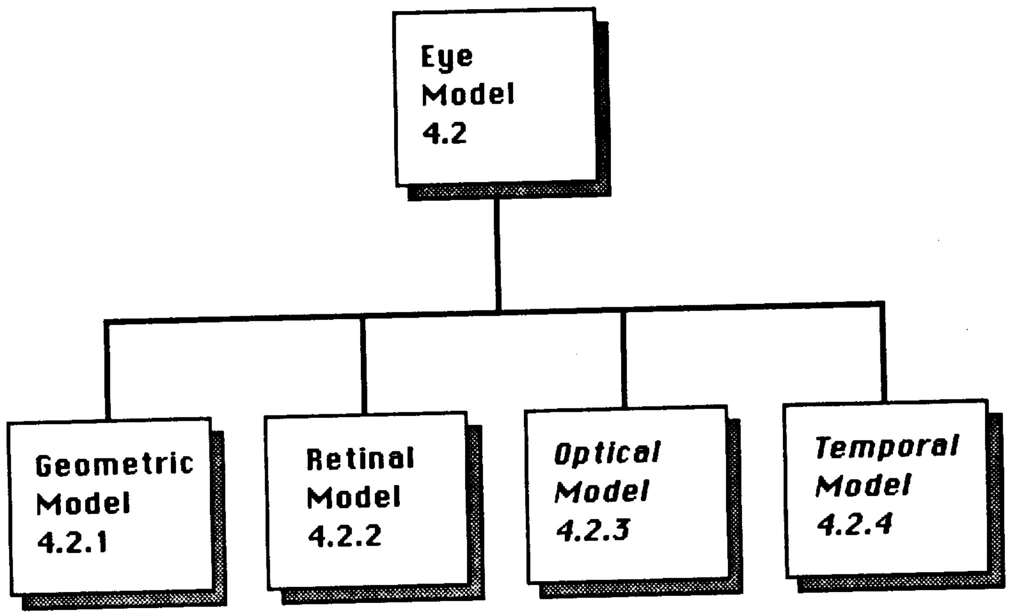


Figure 10. Eye Model

5.2.2.1.4.2 Eye Model

The eye model has four submodels (see figure 10). The submodels are a natural way to group the data structures that describe static features such as eye separation or photo receptor mosaic, with the functions for computing the current value for time dependent data such as pupil size based on those data structures.

5.2.2.1.4.2.1 Geometric Model

This model describes the eye in terms of its geometric properties and current position and orientation. This information is currently spread out over the entire system. In a future release it may be accessed in one place so that specification would be easier and more explicit. Some examples of geometric data are as follows:

Geometric Attributes
Orientation and Position
Size, Focus, Major and Minor Axis
Interpupillary Distance
Pupil Diameter
Motion Limits
Perimetry Geometry
Limiting Facial Structures
Goggles, etc.

5.2.2.1.4.2.2 Retinal Model

The user is required to design a retinal model. He can choose for simplicity the default model which is simply a retina with a fovea and biological blind spot. If the user wishes to investigate a unique retinal model he can enter retina specific data such as scotomas or unusual distribution functions. Some examples of this kind of data is:

Distribution Functions for Cones and Rods
Macular region
Receptive Fields
Spectral Sensitivity
Rod Sensitivity Function
Cone Sensitivity Function
Binocular/Monocular Map
Retinal Mapping Functions
Color Vision

5.2.2.1.4.2.3 Optical Model (not implemented)

This model describes the optical properties of the left and right eyes. The default data is supplied by the "standard eye" (see appendix), but again the user can enter his own. Most values by default are set so as not be evaluated eg. not contribute to the computation if so desired. Mostly this Model is not complete. Here are a few of the features you would expect to find in this model:

Accommodation
Index of Refractions
Optical Axis
Fixation Point
Diopter Range
Convergence Angle
Refractive Errors
Aberrations
Attenuation such as: Vitreous Fluids, Cornea, Lens
Scatter and Glare due to Vitreous Fluids, Cornea, Lens
Veiling Glare and Discomfort Glare

5.2.2.1.4.2.4 Temporal Model (not implemented)

This model would include functions such as:

Sensitivity to light
Light Adaptation
Temporal Sensitivity
Spatial Sensitivity
Eye Movements

JACK DIRECTORY

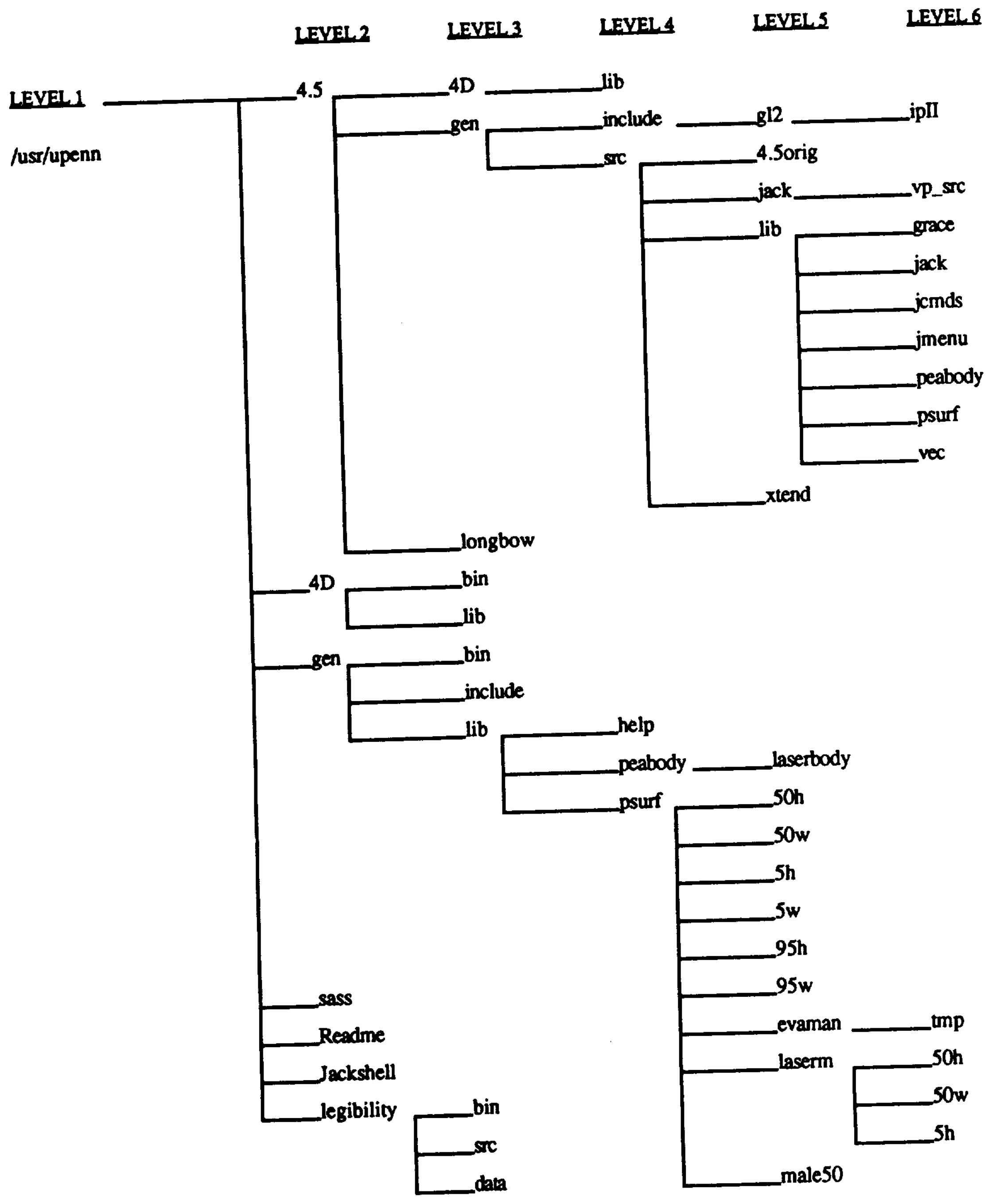


Figure 11. Jack Directory Structure

5.2.2.1.5 Directory structure

Figure 11 shows the directory structure of Jack as distributed with the vision system. See the Jack User documentation for further details.

The following data files are distributed with the vision model. These files are generated by fcons when run on the legibility model data files. They are in a format that can be read by the retina editor and should reside on the directory you start up VEJI from.

QO_i1_l1_d1.cons	QO_i4_l2.54_d1.cons	qo_i3_l1_d1.cons
QO_i2_l1_d1.cons	QO_i4_l2_d1.cons	qo_i4_l1.5_d1.cons
QO_i3_l1_d1.cons	qo_i0_l1_d1.cons	qo_i4_l1_d1.cons
QO_i4_l1.5_d1.cons	qo_i1_l1_d1.cons	qo_i4_l2.54_d1.cons
QO_i4_l1_d1.cons	qo_i2_l1_d1.cons	qo_i4_l2_d1.cons

The following makefile is used to remake a new version of cortex the name of the legibility model executable. Use the command **make** in the directory with the source code. This in most cases will be in UPENN/legibility/src.

Makefile:

```
----------------------------------------------------------
CFLAGS= -g -float
FFLAGS= -g
LDFLAGS=
LIB= -lm
#SGI machines
LIB= -lm -lmalloc -lgl
DISPLIB=
INCLUDE= imgdec.h imgmacro.h

EXECUTABLES=   cortex

IMGLIBSOURCES=  imgio.c imgalloc.c imgconv.c arralloc.c misc.c  fimgimg.c reflect.c
redexp.c fimgopmike.c  filtsub.c dispimg.o kernel.c


IMGLIBOBJ= $(IMGLIBSOURCES:.c=.o)

all:        $(EXECUTABLES) imglib.a

imglib.a:     $(IMGLIBOBJ)
            ar rv $@ $(IMGLIBOBJ)
            ranlib $@

$(EXECUTABLES): $$@.o imglib.a
            $(CC) -o $@ $@.o imglib.a $(CFLAGS)  $(LIB)

$(IMGLIBOBJ):  $(INCLUDE)
-----------------------------------------------------------------------------------------
```

The following is a listing of all the files that make up cortex (legibility model executable). This in most cases they will be in UPENN/legibility/src.

arralloc.c	fcons.c	imgalloc.c	kernel.c
btest.c	filtsub.c	imgconv.c	malloc.c
corfish.c	fimgimg.c	imgdec.h	misc.c
cortex.c	fimgop.c	imgio.c	redexp.c
dispimg.c	fimgopmike.c	imgmacro.h	reflect.c

The following makefile is used to rebuild a new version of jack. The executable will be named djack.

Makefile:

```
--------------------------------------------------------------------------
NAME = djack
VERSION = 4.6
FULLNAME = $(NAME)-$(VERSION)
BINDIR = ../../../4D/bin
EXE = $(FULLNAME)

OBJ =  menu.o retina_menu.o retina.o options.o vision_info_menu.o\
 editor_menu.o peawin.o ret_editor.o setstate.o viewcones.o body.o
SRC = $(OBJ:.o=.c) Makefile
LIBS = -L$(LIBDIR) -ljmenu -lgrace -ljcmds -ljack -lpea -lpsurf\
 -lalt -lgio -lvec -lrle -leditor -lgl_s -lsun -lbsd -lmalloc -lm -lc_s

all: $(EXE)

$(EXE) : $(OBJ) Makefile
	$(CC) -o $(EXE) $(LDFLAGS) $(CFLAGS) $(OBJ) $(LIBS)

$(OBJ) : $(INCLUDEDIR)/jack.h

include $(INCLUDEDIR)/make.h
--------------------------------------------------------------------------------
```

The following files are modified Jack files or new additions to Jack. They are the files used in the above makefile to build the new executable. The new files are printed in bold letters. If any Jack source file was modified the modification was delineated by comments with the key words A^{3}I in them. The modified files were removed from their normal Jack directories and placed here.

body.c	options.c	**retina_mod.c**
editor_menu.c	peawin.c	**setstate.c**
ret_editor.c	**viewcones.c**	**info_menu.c**
retina.c	**vision_info_menu.c**	
menu.c	retina_menu.c	

The following makefile builds a new VP library that is used to link with Jack. Issue the command **make** in the directory $UPENN/4.5/gen/src/jack/vp_src and a new library will be made.

Makefile:

```
-------------------------------------------------------------------------------
SHELL=/bin/sh
LIB= ../../../../4D/lib/libeditor.a
ARFLAGS = rvs

OBJ = \
      vector.o \
      mouse.o\
      list_light.o\
      draw.o\
      retina_light.o\
      init.o \
      eye.o\
      message.o \
      view.o \
      window_light.o

SRC = $(OBJ:.o=.c) Makefile

all: $(LIB)

$(LIB): $(OBJ)
      ar $(ARFLAGS) $(LIB) $?

ranlib: $(LIB)
      ar ts $(LIB)
```

These files make up the VP library.

```
Makefile       draw.c        eye.c
eye.h          init.c        list_light.c
 memory.c      message.c     mouse.c
retina_light.c test.obj      vec.h
vector.c       window_light.c
-------------------------------------------------------------------------------
```

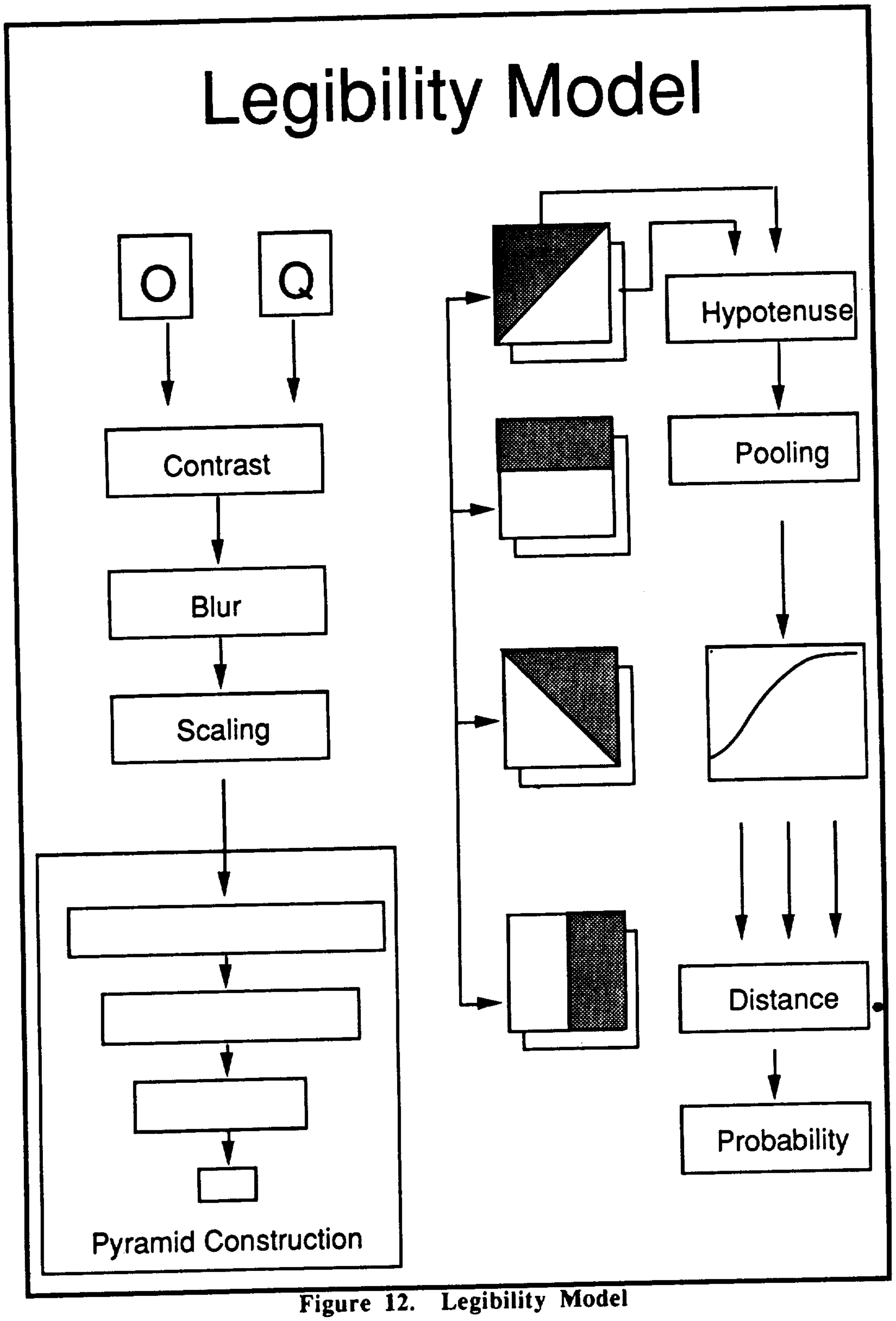


Figure 12. Legibility Model

5.2.2.2 Detailed Design of Compilation Units

5.2.2.2.1 The Legibility Model

A more detailed overview of the legibility that will facilitate understanding the following discussion is provided above in Figure 12.

5.2.2.2.1.1 Input parameters and stimulus format

The model, as currently formulated, takes as input one or two image files, and a number of optional lighting and observer state parameters. These parameters are listed here with the default value and units in parentheses:

Screen luminance (10.0 foot-lamberts)
Illuminance (0.0 foot-candles)
Eccentricity of displayed stimulus (0 degrees)
Fixation depth (741.12 mm)
Stimulus depth (741.12 mm)

The image files start with two four-byte integers indicating the width and height of the image in pixels, followed by rows of pixel values in floats. The images can be any size, although should be at least 256x256 to allow filtering within a large enough range of different frequency bands. The conversion factor from pixels to mm is currently fixed in the software as 13.21 pix/mm. With only one image as input, the software calculates the probability of detecting that stimulus; with two images, the standard discrimination probability is calculated.

In the current version of the model, pixel values are required to range from -1.0 to 1.0, with the maximum absolute value indicating the contrast of the stimulus. For example, a sine grating stimulus with peaks at 0.5 would have a contrast of 0.5. To remove the need for this convention, a model stage which computes contrast from arbitrarily scaled input images is needed, but has not yet been implemented.

5.2.2.2.1.2 Front end calculations

Several initial transformations on the input images are performed prior to the linear filtering stage of the model. These transformations model the effect of changes in fixation depth, veiling luminance, and fixation eccentricity.

5.2.2.2.1.3 Fixation depth

In order to account for changes in effective image resolution with changes in the difference between image depth and fixation depth, we used geometrical optics to calculate the size of the blur circle, and then pre-filtered each input image with this disk-shaped convolution kernel. This calculation requires knowledge of the distance from the exit pupil to the imaging surface (i.e., the retina), which we took as 20.3 mm from Westheimer (1986). It also requires an estimate of pupil size. For this, we wrote a simple interpolation routine to estimate pupil diameter at any light level from a table published in Hood and Finkelstein (1986).

5.2.2.2.1.4 Contrast reduction

Veiling luminance, caused by the reflection of ambient light by the display screen, reduces the effective contrast of displayed information. We are modeling the screen face as a perfectly lambertian surface with a reflectivity of 10\%. This assumption implies that an illuminance of 10 fcd will result in a veiling luminance of 1 fL. We are defining contrast as:

$$c = (\text{lmax} - \text{lmin})/(\text{lmax} + \text{lmin})$$

where lmax and lmin are the maximum and minimum displayed luminances. Given this definition, the addition of a veiling luminance v to both lmax and lmin changes the contrast by a factor $1/2v$.

5.2.2.2.1.5 Eccentricity scaling

Abundant psychophysical evidence shows that contrast sensitivity remains roughly constant across the visual field, if the grating patch is scaled up by a linear function of eccentricity. This strongly suggests that processing is similar across the visual field, except for a linear scaling up of sensor size towards the periphery. In order to model the effect of this change in sensor size, we found it more convenient to scale down the size of the input images as a function of eccentricity, rather than to scale up the size of the sensors. We used the scale factor k of 0.4 quoted by Watson (1983), so that the scaling of input images as a function of eccentricity e is

$$e = 1.0/(1.0 + ke).$$

5.2.2.2.1.6 Linear filtering

5.2.2.2.1.6.1 Pyramid decomposition

In order to filter within a range of different frequency channels, the input image is first decomposed with a gaussian pyramid into channels separated from each other by one octave, The frequencies we chose for these channels are identical to those used by Watson (1983); i.e., 32 through 0.5 c/d, corresponding to seven octaves or equivalently, seven pyramid levels.

5.2.2.2.1.6.2 Computing filter gains

The human visual system is not equally sensitive at all frequencies. A plot of contrast detection threshold as a function of spatial frequency shows roughly an inverted-U shape, with a peak at roughly 2 c/d, and complete loss of sensitivity by approximately 60 c/d. Moreover, as shown by van Nes and Bouman (1967), the shape of this contrast sensitivity function changes with retinal illuminance. To model these dependencies, the image component in each frequency channel is weighted by a gain factor appropriate for the retinal illuminance, before any oriented filtering is performed.

Retinal illuminance (in photopic trolands) is calculated as the amount of light incident on the cornea (in cd/m^2) times the pupil area (in mm^2), where the light incident on the cornea is assumed to be the screen luminance plus the veiling luminance, appropriately converted from fL to cd/m^2. The gain at each frequency is then calculated directly from the van Nes and Bouman data with a simple log interpolation function to return sensitivities at retinal illuminances other than those reported in the data. For example, if the threshold modulation for a 1 c/d grating were 1% at 10 tds and 5% at 1 tds, then we interpolate the threshold at 3.16 tds (half the log distance from 1 to 10) to be 2.23% (half the log distance from 1% to 5%). This direct calculation is possible only under the assumption of no summation among different frequency channels; any assumed summation would require a more complicated relationship between the contrast sensitivity function and the gain of each channel.

5.2.2.2.1.6.3 Steerable filtering

For convenience and speed of oriented filter operation, we use the steerable filters of Freeman and Adelson (1990), which allow separable calculation of linear filter responses at any orientation and phase. The filters implemented here, a second derivative of a gaussian and its Hilbert transform, have a log bandwidth at half height of approximately 0.7 octaves. This is within the range of bandwidths inferred psychophysically (e.g., Watson and Robson, 1981). We are using four orientations (0, 45, 90, and 135 degrees) and two phases (sine and cosine), for a total of eight oriented filter responses per pyramid level. The orientation bandwidth of these filters (i.e., the range of angles over which the filter output is greater than one half the maximum) is approximately 65 degrees. This figure is slightly larger than the 40 degree tuning of monkey simple cells reported by Devalois et al (1982), and the 30 to 60 degree range reported psychophysically by Phillips and Wilson (1984).

5.2.2.2.1.6.4 Energy calculation

During the early stages of model testing, we found that the detectability of a simple edge could change dramatically with small changes in edge position. To combat this problem, a small amount of spatial summation was added by computing energy after the linear filtering stage. That is, corresponding sine and cosine filter responses were combined as:

$$e(x_i) = \sin^2(x_i) + \cos^2(x_i)$$

where x_i is a linear filter response, indexed over filter position, orientation, and frequency band.

5.2.2.2.1.6.5 Point non-linearity

Nachmias and Sansbury (1974) showed that the results of a grating contrast discrimination experiment, when plotted with threshold contrast increment as a function of the base contrast from which the increment threshold is being measured, produce a dipper-shaped curve. These authors argued that the results can be modeled by assuming a sigmoid non-linearity following a linear detection mechanism. The decision mechanism has available to it only the output of this non-linearity, and reliably discriminates between inputs of two different contrasts when the difference in outputs is greater than some threshold.

To quantitatively model the dipper-shaped contrast discrimination curve, we follow Legge and Foley (1980) in using a non-linear transducer of the form:

$$T(L_i) = r|L_i|^n \,/\, (L_i^2 + s^2)$$

where T is the non-linear transducer output, L_i is the linear filter response (indexed as above), r is an overall gain-setting parameter, n is a real number greater than 2 (2.4 here), and s is a semi-saturation constant (0.0075).

The contrast detection thresholds used in the legibility model , shown in Figure 13, are from van Nes and Bouman, 1967.

		Frequency (c/d)						
		32	16	8	4	2	1	0.5
Retinal Illuminance (td)	.0009	999.0	999.0	999.0	46.21	19.88	14.11	12.49
	.009	999.0	41.69	33.66	12.34	6.76	5.31	4.88
	.09	75.86	14.81	6.44	3.57	2.24	1.69	1.54
	0.9	28.96	3.23	1.52	0.973	0.692	0.763	1.29
	9.0	8.91	1.33	0.579	0.352	0.350	0.629	1.29
	90	2.63	0.551	0.284	0.225	0.318	0.629	1.29
	900	1.40	0.450	0.255	0.216	0.318	0.629	1.29

Figure 13. Contrast Detection Thresholds

Pupil diameter in mm were attained via table lookup. The values used were for viewing a white lambertian surface at illuminances:

3.426e-6 fcd
3.426e-4 fcd
...
3.426e4 fcd

and the pupil diameter was set to one of the following values: 7.1, 6.6, 5.5, 4.0, 2.4, 2.0. Since this model is only accurate for photopic vision the larger pupil diameters are never used.

The depth of eye in mm, from exit pupil to retina was set at 20.3, an accepted distance. The cortical scaling parameter was set according to (Yap, Levi, and Klein, 1987) at 0.77. The retinal scaling factor was set according to (Watson, 1983) at 2.5.

5.2.2.2.2 Vision Enhanced Jack Interface Functions

5.2.2.2.2.1 The VVF Display

The VVF is a three-dimensional graphical construct. In the VVF display, three-dimensional objects can be created by using the menu or by reading in a file of objects. Newly created objects have a default position in the VVF, but can be moved to any location in the VVF. They can also be connected to build compound objects out of simple objects. Objects are defined as a type, and each has a dynamically allocated database that contains information about the vertices, faces, and location of the object.

As the VVF is a three-dimensional construct, we need to specify projection and viewing information. The VVF is displayed as a perspective projection, the parameters of which are stored in the window's database. The view reference point and center of projection are also stored in the window's database, and can be changed interactively to look at any point in the VVF from any location.

Each time a new VVF window is created, the window is assigned default values for the projection and view, but it inherits the objects that already exist in the VVF. Essentially, opening a new VVF window gives the user the ability to view the VVF in a different way.

5.2.2.2.2 The Retina Display

The retina display illustrates the dynamic relationships of three types of objects with respect to the visual axis. By default, it shows the retinal images that are formed by objects in the VVF. The orientation of these images varies with changes to the fixation point and with orientation and position changes of the head. A subset of retinal images, that form the second object type, are formed from objects that are fixed with respect to the head. Since helmet mounted devices, nose bridge, glasses, etc, are stationary to the head coordinate system, this object type remains stationary in the retina display unless the fixation point is changed. The retina display can also illustrate areas that are characteristic of the retina, and are therefore fixed in location and orientation even when fixation and the head position is varied. The fovea and the natural blind spot are examples of fixed data. (We'll refer to fixed data as retinal objects to distinguish them from the variable retinal images discussed above.) The proper dynamic relationships between the data types are maintained as the user manipulates the fixation point.

The user can open additional retina windows as needed, and the retina displays can be interactively customized to make it easier to see data of interest. For example, each retina window can display either the right or left retina, or the two superimposed on each other. Left retina data is displayed in shades of green, right retina data is displayed in shades of red, and areas of overlap are shown in yellow.

The user can build up a database of retinal objects e.g. cone density data, iso-focus contour, etc. This is the means by which the legibility data is brought into VMT. Retina objects can also be interactively drawn directly onto the right, left or both retinas.

An important difference between retinal images and retinal objects, is that the objects can be retrojected (discussed below) into the VVF. There is no need to retroject retina images because they have the object that generated the image there already.

5.2.2.2.3 Field of View Cones

The designer can set the solid angle for the field of view of interest. Semi-transparent view cones are then projected along the current visual axis for the right and left eyes into the crewstation. The convergence of the cones is at the fixation point. The intersection of these cones and the crewstation delineates the area that falls within the current field of view settings.

5.2.2.2.4 Total Field of View Plots

Of major importance in vehicle design is the area visible out the window. Total field of view plots provide the designer with a 360 degree plot of external visibility. It is possible to read information eg. over-the-nose visibility, directly from these plots. The results can be seen immediately as the designer explores effects of various body types, widows designs, etc.

5.2.2.2.2.5 Retrojections

Retinal objects can be selectively retrojected into the VVF. By retrojecting a retinal object, the user can see where it intersects the VVF. On a computer that supports the rendering of transparent surfaces, the retrojection is drawn as a semi-transparent volume in the VVF. On computers that don't support transparent surfaces, the retrojection volume is outlined with a series of rays (lines) that are drawn from the eyeball(s) out into the VVF.

5.2.3 External Interface Detailed Design

The vision model has not been integrated with the rest of MIDAS simulation. In the next phase it will follow the interfaced standards developed for MIDAS communications.

5.2.4 Coding and Implementation Notes

6.0 USER'S GUIDE

The Vision User Guide has two main parts. The first explains how to use the David Sarnoff Legibility Model (LM) software to produce legibility data for display in Jack The second part explains the use of the vision specific options in the A^3I version of Jack, hereafter referred to as the Vision Enhanced Jack Interface (VEJ I). The vision enhancement was a result of a rewrite of an earlier program called VP for Volume Perimetry from the Lighthouse Research Laboratory. The vision options are not standard and are not normally released with new versions of Jack. They are also not maintained or documented by the University of Pennsylvania. Further releases of Jack may be incompatible with the vision options.

6.1 OVERVIEW OF PURPOSE AND FUNCTION

Geometrical data such as the volume field of view , occlusions, facial geometry and helmet margins can also be projected into the cockpit with respect to the coordinates of the aviator's eyes and fixation point . The intersections of the projections with objects in the crewstation, delineate the area of coverage, masking, or occlusion associated with the objects.

Objects in the crewstation space can be projected onto models of the operator's retinas. These projections can be used to provide the designer with the retinal coordinates and the visual angles subtended by objects in the crewstation space. Both the right and left eye retinal projections are mapped. The retinal map is yoked to the fixation point and changes as the fixation point is interactively manipulated. Performance contours on the retinas can also be indicated thus aiding the designer in understand the limitations to visibility imposed by retinotopic processing.

6.2 INSTALLATION AND INITIALIZATION

To install the vision enhanced version of Jack, follow the normal installation instructions provided with Jack . There are some additional legibility data files that were created with the Sarnoff model that should be copied to the same directory that you execute Jack from. These files have names similar to QO_i1_l1_d1.cons and contain the precomputed legibility predictions contours.

6.3 STARTUP AND TERMINATION

For startup and termination instruction for the vision enhanced version of Jack see the Jack documentation. The LM is called from the Unix prompt and is not dependent on any environmental variables. To start LM type the filename (cortex) followed by the input files (pixel descriptions of the symbols to discriminate between followed by any additional options. It will run to completion without further intervention.

6.4 FUNCTIONS AND THEIR OPERATION

6.4.1 Sarnoff Legibility Model Functions

To run LM execute the file named cortex. This should be in the directory called $UPENN/4.5/legibility/src. You call the program by typing:

```
cortex < input1   -l lum   -i illum -e ecc -d dfix -ds dstim input2
```

Where:

- **input1** is the bitmap image file for the first image
- **lum** is display luminance in foot-lamberts (fL)
- **illum** is ambient illuminance in foot-candles (fcd)
- **ecc** is eccentricity of displayed stimulus, in degrees
- **dfix** is the fixation distance in mm
- **dstim** is stimulus distance in mm
- **input2** is the bitmap image of the second image to be desciminated from the second image. If there isn't a second image than a second copy of the first is used.

In the absence of an explicit input parameter the following defaults are used.

illum=0.0, ecc=0.0, lum=10.0, dfix = 741.12, dstim = 741.12;

a typical call would look like this:

```
cortex < large_Q_file -l 10 -i100
```

The files pyrg.fir and steer.fir must be present in the same directory as cortex. These files are used as date for the filters used in the model.

Here is what the output of cortex means: Without the -p 1 flag, the output is simply eccentricity and probability correct discrimination. By running cortex for a range of eccentricities for each combination of luminance, illuminance, and fixation distance, It generates the kind of data files you can see in the $UPENN/4.5/legibility/data/*.dat files.

When you run cortex with the -p 1 flag, you get a lot of intermediate results that can be used for debugging. On the first line:

- **veil** shows the screen luminance plus the output of veiling illuminance reflected off the screen.
- **ret** shows the retinal illuminance in trolands, calculated as a function of pupil diameter and veil (the veiling luminance).
- **fcon** is the contrast reduction in the image, resulting from the veiling luminance.
- **fscale** shows the inverse of the retinal and cortical magnification factors needed to scale the frequency and size of the sensors in the periphery of

the visual field.

Each subsequent line of output shows the results of the probability calculation for each different level of the pyramid.

Prob is the final probability result
dist is the the distance measure in detector output space, from which the probability is calculated
lev is the pyramid level
ori, x, and **y** index the orientation and position of the detector which showed the maximum sensitivity to the difference between the two input stimuli.
A **-1** on any line indicates that no detector at that pyramid level was at all sensitive to the stimuli.

The files containing the input characters are 256x256. The characters themselves are composed of 5x5 blocks of pixels for each pixel in the longbow specs. So, for example, the size a characters are 55 x 35 pixels. The size convention I have used in the model are that the stimuli are 30" from the observer, and that at this distance, one degree of visual angle corresponds to approximately 170 pixels.

The user can generate contour data (format for VEJI) from these files directly, using a piece of software called fcons. This program is in the same directory as cortex. You should be able to compile it directly (with the -lm flag) and use it. You call it as fcons n < in.dat > out.cons where n is the number of lines in the input data file, in.dat is the input data file, and out.cons is the output file of contours.

You call it as

fcons n < in.dat > out.cons

where n is the number of lines in the input data file, in.dat is the input data file, and out.cons is the output file of contours.

Retina Display

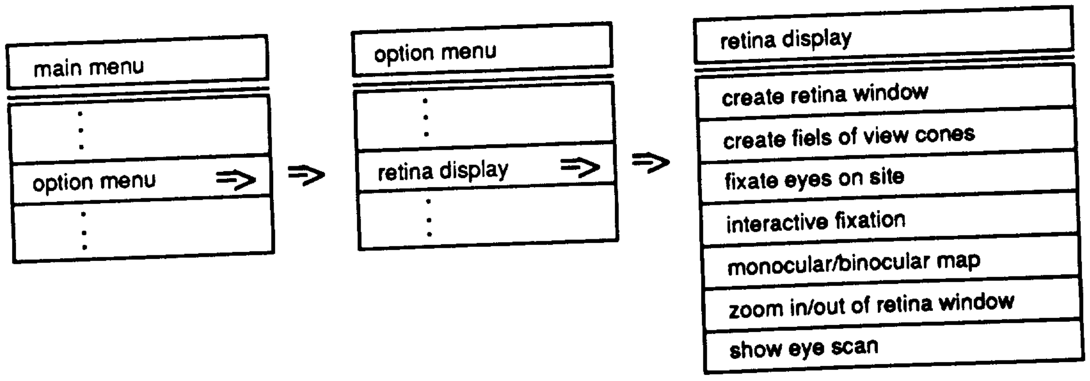


Figure 14. Retina Display Menu

6.4.2 VEJI Options

6.4.2.1 Retina Display

The user's interface is consistent with the rest of the Jack interface. The user accesses the vision model's options by moving through the menus. The user must pull out the option menu under the main popup menu in Jack. This will result in many special options being displayed. The relevant ones to the vision models are: retina display, retina editor and vision information. Referring to Figure 14, the first set of options (retina display) addresses the user's need to manipulate the fixation point and visualize the area of coverage of the visual field as the fixation point is swept around the environment.

6.4.2.1.1 Create Retina Window

To view the environment as seen from the aviator's eyes select the "create retina window" option. The user must then select a figure with eyes. Currently only the polybody figures have eyes. After selecting a figure with eyes, the user must then open a window anywhere on the screen. Try and pick a location and size that will not occlude areas of interest. This window will then maintain a view as seen from the selected figure. The red figures correspond to the right eye and green figures to the left. The mapping of objects is done in polar coordinates with the fovea at the center ie. (0,0). Each of the concentric rings are in ten degree increments, with outer most being 90 degrees. This represents the limits of the periphery. The magenta disk at the center represents the macular region (~ 5 degrees). The red/green disk represents the biological blind spot for the left or right eye.

6.4.2.1.2 Create Field of View Cones

Viewcone projection is an area that is under revision. Therefore there are two approaches under investigation at this time. The first options is "create field of view cones". This option provides the user a mechanism for visualizing the intersection of predescribed psurf formatted viewcones with the environment. The viewcones provided were created using a compatible CAD package or Jack primitives and are psurfs just like other objects in Jack's environment. The user can select from a sixty degree view cone with nose cut, or a long and a short five degree view cone. These view cones are constrained to the the figure's eye and will change as the fixation point for the figure is manipulated. They are semi-transparent and are colored red for right eye, green for left eye, and yellow for the intersection (binocular). This options is intended to provide access to a database of view cone somatotypes.
To use this option, after selecting "create field of view cones", the user must select a figure with eyes that the field of view will be projected from. Then the user must select the type of cone desired. Once these cones are created they can not be deleted. You may project more than one type of view cone if desired.

6.4.2.1.3 Create Monocular Field of View Cones.

Many times a user will want to investigate an arbitrary angle field of view. In this case there may not be a predefined viewcone with the desired angle/size. The create monocular field of view cones differs from the "create field of view cones" option by allowing the user to project an arbitrary angled field of view cone. The angle is set by user in the vision information section (explained later) and is turned on by selecting this option. The view cone is projected out of only the left eye and is semi-transparent green.

6.4.2.1.4 Fixate eyes on site

This options will set the visual axes of a figure to converge on a predefined site. Defining a site is described in the Jack User's manual. Once a site is defined the user can command a figure with eyes to fixate on that site by selecting this option. The user will be asked to select the figure to do the fixation and the site to fixate on. Then by pressing the left mouse button the figure will do the fixation. Fixations are constrained to the joint limits of the figure's eyes and will only move as far as possible if the site is outside the limits.

6.4.2.1.5 Interactive fixation

This option allows the user to interactively sweep a figure's visual axes throughout the environment, thereby intersecting view cones with objects of interest. Any time the fixation point is changed the Retina Display window is updated. This options works much like the fixation on site option, but instead of selecting a site to fixate on the user manipulates the fixation site interactively. To move the fixation point along any single axis press one of the mouse buttons, to move it in a plane select two mouse buttons. The fixation point is moved just as any other object in Jack. See the Jack User Manual for more details.

6.4.2.1.6 Monocular/Binocular Map

This option helps declutter the Retina Display window by toggling through right, left and both eye projections. By selecting this option the first time the user projects in the retina Display only the right eye's image. Selection this option again projects only the left eye's image. Selecting the option again returns the users to projecting both eye images.

6.4.2.1.7 Zoom in/out of retina window

Selecting this option zooms the retina Display in so that only the center 30 degrees are visible. This mode is helpful for seeing detail about object size and retina location. Selecting this option again returns the user to display the full 90 degrees.

6.4.2.1.8 Show eye scan

This option is used to trace the eye's fixation point as it is moved around the environment. As the MIDAS simulation progressed the fixation point would trace out a 3D curve in space associated with the fixation point location. Since there is a bug in the Jack software for tracing objects this function no longer works correctly. It will be fixed in a future release.

Retina Editor

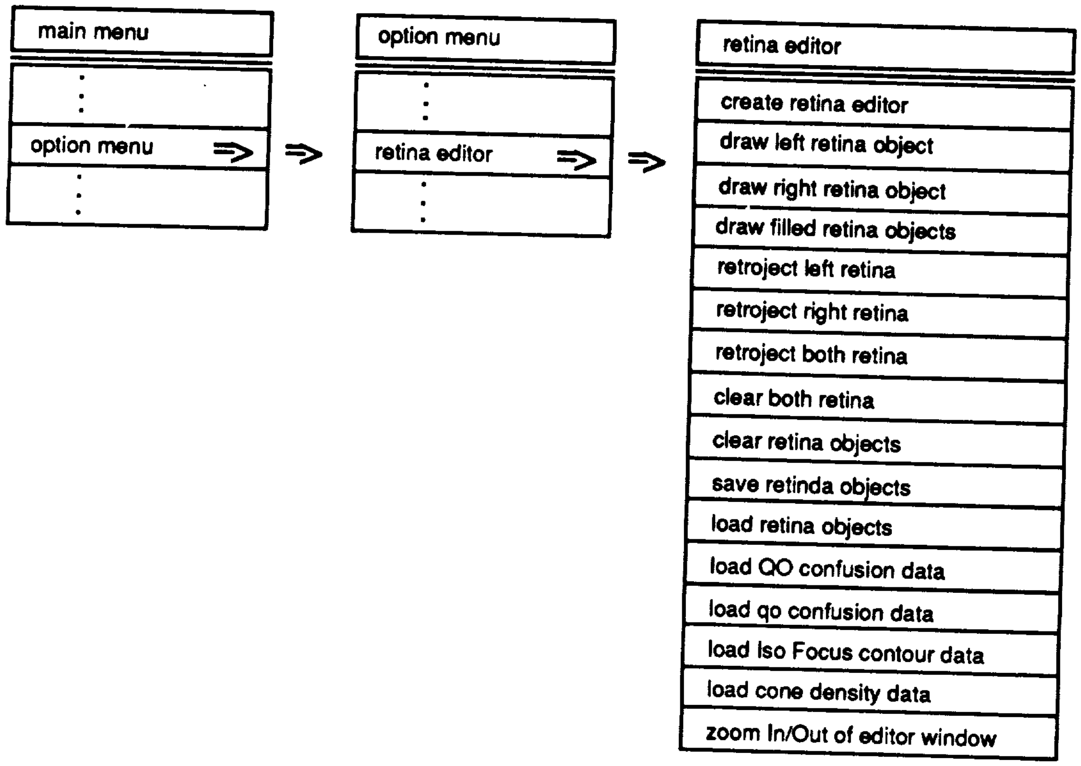


Figure 15. Retina Editor Menu

6.4.2.2 Retina Editor

See Figure 15 for an overview of the functions available under this option.

6.4.2.2.1 Create Retina Editor Window

The retina editor window provides a way to plot visual relevant performance data in a fovea centric polar coordinates map, similar to the retina display. Whatever is drawn or read into this window can also be projected into the environment. To create a retina editor window select this option. The normal SGI window opening and position is then performed. The user should try to position this window some place where it does not occlude useful information. Remember the user can always resize the Jack screen to anything size that is appropriate. The editor window is not an integrated Jack window and only gets updated when an event occurs that would change what is displayed in the window.

6.4.2.2.2 Draw left/right retina object

The draw option is used to draw arbitrary contours onto the retina editor. After selecting either left or right draw options move the mouse cursor over the point on the editor window that you wish to enter as the first point, press the left mouse button. Now move to the next point and press the left mouse button. Continue entering points until you have entered all the data points and then press the right mouse button to close the contour. The contour will be colored red for right, green for left retina object .

6.4.2.2.3 Draw filled retina objects

This option toggles back and forth between drawing the data as lines or polygons. The polygons are semi transparent and interior contours can be distinguished from overlapping polygons.

6.4.2.2.4 Retroject left/right/both retina

This option turns on the retrojection (projection into Jack's environment) of data loaded or drawn onto the retina editor. The projection of this data is along the visual axis of the figure and extends to the plane positioned at the fixation point and normal to the visual axis.

6.4.2.2.5 Clear retina objects

Selection of this option clears all data on the retina editor.

6.4.2.2.6 Save/Load retina objects

Selection of these options allows a user to save the data that has been drawn onto the editor or load in new data.

6.4.2.2.7 Load QO/qo confusion data

These options are a special case of the above load option. They were put on the menu only to make it easier to demonstrate the vision model. Instead of taking as input a filename, this option builds the correct file name for the current conditions. The conditions that effect the file that is load are: ambient illumination, stimulus lumens and character size. Light are set in the vision state information settings. Character size is determined by your selection of large (QO) or the small (qo) letters in the option name. If any of the light settings are changed after the data is loaded you must clear the editor of the old data and reload the confusion data again.

6.4.2.2.8 Load Iso Focus / Cone density contour data

This option is the same as the above options for the loading of confusion data, except that the data does not depend on the ambient illumination and stimulus lumens and therefore doesn't need to be cleared and reloaded when they change.

6.4.2.2.9 Zoom in/out of editor window

This option zooms in/out of the retina editor window. Some of the performance data can cover only a small portion of the retina window when this occurs it is useful to zoom in. Selecting this option again zooms out.

Vision Information

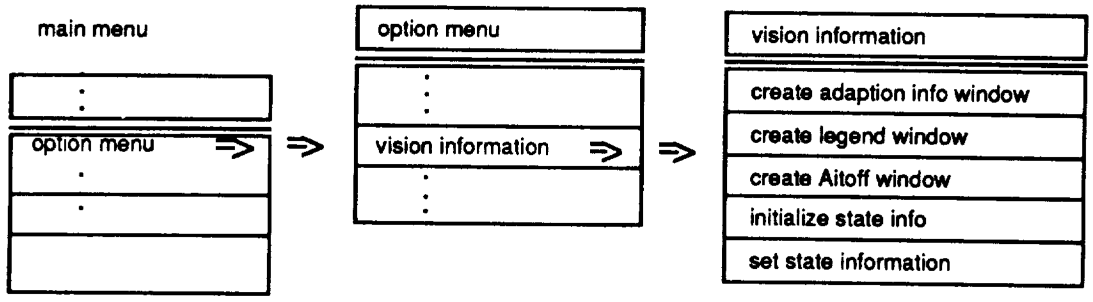


Figure 16. Vision Information Menu

6.4.2.3 Vision Information

See Figure 16 for an overview of the functions available under this option.

6.4.2.3.1 Create adaptation info window

This option reflects work in progress. The purpose for this window is to show the current state of the observer. As factors such as ambient lighting changes the observer state is assumed to be adapted instantaneously. The assumed current state is sometimes of interest for the user. It may also be useful to investigate mis-adaptive states.

6.4.2.3.2 Create legend window

This window will be used to provide a legend for color used in the display of data. Generally speaking green is associated with attributes of the left eye, red with the right eye and yellow with both. The idea is to provide a standard way to communicate the meaning of complex data in a consistent way. This window is not completed at this point.

6.4.2.3.3 Create Aitoff window

Aitoff charts are used widely by industry to standardize visual parameters based on the coordinates of the design eye. To use this option the user must now the coordinate of the design eye and define a site at that location. The user can then get an Aitoff graph relative location. This options plots everything in the environment and therefore produce a very cluttered plot.

6.4.2.3.4 Initialize state information

This option should always be done before attempting to use any of the vision options. It is best to include this in the startup script for the Jack environment.

6.5 ERROR AND WARNING MESSAGES

Error and Warning messages are done in the same way Jack handles them. See Jack for more information.

6.6 RECOVERY STEPS

Recovery Step are done in the same way Jack handles them. See Jack for more information.

7.0 ABBREVIATIONS AND ACRONYMS

MFD	Multi-Function Display
VVF	Volume Visual Field
MIDAS	Machine Integration Design Analysis System
JACK	Not an acronym (just looks like one)
VEJI	Vision Enhanced Jack Interface
VP	Volume Perimetry
LM	Legibility Model

8.0 GLOSSARY

9.0 NOTES

9.1 FUTURE DIRECTIONS

9.1.1 Illumination Model

Illumination modeling is undergoing rapid progress towards more realistic computational models. There are many tradeoffs in the illumination algorithms between accuracy, computation time and noticeable differences. Emphasis can be placed on reaching pleasing or artistic results; thus providing an illusion of reality. The algorithms implicitly consider the accuracy of the display median to limit the amount of computation to that which differences are detectable on most graphic displays. Emphasis could be shifted to model the laws of physics that rigidly govern the propagation of electro-magnetic radiation as accurately as possible. Our vision model will require a high degree of fidelity to the physical world to limit the errors in other dependent computations. Many of the techniques to follow will not execute in anywhere near real-time. They will however, have intermediate results that may be used for displaying approximate graphics.

The accurate presentation of data is confounded by the limited display capabilities of the CRT. A CRT is capable of displaying only a subset of colors and intensities ranges. It is also of finite resolution. What can be presented on a CRT is not the same stimuli that would be experienced if one were actually there. For this reason the graphic image will only be used for rough approximations where a large number of possible solutions are being evaluated. After selection of a specific solution it can be reevaluated at higher fidelity.

The development of an accurate illumination model has a profound impact on the rest of the simulation. The physical response to electromagnetic waves of the surface material of every object in the simulation must be known. The orientation of all objects must be known to calculate the visibility to illumination sources and the viewer. Attenuation factors of the environment in which the viewer and illumination sources are embedded in must be included in the model. Attributes of the illumination sources must also be modeled. This implies that every object (even geometrically simple ones) will need a relatively large amount of illumination data in addition to the normal geometric data.

The theoretical basis for the illumination model will be a hybrid of several complementary techniques. Forming the basis for the modeling of diffuse illumination will be that of a hemispherical radiosity model. This model was first used to study heat transfer between elements in furnaces or on a spacecraft. The radiosity model has been recently extended by Goral at Cornell University to the area of computer graphics. Other researchers, Cohen, Greenberg,and Hall, have refined the model to handle special cases such as high radiosity gradient artifacts found in Goral's original model.

The radiosity model provides a solution for diffuse and emissive radiant energy but does not model specular reflectance and refraction. For this type of illumination a ray tracing technique is more appropriate. This model simulates lights with rays and propagates them throughout the environment. Surfaces reflect or transmit these rays depending on their surface attributes. Proper attention is paid to angle of incidence and refraction. Ray tracing is CPU intensive and therefore will be used for final analysis rather than for interactive evaluation of the design solution space.

Roy Hall's illumination model presented in his book, *Illumination and Color in Computer Generated Imagery*, is the most complete illumination model to date. Glassner (*An introduction to Ray Tracing*) codifies Hall's model and suggests how to extend it to integrate diffuse transmission of light. (see appendix) This is the illumination model that will be used to vary light parameters in our vision model. The error between Hall's model and the Silicon Graphic's light model (see appendix) will be explored. The size of this difference will determine the appropriateness of this model for use in approximating solutions.

Special procedures can be provided to model local phenomena that can be precomputed. For example, the glare due to the sun could be precomputed for various angles and a set of light masks could be overlaid on a CRT to model the expected glare.

The following provide the data structures necessary for the afore-mentioned algorithms to be computed.

General Illuminators
This module contains the data structures and access functions for the illuminator data. These data may be static but could also be set by the output of some other simulation module. Default data for illuminators will be supplied from the IES Lighting Handbook. Some examples are as follows:

Direction Vector
Energy Spectrum
Characteristics
Location
Geometry
Reflector type

Sun, Moon
The sun and moon are special case of illuminators and data will be attained from a TBD source.

Flares, Lasers
Flares and lasers present some special problems in vision and need to be addressed differently in the model. These data will be very time dependent in both the spatial and amplitude domains. The effects of narrow frequency ranges, strong lighting contrast and filters make these illumination sources difficult to model.

Light Loss Factors
This module contains the data structure and functions to model loss of light from some of the major causes. Shadows will comprise a significant portion of the light loss cases. These cases are addressed as part of Hall's illumination model. The others will be modeled from data out of the IES Lighting Handbook.

Shadows types
The major types are shadows are: Umbras, Penumbras, Geometric Attenuation

Non Recoverable Factors
The non recoverable factors are those features of luminaries that cause deviations from the controlled laboratory but are not corrected by light maintenance procedures. The IES Lighting Handbook identifies these factors as important when computing light calculations and states that they are multiplicative. The total light loss is the product of these factors. Temperature Factor, Line Voltage Factor, and Surface Depreciation.

Recoverable Factors
The recoverable factors are like non-recoverable factors with the exception that they can be corrected by proper light maintenance. Some examples are: Lumen Depreciation Factor, Dirt Depreciation Factor, Lamp burn out Factor.

Workspace Model
The Workspace model is comprised of the data description for the objects that exist in the environment. In addition data about the current state of the environment is maintained. Objects are a generic name for an entity in the simulation. An object can be used as a stimuli to the vision model. An object has at least the following slots:

> The orientation and positional data are maintained in world coordinates for each object.
>
> A material definition is provide for each polygon that comprises an object. This material definition is needed for the radiosity model. Material properties include: Emission color, Ambient reflectance, Diffuse reflectance, Specular reflectance, Shininess (specular light scattering exponent) Alpha (transparence) and color.
>
> A geometric description including information such as vertices coordinates is maintained for all objects.
>
> Many objects in the environment can exist in various states. For those objects that posses this characteristic the associated state variables are maintained in the object data structure.

An important factor for target detection is Atmospheric Attenuation. The visibility of a distance object is very much dependent on conditions in the atmosphere. The predominant influencing factors are: Precipitation, fog, clouds, smoke, haze, etc. Attributes and procedural descriptions will be stored here to describe these effects.

Texture Maps are required for added accuracy for many of the detection algorithms. An example of where texture maps may be used would be target masking. Many targets use some form of camouflage to make detection more difficult. It will be important to model target masking to attain accurate results. The problem of detection is more general than intentional masking. The detectability of any object depends to a large extent on the contrast between itself and its background. Texture maps are necessary to generate realistic backgrounds, eg. terrain or sky, from which an object's detectability will calculated.

10.0 APPENDICES

APPENDIX A — FIGURES

BINOCULAR VISION MODEL

Sarnoff Vision Model Input

55 x 35 pixel characters

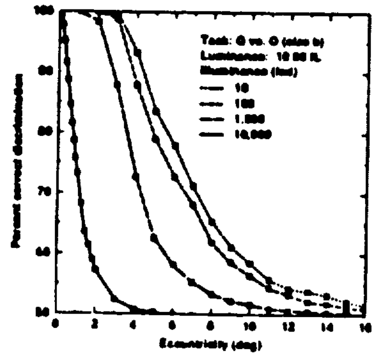
Character size

Ambient light

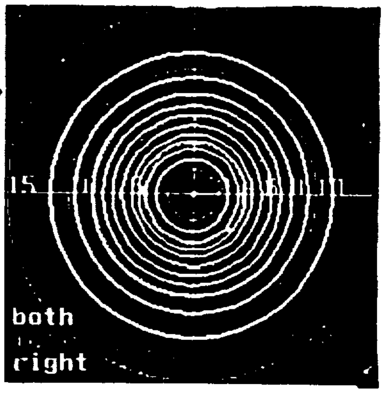
Stimulus luminance

Fixation point and delta

Performance data



Retinal editor



Other data

User input

Published data

Data projection

Vision model environment

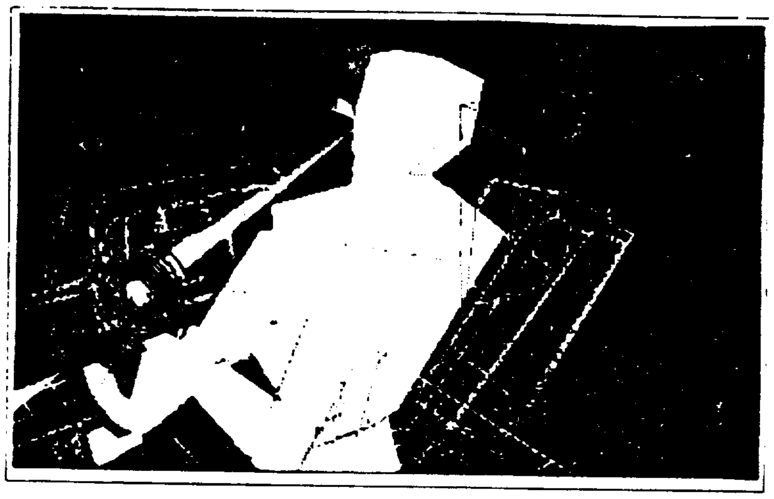
User Interface

Visual attention
- Fixation
- Field of view
- Depth of field

Data parameters

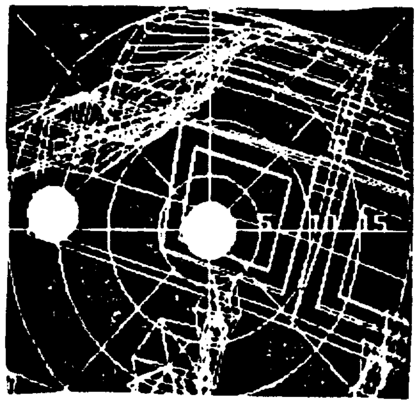
Performance data projections

Object space to retinal space projection



Object space projections

Binocular foveal centric projection



Contributors

Dr. James Larimer
NASA Ames Research Center

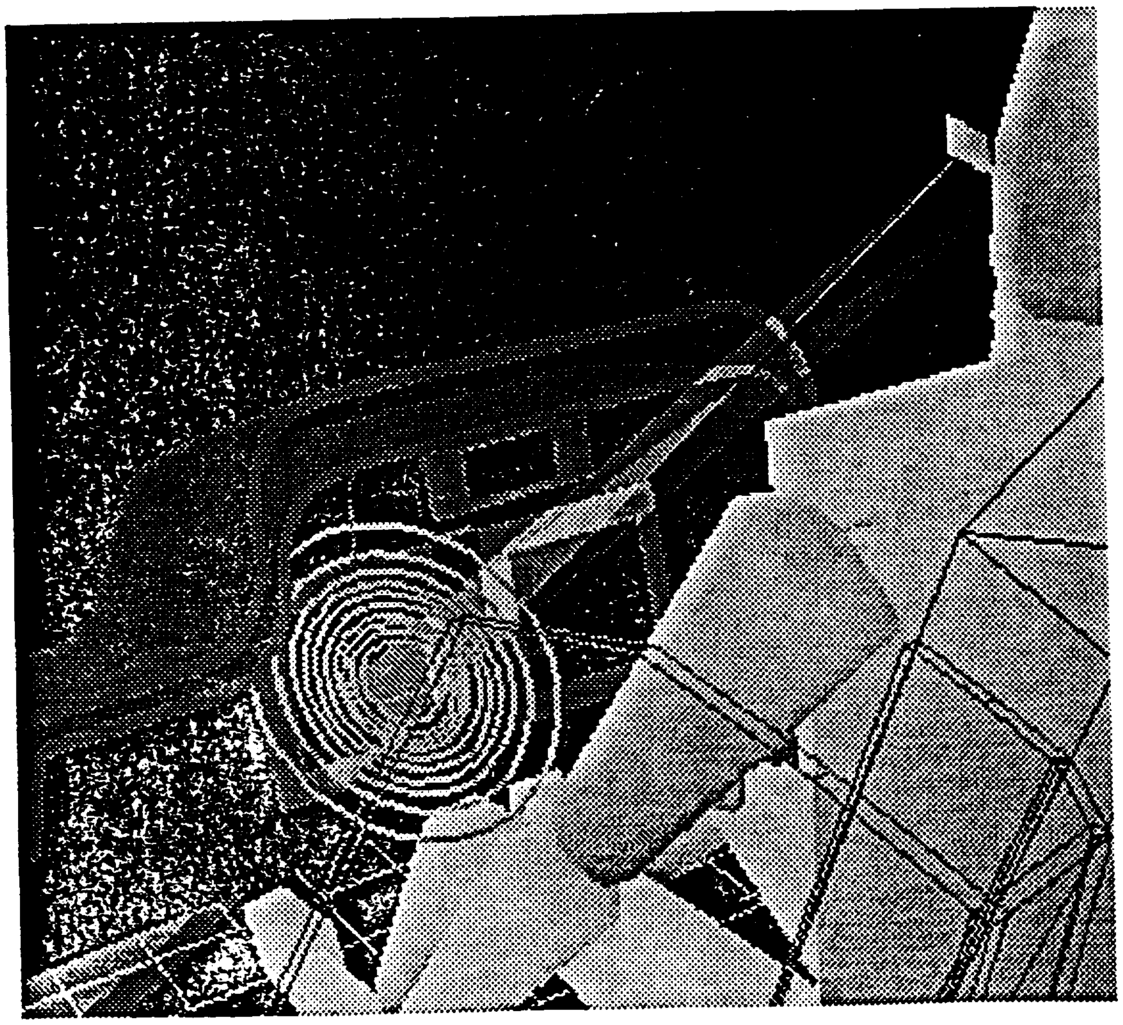
Mike Prevost
Sterling Software

Dr. Aries Arditi, Steve Azueta
Light House N.Y.

Dr. James Bergen, Jeff Lubin
David Sarnoff Labs

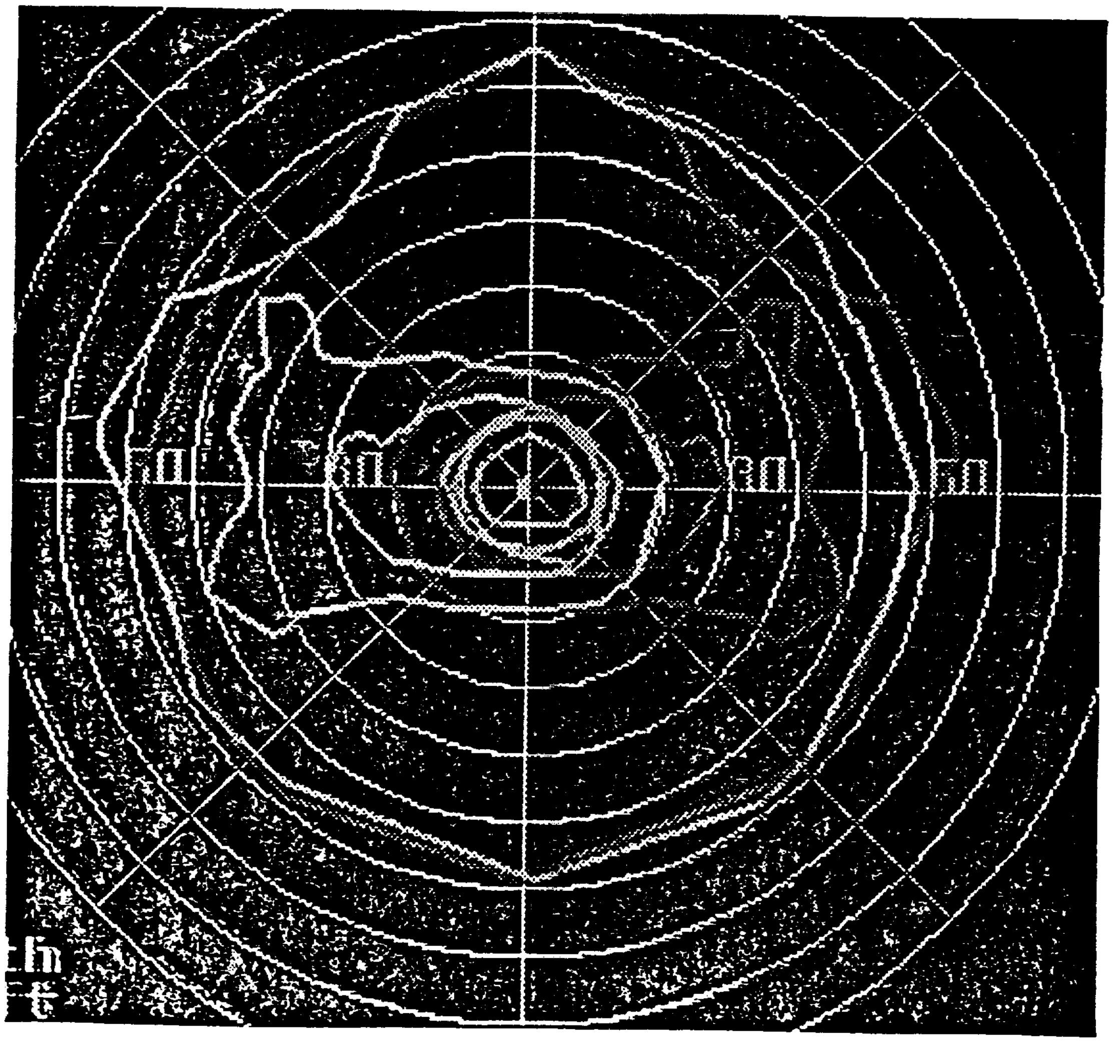
Dr. Norman Badler, Cary Phillips
University of Pennsylvania

Projection of Legibility Contours

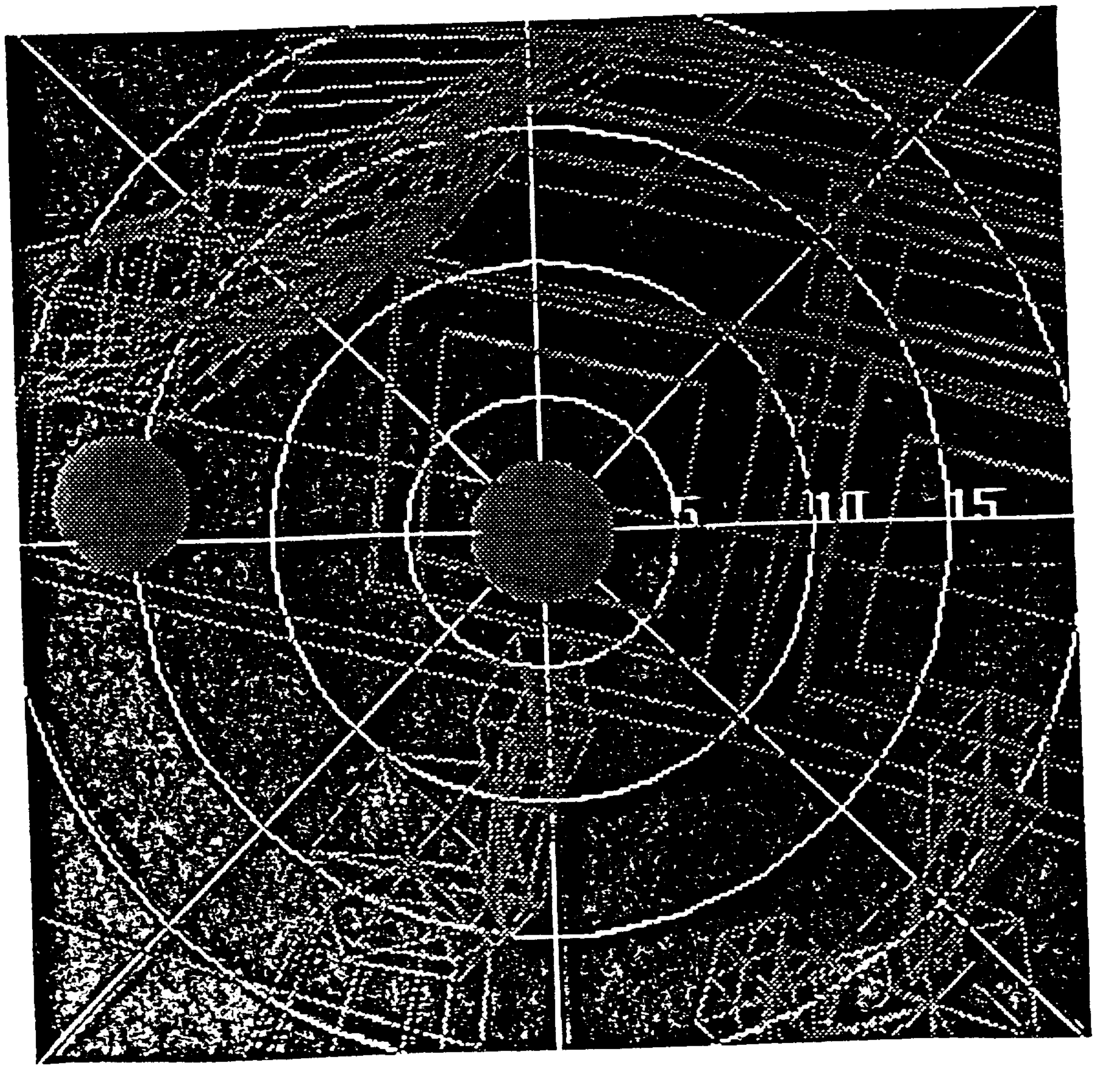


Retinal Editor Window

Cone Density Data



Fovea Centric Projection



APPENDIX B — SARNOFF LEGIBILITY MODEL

Sarnoff Cockpit Display Visbility Modelling for NASA A^3I Project

Jeffrey Lubin James R. Bergen

November 13, 1990

1 The Task

Sarnoff's task in the A^3I project is to develop a quantitative, easily computable model of instrument visibility within an aircraft cockpit environment. The purpose of the model is to provide visibility data to display designers, who need quantitative information on the effect that their design choices will have on crew performance over a broad range of mission scenarios. Given this range, the visibility of alphanumeric and other information depends not only on the spatial configuration of the display, but also importantly on lighting parameters such as light level in the cockpit and the observer's state of adaptation. The model must therefore accurately assess the effect of such parameters, and convey this information to the display designer in an easily understandable form.

2 Background

A number of visibility models have already been developed, by workers in both the applied vision and basic psychophysics communities. Each can successfully predict human performance within a restricted range of stimulus and task domains.

2.1 Applied psychophysics

An early success in applied vision was the JND Model of Carlson and his associates (Carlson and Cohen, 1974; Carlson and Klopfenstein, 1985). In this model, an input image is decomposed via a one dimensional fourier transform into a number of spatial frequency bands. These filtered bands are then perturbed by various noise sources, squared, and spatially integrated. Changes in the output of this process from one member of a pair of images to the other provide a simple perceptual measure of the visibility of differences between the two images. This model has successfully predicted the visibility of changes in edge sharpness and of various display artifacts, among other things. The disadvantages of the model are that it is spatially one dimensional, and is somewhat complicated to compute, since a noise parameter must be adjusted for each change in display parameters such as luminance and display size. A more recent variant of this model, the Barten (1987) SQRI Model, solves some of the complexity problems by introducing polynomial approximations for the changes in human sensitivity to changes in display parameters.

2.2 Basic psychophysics

Similar models have been introduced into the basic psychophysics literature by Wilson and his colleagues (e.g., Wilson, McFarlane, and Phillips, 1983; Wilson and Regan, 1984), based on the threshold model of Wilson and Bergen (1979), and by Legge and Foley (Legge and Foley, 1980; Foley and Legge, 1981). These models successfully predict human performance in simple psychophysical tasks such as grating contrast detection and discrimination. In all of these models, the input image is first decomposed into independent spatial frequency channels by a set of linear filters. The output of each filter is then put through a sigmoid non-linearity, the shape of which matches very closely that of the non-linearity imposed by the noise and squaring steps of the JND Model.

2.3 Two-dimensional models

All the models described above are spatially one-dimensional; that is, they predict sensitivity to spatial variation in one dimension only. Watson and his colleagues (Watson, 1983; Ahumada and Watson, 1985; Nielsen, Watson, and Ahumada, 1985) have implemented a model which generalizes the linear filtering stage of these models to two dimensions. Each filter is a two-dimensional gabor function, with a number of different scales, orientations, and phases of filtering at each point in the two-dimensional visual field, and an increase in the overall scale of filtering as a function of eccentricity. The model has been validated on some detection and discrimination data. One limitation is that it is only accurate at stimulus levels near detection threshold since, unlike the other models described above, there is no point non-linearity after the linear filtering stage.

2.4 Combining information across channels

One problem all these models must face is how to combine information across a large number of different filtering channels, so that a uni-dimensional value for human performance on a discrimination task can be obtained. In other words, from the large dimensional space represented by the channel outputs, the models must derive something like a single value for the probability of detecting a difference between a pair of stimuli.

One way to perform this reduction of dimensionality is to base the performance measure only on the single channel which shows the maximum change in output from one member of the stimulus pair to the other. This "maximum-of" decision rule is implicit in most of the basic psychophysics modelling, and is motivated by the hope that the simple stimuli usually used in that paradigm are sufficiently localized in the space of channel outputs so that only one channel governs performance, regardless of the degree to which channel outputs are combined.

Other models, like the Watson model, use variants of an optimal Bayesian classifier at the channel combination stage. Given the assumption that each channel output is perturbed by zero mean, unit variance Gaussian noise, the detectability of a pattern by an optimal observer is directly proportional to the euclidean distance of the pattern's feature vector from the origin of the channel output space. For an uncertain observer, detectability can be modeled in the same way, but with an exponent higher than the euclidean 2.

3 Sarnoff Approach

3.1 Basic strategy

Sarnoff's approach to the NASA legibility modeling task has been to augment the one-dimensional discrimination models of Carlson, Barten, and the basic psychophysics community to include the two-dimensional spatial processing used in the Watson detection model. In addition, because the NASA task requires performance prediction over a wide range of lighting conditions and observer perceptual states, the Sarnoff model includes a front end that preprocesses the input images to model the effects of changes in illuminance, screen luminance, and observer fixation location in three-dimensional space.

3.2 Choice of performance measure

The performance measure chosen for the Sarnoff model is probability of a correct discrimination between two input images; e.g., between two alphanumeric characters. Other performance measures are of course possible; one common measure from the applied psychophysics community is image quality, expressed in units of just-noticeable differences (JNDs) between the two images. In fact, the probability measure used in the Sarnoff model is derived from a JND-like measure, as will be described below. Another potentially useful measure is the reaction time required for correct discrimination. However, this measure has received little attention to date in psychophysics research, and its use here would thus require a large amount of additional data collection and modelling.

Another important consideration in the formulation of the Sarnoff model is that the performance measure be conservative. That is, display design engineers need a measure that would allow them to rule out very bad designs, but would let marginal designs pass for additional testing. In other words, it is better to err on the side of overestimating rather than underestimating the probability of successful discrimination.

3.3 Incremental improvement

Finally, the model has been constructed in such a way as to allow incremental improvement in its ability to predict performance among complex stimuli.

For example, the model as currently formulated can accurately predict performance only among static, monochromatic images. However, by replacing the model's spatial filters with a set of spatio-temporal filters, performance measurement among moving stimuli would be possible. The strategy in model development is therefore to validate the simple model on simple stimuli, and then incrementally build in model complexity to handle the more complex stimuli.

4 The Model

Figure 1 is a schematic diagram showing the different stages of the visibility model. In this section, each stage of the model will be described in detail.

4.1 Input parameters and stimulus format

The model, as currently formulated, takes as input one or two image files, and a number of optional lighting and observer state parameters. These parameters are listed here with the default value and units in parentheses:

Screen luminance (10.0 foot-lamberts)
Illuminance (0.0 foot-candles)
Eccentricity of displayed stimulus (0 degrees)
Fixation depth (741.12 mm)
Stimulus depth (741.12 mm)

The image files start with two four-byte integers indicating the width and height of the image in pixels, followed by rows of pixel values in floats. The images can be any size, although should be at least 256x256 to allow filtering within a large enough range of different frequency bands. The conversion factor from pixels to mm is currently fixed in the software as 13.21 pix/mm. With only one image as input, the software calculates the probability of detecting that stimulus; with two images, the standard discrimination probability is calculated.

In the current version of the model, pixel values are required to range from -1.0 to 1.0, with the maximum absolute value indicating the contrast of the stimulus. For example, a sine grating stimulus with peaks at ±0.5 would have a contrast of 0.5. To remove the need for this convention, a model stage

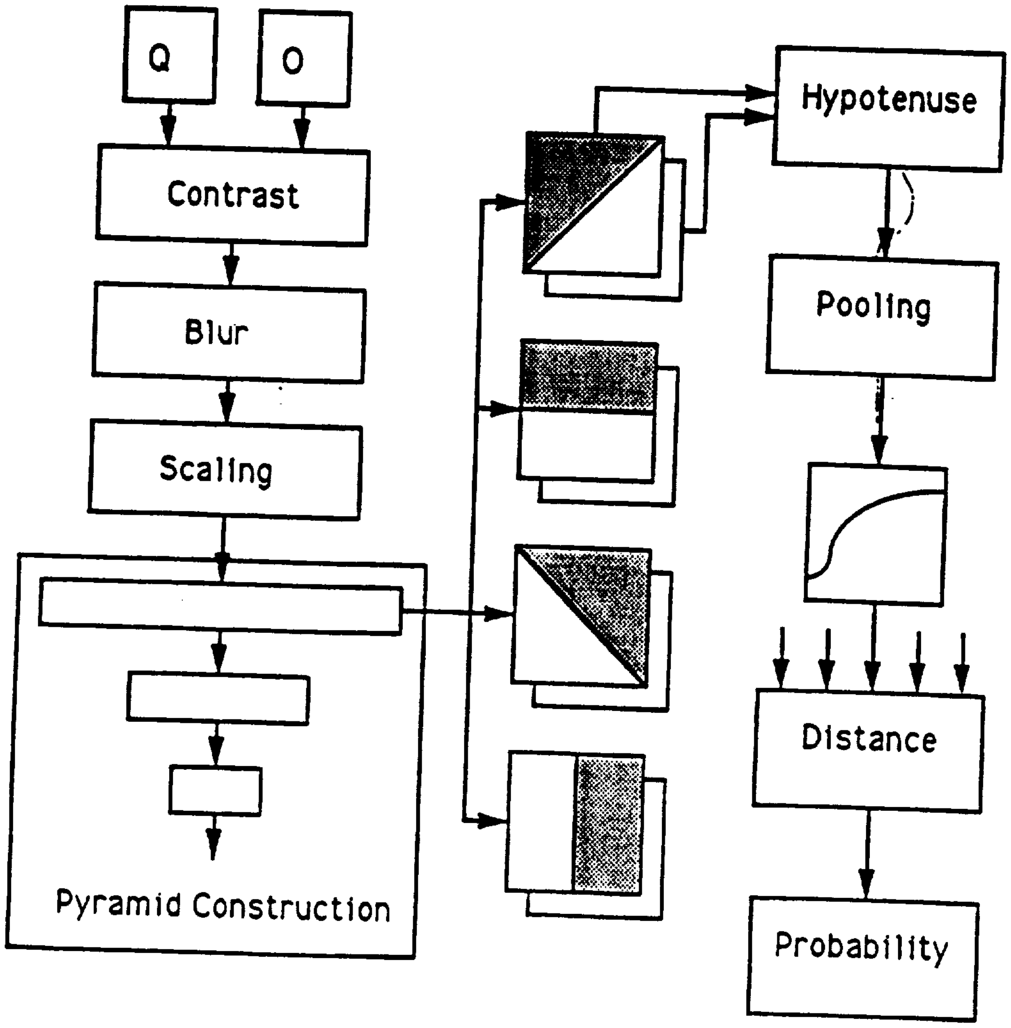


Figure 1: Visibility model flow diagram.

which computes contrast from arbitrarily scaled input images is needed, but has not yet been implemented.

4.2 Front end calculations

Several initial transformations on the input images are performed prior to the linear filtering stage of the model. These transformations model the effect of changes in fixation depth, veiling luminance, and fixation eccentricity.

4.2.1 Fixation depth

In order to account for changes in effective image resolution with changes in the difference between image depth and fixation depth, we used geometrical optics to calculate the size of the blur circle, and then pre-filtered each input image with this disk-shaped convolution kernel. This calculation requires knowledge of the distance from the exit pupil to the imaging surface (i.e., the retina), which we took as 20.3 mm from Westheimer (1986). It also requires an estimate of pupil size. For this, we wrote a simple interpolation routine to estimate pupil diameter at any light level from a table published in Hood and Finkelstein (1986).

4.2.2 Contrast reduction

Veiling luminance, caused by the reflection of ambient light by the display screen, reduces the effective contrast of displayed information. We are modelling the screen face as a perfectly lambertian surface with a reflectivity of 10%. This assumption implies that an illuminance of 10 fcd will result in a veiling luminance of 1 fL. We are defining contrast as

$$c \equiv (\mathrm{lmax} - \mathrm{lmin})/(\mathrm{lmax} + \mathrm{lmin})$$

where lmax and lmin are the maxmimum and minimum displayed luminances. Given this definition, the addition of a veiling luminance v to both lmax and lmin changes the contrast by a factor $1/2v$.

4.2.3 Eccentricity scaling

Abundant psychophysical evidence shows that contrast sensitivity remains roughly constant across the visual field, if the grating patch is scaled up by a linear function of eccentricity. This strongly suggests that processing is similar across the visual field, except for a linear scaling up of sensor size towards the periphery. In order to model the effect of this change in sensor size, we found it more convenient to scale down the size of the input images as a function of eccentricity, rather than to scale up the size of the sensors. We used the scale factor (k) of 0.4 quoted by Watson (1983), so that the scaling of input images as a function of eccentricity (e) is

$$1.0/(1.0 + ke).$$

4.3 Linear filtering

4.3.1 Pyramid decomposition

In order to filter within a range of different frequency channels, the input image is first decomposed with a gaussian pyramid into channels spearated from each other by one octave, The frequencies we chose for these channels are identical to those used by Watson (1983); i.e., 32 through 0.5 c/d, corresponding to seven octaves or equivalently, seven pyramid levels.

4.3.2 Computing filter gains

The human visual system is not equally sensitive at all frequencies. A plot of contrast detection threshold as a function of spatial frequency shows roughly an inverted-U shape, with a peak at roughly 2 c/d, and complete loss of sensitivity by approximately 60 c/d. Moreover, as shown by van Nes and Bouman (1967), the shape of this contrast sensitivity function changes with retinal illuminance. To model these dependencies, the image component in each frequency channel is weighted by a gain factor appropriate for the retinal illuminance, before any oriented filtering is performed.

Retinal illuminance (in photopic trolands) is calculated as the amount of light incident on the cornea (in cd/m^2) times the pupil area (in mm^2), where the light incident on the cornea is assumed to be the screen luminance plus the veiling luminance, appropriately converted from fL to cd/m^2. The gain at each frequency is then calculated directly from the van Nes and Bouman data with a simple log interpolation function to return sensitivities at retinal illuminances other than those reported in the data. For example, if the threshold modulation for a 1 c/d grating were 1% at 10 tds and 5% at 1 tds, then we interpolate the threshold at 3.16 tds (half the log distance from 1 to 10) to be 2.23% (half the log distance from 1% to 5%). This direct calculation is possible only under the assumption of no summation among different frequency channels; any assumed summation would require a more complicated relationship between the contrast sensitivity function and the gain of each channel.

4.3.3 Steerable filtering

For convenience and speed of oriented filter operation, we use the steerable filters of Freeman and Adelson (1990), which allow separable calculation of linear filter responses at any orientation and phase. The filters implemented here, a second derivative of a gaussian and its Hilbert transform, have a log bandwidth at half height of approximately 0.7 octaves. This is within the range of bandwidths inferred psychophysically (e.g., Watson and Robson, 1981). We are using four orientations (0, 45, 90, and 135 degrees) and two phases (sine and cosine), four a total of eight oriented filter responses per pyramid level. The orientation bandwidth of these filters (i.e., the range of angles over which the filter output is greater than one half the maximum) is approximately 65 degrees. This figure is slightly larger than the 40 degree tuning of monkey simple cells reported by Devalois et al (1982), and the 30 to 60 degree range reported psychophysically by Phillips and Wilson (1984).

4.3.4 Energy calculation

During the early stages of model testing, we found that the detectability of a simple edge could change dramatically with small changes in edge position. To combat this problem, a small amount of spatial summation was added by computing energy after the linear filtering stage. That is, corresponding sine and cosine filter responses were combined as

$$e(x_i) = \sin^2(x_i) + \cos^2(x_i)$$

where x_i is a linear filter response, indexed over filter position, orientation, and frequency band.

4.4 Point non-linearity

Nachmias and Sansbury (1974) showed that the results of a grating contrast discrimination experiment, when plotted with threshold contrast increment as a function of the base contrast from which the increment threshold is being measured, produce a dipper-shaped curve. These authors argued that the results can be modelled by assuming a sigmoid non-linearity following a linear detection mechanism. The decision mechanism has available to it only the output of this non-linearity, and reliably discriminates between inputs of two different contrasts when the difference in outputs is greater than some threshold.

To quantitatively model the dipper-shaped contrast discrimination curve, we follow Legge and Foley (1980) in using a non-linear transducer of the form

$$T(L_i) = \frac{r|L_i|^n}{|L_i|^2 + s^2}$$

where T is the non-linear transducer output, L_i is the linear filter response (indexed as above), r is an overall gain-setting parameter, n is a real number greater than 2 (2.4 here), and s is a semi-saturation constant (0.0075). Given that our energy calculation described above already squares the linear filter resonses, the transducer output can be expressed as

$$T(e_i) = \frac{re_i^{n-2}}{e_i + s^2}$$

where e_i is short for $e(x_i)$ in the energy expression above.

4.5 Decision stage

4.5.1 Distance calculation

We are assuming no summation among transducer outputs, and a decision mechanism in which discrimination is governed by the pathway whose transducer output shows the maxmimum change between the two stimulus presentations. Although these assumptions are probably not correct in detail, they dramatically simplify model theory and calculations, and thus are useful as a first pass at the truth.

One way of expressing this maximum change assumption is that the discriminability between two images is assumed to be proportional to a non-Euclidean distance between the points representing the two images in the space of transducer outputs. That is,

$$D(s_1, s_2) = \sqrt[Q]{\sum_{i=1}^{n} [T(e_i(s_1)) - T(e_i(s_2))]^Q}$$

where s_1 and s_2 are the two images, and n is the number of different transducer channels. If $Q = 2$, this expression returns the Euclidean distance between points in the transducer output space. As $Q \rightarrow \infty$, the distance metric gets closer and closer to the maximum change model described above.

4.5.2 Distance to probability

This distance measure can be used in the following generalization of the Nachmias and Sansbury model: Two images are reliably discriminated whenever the result of the distance calculation for the two images is greater than some threshold.

This simple generalization is sufficient to model results in which discrimination thresholds are measured as a function of a change in a stimulus parameter (e.g. contrast, as in the Nachmias and Sansbury results, or spatial frequency, as in the edge transition results to be discussed below.) For these tasks, the threshold distance can be understood as the distance which gives a 75% probability of discrimination among the two stimuli. However, for the cockpit display visibility modelling, the relevant performance measure is the probability of discrimination given two images, not the required change between two images given a fixed probability. Therefore, a mapping between distance and probability is required, not for just a single value of distance and probability, but along the entire range of each.

We have generated this complete mapping from distance to probability using two sets of data: Contrast detection psychometric functions from Foley and Legge (1981) and contrast discrimination functions from Legge and Foley (1980). In the former data set, probability of detecting a sine grating is plotted against the contrast of that grating. Foley and Legge showed that these data are well fit by an expression

$$p(C) = 100 - 50\exp(-aC^b)$$

where C is contrast, and a and b are parameters, fitted to one observer as $a = 52.60$ and $b = 3.0$.

For the latter data set, a good fit is obtained using the transducer function described above. This transducer expresses distance as a function of linear filter output, but can be expressed equally well as a function of grating contrast, since linear filter output is proportional to contrast. This means we can express both distance (i.e., transducer output) and probability as a function of contrast, and have only to invert the transducer function to obtain an expression for probability as a function of distance. We were not able to solve this problem analytically, but instead generated an accurate computational solution that was incorporated into the visibility model software as a lookup table.

To test this mapping, we applied the model to predict psychometric functions for contrast discrimination, also published in Legge and Foley (1981). In these functions, probability for detecting a change in contrast of a sine grating at threshold is plotted against that change in contrast. The model predicted these functions very accurately.

4.6 Position uncertainty

One important task for performance measurement in a cockpit display environment is character discrimination as a function of eccentricity. However, predictions on O vs. Q discrimination from the model as described so far incorrectly assert near perfect discrimination out to as much as 16 degrees, whereas in reality, probability of successful discrimination among these two characters falls to chance by about 5 degrees.

Because of the lack of spatial pooling in the model, the task of discriminating between O and Q becomes, for the model, the task of simply detecting the diagonal segment in the Q, a task which could in fact be performed reliably out to 16 degrees. But for letter discrimination, it is not sufficient to simply detect all the component features; accurate spatial relationships among these features must also be recovered. This led us to consider other psychophysical tasks for which accurate localization is important, most notably the three dot bisection task of Yap, Levi and Klein (1987; JOSA, 4, 8, pp. 1554-1561). Here, as in contrast detection, a linear scaling of image area with eccentricity leads to constant performance across the visual field. However, the scaling factor for three dot bisection is larger by about a factor of three than the scaling factor for contrast detection.

The fact that tasks requiring accurate localization scale by a larger factor than that of contrast sensitivity suggested to us the need for a stage of eccentricity dependent spatial pooling of sensor responses following the linear filtering stage. If filter responses were pooled over a progressively larger area towards the periphery, then the central visual system would become increasingly uncertain of the spatial position of a given image feature. Furthermore, if the gain of the pooling filters were scaled so that the magnitude of the pooled response of a uniform input were unchanged with the size of the input area, then the model would continue to accurately predict contrast sensitivity in the periphery.

When we incorporated these changes into the model, it was able to qualitatively predict published results in letter discrimination, three-dot bisection, and contrast sensitivity in the periphery. After making these changes, we found further verification for our letter discrimination predictions in a recent report by Farrell and Desmarais (JOSA (1990) 7, 1, pp. 152-159).

4.7 Output format

Model predictions are generated in files for which all parameters are fixed, except for eccentricity (i.e., degrees of visual angle off of the fixation point.) The output files thus contain a simple two column table of eccentricity - probability pairs, one such file for each combination of image pair, lighting, and fixation depth parameters.

The initial set of model results included predictions for two different fonts of Q and O (types a and b from the Apache Longbow Crew Systems Interface Document). A screen depth of 30" was assumed, coupled with four different fixation depths: 15", 30", 60", and ∞. For each of the two fonts and four fixation depths, we generated model predictions at the following luminance/illuminance combinations:

luminance	illuminance
10,000 fcd	350.0 fL
10,000 fcd	100.0 fL
10,000 fcd	31.6 fL
10,000 fcd	10.0 fL
1,000 fcd	10.0 fL
100 fcd	10.0 fL
10 fcd	10.0 fL

These illuminances range from bright sunlight (10,000 fcd) to late dusk (10 fcd). The luminances range from the maximum available, as listed in the Apache Longbow document (350 fL) to the minimum daylight mode setting (10 fL).

Figures 2, 3, 4, and 5 show model results for the range of character sizes and lighting parameters described above. In all thse figures, fixation depth is equal to stimulus depth (30").

In order to generate complete discrimination contours from the model output files, the package of model software also contains a routine which generates probability contours from 55% to 95%, and the 99% contour. This routine performs a linear interpolation on the probability values listed in the model output file, to determine the eccentricity at which each contour's probability value would occur. It then computes a set of x,y points (in degrees of visual angle) for a complete circle at that eccentricity. Additional data may, in the future, require refinement of this module to produce contours which deviate from a purely circular shape.

5 Validation

5.1 Edge transition data

The discrimination model has been successfully tested against the Carlson and Cohen (1978) edge transition data, with a fit better than that of Carlson and Cohen's original (JND) model.

In the edge transition task, observers are asked to discriminate a change in sharpness of an intensity edge, as a function of the base sharpness of that edge. More specifically, in the Carlson and Cohen data, the one-dimensional

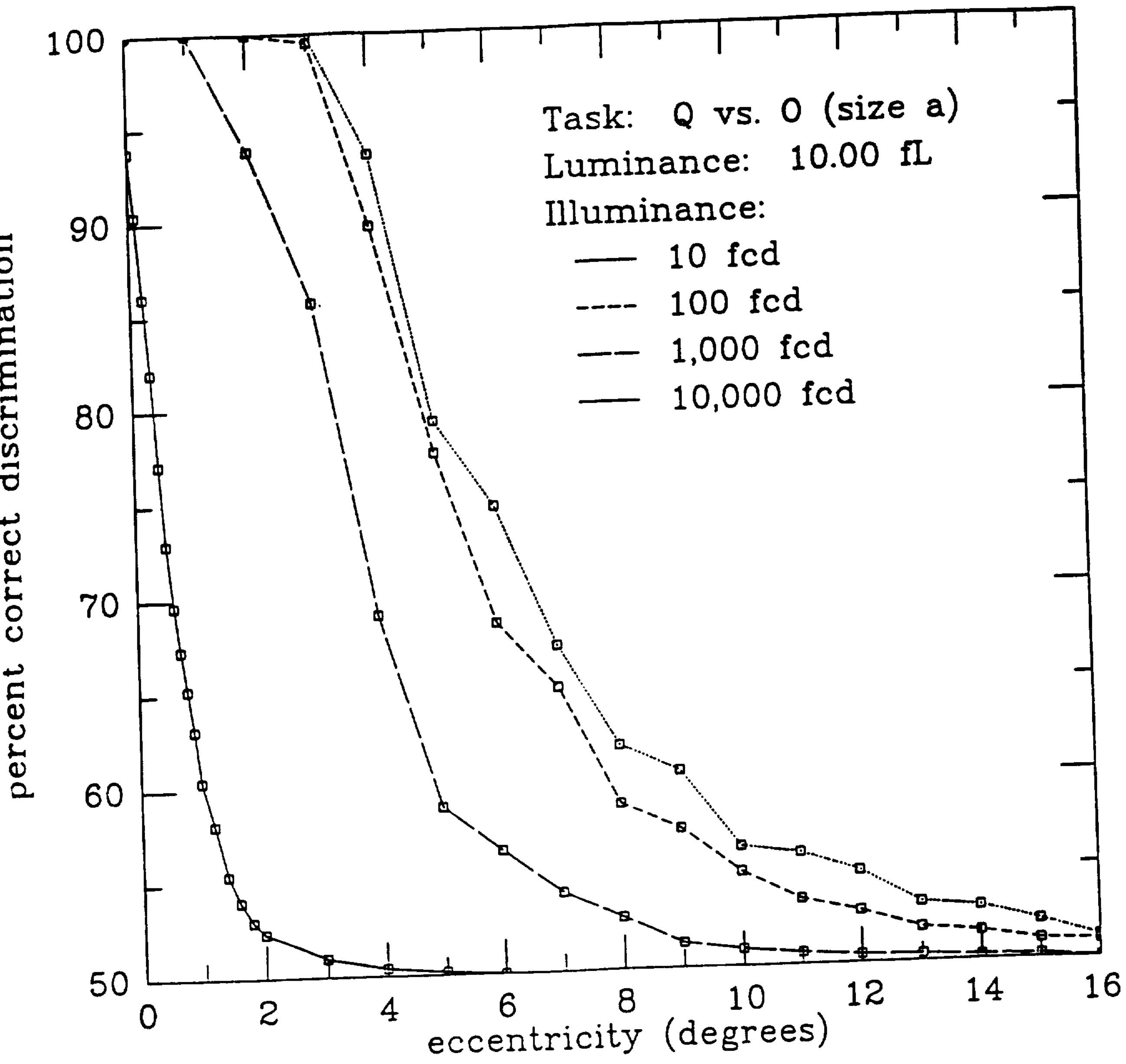


Figure 2: Model predictions: Q vs. O, size a, luminance fixed at 10 fL, four different illuminances.

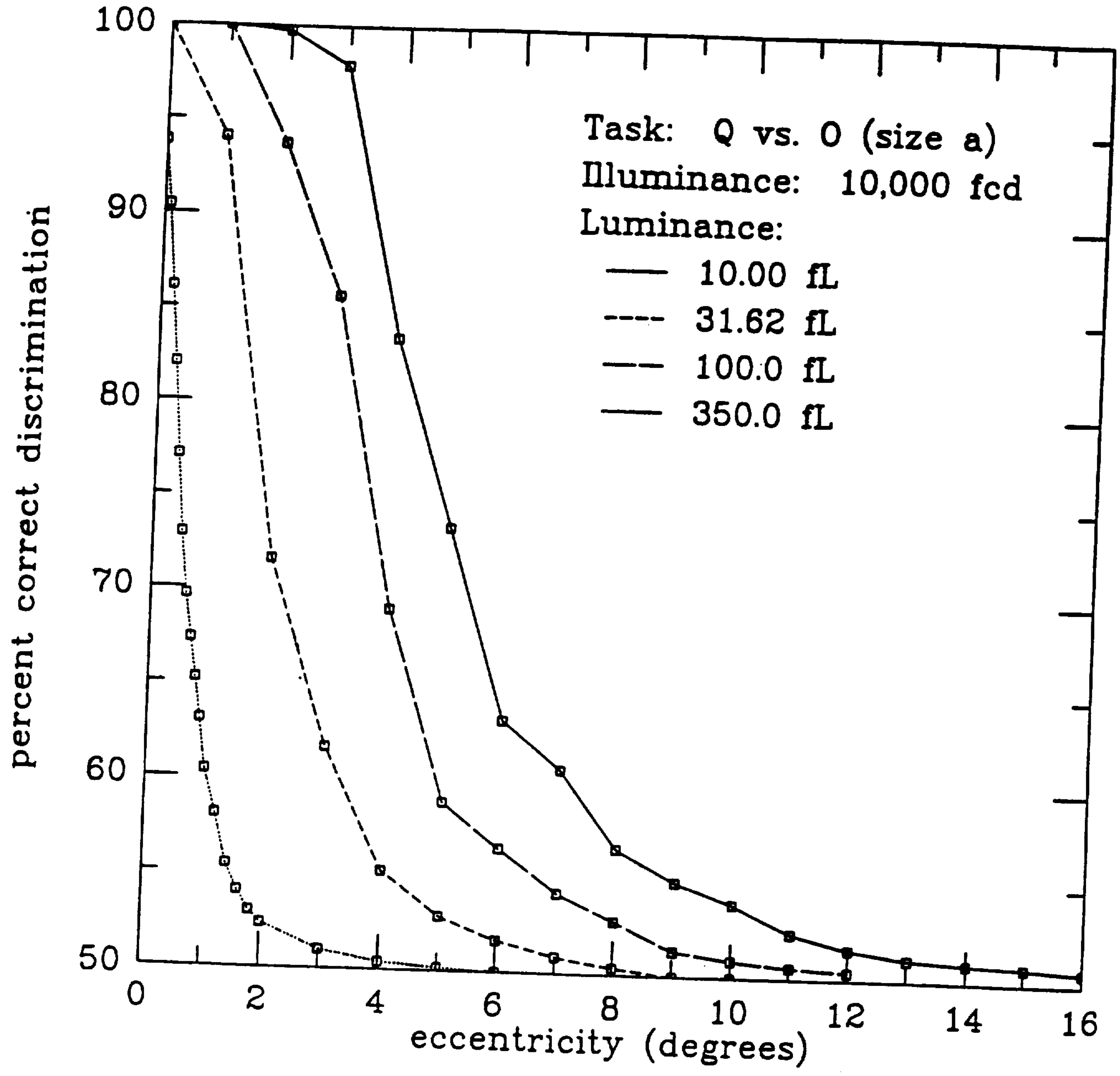


Figure 3: Model predictions: Q vs. O, size a, illuminance fixed at 10,000 fcd, four different luminances.

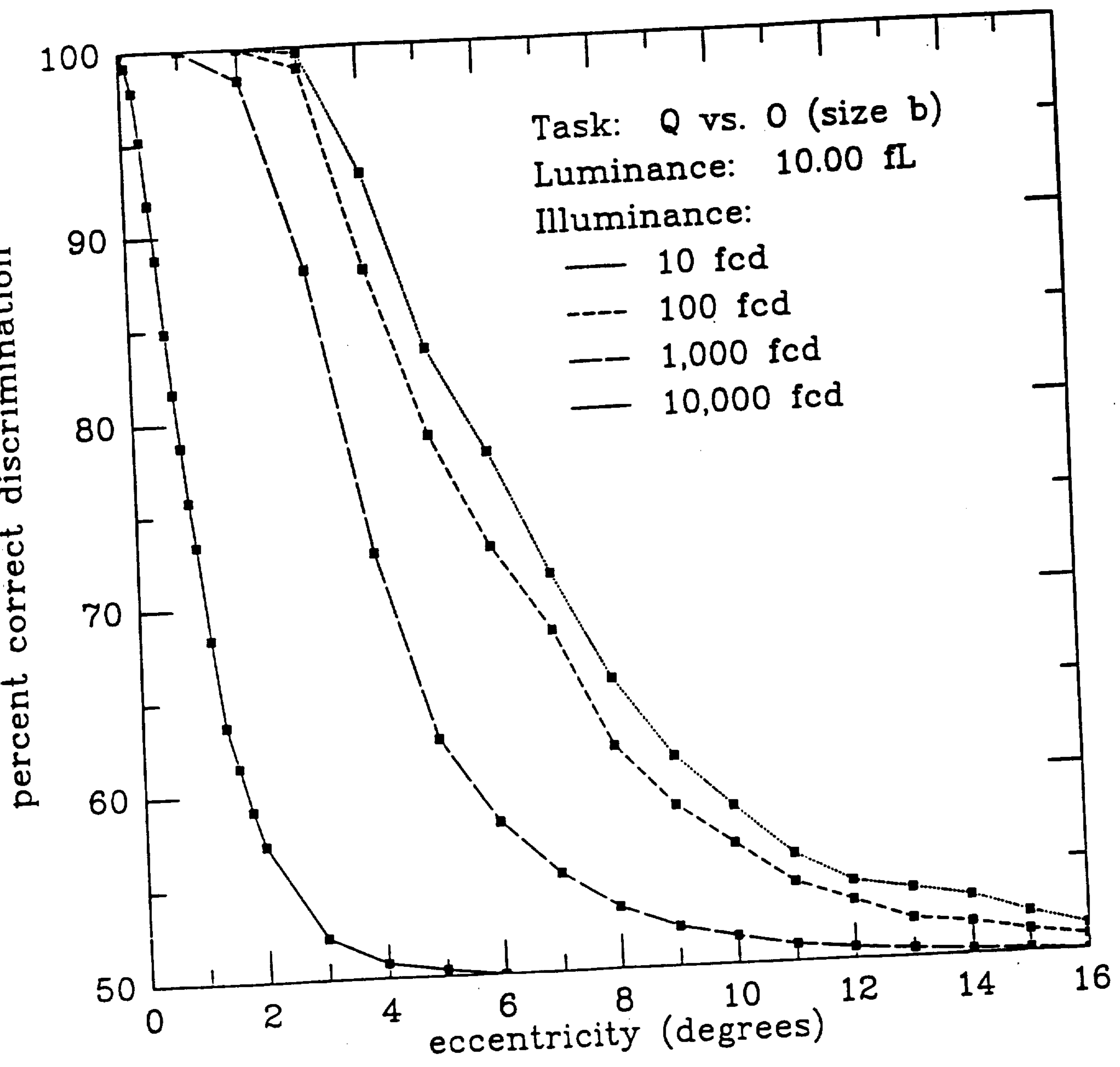


Figure 4: Model predictions: Q vs. O, size b, luminance fixed at 10 fL, four different illuminances.

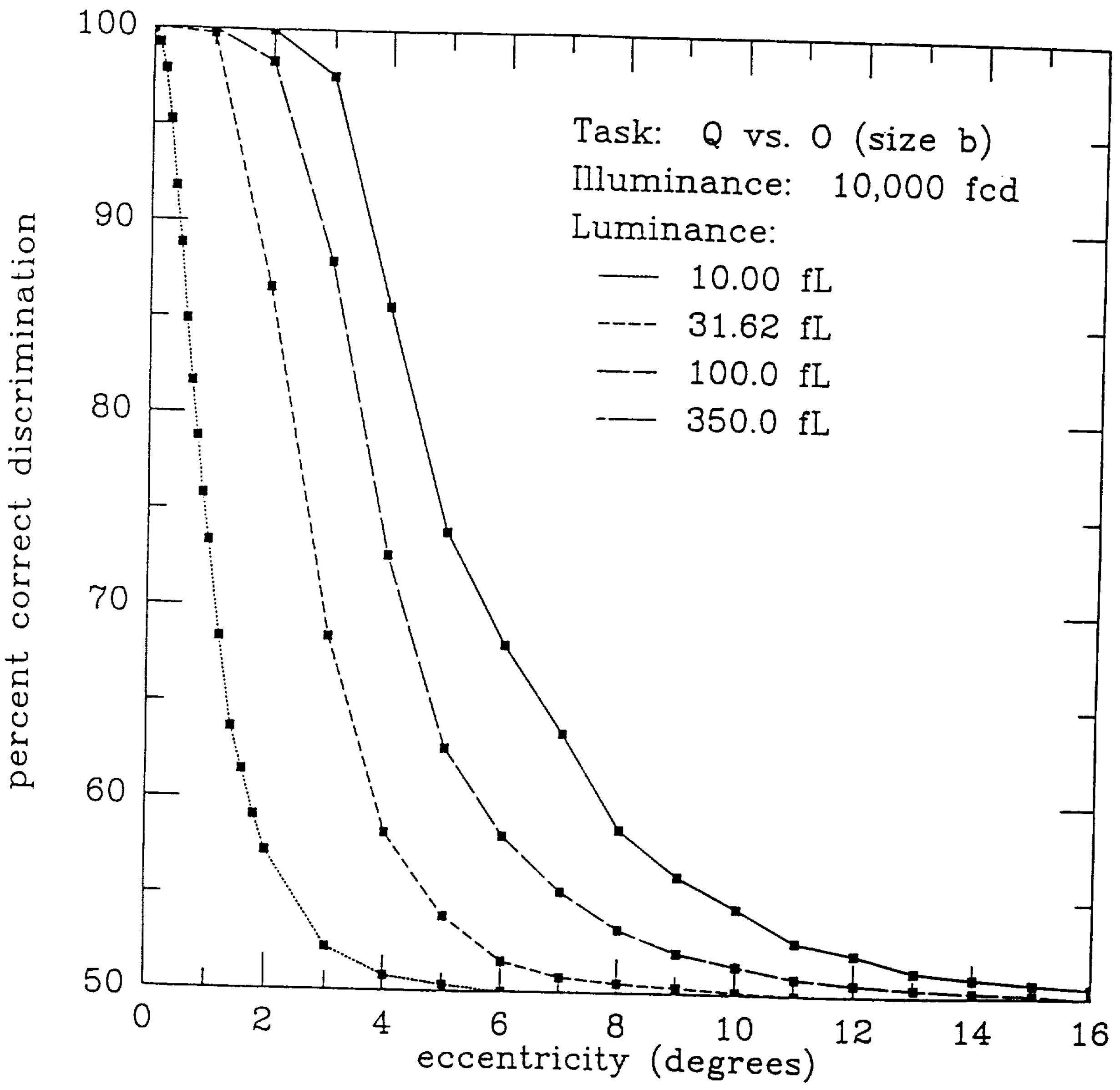


Figure 5: Model predictions: Q vs. O, size b, illuminance fixed at 10,000 fcd, four different luminances.

horizontal cross-section of a vertical edge is generated as an error function (erf), where

$$\mathrm{erf}(x) = (2/\sqrt{\pi}) \int_0^x \exp(-t^2)dt$$

The sharpness of the edge is controlled in the following parameterization for x in the above expression:

$$x = d\pi f_c / \sqrt{\log(2)}$$

Here, d indexes the distance across the edge image in degrees, and ranges from $-w/2$ to $w/2$ (where w is the width of the image in degrees), and f_c is the frequency at which the modulation transfer function for the edge has fallen to one half of its maximum. This parameter f_c thus governs the sharpness of the edge, with higher values giving sharper edges.

Given this parameterization, the psychophysical task can be expressed as follows: From an edge with a fixed f_c value (call it f_x), how much does the f_c have to increase (call the increase df_x) so that the change in edge sharpness is just noticeable. The results were plotted by Carlson and Cohen with $\log(f_x)$ on the abscissa, and $\log(df_x/(f_x + df_x))$ on the ordinate. In general, these results follow a u shape, with the minimum near a frequency of 1 cycle/degree.

Figure 6 shows the edge transition data from Carlson and Cohen, together with their model fit and the fit of the discriminability model described here. The discriminability model was fit by finding the value of df_x at each f_x which gave 75% probability of discrimination between the two edges.

5.2 Character discriminability data

We have begun collecting additional data to test the model's predictions in a character discrimination task. All experiments were performed on the MacIIx. The stimuli, as in the model runs, were uppercase O and Q, adapted from the Mac Helvetica14 font to accurately replicate the size and bitmap pattern of the Size A characters described in the Apache Longbow Crew Systems Interface Document. The screen luminance was variable from between 0 and 13.5 ftL. The subject's distance from the screen is 30 inches. Veiling illuminance for the data collected so far was approximately 10 fcd.

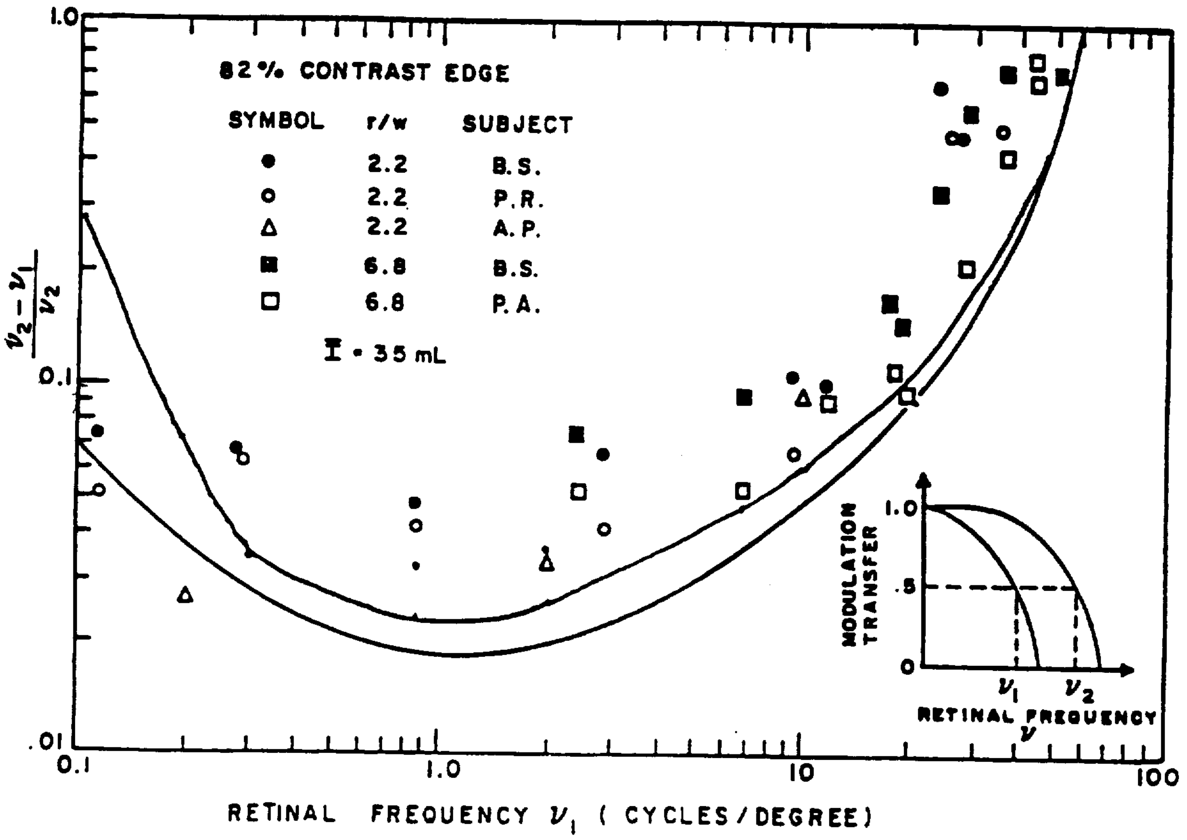


Figure 6: Edge transition: data and model. Upper curve shows the predictions of the visibility model described in this report; lower curve shows the predictions of the JND model of Carlson and Cohen.

A small fixation mark (+) was continuously present at the center of the screen. During each trial a single character, O or Q, was flashed on for 167 msec, at an eccentricity e either to the right or to the left of the fixation point. This direction, as well as the identity of the character, was randomly varied from trial to trial. The eccentricity e remained constant within a session, as did the luminance of the characters and background. The subject's task was to identify, with a keypress, which of the two characters was presented. A session ended when each of the four possible combinations of character identity and position has been presented n times, where n is typically 50.

Preliminary data from this paradigm are shown in Figure 7.

In this figure, the three connected points marked by the heavier crosses show the data for one subject, under lighting conditions of 10 fcd illuminance, and 6 fL screen luminance. Also shown in the figure, marked by filled squares, is the model run with lighting conditions most similar to those of the data run; i.e., 10 fcd, 10 fL. As shown in the figure, the slope of the falloff in performance with eccentricity is in good agreement with model predictions. Not surprisingly, though, there is a "dc shift" between model and data; that is, the eccentricities at which these reported probabilities occur are *not* in good agreement with the model results.

One obvious possible reason for this discrepancy is that the short stimulus duration used in the experiment (to prevent eye movements) reduces the effective contrast of the stimuli, thereby significantly affecting performance. Since the model as it now stands is blind to changes in stimulus duration, it would not be expected to accurately predict performance for arbitrarily short duration stimuli.

To make sure that an increase in effective contrast does increase human performance (i.e., to make sure that the human results reported above are not already at a performance asymptote), we used a second screen luminance configuration, chosen to maximize the effective contrast available on the MacIIx. In this condition, the characters are dark on a 13.5 fL (maximum possible luminance) background. Results are shown in the figure as empty squares. Under these conditions, the human data are much improved, shifting the performance curve outward by about two degrees.

Shown also in the figure, for comparison purposes, is a model run in which the effective contrast is low; i.e., a 10,000 fcd, 31.6 fL condition. Here, the results are quite similar to the 10 fcd, 6 fL data run. These results suggest that some additional model development is needed to accurately measure

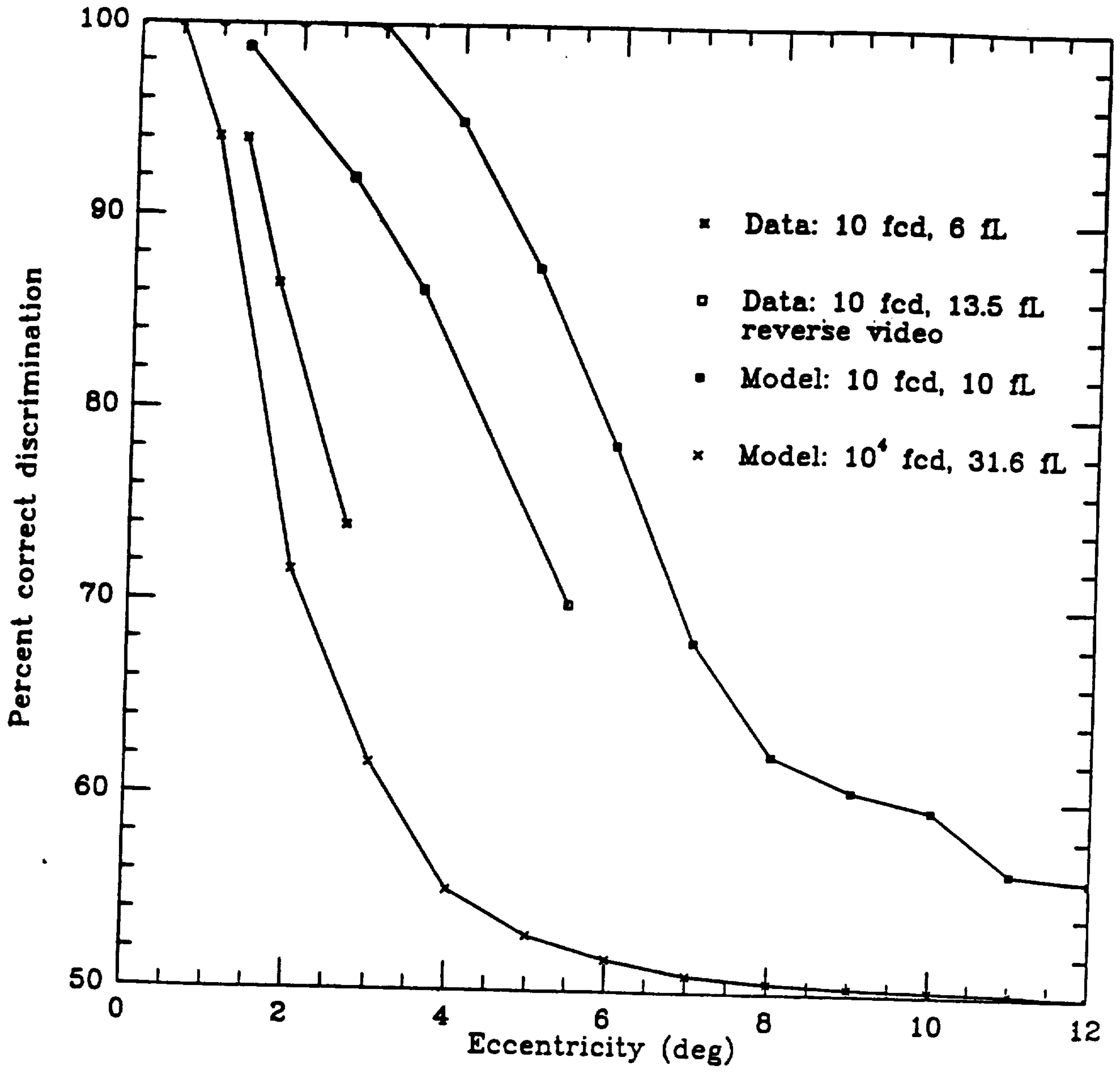


Figure 7: Letter discrimination: data and model.

the effective contrast of a stimulus, given its time course and other relevant stimulus and observer parameters.

6 Future directions

In general, future plans for this project involve both extending the range of model applicability and, through additional data collection and model refinements, improving confidence in model predictions. Specific plans are outlined below.

6.1 Stimulus parameters

6.1.1 Change and motion

An important area for extending the range of the model is into temporal stimulus parameters; i.e., explicitly modeling visual response to stimuli which are moving or otherwise changing in time. The importance of the latter was made clear in the previous section, where it was described that the short time course of the experimental stimuli may be changing their effective contrast. Modeling response to motion would also vastly increase the useful range of stimuli: target detection could be modeled, as well as dynamic cockpit display events such as flashing lights or rapidly deflecting meter needles.

6.1.2 Color

Extending the model into the chromatic domain is relatively straightforward, and useful for the same reasons cited above for change and motion detection.

6.2 Tasks

6.2.1 Discrimination

The model as it now stands can generate predictions for discrimination tasks other than character discrimination. Tasks such as target detection and meter needle position discrimination can be investigated, even before the model is extended into the motion or color domains.

6.2.2 Localization

Another important task, especially for target detection, is stimulus localization; i.e., the accuracy with which stimulus position can be reported. Because of the areal summation currently built into the model, accurate localization of stimuli will decrease as a function of stimulus eccentricity. Additional research will be required to determine other relevant stimulus, task, and observer parameters affecting performance in this task.

6.3 Performance measures

6.3.1 Probability and confusion matrices

Currently, the model generates an estimate of probability correct for discrimination between a pair of stimuli. A useful and straighforward generalization would be the production of a full confusion matrix for members of a set of allowable responses. This performance measure would become especially important if the vision model were given the responsibility of providing to other modules the identity of a percept, rather than just a measure of success in a simple discrimination task.

6.3.2 Response time

Another possible performance measure is the time required to perform a vision-based task. The usefulness of accurate modeling of this performance measure depends on the level of temporal detail required by other MIDAS modules. Since visual response times are almost always in the 100 to 500 millisecond range, accurate modelling would be irrelevant if other modules were operating in, for example, the 5 to 10 second range. Moreover, modeling response time would probably require significant additional effort to extend the current model.

6.4 Observer state parameters

6.4.1 Adaptation

The dynamics of light adaptation is a complex research question, to which entire careers have been devoted. However, much would be gained from a simple model in which the decay parameter for recovery from a change in lighting depends on the absolute light level and the size of the change. Moreover, modeling can also be simplified if the time scale over which recovery is modeled is restricted to a specific range; e.g., 100 to 1000 msec. However, it should be noted here that the addition of light adaptation to the vision model puts additional responsibilities on other MIDAS modules as well, since some module would be required to provide the vision model with ongoing estimates of lighting conditions, based for example, on sun position, cloud cover, and vehicle position.

6.4.2 Resource loading

Resource loading is an ill-defined but important parameter affecting pilot performance. One way to start investigating this parameter is to consider performance in dual task situations, about which a moderate amount of results has been reported (e.g., Pashler, Cognitive Psychology, 1989). The general strategy here is to consider resource loading not in terms of a homogeneous capacity model, but in terms of specific interactions among specific kinds of tasks.

6.5 Model refinement

6.5.1 Contrast gain control

Recent results (Lubin, ARVO, 1989-1990) suggest a model stage in which outputs of different spatio-temporal frequency channels set the gain of neighboring channels via a division-like operation. These data are relevant to the visibility model because they provide evidence against the simple independence of frequency channels in the current model. Inclusion of such a gain-setting stage would improve model performance across most discrimination tasks.

6.5.2 Detection vs. localization scaling

These two different stages of scaling were already discussed in the model description section above: The model contains an earlier stage at which sensor size scales as a function of eccentricity, and a later stage at which sensor responses are pooled as a different function of eccentricity. For simplicity, this second stage of scaling is currently implemented as a scaling down of the distance in the transducer output space. However, this simplification does not always accurately model the presumed scaling operation, since it decreases the output distance even between spatially distant pairs of stimuli, for which no pooling would be expected. An implementation more isomorphic with the presumed operation is thus required.

6.6 Data collection

6.6.1 Model verification

Data collection is ongoing. Current plans include extending the range of character discrimination data to other lighting conditions and character pairs. Performance in other cockpit display tasks, such as detecting a change in meter needle position, will also be measured. Moreover, as the model is extended to other performance measures and stimulus and observer parameters, additional data collection will become necessary. Tasks such as moving target detection, with performance measures such as the response time to accurately localize the target, may be undertaken.

6.6.2 Normative data

Another useful set of data would be normative data; that is, data for a wide range of individuals within the relevant subject population, so that variance estimates for different performance measures could be obtained. Such estimates would help establish more accurately the probability of success in different mission scenarios.

APPENDIX C — LIGHTHOUSE VP SOURCE CODE DOCUMENTATION

1 Organization Of The Source Code

1.1 The VP Executable Program

All of the source code, with the exception of `main.c`, is used to generate a number of libraries. When `main.c` is compiled with these libraries, the VP executable program is produced.

Environment Variables
There are several environment variables that need to be set in order to properly compile and run VP. These variables should be set by running the command file `.vpenv` in the `VP` root directory. The first step, though, is to edit `.vpenv` so that the environment variable `VP` is going to be set to the VP root directory. For example, if the VP directory hierarchy is rooted in `/usr/foo`, edit the line that begins "`setenv VP`" to read

```
setenv VP /usr/foo/vp
```

Once this is done, the rest of the program's environment variables can be set by executing `.vpenv` with the UNIX `source` command.

1.2 The Source Code

With the exception of the main program (`vp/src/main.c`) all of the source code is used to make the VP library files. Each subdirectory of `vp/src` contains the files that are used to make one library of routines. For example, the source files in `vp/src/model` are used to make the library `vp/lib/libmodel.a`. All of the header files are kept in `vp/lib`.

The Main Program: `vp/src/main.c`
The function of the main program `vp/src/main.c` is straightforward. First it calls `initialize` (which opens and initializes all of the initial windows) and then it calls `menu_loop` (which processes intermediate events while waiting for a menu event).

The `vpmain` ***Library***
The routines in the `vpmain` library take care of much of the overhead in the program. Start-up initialization, window maintenance, message and error handling, event handling and list operations are implemented in `libvpmain.a`. The C files described in this section make `libvpmain.a` and are found in `vp/src/vpmain`.

`event.c` — Contains the function `process_events`, which waits for an event to queue and processes it accordingly. It returns a structure of type `Event`.

`grid.c` — Contains the routine that draws the scale grid.

`init.c` — Opens the start-up windows and initializes the eyes.

`list.c` — Functions for operations on two-way open-ended and circular linked lists and stacks.

`menu.c` — The menus for the message window and environment windows are defined here. (Note that all environment windows have the same menu.)

`message.c` — Functions for displaying text in the message window, maintaining the message stack and getting input text.

`view.c` — Contains the routines that change the view in environment windows.

`window.c` — Functions that open and close the message and environment windows and find data in the window list.

The **`model`** ***Library***
The model library (`/vp/lib/libmodel.a`) provides the routines for creating, maintaining and drawing the model hierarchy. This would also include the editing routines, which allow the user to move models to new locations and delete them from the environment, and the lighting routines.

`create.c` — Contains the routines that "create" models, which essentially means reading the model file and adding the model's data structure to the model tree.

`draw.c` — Contains the routines that draw the model hierarchy and individual models.

`edit.c` — Contains the routines that move and delete models.

`lexer.c` — The lexical analyzer used by the model file parser.

`light.c` — The routines that define and control lighting.

`parse.c` — The model file parser.

`pick.c` — The routines used to "pick" a model that is drawn on the screen.

`symbol.c` — Symbol table support for the parser.

`tree.c` — Tree traversal routines that apply a function (specified as an argument to the traversal function) to each node in the tree.

`utils.c` — Various general utilities.

The **`vector`** ***Library***
The vector library (`vp/lib/libvector.a`) contains routines for vector and matrix operations. (Some vector operations are performed with macro substitution and are defined in `vp/include/vector.h`.)

`matrix.c` — Routines that perform operations on homogeneous transformation matrices.

`vector.c` — Routines that perform vector operations.

The **`eye`** ***Library***
The eye library (`vp/lib/libeye.a`) contains the all of the routines that are associated with the eyes and vision. It is important to note that even though retina window routines are defined in `libeye.a`, they must use the window support that is provided in `libvpmain.a`.

`eye.c` — Contains initialization and fixation routines.

`menu.c` — Defines the menu for the retina window. Note that some eye operations are executed from the environment window menu.

`retina.c` — Routines that create, draw and delete retinal objects.

`window.c` — Routines for opening and drawing retina windows.

2 Lists, Stacks and Trees

Lists and stacks are defined in `list.h` and supported by functions in `list.c`. These data structures are dynamically implemented so that there are no predefined limits on their sizes.

2.1 Lists

A generic list node is defined in `list.h` as

```
struct list_node {
     void              *dat;
     struct list_node *prev;
     struct list_node *next;
};
```

This structure is the basis of the program's open-ended lists, circular lists and stacks. The fields `prev` and `next` point to the previous node and the next node, respectively. The use of the `prev` pointer allows for two-way operation. The data field, `dat`, is a pointer to `void`. Having a `void` pointer as a data field is what gives the list node its generic quality. Any pointer can be converted to `void` through a cast and then converted back to its original type through a cast without damage. For convenience , `List` is a type defined in `list.h` as a pointer to `struct list_node`.

The most basic list supported by VP is an open-ended linked list. The prototypes for the linked list support functions are shown below.

```
void     list_append(List *list, void *data);
Boolean  delete_listnode(List *list, void *data);
void     delete_list(List *list);
```

The argument `list` points to the head of the list. `data` points to the data that is to be added to or deleted from the list. The function `list_append` is used to append an item to the end of a list, or to create a new list when `list` is equal to `NULL`. `delete_listnode` removes the list node containing `data` from the list. It returns `FALSE` if the list was empty (equal to `NULL`) or if `data` wasn't found in the list. `delete_list` deletes an entire linked list.

Circular lists are implemented in very much the same way, except they don't have a beginning or an end. The prototypes for the circular list support functions are shown below.

```
Boolean  insert_circlelistnode(List *list, void *data);
Boolean  remove_circlelistnode(List *list, void *data);
```

`insert_circlelistnode` is used to add data to the list. A new list node for `data` is created, and it is inserted into the list immediately after the node pointed to by `list`. `remove_circlelistnode` deletes the node containing `data` from the list. If the list was empty or if `data` wasn't found, `remove_circlelistnode` returns `FALSE`.

2.2 Stacks

A stack is implemented as a special case of a linked list in which the end of the list is the top of the stack. Type `Stack` is defined in `list.h` to be exactly the same as `List` — a pointer to `struct list_node`. The prototypes of the functions that are used to push and pop the stack are listed below.

```
void push(Stack *stack, void *data);
void *pop(Stack *stack);
```

push creates a stack node for data and pushes it onto stack. What actually happens is that push appends the data to the end of the list and sets stack to point to the end of the list, which is the new top of the stack. To pop the stack pop assigns the data in the last list node to a temporary variable, deletes the last node from the list, assigns the new end of the list to stack and returns the former top of stack (which was saved in a temporary variable).

Trees
A generic tree node is defined in vp/include/list.h as follows:

```
typedef struct tree_node *Tree;
struct tree_node {
     void  *dat;
     Tree  *prev;
     int   numlinks;
     Tree  *link;
};
```

3 VP Windows

Each window has a number of parameters that must be known by the program in order for it to be drawn properly. Information about the projection and view transformations, for example. The program must also know exactly what is to be drawn in the window. This section deals with the issues of window data and the implementation of drawing the windows.

3.1 Window Organization

Each window that is opened by the program has a `Window` structure associated with it. The `Window` structure (shown below) is defined in `vp.h`.

```
struct window {
     long     wid;                         /* Graphical window id.
     */
     int      menuid;                      /* The window's menu */
     char      *title;                      /* Displayed in title
     bar. */
     void      (*draw_wintype)(Window  *w);       /*  Draws  the
     window */
     float    bkgnd[4];                    /* Background color. */
     Xfrm      cop;                     /*  Center  of  projection  of
     view. */
     float    vrp[3];                  /* View reference point. */
     Angle    fovy;                    /* Vertical field of view */
     float    aspect;                  /* x:y aspect. */
     Model    *vcam;                   /* Camera model at cop. */
     Tree     modeltree;               /* Tree of 3D object models.
     */
     List     objects;                 /* Generic list. */
     struct {                          /* Various flags. */
          unsigned int     wintype : 2;
          unsigned int     is_main : 1;
          unsigned int     is_lit  : 1;
     }        status;
};
typedef struct window Window;
```

Some of `Window`'s members apply to every window, while others are only needed by certain types of window. Below is a short description of the structure's members.

`wid` — The graphical window identifier.

`menuid` — Each window uses its own menu, and this is the identifier of its menu. This helps to eliminate the need for a single massive menu.

`title` — The title that is displayed in the window frame's title bar.

`draw_wintype` — Points to the function that draws the type of window that is specified in `status.wintype`. Each type of window is drawn by a specific function. At the time that the window is being opened, the appropriate drawing function is assigned to `draw_wintype`.

`bkgnd` — The background color of the window.

`cop` — A transformation matrix that specifies the window's center of projection and describes the view. Used only by three-dimensional windows.

`vrp` — A vector specifying the view reference point of a three-dimensional window.

`fovy` — Specifies the vertical field of view for perspective projection in a three-dimensional window.

`aspect` — Specifies the field of view in the horizontal direction. It is equal to the x:y aspect ratio of the window's dimensions in screen coordinates.

`vcam` — A camera icon that is drawn at the window's `cop`. This can only be seen from other environment windows, and acts as an aid in relating to the view.

`modeltree` — The tree of graphical objects that are part of a window's three-dimensional environment.

`objects` — A list that can be used for any purpose. It has no dedicated use.

`status` — A set of status flags. `status.wintype` denotes the "type" of window — `THREE_D` for an environment window, `RETINA` for a retina window, or `MESSAGE` for the message window.

The Global Window List
With the exception of the message window, there is no predetermined limit on the number of windows that can be opened in the program. The user can open as many environment and retina windows as practicality (and the computer's memory) will allow. VP keeps track of all of the windows by keeping them all in a single list. If opened properly, each window will have a `Window` structure in the global window list `WinList`. `WinList`, declared in `vp.h` as type `List`, is a circular list that contains pointers to each window's `Window` structure. The order of the windows in the list is meaningless. The only requirement is that each existing window appear in `WinList` exactly once. As windows are opened, they are added to the list. Likewise, as they are closed they are deleted from the list.

Opening a Generic Window
Each type of window in VP has a specific window-opening function. `open_environment_window` opens an environment window, and `open_retina_window` opens a retina window, for example. What all of the specific window-opening functions have in common, though, is that they each begin by calling `open_window`. The purpose of `open_window` is to create a generic window that has no specific "type" or function — it returns a `Window` pointer to a basic window that is ready to be customized. Its prototype is listed below.

```
    Window *open_window(char *title, int corners[4]);
```

The first parameter specifies the title that is to be displayed in the title bar of the window. The second parameter is an array that contains the coordinates of the window's borders on the screen. The first two elements of the array are the screen x values of the window's corners, while the third and forth elements are the screen y values of the window's corners.

`open_window` starts by allocating memory for a `Window` structure, and then adds it to the global window list. Then it opens a window, configures the graphics — the display mode is configured to RGB double buffer mode M and queues window maintenance and mouse devices.

To draw a list other than `WinList` in which the order of drawing is not a consideration, use `draw_other_windows` as shown below.

```
draw_other_windows(NULL, subset);
```

This line of code would draw every window in the list `subset` whose `Window` structure address is not equal to `NULL`. Of course, there should never be a `NULL` entry in a window list.

3.2 The Message Window

The message window is dedicated to displaying text — primarily instructions and error messages. (At this time, the message window is limited to displaying a single line of text.) Only one message window is opened and it cannot be closed.

The message window is opened in `initialize` with a call to `open_message_window`, which is defined in `vpmain/window.c`. Its prototype is listed below.

```
Window *open_message_window(char *title, int corners[4]);
```

The window is tall enough to accommodate a line of text, and as wide as the display screen. The drawing function that gets assigned to the `draw_wintype` field of the message window's `Window` structure is `draw_message_window`.

The Message Stack
The message window uses a stack to maintain text that it needs to "remember". The stack is declared in `vpmain/message.c` as follows:

```
static Stack msg_stack;
```

`draw_message_window` simply draws the text that is at the top of the stack. New text is displayed by pushing it onto `msg_stack`. Similarly, popping the stack will replace the new text with the previous text. The functions used for editing the message stack are defined in `vpmain/message.c`; their prototypes are listd below.

```
void push_message(char *newmsg);
char *pop_message(void);
```

Both of the functions call `draw_message_window` to display the updated message. `pop_message` returns a pointer to the popped text.

3.3 The Environment Window

Environment windows display a two-dimensional rendering of three-dimensional object data. Three-dimensional graphical objects are stored in constructs called `Models`, and collectively they compose the program's graphical object environment. The models are stored in a single tree structure that is shared by all of the environment windows. Thus, each existing environment window will render the same group of models. The only difference between different environment windows is the view from which the models are beeing seen. The program places no explicit limit on the number of environment windows that can be open simultaneously.

Opening An Environment Window
An environment window is opened with a call to `open_environment_window`. The function is defined in `vpmain/window.c`; its protype is listed below.

open_window also initializes some of the members of the new Window structure: the window identifier (returned by winopen) is assigned to the wid field, the title (passed as the first argument) is assigned to the title field, and the modeltree and objects fields are set equal to NULL. open_window ends by returning a pointer to the new Window structure.

Drawing Window Lists

Each type of window has a corresponding function that draws that type of window. As an example, the lines

```
draw_retina_window(retwin);
swapbuffers();
```

would draw the retina window associated with the Window structure pointed to by retwin. Often though, it is necessary to redraw all of the windows without even knowing how many there are or what type they are. This can be done by calling the function draw_all_windows. Its prototype is shown below.

```
void draw_all_windows(void);
```

draw_all_windows, located in vp/src/vpmain/window.c, draws all of the windows in the global window list. It starts by saving the identifier of the current graphics window in the local variable originalwin. Then it traverses WinList, drawing each window with the lines

```
WINDOW(WinList)->draw_wintype(WINDOW(WinList));
swapbuffers();
```

WINDOW is a macro defined in vp.h that casts a List node's dat field to a pointer to Window. So for each Window structure in the global window list, draw_all_windows calls the window drawing function specified in the structure's draw_wintype field. Note that all of the window drawing functions must have the same parameter — a pointer to Window. After the entire list has been traversed, the current graphics window is reset to the identifier stored in originalwin, and control is returned to the calling function.

As there is no specific order in the global window list, and the node to which WinList points frequently changes, the order in which draw_all_windows draws the windows is uncertain. Sometimes though, it is desirable to consistently draw the same window first, as is the case when interactively moving a model or setting the view in an environment window. It is also sometimes desirable to draw a subset of the global window list WinList. The function draw_other_windows serves both purposes. Its prototype is listed below.

```
void draw_other_windows(Window *win, List others);
```

There are two key differences between this function and draw_all_windows. First of all, instead of traversing the list pointed to by WinList, it traverses the list specified by others. The second difference is that it draws all of the windows in the others list, except for the window specified by win. Consider the code listed below.

```
draw_environment_window(envwin);
swapbuffers();
draw_other_windows(envwin, viewlist);
```

In this example, the environment window specified by envwin would be drawn first. Then draw_other_windows would draw all of the windows in the circular list viewlist, except for the window specified by envwin.

```
Window *open_environment_window(char *title, corners[4]);
```

The function opens the window with a call to `open_window` and then tailors it into an environment window by setting the view and "creating" a model environment.

In `open_environment_window`, the window's matrix mode is set to `MVIEWING`. (This separates the projection matrix from the matrix stack that is used for view and modelling transformations.) Environment windows use a perspective projection

Setting The View
The initia

Drawing The Environment
Environment windows are drawn by the function `draw_environment_window`, which is defined in `vpmain/window.c`. Its protype is listed below.

```
void draw_environment_window(Window *w);
```

The argument `w` points to the `Window` structure of the environment window that is to be drawn. The function starts by setting the current graphics window to `w->wid` and clearing the window with the color specified in `w->bkgnd`. The z-buffer is also cleared. The code that sets the projection and the view and draws the tree of graphical models is straightforward:

```
perspective(w->fovy, w->aspect, NEAR, FAR);
inverthomomat(view.mat, w->cop.mat);
loadmatrix(view.mat);
scale_grid();
draw_model_tree(w->modeltree);
```

`perspective` sets the projection according to the window's vertical and horizontal fields of view. The view is set by loading the inverse transpose of `w->cop` onto the matrix stack. Then the reference grid is drawn centered at the world origin with a call to `scale_grid`. Finally, `draw_model_tree` draws the tree of graphical models.

3.4 The Retina Window

/** under reconstruction **/

4 Models

A three-dimensional graphical object model in VP is stored in a data structure of type `Model`. Models are collectively stored in a tree structure that is common to each environment window. The model itself is composed of a tree of one or more object primitives called *segments*. A model segment is stored in a data structure of type `Segment`, which specifies lists of vertices that make the polygonal faces of the segment. Models are created and added to the environment through a parser that reads model description files that conform to VP's model description grammar.

The Model Description Language
The model description language that VP employs is expressed in Backus-Naur Form in Figure 4.1 below.

```
  start → modlist eof
modlist → mdef modlist
          | ∈
   mdef → model id '{' xfrm seglist '}'
seglist → sdef seglist
          | ∈
   sdef → segment id '{' xfrm prim seglist '}'
   xfrm → tmat
          | xlist
          | ∈
  xlist → translate '(' num ',' num ',' num ')'
          | rotate '(' num ',' num ',' num ')'
          | scale '(' num ',' num ',' num ')' ';' xlist
          | ∈
   tmat → tm '(' '(' num ',' num ',' num ','
   num ')' ','
       '(' num ',' num ',' num ',' num ')'','

       '(' num ',' num ',' num ',' num ')'','

       '(' num ',' num ',' num ',' num ')')'';'
   prim → id '(' ')' ';'

     id → letter(letter|digit)*
    num → digit+
 letter → [a-zA-Z_]
  digit → [0-9]
```

Fig. 4.1. The grammar for VP's model description language.

The grammar allows for one or more models to be defined in a single file. Each model definition begins with the keyword `model`, which is immediately followed by the name that is to be assigned to the model. (See the grammar production for *mdef*.) Then transformations for the model's location and orientation can be specified, and finally the segment tree is defined. The segment tree is a group of one or more segment definitions. Each segment definition begins with the keyword `segment`, which is followed by the segment's name. Then the segment may be given a transformation, and the primitive is specified. The primitive is essentially the name of a text file that contains the vertex coordinates and face vertex lists. A primitive file must have the extension `.seg`, but as seen in this example, the extension is not used in the model description file. The following code is a definition of a cube model.

```
model cube_model
{
        segment cube_segment {
                cube();
        }
}
```

In this description, a one-segmented model named `cube_model` is created at the world origin. It is placed at the world origin because the identity matrix is assigned to models and their segments by default when no transformation is specified. The `cube()` primitive tells the parser to get the vertex and face information from the file `cube.seg`, which is shown below.

```
1 1 1
1 1 0
1 0 0
1 0 1
0 0 1
0 0 0
0 1 0
0 1 1
*
1 2 3 4
3 6 5 4
5 6 7 8
1 8 7 2
1 4 5 8
2 7 6 3
*
```

Each line before the first asterisk specifies the *x*, *y* and *z* coordinates of a vertex. Each line after the asterisk lists the index of each vertex that makes up a single face. `cube.seg` describes a unit cube.

Transformations may be described in one of two ways. One way is to use the matrix operators `translate`, `rotate` and `scale`. The alternative is to directly specify the complete transformation matrix. The two methods are exclusive — they cannnot be combined. (See the grammar production for `xfrm`.) The following example is will serve as an illustration:

```
model eyes
{
        tm ((0.7071, 0, -0.7071, 0),
            (0, 1, 0, 0),
            (0.7071, 0, 0.7071, 0),
            (0, 10, 0, 1));

        segment left_eye
        {
                translate(3.2, 0, 0);
                scale(0.44, 0.44, 0.44);
                sphere();

                segment cornea
                {
                        translate(0, 0, 0.7);
                        scale(0.25, 0.25, 0.25);
                sphere();
            }
        }

        segment right_eye
        {
            translate(-3.2, 0, 0);
            scale(0.44, 0.44, 0.44);
            sphere();

            segment cornea
            {
                translate(0, 0, 0.7);
                scale(0.25, 0.25, 0.25);
                sphere();
            }
        }
    }       /* model eyes */
```

The eyes model describes two "eyeballs" whose centers are 6.4 cm. apart. The keyword tm at the top of the model description indicates that the given matrix should be used as the model's coordinatè transformation. The tm matrix above will place the model eyes 10 cm. above the world origin, and the model will be rotated 45 degrees about the y axis.

Each eyeball is constructed with two instances of the sphere() primitive. (In this example, the segment file sphere.seg describes a sphere that is 5 cm. in diameter.) The segment left_eye is located at (3.2, 0, 0) with respect to the origin of the model. scale is used to transform the sphere() primitive to the proper size. *Note that* scale *has immediately local scope — it is not carried through the hierarchy as translation and rotation are.* left_eye has the nested segment cornea. Each nested segment is a descendent in the hierarchy. Therefore, cornea will be transformed with respect to left_eye. The cornea segment is a smaller sphere that intersects the front of the larger sphere to produce a more more reallistc eyeball. right_eye is exactly the same as left_eye, with the exception that it is located at (-3.2, 0, 0) with respect to the origin of the model.

VP's description language is similar to the QuickModel format developed by Alias Research. One major difference is that VP's model description language does not yet support spline curves and surfaces. Another difference is that QuickModel does not support hierarchical model descriptions, while VP does. The model description language is still under development - efforts are being made to give it spline curve and surface support, and to make it more compatible with the QuickModel format.

The Model *Definition*

The structure that specifies a three-dimensional graphical model is defined in vp/include/model.h. It is defined as type Model as shown below.

```
    typedef struct model Model;
    struct model {
        char   *name;       /* Unique model name. */
        Tree   treenode;    /* Tree node that contains model. */
        Flags  status;        /* Various status flags. */
        Xfrm   local;         /* Coordinate transformation. */
        Xfrm   global;        /* Coordinate transformation. */
        Tree   segtree;       /* Segments that compose the model.
        */
    };
```

The members of `struct model` are briefly described below.

> `name` — A unique name used to identify the model. The string in the `name` field is a concatenation of the model's name (as specified in its description file) and a numerical index that is assigned in a call to `name_model`.
>
> `treenode` — A pointer to the tree node in which this model is stored.
>
> `status` — Various status flags. At this time, the only flag actually used is `status.is_lit`. The model is rendered with as a solid if the `status.is_lit` flag is true at the time the model is being drawn. Otherwise, it is drawn as wireframe.
>
> `local` — A homogeneous transformation matrix that specifies the location and orientation of the model with respect to its immediate predecessor in the model tree.
>
> `global` — A homogeneous transformation matrix that specifies the model's location and orientation with respect to world origin.
>
> `segtree` — The tree of object primitives that compose the model.

The vertices and faces of the model are actually stored in the tree of segments specified by the `segtree` field of the `Model` structure. A segment is stored in a data structure of type `Segment`, which is defined in `vp/include/model.h`. The structure is shown below.

```
    typedef struct segment Segment;
    struct segment {

        char   *name;         /* Segment name */
        Xfrm   local;         /* Coordinate transformation */
        Xfrm   global;        /* Coordinate transformation */
        float  color[4];      /* Used when drawing as wireframe
        */
        int    nfaces;        /* Number of polygons in the
        segment */
        Face   *facetable;    /* Vertex lists for each face */
        float  *vertextable;  /* Vertex coordinates */
    };

    typedef struct face Face;
    struct face {
        int    nvertices;     /* Number of vertices in the face
        */
        int    *vtable;       /* List of the face's vertex
        indices */
        float  normal[3];     /* The face's surface normal */
    };
```

The members of `struct segment` are briefly described below.

`name` — The name of the segment as specified in the model description file.

`local` — A homogeneous transformation matrix that specifies the location and orientation of the segment with respect to its immediate predecessor in the segment tree.

`global` — Not yet implemented, this field will be the homogeneous transformation specifying the location and orientation of the segment with respect to world origin.

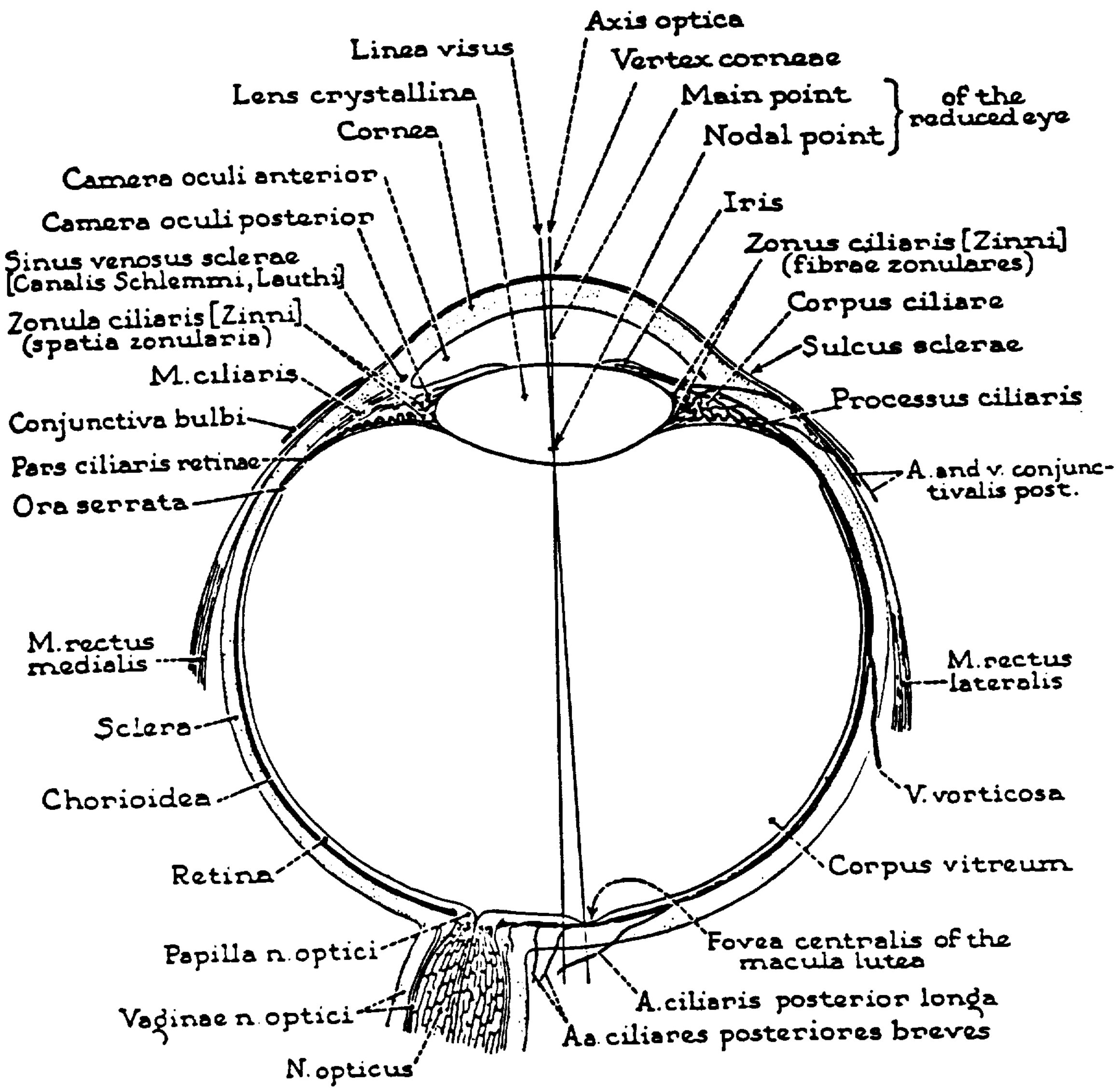
`color[4]` — Specifies the color of the segment. (Only applies when the segment is being drawn as a wireframe.)

`nfaces` — The number of faces in the segment.

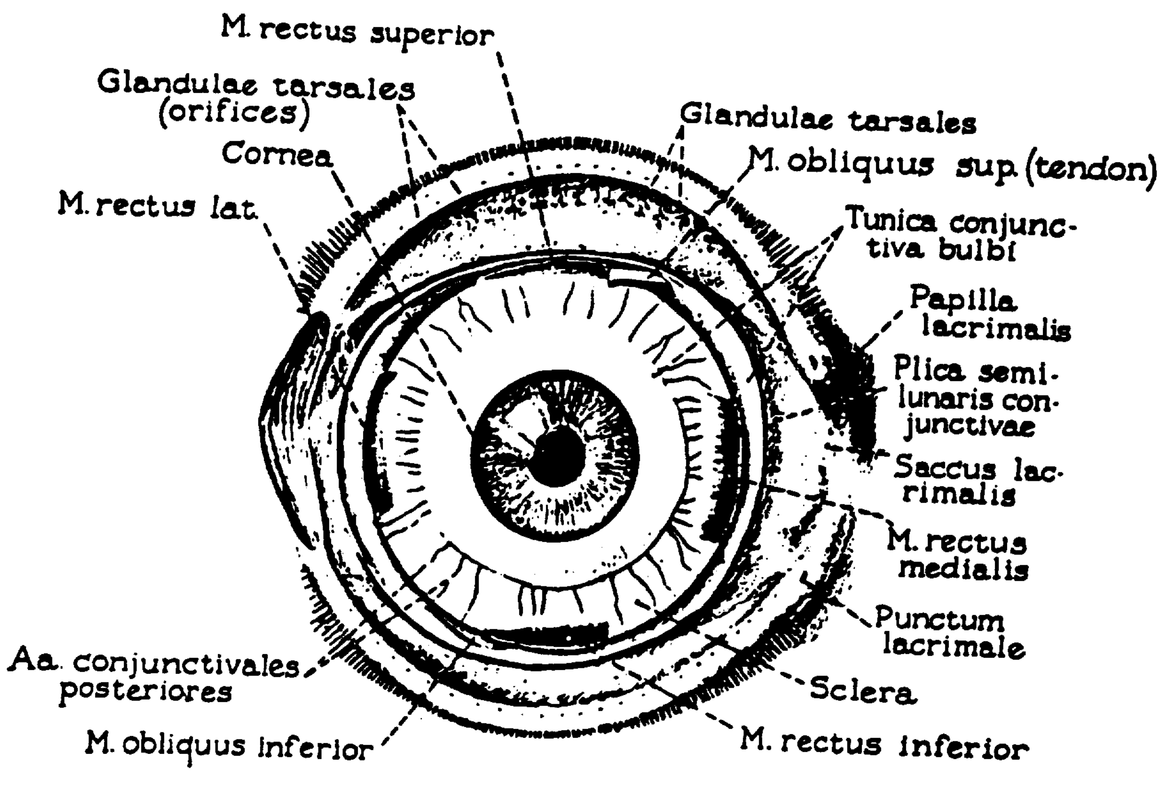
`facetable` — A table of `Face` structures. Each face in the segment has a structure that is pointed to by an entry in `facetable`. `Face` structures specify the face's surface normal and the list of vertices that make up the face.

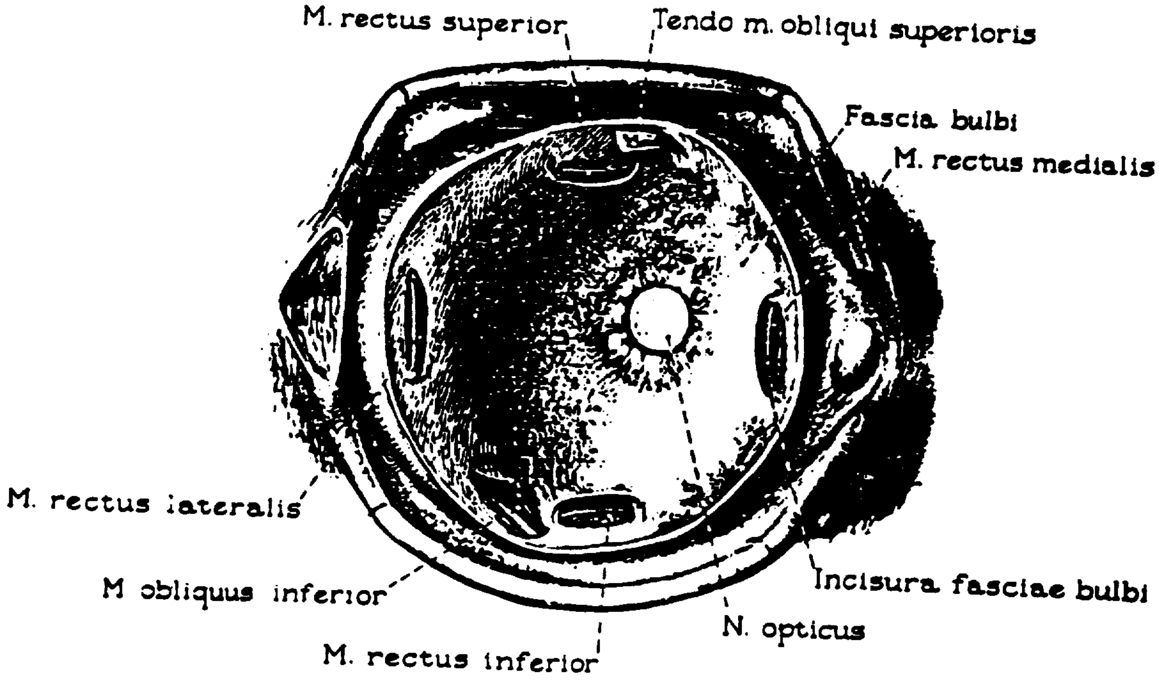
`vertextable` — A table of the `x`, `y` and `z` coordinates of each vertex in the segment.

APPENDIX D — EYE PHYSIOLOGY



Section through the right eye: schematic (after Spalteholz).





Eye: conjunctiva, fascia, and muscles.

In the upper figure the lateral angle has been cut and the eyelids drawn apart; the conjunctiva has been incised and peeled away to expose the ocular muscles. In the lower figure, the fascia is exposed by removal of the ocular bulb (the optic nerve, cut). (From Warren: Handbook of Anatomy, Harvard University Press.)

Annex I

Army-NASA Aircrew/Aircraft Integration Program: Phase IV

A^{3}I

Man-Machine Integration Design and Analysis System (MIDAS)
Software Detailed Design Document

Aerodynamics / Guidance Module

prepared by

Alex Chiu

Table of Contents

Table of Contents

MAN-MACHINE INTEGRATION DESIGN & ANALYSIS SYSTEM (MIDAS) SOFTWARE DETAILED DESIGN DOCUMENT PHASE IV:

AERODYNAMICS / GUIDANCE MODULE

1.0 INTRODUCTION

1.1 IDENTIFICATION OF DOCUMENT

This is the Software Product Specification for the aerodynamics/guidance module of the MIDAS Software System.

1.2 SCOPE OF DOCUMENT

This document is primarily focused on the enhancement of the aerodynamics/guidance module (AGM) developed in the Phase IV development stage. The body of AGM remains the same as that of Phase III. To understand the theory of AGM the user is urged to refer to the Phase III AGM documentation. AGM was originally developed and implemented in Fortran by Analytical Mechanics Associates (AMA). For detailed descriptions of the AMA developed AGM, the reader is referred to the AMA Report 252-3 prepared by Phaetak and Tran listed in Section 2.1. The reader is assumed to be familiar with the basic concepts of aerodynamics and control theory, UNIX, Fortran, and C.

1.3 PURPOSE AND OBJECTIVES OF DOCUMENT

The purpose of this document is to serve as a technical reference focusing on the Phase VI enhancement of AGM. This document contains high level information as well as implementation detail useful to readers who need to understand the Phase IV AGM module.

2.0 RELATED DOCUMENTATION

2.1 APPLICABLE DOCUMENTS

The following documents are referenced herein and are directly applicable to this volume:

Army-NASA Aircrew/Aircraft Integration Program (A^{3}I) Software Detailed Design Document: Phase III, Contractor Report 177557, NASA Ames Research Center, Moffett Field, California 94035-1000, June 1990.

A. Gessow, G. C. Myers, Jr., *Aerodynamics of the Helicopter*, Federick Ungar Publishing Co., New York, December, 1952

M. S. Lewis, E. W. Aiken, *Piloted Simulation of One-On-One Helicopter Air Combat at NOE Flight Levels*, USAAVSCOM Technical Report 85-A-2, NASA Ames Research Center, Moffett Field, California, April, 1985

Anil V. Phatak, Hien H Tran, *A^3I Autopilot/Guidance Program Homing/Path Guidance with Turn-Straight-Turn Option* (Version TGUIDAP2) AMA Report 252-3, Mountain View, California, March, 1988

Silicon Graphics Inc., "IRIS-4D Series Fortran Programming Language", Version 1.0, Mountain View, California, 1988

3.0 CONCEPT

3.1 DEFINITION OF THE AGM MODULE

The Phase IV AGM module consisted of two portions. The guidance portion computes the controls required to steer the aircraft to the desired location. The aerodynamics portion computes the new position after having applied the controls computed by the guidance portion for certain time interval, the size of a tick in Phase IV.

3.1.1 Purpose and Scope

The Phase IV AGM module served the purpose of providing a high fidelity aerodynamics model with guidance capabilities in support of MIDAS in conducting research activities to address human factors issues. The AMA-developed AGM is linear and partly decoupled, in the sense that the collective control has no effect on the yaw, pitch, or roll movement. The wind effect on the helicopter behavior was not taken into consideration as well. The model has been enhanced by AMA with the addition of a Turn-Straight-Turn guidance scheme. The enhanced AGM was ported to the A^3I's Silicon Graphics Workstations in previous phases.

3.1.2 Goals and Objectives

The AMA-developed AGM represents a rather generic helicopter aerodynamics model with guidance capability. The Phase IV goal of AGM was to improve the fidelity of the role it plays in the MIDAS simulations by enhancing the AMA-developed AGM to possess the terrain knowledge. The following objectives needed to be achieved. First, a new piece of DMA terrain with the necessary features specified by the Symbolic Operator Model needed to be rendered using the techniques developed by a former A^3I graphics staff member in previous phases. Then, the new piece of terrain needed to be integrated with AGM. Finally, it needed to provide capabilities to calculate the terrain elevation at a given location (x,y), to check line of sight for two given points, and to introduce intermediate waypoints when necessary. It has to provide the Symbolic Operator Model with the controls computed by the guidance portion, and drive the displays of the VEST Pilot, VEST Copilot, and Views modules.

3.1.3 Description

For AGM to provide the Symbolic Operator Model with the computed controls, its structure was modified from closed loop to open loop as described below. Each cycle begins with the guidance portion computing the controls based on the current position and the next waypoint, but does not pipe the computed controls directly into the aerodynamics portion. Instead, the controls are first sent across the network to the Symbolic Pilot Model, which accepts or rejects the controls based upon his available resources. The Symbolic Pilot Model sends a message back to AGM indicating whether the computed controls will actually be used. Based on this message, the aerodynamics portion computes the new position and orientation, and the current cycle is completed.

To integrate the terrain with AGM, the AGM module was enhanced to be capable of reading the terrain data and storing them in a proper place for future access.

The controls actually used and the aerodynamic parameters were sent across the network to drive the graphics displayed by the VEST Pilot, VEST Copilot, and Views.

The network interfaces required for AGM to interact with other MIDAS modules were provided by the Communication module.

3.2 USER DEFINITION

The AMA-developed AGM may or may not be adequate, depending on the requirements of your application. It was inadequate when used in MIDAS due to the lack of the capabilities to avoid flying into terrain. Most of the aerodynamics models behave like the AMA-developed AGM. Therefore, the enhancement of the AGM model to possess the terrain knowledge accomplished with MIDAS might be useful to users who attempt to accomplish similar task. The capabilities developed could also aid the vision model in performing the line-of-sight checking.

3.3 CAPABILITIES AND CHARACTERISTICS

The enhanced AGM is capable of calculating the elevation for any given point of the terrain. This capability is required to drive the altimeter. The capability of checking the line of sight for any two points can be used in assisting the pilot in carrying out his mission. The introduction of intermediate waypoints prevents the helicopter from flying underground. The intermediate waypoints are introduced prior to the commencement of simulation based on the input waypoints specified by the pilot model and the terrain data. When the simulation begins the helicopter will traverse through the waypoints contained in the expanded waypoint list .

3.4 SAMPLE OPERATIONAL SCENARIOS

The following sample operational scenario gives the reader a flavor of how to run the enhanced AGM.

The input required by the AMA-developed AGM consists of a set of waypoints. Each waypoint contains not only the coordinates (x,y,z) but the speed and heading of the helicopter at that point as well. The input for the enhanced AGM requires a slightly different form. The user needs to select a set of two dimensional waypoints (x,y) from the terrain and specifies the desired altitude above terrain at each of these points. Also, the desired speed and heading of the helicopter at each of the selected waypoints need to be specified.

Now the user runs the AGM module. It first reads the user-specified input data. Then it calculates the terrain elevation for each of the selected waypoints and adds to it the altitude above terrain specified at this point to yield the absolute altitude. The line of sight checking is then performed for each pair of consecutive waypoints. If the line of sight for a pair of consecutive waypoints is not clear proper intermediate waypoints will be introduced. Proper speeds and headings need to be specified at the introduced waypoints as well. The waypoints contained in the new list will become the ones the helicopter has to traverse.

The initialization process always positions the helicopter at the very first waypoint. The guidance portion suggests to the Symbolic Operator Model the controls it computed to steer the helicopter to the next waypoint. The Symbolic Operator Model notifies AGM whether the pilot has available resources to perform the suggested control movements. If the resources are not available, controls used in the previous cycle will be used in the current cycle. The aerodynamics portion then computes the new position and orientation based on the controls actually used. Then the controls and aerodynamic parameters are sent across the network to drive the displays. This process continues until the simulation is over.

4.0 REQUIREMENTS

4.1 REQUIREMENTS APPROACH AND TRADEOFFS

The Phase IV AGM module needed to fulfill the following software requirements. First, it needed to be able to read the data of any DMA terrain and store them in a proper place for easy access. Secondly, efficient schemes needed to be devised (1) to interpolate terrain elevation at a two dimensional point based on the known elevations at vertices, (2) to check line of sight, and (3) to introduce intermediate waypoints. Finally, capabilities for exchanging information with other MIDAS modules needed be provided.

4.2 EXTERNAL INTERFACE REQUIREMENTS

The Phase IV MIDAS is a truly distributed system. Its modules are distributed over different computer platforms. To allow the AGM module to exchange information during a simulation, four interfaces need to be developed. These four interfaces allow it to communicate with the VEST Pilot, VEST Copilot, Views, and the Symbolic Operator Model. These interfaces are provided by the Communication module. Data structures are designed to contain the controls and aerodynamic parameters. For more detailed information, refer to Annex J, Communication Module.

4.3 REQUIREMENTS SPECIFICATION

4.3.1 Process and Data Requirements

The input waypoint list required by the Phase IV AGM module was described in section 3.4. The two dimensional coordinates x and y, altitude above terrain, speed, and heading are all represented by floating point numbers. For data structures which hold the controls and aerodynamic parameters refer to the header file dp.h, as described in Annex J, Communication Module.

4.3.2 Performance and Quality Engineering Requirements

The AGM module is expected to exhibit reasonably good performance so that it can fit in the MIDAS simulations. The computation of controls and the integration of the governing equations over a tick size should not become the bottleneck when participating in the simulation. Portability is a major consideration during the development stage. The network communication between the AGM module and other MIDAS modules should be bi-directional and reliable. Most importantly, it is necessary to keep the helicopter from flying underground.

4.3.3 Implementation Constraints

The AGM module was developed on the government-furnished equipment, which includes the Symbolics and Silicon Graphics IRIS workstations. Because the SGI workstations provide better Fortran environment for scientific computing, the Fortran AGM module was designated to run on the SGI workstations. The network interfaces available on the SGI workstations are written entirely in C. Therefore, interfaces are needed for the Fortran AGM module to communicate with the C network interface. Fortunately, the SGI workstations do provide ideal tools -- the data blocks and interlanguage calls to implement the C-Fortran and Fortran-C interfaces.

5.0 DESIGN

5.1 ARCHITECTURAL DESIGN

Figure 1 shows the top level configuration of the Phase IV AGM module along with the MIDAS modules it directly interacts with, such as the Symbolics Pilot Model, VEST Pilot, VEST Copilot, and Views. Conceptually, the AGM module consists of four major subcomponents : terrain, guidance, aerodynamics, and network interfaces.

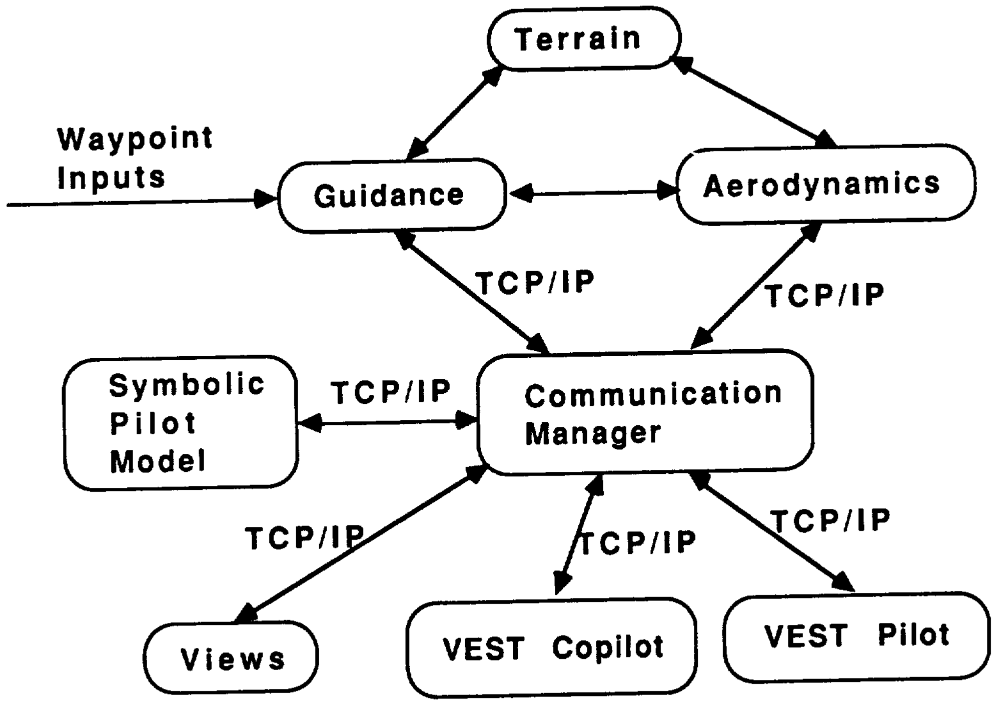


Fig. 1

Figure 1. Aerodynamics/Guidance Module Configuration

The network interfaces provided by the Communication module were implemented based on the TCP/IP protocol. The C-Fortran and Fortran-C interfaces are used in facilitating the communications between the guidance/aerodynamics and the network interfaces. As mentioned in Annex J, Communication Module, the AGM module communicates with other MIDAS modules through the communication manager. Data structures are defined to hold the controls and aerodynamic parameters and are passed around among the MIDAS modules.

5.1.1 Design Approach and Tradeoffs

It was decided to develop the terrain-related capabilities from scratch, because it only needs simple mathematics, knowledge of UNIX and C programming language, and C-Fortran interfaces. During the development phase of these capabilities, modularity, portability, and easy maintenance are the major considerations. The architecture should be flexible enough to allow the user to replace the MIDAS terrain with his/her desired DMA terrain.

The replication of the terrain in the AGM module was a tradeoff between speed and memory. In Phase IV and the previous phases, the terrain was part of the Views module. The AGM module could have fetched the terrain data across the network from the Views module at the expense of busier network traffic. However, as a result of the advances of the memory technology, memory can be acquired at a fairly reasonable cost. It was decided to enhance the AGM module to include the terrain for easy access. The integration of terrain with AGM represents an effort toward improving the AGM module to aid in conducting human factors research activities.

The acquisition of a parallel computer can change the entire structure of MIDAS. For example, the replication of terrain can be eliminated. The global memory space available on parallel computers is an ideal place to store intact data bases, such as terrain. And all modules running on the parallel computer processors have access to the global memory space without appeal to the Ethernet. Therefore, the network communications across the Ethernet will be totally eliminated and the performance can be expected to improve drastically. Of course, smart schemes for modules to store/fetch data into/from the global memory space need to be developed.

5.1.2 Architectural Design Description

To meet the requirements stated earlier, the architecture of the AGM module is decomposed into the following subcomponents. The first subcomponent performs the terrain-related tasks. The tasks include reading the AGM input file and the terrain data, storing the coordinates of vertices in an array of data structure, and introducing proper intermediate waypoints, if necessary. The expanded waypoint list is input to the second subcomponent, the guidance subcomponent, which computes the controls based on the current helicopter position and the next waypoint. The controls include the cyclic, collective, and pedal movements. The third subcomponent is the network interface furnished by the Communication module to facilitate the communications with the Symbolic Pilot Model, VEST Pilot, VEST Copilot, and Views. The network interface sends the control values across the network to the Symbolic Operator Model. The pilot makes a decision on the control movements based on the resources available. The controls actually used are then piped into the fourth subcomponent, aerodynamics, which integrates the equations of motion to yield new position and orientation. The network interface sends the controls actually used and the aerodynamic parameters across the network to drive the graphics modules.

5.1.3 External Interface Design

The external interfaces required for the AGM module to communicate with the rest of MIDAS system were provided by the Communication module. Refer to Annex J, Communication Module, for details.

5.2 DETAILED DESIGN

The Phase IV AGM software was designed to have four code units. These four code units map perfectly with the subcomponents described earlier. The code units are described in detail in later sections.

5.2.1 Detailed Design Approach and Tradeoffs

The design of the AGM module must take into consideration that the MIDAS system allows the user to change the simulation scenario. For example, the user might want to use a particular piece of terrain, a different guidance model, or a different aerodynamics model for his particular mission. The design of the AGM module provides the flexibility to allow the user to make these changes easily.

The subcomponents of the AGM module were implemented in a truly modular fashion and the interfaces among them are very friendly. Since most of the guidance and aerodynamics models are written in Fortran, an interface was implemented for the guidance and aerodynamics models to interact with other subcomponents written in C. The interface was written based on the interlanguage calls and data blocks available on the SGI workstations. Refer to "IRIS-4D Series Fortran Programming Language" listed in section 2.1.

Because the AGM module needs to communicate with the communication manager, the network connection process was included in the AGM software.

5.2.2 Detailed Design Description

5.2.2.1 Compilation Unit Design and Traceability to Architectural Design

The vertices extracted from the DMA DTED tapes were grouped to form polygons. Usually, a polygon has three vertices. A polygon may have four vertices if they are coplanar. Because the terrain adopts polygonal representation, a data structure is defined for a polygon in a header file which contains the number of vertices and the coordinates of each vertex. An array of this data structure is declared in the main function that holds the information of polygons that make up the entire terrain. The array is declared to be global so that it is accessible to both the guidance and aerodynamics subcomponents.

The run-time AGM module consists of the following processes : initialization process, connection process, guidance computation, aerodynamics computation, data passing. The main C function was written which reflects the control flow of these processes. Prior to executing the command to run the AGM module, the communication module should have proceeded to the stage ready to accept requests for connection.

The initialization process does the following things. It opens the file that contains the polygonal representation of terrain, reads the data, and stores the data in the global array of data structure. It then reads the AGM input file (see AMA Report 252-3 for details) and introduces appropriate intermediate waypoints, if necessary.

When the initialization process is completed, it proceeds to the network connection stage. At this point, the communication manager should be waiting to accept connection requests. The AGM module sends connection request to the communication manager. When the AGM module has been successfully connected to the communication manager, it waits for the first incoming message from the communication manager.

The AGM module then starts the guidance-aerodynamics cycle. A new important capability the Phase IV AGM module has is to allow the pilot to change the route of flight during the simulation. If the AGM module receives a new route of flight, it introduces a new set of intermediate waypoints. The new waypoint list will replace the old list and the helicopter now has to traverse the new set of waypoints. At the end of each cycle, the controls and aerodynamic parameters are sent across the network to drive the graphics modules.

5.2.2.2 Detailed Design of Compilation Units

The following brief description gives the reader a flavor of the algorithms underlying the terrain-related capabilities.

The elevation of a terrain point (x, y) was calculated as follows. First, identify the polygon the point is on. Then, obtain the plane equation for the polygon from its vertices. Finally, solve for z by substituting x and y into the plane equation.

To check the line of sight for two given points (x1, y1, z1) and (x2, y2, z2), first project the 3d line segment connecting (x1, y1, z1) and (x2, y2, z2) onto the horizontal x-y plane, i.e., form the 2d line segment (x1, y1) and (x2, y2). Identify the set of polygons covered by the 2d line segment. The line of sight is not clear if any polygon in the set lies above or intersects the 3d line segment, and is clear if the polygons in the set all lie below the 3d line segment. Here, a line segment is said to intersect a polygon if the intersection point is part of the line segment and within the polygon.

The algorithm used in checking the line of sight is also used in the process of introducing intermediate waypoints. Suppose we are given a line segment defined by the starting point P1 (x1, y1, z1) and the end point P2(x2, y2, z2). First, identify the set of polygons covered by the projected 2d line segment (x1, y1) and (x2, y2). Cycling through the polygons in the set, if a polygon intersects the 3d line segment (see above for its definition), an intermediate waypoint needs to be introduced. The introduced waypoint will be the intersection point elevated a certain height in the z direction. At this stage, purge those polygons that have been processed off the polygon set. The newly introduced waypoint and P2 define a new line segment. The waypoint introduction process for the given points P1 and P2 continues with the new line segment and the purged polygon set, until the line of sight is clear.

5.2.2.2.1 Detailed Design of Initialization Unit

The following C functions and Fortran subroutines are involved in this unit: read_ter_data(), openfiles_(), tdinput_(), init_dp(), and get_dp_types().

The interlanguage call technique available on the SGI workstations is used in this unit for a C function to call Fortran subroutines. Those procedures with names ending with underscore are Fortran subroutines. Basically, read_ter_data() reads the terrain polygons. openfiles() and tdinput_() read the input data and invokes functions contained in chk_los.c to introduce intermediate waypoints. Init_dp() initializes the AGM-related data structures defined in the data pool. Get_dp_types() contains the message type and the source and destination modules for all AGM-related messages.

5.2.2.2.2 Detailed Design of Connection Unit

Establish_network() is the major function involved in this unit. This function follows the standard TCP/IP network connection procedure. It involves creating sockets, binding port numbers and/or network addresses to sockets, and sending connection request to the communication manager. The socket number returned from successful connect call is used for later subsequent data transmission.

5.2.2.2.3 Detailed Design of Guidance and Aerodynamics Units

These two units are described in detail in the AMA Report 252-3.

5.2.2.2.4 Detailed Design of Data Passing Unit

This unit contains functions, which can be found in the file network.c, to perform the following tasks.

* Update the related data structures with the guidance-computed controls and aerodynamic parameters.
* Send the updated data structures across the network to modules that need them.
* Detect incoming messages on the socket.
* Retrieve incoming messages and respond to them.

5.2.3 External Interface Detailed Design

The external interfaces are provided by the Communication module. Messages that contain information of the route of flight, controls, and aerodynamic parameters are defined in the data pool header file dp.h and get passed during the simulation. Described herein are these messages.

Three control-related messages are defined. One is to hold the demanded collective, pedal, and x- and y-cyclic controls, i.e., those computed by the guidance subcomponent. The second one is to hold the decision made by the pilot model on the use of demanded controls. The third one is to hold the controls actually used.

Messages are also defined to hold waypoint and route of flight information. Different waypoint data structures are defined for use by different modules. Same holds for the route of flight and the aerodynamic parameters. See dp.h for more detail.

5.2.4 Coding and Implementation Notes

The compiler options used included -G 8, -Olimit 1024, -lm, -lsun, -lbsd, and -lc_s.

6.0 USER'S GUIDE

6.1 OVERVIEW OF PURPOSE AND FUNCTIONS

The AGM module is a linear decoupled aero/guidance model. It has been tested and accepted by the aerodynamics community. The ICAB Simulator at Ames used the model to simulate the helicopter aerodynamics. However, it by no means satisfies missions that require aggressive maneuvers. The lack of the nap-of-the-earth capability is considered a major drawback. It also lacks the terrain-following capability.

The integration of the terrain into the AGM model and the development of the capabilities described earlier represent a step toward improving the its fidelity. The line-of-sight capability can be integrated with the vision and pilot models to aid them in performing their tasks.

6.2 INSTALLATION AND INITIALIZATION

The default setup designates the AGM module to run on Starfish (IRIS/4D 220). Prior to executing the command to run the AGM module, the communication module should have progressed to the point ready for accepting connection requests. When the message indicating the communication module is ready appears on the screen of the workstation on which it runs, issue the AGM command "run_agm". And this is all you need to do! If the user runs an a participating module prior to the point the communication module is ready, the module will exit after a number of attempts to connect to the network server. The AGM module will proceed in the way described earlier.

6.3 STARTUP AND TERMINATION

The required input data are clearly described in AMA Report 252-3. Once this file is prepared, issuing the command "run_agm" on the designated workstation is all you need to do to run AGM. When the simulation has progressed to the end of the duration specified by the user, the simulation executive notifies all modules of the termination.

No special procedures are provided when it encounters abnormal conditions. In such cases, just re-run the simulation.

6.4 ERROR AND WARNING MESSAGES

Certain check points were included in the connection unit to ensure successful connection between the AGM module and the communication module. Also, the waypoints are also checked against the terrain boundaries. If any waypoint is out of range, a warning message appears.

6.5 RECOVERY STEPS

It is suggested that the user follow the User's Guide section to restart the AGM module if any abnormal situation occurs.

7.0 NOTES

7.1 LESSONS LEARNED

During the development stage of the Phase IV AGM module, the developer learned that the integration of the terrain with the AGM model paved the way for future development of the terrain-following capability. It needs to be determined whether the model currently used can meet the mission requirements in future phases, as the pilot model developer has complained of the inadequacy, relative to the mission complexity, of the current model for two phases.

7.2 FUTURE DIRECTIONS

To improve the AGM module, the following things might be worth pursuing. It needs to be determined whether to develop desired capabilities such as NOE and terrain-following capabilities or locate existing models which already have these capabilities. The model with NOE capability used by another task at Ames sounds attractive and is worth exploration.

Annex J

Army-NASA Aircrew/Aircraft Integration Program: Phase IV

A^3I

Man-Machine Integration Design and Analysis System (MIDAS)
Software Detailed Design Document

Communications Module

prepared by

Alex Chiu

Table of Contents

Table of Contents

MAN-MACHINE INTEGRATION DESIGN & ANALYSIS SYSTEM (MIDAS) SOFTWARE DETAILED DESIGN DOCUMENT PHASE IV:

COMMUNICATIONS MODULE

1.0 INTRODUCTION

1.1 IDENTIFICATION OF DOCUMENT

This is the Software Product Specification for the communication module of the MIDAS Software System.

1.2 SCOPE OF DOCUMENT

This document describes the requirements and detailed design of the phase IV MIDAS communication module. The reader is assumed to be familiar with the Genera and UNIX environments, Lisp and C programming languages, TCP/IP protocol, and basic concept of Local Area Network. Familiarity with previous phases of development is definitely helpful, but not required.

1.3 PURPOSE AND OBJECTIVES OF DOCUMENT

The purpose of this document is to serve as a technical reference focusing on the integration of MIDAS modules achieved in phase IV. This document provides high level information to any reader who only needs a top level understanding of the communication module. It also contains implementation detail useful to users involved in modifying the communication software to meet their needs.

2.0 RELATED DOCUMENTATION

2.1 APPLICABLE DOCUMENTS

The following documents are referenced herein and are directly applicable to this volume:

Silicon Graphics Inc., "IRIS-4D Series Communications", Version 1.0, Mountain View, California, 1988

Symbolics Genera 7.* Manual, Vol. 5, "Streams, Files, and I/O"

Symbolics Genera 7.* Manual, Vol. 9, "Networks"

2.2 INFORMATION DOCUMENTS

3.0 CONCEPT

3.1 DEFINITION OF THE COMMUNICATION MODULE

The phase IV communication module was defined to be the one responsible for providing reliable network services to facilitate message passing between the MIDAS modules distributed over various computers.

3.1.1 Purpose and Scope

The Phase IV development of MIDAS focused on the integration of modules so that simulations could be carried out to show quantitative (loading and timelines) and qualitative (equipment design characteristics) differences between the Apache A and Apache Longbow for a common set of mission tasks. The purpose the communication module served in phase IV was to devise viable schemes to integrate various model-based MIDAS modules and ensure successful simulation. The schemes devised are intended to be general and adaptive so that they can be applied to achieve integration for similar type of problems with minimum modification.

3.1.2 Goals and Objectives

The goal of the phase IV communication module was to develop a framework for large scale distributed system integration so that MIDAS modules could be integrated and simulations could be performed to address important human factors research problems. The framework is intended to yield a modular environment that eases both the task of integrating the model-based MIDAS modules and the job of maintaining and expanding the MIDAS system.

The objective of the phase IV communication module was multifold. First, appropriate methods needed to be devised to coordinate activities of MIDAS module distributed over different workstations. Secondly, capabilities should be provided for MIDAS modules to exchange information during a simulation.

3.1.3 Description

Conceptually, the communication module was designed, as shown in Figure 1, to network all MIDAS modules together so that messages can be passed around between them. Needless to say, it has to assure that the receiving side receives precisely what the sending side sends.

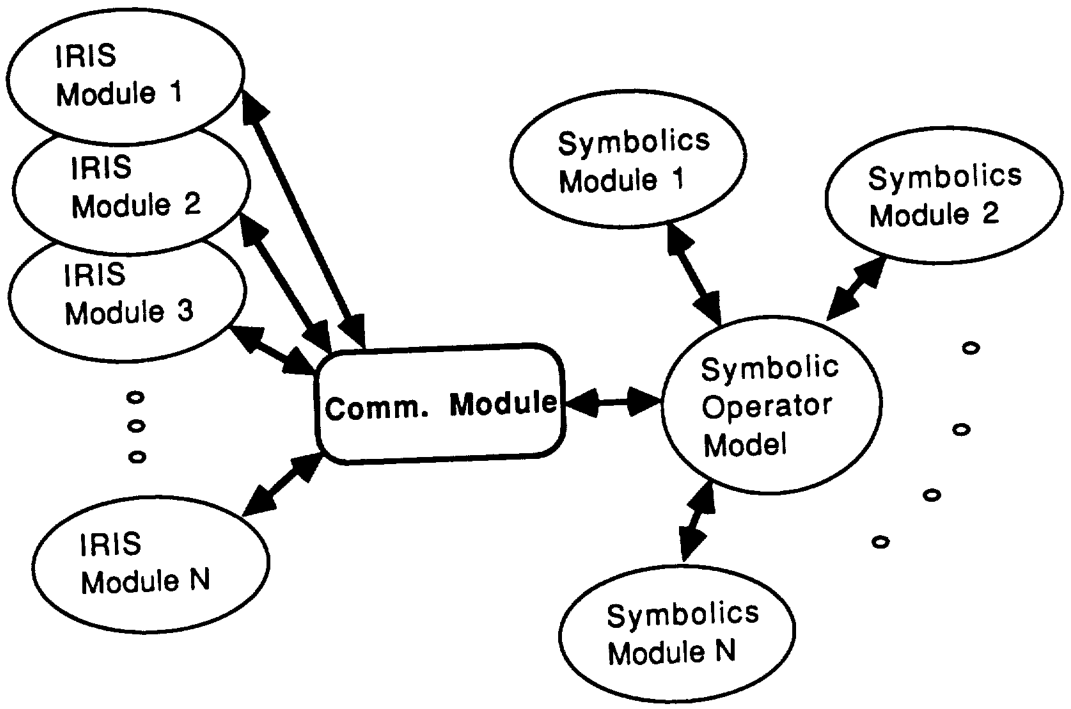


Fig. 1

Figure 1. Communication Module Overview

Because of the way the Symbolics modules were configured, as described in the documentation for the Scheduler and the Task Loading Model, the Symbolic Operator Model was the only Symbolics module that directly interacted with the Communication module. The other Symbolics modules designed to interface only with the Symbolic Operator Model. Therefore, the Communication module needed to provide sufficient links, as shown in Figure 1, to allow any MIDAS module to interact with the rest of the MIDAS system.

3.2 USER DEFINITION

The configuration and tools developed in this module are useful to designers who need to perform large scale system integration of various model-based modules distributed over different platforms. Also, users who need to write communication applications software to facilitate bi-directional message passing between the workstations like those used in the MIDAS system need not reinvent the Wheel.

3.3 CAPABILITIES AND CHARACTERISTICS

The architecture, to be described in later sections, contrived for use in phase IV can benefit the designer in many ways. It yields a very modularized environment which eases the job of maintaining and expanding the system. It also serves as a convenient mechanism for the designer to add/delete component modules to/from the existing design. Simulations can be carried out so that the designer can compare the results for different designs.

The communication module is capable of bi-directional transmission of character strings and numeric data between the IRIS workstations, between the Symbolics workstations, and between an IRIS workstation and a Symbolics workstation. This module runs under Genera 7.2 (in fact, it also runs under Genera 8.0) and UNIX System V with Berkeley extension BSD 4.3. Modification may be required for the module to run under different versions of Genera or UNIX/BSD. The module also provides the capability to allow the user to select participating modules prior to the commencement of a simulation.

3.4 SAMPLE OPERATIONAL SCENARIOS

The following sample operational scenario gives the reader some flavor how a simulation runs.

Prior to running a simulation, the user selects the participating modules and designates an appropriate workstation for each of them to run on by editing a text file. Also specified in the text file is the duration of the simulation in terms of ticks. Let us assume both the text file and the communication module reside on the same workstation, IRIS 4D 220, say. Suppose the Symbolic Operator Model and the Aero/Guidance model were selected to participate in the simulation and were designated to run on the Symbolics 3675 and the IRIS 4D 220, respectively.

The user now brings up the communication module. It first reads the text file to get the duration of the simulation and the participating modules. Then, the connection process of establishing necessary communication channels is initiated. In phase IV, only one stream was used in each channel for message passing between the communication module and each of the participating modules. The communication module progresses to a waiting status ready for accepting requests for connection from the participating modules. While it is waiting, the user brings up the participating modules on their designated workstations. The first thing they do is issue request for connection to the communication module. The connection process continues until all participating modules have been successfully connected.

When the connection process is completed, the communication module broadcasts tick zero to all participating modules. Now is the time for each participating module to execute its initialization process, including sending out input data needed by other modules. In our sample scenario, the Symbolic Operator Model sends the flight path specified in terms of waypoints to the Aero/Guidance model; each waypoint contains a terrain point (x, y) and the associated parameters at (x,y) : altitude above terrain, airspeed, and heading. When a participating module is finished with initialization, it sends an initialization-done flag to the communication module. When the communication module has collected all initialization-done flags, it increments the tick by one and broadcasts the incremented tick to all participating modules.

The simulation scenario gets updated from this point on. Each participating module progresses for the duration of a tick size and updates the associated data structures at the end of the tick. When next tick begins, the participating modules exchange the updated data structures. The simulation continues in this way for as many ticks as the user specified in the text file.

4.0 REQUIREMENTS

4.1 REQUIREMENTS APPROACH AND TRADEOFFS

The phase IV communication module needed to fulfill the following software requirements. First, it needed to serve as the control center which ensured that all participating modules were coordinated during a simulation. Secondly, it was expected to function as the message routing center. Thirdly, capabilities for exchanging information between MIDAS modules distributed over various workstations should be provided. Finally, it should yield a pleasant environment for integrating existing MIDAS modules distributed over different platforms.

4.2 EXTERNAL INTERFACE REQUIREMENTS

To allow the MIDAS modules to exchange information during a simulation, three interfaces need to be developed, taking into consideration the configuration of the Symbolics modules as described in section 3.1.3. The first interface is to allow an IRIS module to exchange information with the rest of the IRIS world. The second interface is to allow a Symbolics module to exchange information with the rest of the Symbolics world. The third interface is to link the Symbolics world with the IRIS world so that every module can exchange information with other MIDAS modules. A nice feature, not requirement, for the external interfaces to possess is the capability to detect incoming messages.

4.3 REQUIREMENTS SPECIFICATION

4.3.1 Process and Data Requirements

The phase IV communication module adopted a very simple way for the user to specify the input data necessary to run a MIDAS simulation. The required input data include the duration of the simulation in terms of ticks, the participating modules, and the workstations they run on. The specification was done by editing a text file prior to the commencement of a simulation. When the command to run the simulation is executed the communication module first reads content of the input file and then triggers the simulation process.

The input data do not have to be specified through a text file as described above. Windows, menus, icons, and dialog boxes could be used to achieve the same purpose.

4.3.2 Performance and Quality Engineering Requirements

The communication module is required to provide reliable network for bi-directional message passing between the MIDAS modules. The simulation speed is required to be interactive, not necessarily real-time. Portability is also a major consideration during the development stage.

4.3.3 Implementation Constraints

All MIDAS modules were developed on the government-furnished equipment, which includes the Silicon Graphics IRIS workstations and the Symbolics workstations. Because the communication module has to network these modules, it was implemented based on the network-related functions available on these workstations.

5.0 DESIGN

5.1 ARCHITECTURAL DESIGN

As mentioned earlier, the phase IV MIDAS system can be thought of as composed of the Symbolics world and the IRIS world. Figure 2 shows in detail the phase IV MIDAS configuration and the decomposition of the communication module into subcomponents. The Symbolics world consisted of the Operator Model, Equipment Model, Task Loading Model, and Scheduler. The IRIS world displayed the VEST Pilot View, VEST Copilot View, and World View. Because the IRIS workstations provide better Fortran environment, the Aero/Guidance Model (AGM) is included in the IRIS world.

Conceptually, the communication module was designed to consist of five major components : simulation executive, communication manager, data pool, and I-S and S-I data transceivers. They constitute an ideal architecture for performing large scale distributed system integration.

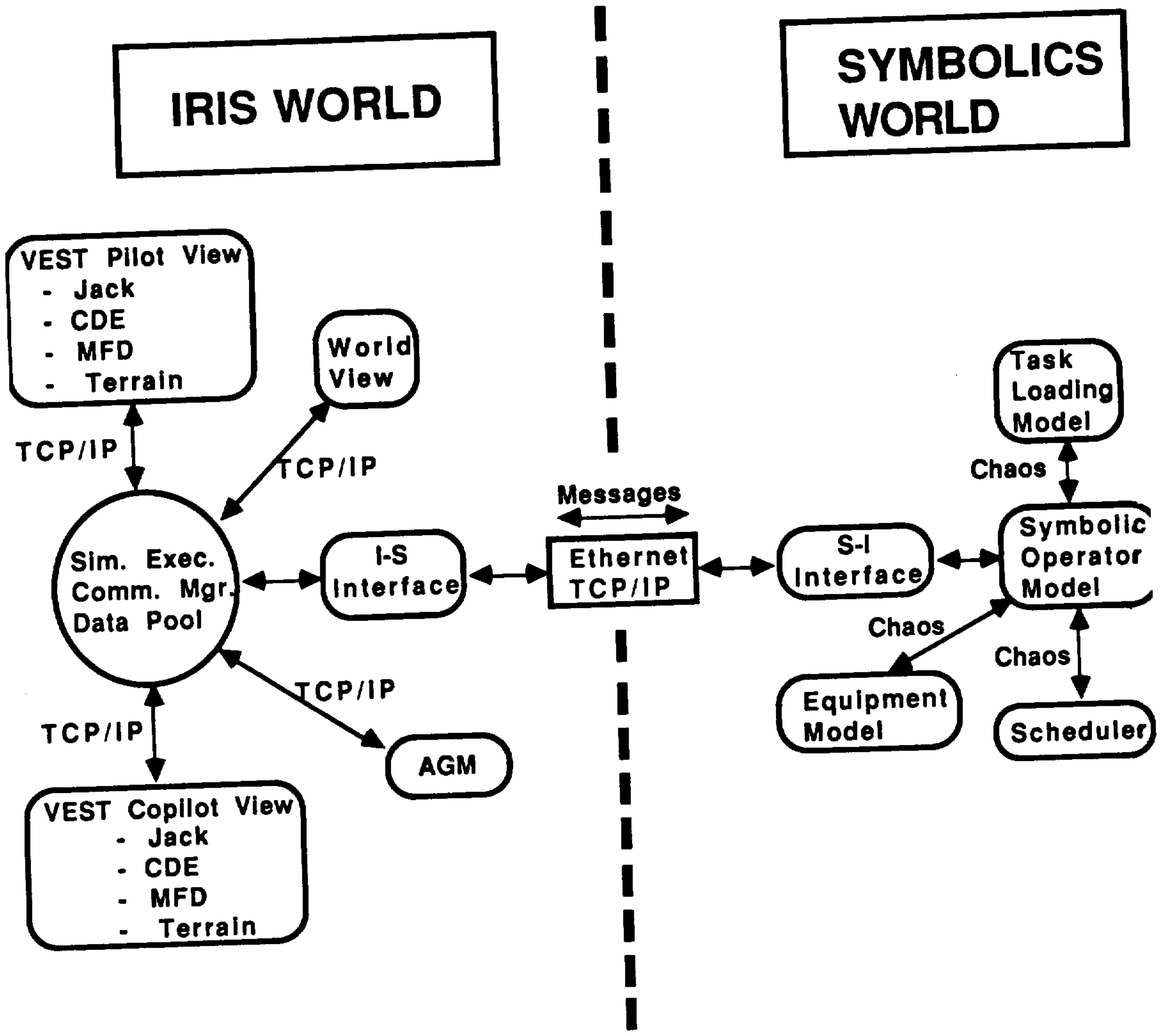


Fig. 2

Figure 2. MIDAS Phase V Communication Architecture

The Symbolics modules are distributed over the Genera-based Symbolics workstations, and the IRIS modules are distributed over the UNIX-based IRIS workstations. The MIDAS workstations include the Symbolics 3675, 3640, and 3620, and the IRIS/4D 220, IRIS/4D 120, and Personal IRIS. All these workstations are networked over the Ethernet. TCP/IP-based links are provided to network the communication module with the rest of the IRIS modules. Similarly, Chaos-based links are provided to network the Symbolic Operator Model with the rest of Symbolics modules. The Symbolics and IRIS worlds are networked by the TCP/IP-based I-S and S-I data transceivers.

The following sequence of processes comprises the run-time simulation : initialization process, connection process, and data passing. The scenario given in section 3.4 represents the standard procedure a simulation goes through.

5.1.1 Design Approach and Tradeoffs

The communication module adopted the tick-based simulation approach, which provides a simple coordination method for distributed systems.The shortcoming of this approach is that it bottlenecks the entire system to the slowest process. Another candidate is the event-driven approach, which could result in potential speed up of the simulation at the cost of requiring more complex coordination method.

During a simulation, the MIDAS modules, distributed over different computer workstations, have to exchange information frequently. Some strategies were adopted to yield better performance, not necessarily real-time but at least interactive. First, information to be exchanged was represented by data structures that got passed around between the IRIS workstations. This is much faster than sending each field byte by byte. However, because of the byte swapping problem it was not possible to send a data structure as a whole between a Symbolics workstation and an IRIS workstation. Fast ways of sending messages between heterogeneous workstations need to be developed in future phases.

5.1.2 Architectural Design Description

The simulation executive serves as the simulation control center. The communication manager serves as the message routing center and the data pool is a cache of the communication manager. The communication manager was designed to possess the knowledge of destination modules for all messages. All messages have to go through the communication manager before they reach destination modules. Direct message passing between the IRIS modules is not allowed. Therefore, if a module needs to send a message to other modules, all it needs to do is to send the message to the communication manager, which then routes the message to the destination modules. This design makes the communication manager very modularized, easy to maintain and expand. To incorporate a new module into MIDAS, it only needs to build the link between the new module and the communication manager.

The inclusion of the data pool as part of the phase IV communication module architecture was because of the potential application of the parallel processing techniques to MIDAS in future phases. The data pool was intended to emulate the global memory space available on most parallel computers for storing intact data as well as the state variables. In phase IV, the data pool was used merely to store the state variables. In fact, it was the only place in MIDAS that holds the most current values of all state variables at any instant. The data pool, residing in the IRIS world, can be accessed by the rest of the MIDAS system through the network. There are cases in a simulation that some module needs values of certain state variables for a small portion of the simulation. For these cases, the module queries the data pool for the values of state variables through the communication manager.

5.1.3 External Interface Design

The messages passed around could include any combination of text strings and numeric data in any form. The external interfaces should be designed to be able to perform this job. A unified definition of messages was contrived and used across the MIDAS modules. A message always has an id, but may or may not have a body, depending upon the purpose it serves. Messages that merely serve "flag" purpose do not have a body. For example, the message which signals the completion of the simulation falls within this category. Other messages have a specific data structure associated with their body. The message id is represented by an integer. Therefore, whenever an incoming message is detected, the external interfaces retrieve the four-byte integer message id, followed by the message body, depending on the message id.

5.2 DETAILED DESIGN

As mentioned earlier, the phase IV simulation adopted the tick-based approach, with the tick size set to one hundred milliseconds. The simulation executive is responsible for incrementing and propagating the tick through MIDAS to trigger individual processes. Every module progresses for the duration of a tick, exchanges information, and notifies the simulation executive of the completion for the current tick.

There were approximately two hundred and fifty state variables shared among the MIDAS modules. These state variables were logically grouped and represented by approximately thirty-five C data structures in the IRIS world. Equivalent representations were defined in the Symbolics world. There were approximately fifty messages defined in the phase IV MIDAS.

The task of providing capabilities for bi-directional message passing between the MIDAS modules was divided into three subtasks. These subtasks were to provide capabilities to allow message passing (1) between the modules within the Symbolics world, (2) between the modules within the IRIS world, and (3) between a Symbolics module and an IRIS module.

Subtask (1) was implemented based on Chaos protocol. Subtask (2) was implemented based on TCP/IP protocol. The I-S and S-I data transceivers together perform subtask (3). The I-S data transceiver is resident in the IRIS world and is responsible for transmitting/receiving messages to/from the Symbolics world. The S-I data transceiver is resident in the Symbolics world and is responsible for transmitting/receiving messages to/from the IRIS world. They also ensure correct byte order when numeric data are passed between these two worlds.

5.2.1 Detailed Design Approach and Tradeoffs

Byte orders are preserved when numeric data are passed between the modules within the same world. However, bytes get re-ordered when numeric data are passed between a Symbolics module and an IRIS module. In phase IV, the S-I data transceiver assumed the responsibility of performing proper byte operations on the numeric data transmitted to or received from an IRIS module. Before transmitting numeric data across the Ethernet, the S-I data transceiver re-positions them so that the I-S data transceiver just collects the bytes in the order received and forms numeric data. After receiving numeric data from the Ethernet, the S-I data transceiver re-positions the bytes in appropriate order and forms correct numeric data. The S-I data transceiver is by no means superior to the I-S data transceiver in performing byte operations. The I-S data transceiver can do the job as well.

Another strategy was adopted to reduce network traffic based on the configuration that the communication manager is the message routing center. Most MIDAS modules did not query the data pool for the data they needed. At any stage during a simulation, if a data structure gets updated by a module, the module immediately sends the updated data structure to the communication manager. The communication manager then updates the corresponding data structure in the data pool and forwards it to destination modules.

5.2.2 Detailed Design Description

5.2.2.1 Compilation Unit Design and Traceability to Architectural Design

Figure 3 shows the control flow of the communication module. It also represents how a simulation proceeds. In particular, it maps well to the sequence of the run-time simulation processes : initialization process, connection process, and data passing.

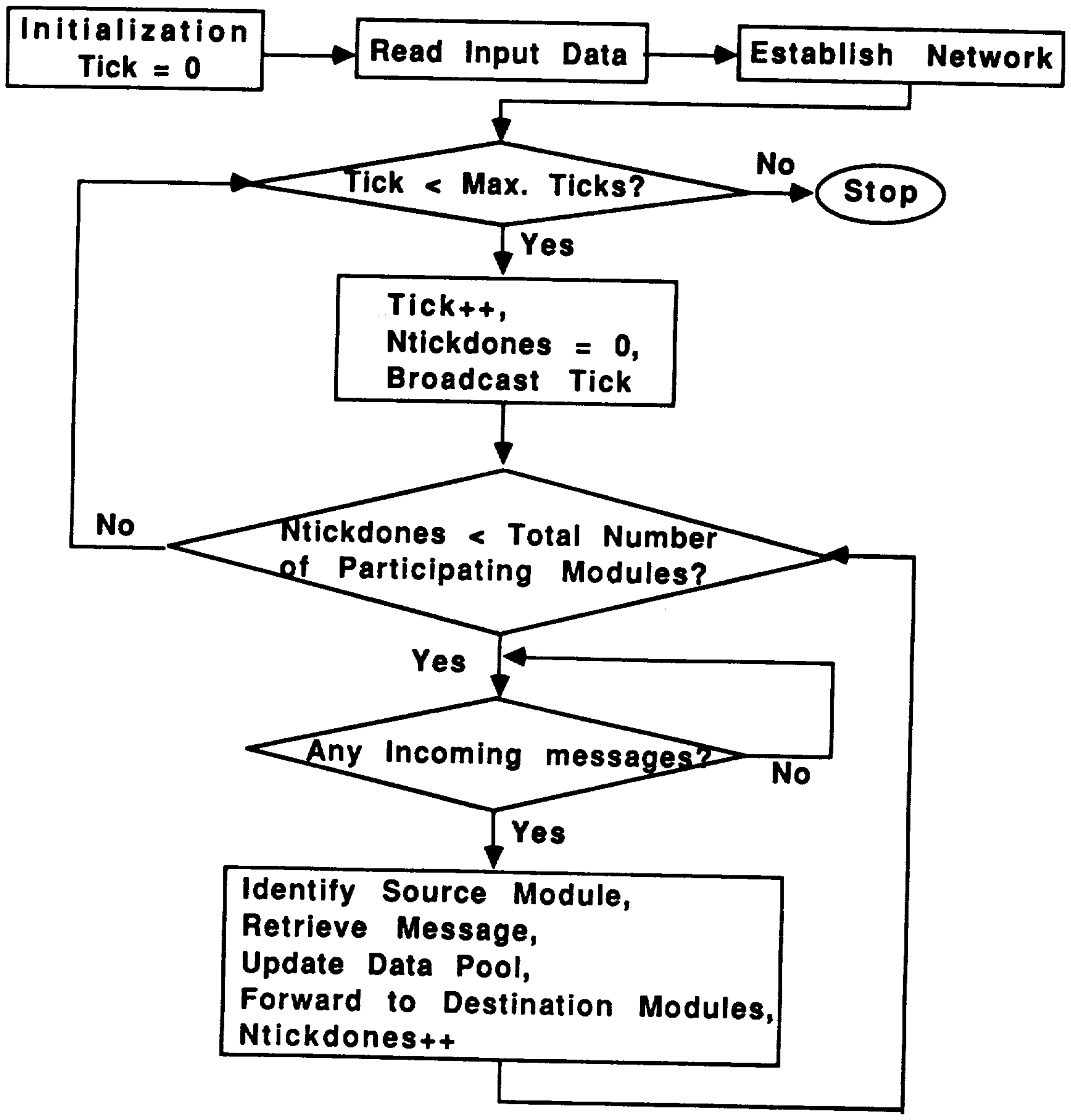


Fig. 3

Figure 3. Control Flow of the Communication Module

5.2.2.2 Detailed Design of Compilation Units

As mentioned earlier, prior to running a simulation the user needs to specify the duration of the simulation in terms of ticks, the participating modules, and the workstations they run on, by editing a text file. If the file did not exist and the command to run simulation was executed, the communication module would stop and alert the user that file was not found.

Each message was identified by an appropriate symbol. A set of C data structures was defined; these data structures represent the groups of state variables mentioned earlier. The message ids and the set of data structures were included in the header file dp.h, which was used not only by the communication module but by other modules, such as VEST Pilot, VEST Copilot, and View. Listed below are the the software units that actually perform the run-time simulation as described in the architectural design section.

5.2.2.2.1 Detailed Design of Initialization Unit

Five C functions are involved in this unit: main(), init_proc_structs(), init_dp(), get_procs(), and get_dp_types(); they are all contained in the file comm_mgr1.c.

A data structure was defined and contained in the header file port.h to hold important network information, such as the port and socket numbers, for all participating modules. An array of this data structures was declared to store the network information of all participating modules. The main function first calls init_proc_strucs() to initialize this array of data structures. Init_dp() initializes the data structures defined in dp.h. Get_procs() reads the input file that contains the user-selected participating modules and the workstations these modules run on. Get_dp_types() contains the source and destination modules for each data structure.

5.2.2.2.2 Detailed Design of Connection Unit

Establish_network() is the major function involved in this unit. This function follows the standard procedure to bring the communication manager to the state ready for accepting connection requests from other modules. It involves creating sockets, binding port numbers and/or network addresses to sockets, and listening for and accepting connection requests. This process continues until all participating modules have been connected to the communication manager. The socket numbers returned from successful accept calls are stored for use in subsequent data transmission.

5.2.2.2.3 Detailed Design of Data Passing Unit

Update_dp.c and send_dpdata_to_dests.c are the major files involved in this unit. Update_dp.c contains the C functions for the communication manager to detect the sockets that have incoming messages. The communication manager processes them one at a time. First, it retrieves the message id, and possibly the message body, depending on the message id. Then it updates the corresponding data structure in the data pool. Send_dpdata_to_dests.c contains the C functions to forward the updated data structures to destination modules. This process continues for as many ticks as the user desires.

5.2.3 External Interface Detailed Design

The message id is represented by an integer, and the message body by a data structure. Therefore, whenever an incoming message is detected, the external interfaces retrieve the four-byte integer message id, followed by the message body, depending on the message id.

5.2.4 Coding and Implementation Notes

The compiler options used included -lsun, -lbsd, and -lc_s.

6.0 USER'S GUIDE

6.1 OVERVIEW OF PURPOSE AND FUNCTIONS

The whole point of the phase IV communication module was to develop a suite to integrate the MIDAS modules, which are written in Lisp, C, and Fortran and distributed over the IRIS and Symbolics workstations. The communication software was written in C and Lisp and provided reliable bi-directional data transmission.

6.2 INSTALLATION AND INITIALIZATION

When the user runs the MIDAS simulation, he should always first bring up the communication module and let it progress to the point where a message appears on the screen indicating it is ready to accept connection requests. At this moment the user brings up the participating modules. Otherwise, the participating module would exit after a number of attempts to connect to the server simply because the communication module was not available.

By default, VEST Pilot is designated to run on Coral (IRIS/4D 120), VEST Copilot on Starfish (IRIS/4D 220), Views on Urchin (Personal IRIS), Aero/Guidance on Starfish, and the Symbolic Pilot Model on Barracuda (3675).

To bring up VEST Pilot, the user types "mgflt.coral -DATA/lbcoralsim" on Coral. VEST Pilot will perform the necessary setup automatically. When the setup is done, mouse click the CDE item from the MultiGen menu bar, drag the mouse down to Ethernet, and release the mouse. The VEST Pilot will try to connect to the communication module. If the connection is successful, a message indicating so will appear on the workstation on which the communication module runs. To bring up VEST Copilot, type "mgflt.starf -DATA/lbstarfsim". To bring up Views, type "../urchin/mgflt -DATA/ahurchinsim". VEST Copilot and Views follow the same procedure to connect to the communication module. To connect the Symbolic Pilot Model to the communication module, simply type (conn-to-which-4ds? 'starfish 'coral) on Barracuda.

When all participating modules have been successfully connected to the communication module, the simulation progresses and continues for as many ticks as the user specified.

6.3 STARTUP AND TERMINATION

Prior to running a simulation, the user has to have an input file which contains the duration of the simulation in terms of tick, the participating modules, and the workstations these modules run. Once this file is ready, follow the steps described in the User's Guide section to start the simulation. The simulation will terminate when it has gone through the duration specified in the input file.

No special procedures are provided when MIDAS encounters abnormal conditions. In such cases, it is suggested that the user bring down the communication module posterior to the participating modules.

6.4 ERROR AND WARNING MESSAGES

Certain check points were included in the connection unit to make sure all participating modules had been connected successfully to the communication module. Similarly, check points were also inserted to make sure the communication module reads enough bytes from the sockets.

6.5 RECOVERY STEPS

It is suggested that the user follow the User's Guide section to restart the simulation if any abnormal situation occurs.

7.0 NOTES

7.1 LESSONS LEARNED

During the development stage of the phase IV communication module, the developer learned that the task of integration is a horrendous and challenging job. It requires seamless collaboration among the module developers. And the individual modules ought to be well developed before the attempt of integration. Hope that this will be improved in future phases if integration remains as a focal point of MIDAS.

The overall performance in phase IV required about a second clock time to process a tick. Seven integrated modules, networks, graphics, and computation contributed to this performance. Networks and graphics were probably the main contributors.

7.2 FUTURE DIRECTIONS

To improve the MIDAS performance, the following things might be worth pursuing. Explore alternatives to transmit numeric data between a Symbolics workstation and an IRIS workstation. In particular, it would be nice if data structures could be passed around as a whole between a Symbolics module and an IRIS module. It is worthwhile exploring other communication protocols with less overhead. A parallel computer could even totally eliminate the network bottleneck. Reduce the graphics overhead embedded in MultiGen by developing a package for MIDAS purpose.

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE December 1991	3. REPORT TYPE AND DATES COVERED Contractor Report

4. TITLE AND SUBTITLE Army-NASA Aircrew/Aircraft Integration Program: Phase IV A^3I Man-Machine Integration Design and Analysis System (MIDAS) Software Detailed Design Document	5. FUNDING NUMBERS NAS2-13210
6. AUTHOR(S) Carolyn Banda, David Bushnell, Scott Chen, Alex Chiu, Betsy Constantine, Jerry Murray, Christian Neukom, Michael Prevost, Renuka Shankar, and Lowell Staveland	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Sterling Federal Systems, Inc. 1121 San Antonio Road Palo Alto, CA 94303-4380	8. PERFORMING ORGANIZATION REPORT NUMBER A-92049
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Ames Research Center Moffett Field, CA 94035-1000	10. SPONSORING/MONITORING AGENCY REPORT NUMBER NASA CR-177593

11. SUPPLEMENTARY NOTES

Point of Contact: Robert A. Carlson, Ames Research Center, MS 233-15, Moffett Field, CA 94035-1000; (415) 604-6036 or FTS 464-6036

12a. DISTRIBUTION/AVAILABILITY STATEMENT Unclassified — Unlimited Subject Category 54	12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This report details the capabilities and design approach of the MIDAS (Man-machine Integration Design and Analysis System) computer-aided engineering (CAE) workstation under development by the Army-NASA Aircrew/Aircraft Integration (A^3I) Program. This workstation uses graphic, symbolic, and numeric prototyping tools and human performance models as part of an integrated design/analysis environment for crewstation human engineering. Developed incrementally, the requirements and design for Phase IV (July 89-Oct 90) are described. Software tools/models developed or significantly modified during this phase included: symbolic operator model; scheduler (Z) model; task loading model; symbolic equipment models; visual editor and simulation tool (VEST); display layout analysis; anthropometric model "JACK"; vision models; aerodynamics/guidance and terrain module; and simulation exec., communications module. These components were successfully used during Phase IV to demonstrate the complex interactions and human engineering findings involved with the proposed Apache Longbow multifunction displays.

14. SUBJECT TERMS Computer-aided engineering, Human performance modeling, Crewstation design, Man-machine interface, Human factors engineering	15. NUMBER OF PAGES 528
	16. PRICE CODE A23

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT