NASA Contractor Report 177596

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

Carolyn Banda, David Bushnell, Scott Chen, Alex Chiu, Christian Neukom, Sayuri Nishimura, Michael Prevost, Renuka Shankar, Lowell Staveland, and Greg Smith of Select Systems Analysis

June 1992

National Aeronautics and Space Administration

NASA Contractor Report 177596

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

Carolyn Banda, David Bushnell, Scott Chen, Alex Chiu, Christian Neukom, Sayuri Nishimura, Michael Prevost, Renuka Shankar, Lowell Staveland, and Greg Smith of Select Systems Analysis, Sterling Federal Systems, Inc., 1121 San Antonio Road, Palo Alto, California

June 1992

National Aeronautics and Space Administration

Ames Research Center
Moffett Field, California 94035-1000

Table of Contents

Figures

Army-NASA Aircrew/Aircraft Integration Program: Phase V

Man-machine Integration Design and Analysis System (MIDAS)

SOFTWARE CONCEPT DOCUMENT

1.0 INTRODUCTION

This is the Software Concept Document for the Man-machine Integration Design and Analysis System (MIDAS) being developed as part of Phase V of the Army-NASA Aircrew/Aircraft Integration (A^3I) Program. The approach taken in this program since its inception in 1984 is that of incremental development with clearly defined phases. Phase I began in 1984 and subsequent phases have progressed at approximately 10-16 month intervals. Each phase of development consists of planning, setting requirements, preliminary design, detailed design, implementation, testing, demonstration and documentation. Phase V began with an off-site planning meeting in November, 1990. It is expected that Phase V development will be complete and ready for demonstration to invited visitors from industry, government and academia in May, 1992.

This document, produced during the preliminary design period of Phase V, is intended to record the top level design concept for MIDAS as it is currently conceived. This document has two main objectives 1) to inform interested readers of the goals of the MIDAS Phase V development period, and 2) to serve as the initial version of the MIDAS design document which will be continuously updated as the design evolves. Since this document is written fairly early in the design period, many design issues still remain unresolved. Some of the unresolved issues are mentioned later in this document in the sections on specific components. Readers are cautioned that this is not a final design document and that, as the design of MIDAS matures, some of the design ideas recorded in this document will change. The final design will be documented in a Detailed Design Document published after the demonstrations.

2.0 RELATED DOCUMENTATION

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

Army-NASA Aircrew/Aircraft Integration Program, A^3I: Phase IV, Man-machine Integration Design and Analysis System (MIDAS) Software Detailed Design Document, NASA Contractor Report Draft, June, 1991. Also published as TN-91-8216-000, Technical Note 1, Sterling Software, October, 1991. (Referred to in the text as the MIDAS Phase IV Documentation)

Larimer, J., Prevost, M., Arditi, A., Azueta, S., Bergen, J., and Lubin, J., "Human visual performance model for crewstation design," Proceedings of SPIE - The International Society for Optical Engineering, Vol 1456, Large-Screen-Projection, Avionic, and Helmet-Mounted Displays, (1991).

Prevost, M. & Banda, C. P., "Visualization tool for human-machine interface designers," Proceedings of SPIE - The International Society for Optical Engineering, Vol 1459, Extracting Meaning from Complex Data: Processing, Display, Interaction II (1991).

Shankar, R., "Z-Scheduler: Integrating theories on scheduling behavior into a computational model," Proceedings of the 1991 IEEE International Conference on Systems, Man and Cybernetics (1991).

Staveland, L., "MIDAS TLM: Man-machine Integration Design and Analysis System Task Loading Model," Proceedings of the 1991 IEEE International Conference on Systems, Man and Cybernetics (1991).

3.0 DEFINITION OF MIDAS

MIDAS is an integrated suite of software components that constitutes a prototype workstation to aid designers in applying human factors principles and human performance models to the design of complex human-machine systems. MIDAS is intended to be used at the 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 PURPOSE AND SCOPE

The purpose of MIDAS is to provide design engineers/analysts with interactive symbolic, analytic and graphical information which permits the early integration and visualization of human engineering principles guiding design. 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 of analyses are typically presented graphically and visually to the design engineers.

Phase V MIDAS will include the capability to run selected segments of a simulated mission with all necessary software running on one Symbolics machine and one Silicon Graphics IRIS machine networked together via Ethernet. The purpose of the simulation is to provide an environment in which complex interactions among the various models of the environment, equipment, mission and crew are propagated to show illuminating details of the operator task performance, loading, and sequences/duration based on the specific equipment and context of the simulated mission segment.

3.2 GOALS AND OBJECTIVES

Two major goals motivate the MIDAS development effort in Phase V:

1) to significantly expand the dynamic simulation capability, including enhancing the symbolic representation of human operator performance as well as extending the capabilities of the symbolic equipment models, and
2) to fully integrate all components of MIDAS, not only those that support the dynamic simulation system, but also those that support additional interactive analyses, under a single unified user interface.

3.3 DESCRIPTION

3.3.1 MIDAS FUNCTIONALITY

The tools and models contained within Phase V MIDAS are designed to support three types of application:

1) **Interactive Tools**, in which the user interacts with specific tools to perform selected human factors analyses,
2) **Simulation System**, in which the user can run a portion of a simulated mission to analyze human-machine interactions under dynamic simulated mission conditions to obtain selected mission and operator performance measures, and

3) **Specification**, in which the user specifies the inputs required for the interactive tools and/or the simulation system. User inputs include the function and layout of crewstation equipment, a scenario and operator tasks for the simulated mission and certain human operator characteristics, such as size.

Figure 1 below illustrates the functionality of MIDAS, showing the inputs at the left and outputs at the right. A top level MIDAS User Interface gives the user access to all MIDAS functionality.

Some examples of inputs provided by the user are shown in the diagram, including operator characteristics, mission scenario specifications, including flightpath and operator tasks, and crewstation design information. The grey boxes in the figure represent components of MIDAS that operate on those inputs, yielding intermediate products shown in the center. The top level User Interface provides access to several editors. The Mission Editor is the component designed to accept flightpath specifications and the behavior of other scenario objects. The Activity Editor provides an environment in which to specify the operator tasks which make up the activities in a mission. The resulting product is a nominal mission scenario; the actual scenario is a product of running the simulation. The user specifies functional and physical models of crewstation equipment through the Symbolic Equipment Editor. The user specifies the graphical representation of equipment by means of the Cockpit Design Editor (CDE).

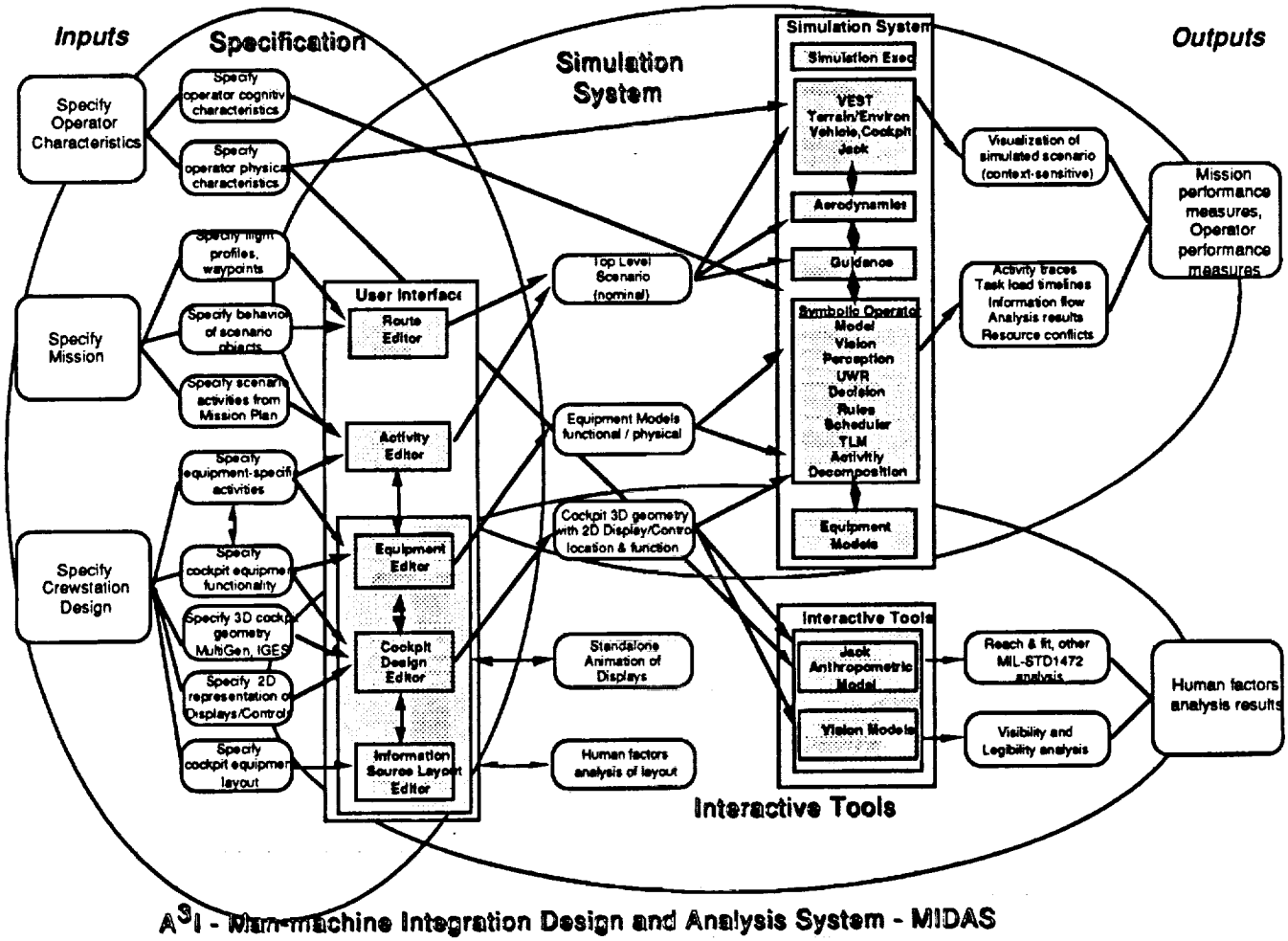

Figure 1. MIDAS Functional Diagram

The mission scenario, operator's cognitive characteristics and the functional/physical equipment models, created by the user in the editors, and shown in the center of the diagram, provide input to the MIDAS simulation system. The components of the simulation system are:

1) the Simulation Executive, which executes the simulation,
2) the Visual Editor and Simulation Tool (VEST), which displays a graphical representation of the world and mission elements as the simulation runs,
3) the Aerodynamics and Guidance models of the aircraft,
5) the Symbolic Operator Model, which include models of Vision, Perception, Motor, Decision by Rules and Decision by Algorithm, an Updateable World Representation, a Scheduler and Task Loading Model and an Activity representation scheme. (Phase V MIDAS will be capable of representing two symbolic operators)
4) the functional and physical models of the crewstation equipment.

As a result of simulating a portion of a mission, the user may obtain outputs, collectively called mission and operator performance measures, including a context-sensitive visualization of the simulated human-machine interaction as well as activity traces, task load traces, resource conflicts, and information flow analysis, as shown in the boxes to the right of the Simulation System.

The Information Source Layout Editor (ISLE) is a MIDAS interactive analysis tool that provides human factors advice on crewstation display layout. The CDE provides a stand alone animation capability to visualize the dynamic characteristics of instruments and displays without having to run a complete simulation. The graphical representation of the crewstation design, created in the CDE, and the physical characteristics of the operator serve as input to the Anthropometric Model (Jack) and the Vision Models, from which the user may obtain human factors analysis results, such as reach a fit and other MIL-STD 1472 analyses, as well as visibility and legibility analyses of the displays.

The extent of the input required to obtain analysis results varies widely. 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.

3.3.2 MIDAS ARCHITECTURE

For Phase V, MIDAS software will run on two machines, a Symbolics XL 1201 and a Silicon Graphics (SGI) IRIS 4D/VGX workstation, linked together by Ethernet. Jack and the Vision Modeling Tool are written in C and reside on the SGI workstation. The specification components are implemented on both machines. The Route Editor (which is part of VEST), and CDE are implemented in C on the SGI, while the Activity Editor and Equipment Editor are implemented in Lisp on the Symbolics machine. The Simulation components are distributed across the two machines. On the SGI, the simulation system data structures and functions are written in C++. VEST (including Terrain, the CDE simulation component, and the models of Environment and Other Objects), Jack, Aerodynamics, Guidance and the Vision agent are implemented in C on the SGI. The Equipment models, the Symbolic Operator Model (including Motor, Pseudo Vision, Perception, Updateable World Representation (UWR), the Decision models, the Scheduler (including the Goal Decomposer) and the Task Loading Model (TLM)), the Mission & SOP component and the Analysis component are coded in Lisp and reside on the Symbolics machine. Each component is described in more detail in section 5.0.

A very top level view of the MIDAS architecture, showing the relationships between components and the distribution of components across machines, is shown in Figure 2 below.

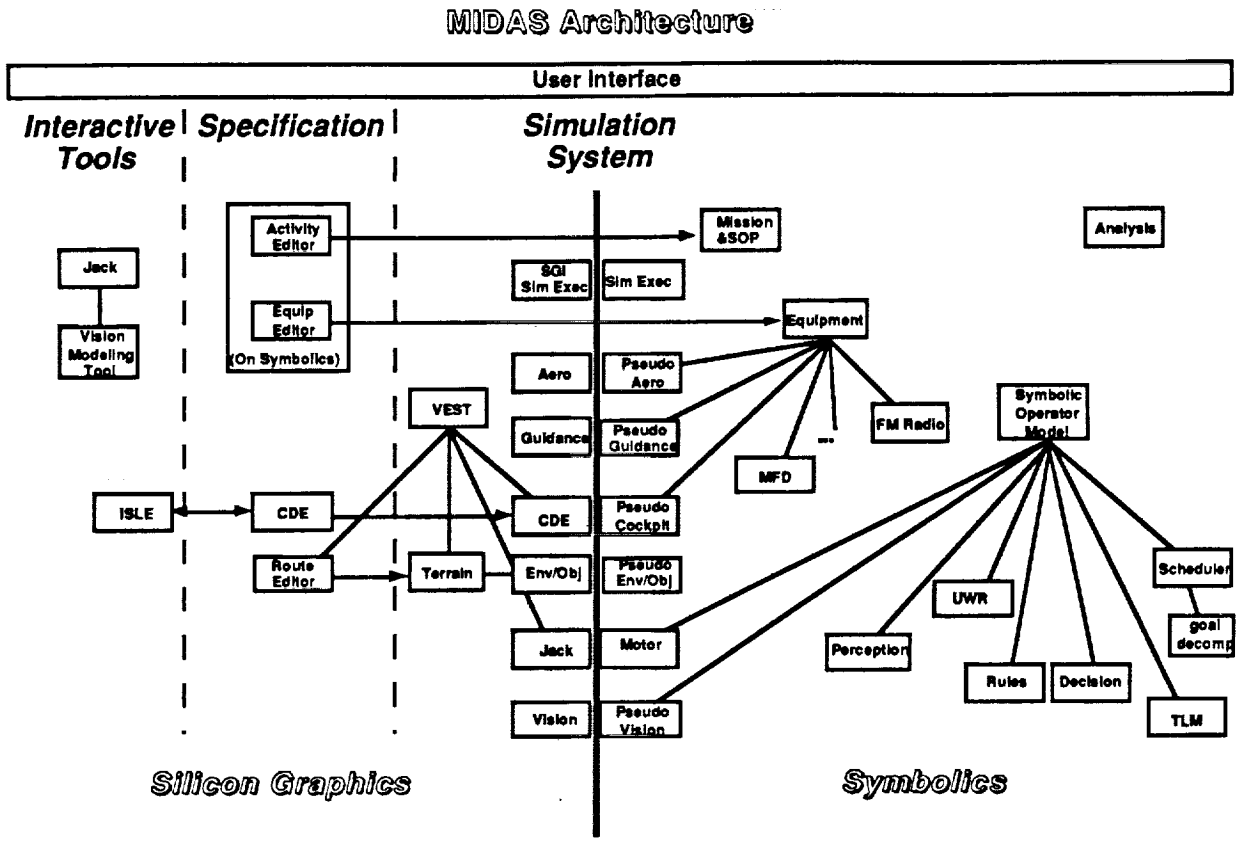

Figure 2. MIDAS Top Level Architecture

3.3.3 MIDAS USER INTERFACE

Although there are three types of application and many components that make up MIDAS, a unified top level MIDAS user interface will provide a similar "look & feel" and operation across components. Implementation of this top level MIDAS interface will be based on Motif and the user's principal entry to MIDAS will be through the Silicon Graphics workstation. Access to the principal functional modules of MIDAS, such as the interactive analysis tools, the various editors for specifying the mission, equipment and operator tasks, and specification of desired simulation analysis results, will be through this top level interface. The user will select the appropriate function, such as Edit Flightpath, for example, from a menu. This menu selection will bring up the Route Editor to enable the user to specify waypoints and other flightpath parameters. The Route Editor is implemented in C as a subcomponent of VEST, but the user will not need to know this. Or, the user may select Edit Mission Activities from the menu. In this case, the Activity Editor will be made available. The user may display the editing window on either the SGI machine, where the top level interface is, or on the Symbolics machine. The Activity and Equipment Editors are written in Common Lisp Interface Manager (CLIM) due to the intimate connection between the editors and the Lisp data structures they support, although it will not be necessary for the user to know where the code resides or how it is implemented. An effort is being made to make the look and feel of the CLIM implementations as much like Motif as possible.

The graphical output during the simulation will be displayed on the SGI machine. Additional information, selected by the user, may be displayed on windows on either machine (via X Windows), depending on the nature of the information and how the displays are implemented. The user may view the Symbolics as a coprocessor and configure all interfaces on the SGI machine, if desired.

In Phase V, due to resource limitations, the user interface to modules, such as Jack, that were implemented in previous phases will be retained. This may result in some differences in look and feel between the top level user interface and the user interface to the module itself. In Phase VI a larger effort will be devoted to user interface issues, and such inconsistencies will be corrected.

4.0 USER DEFINITION

The crew station design process involves professionals from a wide range of disciplines and includes extremely varied activities. For this reason, visualization techniques, such as graphic and iconic representations, are used to foster interdisciplinary communication.

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 Information Source Layout Editor (ISLE), through which the designer could plan and sketch cockpit components, evaluating alternative configurations for displays in the crew station. Those design ideas could be elaborated by using the Cockpit Design Editor to produce a graphical representation of the crewstation displays and controls. With an initial configuration specified, the designer could then put Jack, the Anthropometric Model, in the cockpit and perform some static analyses of reach and fit using figures with different physical characteristics. Several additional MIL-STD 1472 evaluations could also be obtained. 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 its component activities would be defined in the Mission Activity Editor. In addition, s/he would specify both the physical and functional characteristics of the cockpit equipment by selecting or building equipment models and linking them to graphical representations. The designer could also specify certain characteristics of the operator(s). With the mission, operators and cockpit environments 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. Mission tasks could be evaluated as they are executed under simulated mission conditions and task load traces in the four dimensions, visual, auditory, cognitive and motor, could be recorded. Data concerning a large number of mission variables could be collected and analyzed during or after a simulation run. Multiple simulation runs could be run to test sensitivity to certain parameters.

5.0 CAPABILITIES AND CHARACTERISTICS

In this section, each MIDAS component is described individually. Each component is described by the developer principally responsible for its design and implementation.

5.1 INTERACTIVE TOOLS

The Anthropometric Model (Jack), Vision Modeling Tool (VMT) and Cockpit Design Editor (CDE), are implemented in C and ISLE is implemented in C++. These interactive analysis tools, which reside on the Silicon Graphics machine, are described in the follow four sections.

5.1.1 ANTHROPOMETRIC MODEL (Jack)

This section was written by Christian Neukom.

5.1.1.1 Description

Jack is an anthropometric modeling system developed through grants with Dr. Norman Badler at the University of Pennsylvania. The newest version of Jack, 5.4, was released in October, 1991. This software performs two separate functions in Phase V MIDAS:

1) The latest version of Jack with enhancements developed by A^{3}I, as described in this document, will be available to the MIDAS user as an interactive tool for performing analyses of reach and fit and certain other MIL-STD-1472 geometry-based design checks.
2) An earlier version of Jack is incorporated into VEST and animated to provide a graphical representation of the human operator during the simulation.

The enhancements to Jack as an interactive design tool to be developed during Phase V of the A^{3}I Program are described in this section. The integration of Jack software into the MIDAS simulation system is described in section 5.2.3.14.

5.1.1.2 Requirements

Phase V enhancements to Jack as a design tool provided within MIDAS will be developed to provide the following capabilities:

- Reach and fit analysis
- View assessments
- Collision and interference detection
- Selected MIL-STD-1472 geometry design checks, in particular, reach zone analysis.

5.1.1.3 Interfaces

As an interactive design tool, Jack is an independent module called by the user by selecting the appropriate menu item at the top level MIDAS user interface. In Phase V the user interface to the Jack module itself will be supplied by the University of Pennsylvania.

5.1.1.4 Capabilities

The Phase V MIDAS enhancements to Jack include providing reach and fit analysis, view assessments, and collision and interference detection. In Phase V, the newest available version of Jack will be extended to support selected MIL-STD-1472 geometry design checks, in particular, reach zone analysis. In reach zone analysis, the user specifies an instrument or location in the cockpit and the tool classifies that cockpit site according to the following definition of reach zones.

Reach Class Definitions:

> *Reach zone 1.* This is the area where both the smallest aircrew member, with the seat full up and forward, and the largest aircrew member, with the seat full down and aft, can reach and operate any control while they are positioned with their shoulders against the seat back pad, the restraint harness locked, the lap belt snug and without stretching the arm, shoulder or leg muscles as appropriate.
>
> *Reach zone 2.* This is the area where both the smallest aircrew member, with the seat full up and forward, and the largest aircrew member, with the seat full down and aft, can reach and operate any control while they are positioned with their shoulders against the seat back pad, the restraint harness locked, the lap belt snug and using maximum stretch of the arm, shoulder or leg muscles as appropriate.

Reach zone 3. This is the area where both the smallest aircrew member, with the seat full up and forward, and the largest aircrew member, with the seat full down and aft, can reach and operate any control while using the full travel allowed by the unlocked restraint harness but without adjusting the seat position or loosening the lap belt.

Implementation of the various reaches in Jack:

Class 1: Reach chain is hand --> shoulder
Class 2: Reach chain is hand --> waist, with opposite shoulder constrained to seat
Class 3: Reach chain is hand --> waist

Input: neutral seat reference point, seat adjustment, seated height for male and female, equipment list, endeffector.

Output:
- an output file containing a list of equipment with the corresponding classification (i.e. radio left dial, zone 2).
- a display of the above information on screen and a graphical display of the zone classification.

Algorithm:

a) Get user's input.
b) adjust seat
c) Create 95th percentile male figure
d) Seat figure
e) perform reach analysis
f) repeat steps b through c for 5th percentile female figure
g) output result

5.1.1.5 Sample Operational Scenario

A typical scenario of using the anthropometric model (Jack) involves reading an environment file, such as a helicopter cockpit, into Jack followed by creating a human figure of a certain percentile. The figure is then placed in the cockpit seat and diverse reach and fit tests are performed. Various equipment may be selected and classified according to MIL-STD-1472 reach zones. In addition, vision analysis, as described in section 5.1.2, may be performed in the Jack module.

5.1.1.6 Issues

As a consequence of many changes made to the latest version of Jack, Jack has temporarily lost the feature that allowed the user to select a certain percentile of a population. At present, only a default figure is available that corresponds approximately to a 50th percentile male in the NASA Standard 3000. Scaling of individual portions of the figure is possible but not satisfactory. This problem is presently being addressed at the University of Pennsylvania together with a total rework of the Spreadsheet Anthropometry Scaling System (SASS), a spreadsheet-like input and display device for anthropometric data. The total revamp of SASS is expected to take about one year.

5.1.2 VISION MODELING TOOL (VMT)

This section was written by Mike Prevost.

5.1.2.1 Description

The Vision Modelling Tool (VMT) provides the user with interactive tools within the Jack environment with which to explore vision-related crewstation design issues, such as field of view assessments, visibility and legibility analyses and retinal projections of objects in space.

5.1.2.2 Requirements

VMT requirements are as follows:

1) Maintain Existing Capabilities.
2) Enhance control of which objects are drawn in the Aitoff charts.
3) Increase accuracy of Aitoff charts.
4) Incorporate capabilities for light directionality and reflections.
5) Develop capability for specifying fonts needed for legibility analysis.

5.1.2.3 Interfaces

The user interface to VMT is provided as an enhancement to the normal Jack environment and menu interface. The VMT functions can be reached via the "options" menu item.

5.1.2.4 Capabilities

A detailed explanation of VMT's full capabilities is given in Annex H of the MIDAS Phase IV Documentation. In addition to the existing VMT capabilities the following will be added:

1) The rectilinear visual area plots (Aitoff) that are drawn in the current version of VMT are inadequate for two reasons. First, all objects in the environment including Jack and his body parts are drawn. This results in a very cluttered and confusing graph. A mechanism will be provided to the user to allow selection of only those objects s/he is interested in plotting. The second problem with the plots is that only the vertices are plotted correctly and then straight lines are drawn between them. Because a straight line between vertices does accurately represent the projection of the object, a more accurate mode will be provided.

2) The user will be provided feedback on potential reflections relative to the pilot's eye point or the design eye. This will be an interactive feature that allows the designer to vary the location of a light source and then see the effects on the reflection of the light from a given object.

3) A font definition capability for specifying fonts for the Sarnoff legibility model will be provided.

5.1.2.5 Sample Operational Scenario

To use the new Aitoff plotting feature the user must first select a set of objects that s/he wants plotted. The user will then open an output window in which the plot will be drawn. The user can toggle between accurate and approximate modes.

To use the reflection options the user must first select a light source and objects that s/he wants to evaluate for reflection. A reflection axis will be drawn from the reflecting surface with the correct reflection angle as the location of the light source is varied.

5.1.2.6 Issues

Some additional tasks in support of the VMT are required in Phase V:

1) Support, coordinate and integrate legibility model enhancements performed by J. Lubin from the SRI/David Sarnoff Research Labs.

2) Integrate the vision software with Jack version 5.1. This was not a direct requirement but needs to be done.

3) Support, coordinate and integrate volume perimetry model enhancements and eye movement model developed by S. Azueta from the Lighthouse.

5.1.3 COCKPIT DESIGN EDITOR (CDE)

This section was written by Scott Chen.

5.1.3.1 Description

The Cockpit Design Editor (CDE) is an interactive prototype tool for rapidly creating 3-D graphical displays of the instruments. It provides tools to construct and animate traditional gauges and MultiFunction Display(MFD)-type instruments.

5.1.3.2 Requirements

The requirements for the CDE are:

- To make the CDE independent of the MultiGen™ software used in previous phases.
- To provide a user with tools to prototype a cockpit in the 3-D geometry.
- To design a MacDraw-type library to allow a user to sketch a 2-D image of an instrument.
- To allow a user to re-scale a 2-D image of an instrument and map it onto a 3-D polygon.
- To allow a user to build external instrument libraries, which may be written in C.
- To provide tools for a non-programming user to control the animation of cockpit instruments.
- To allow a user to build complex instrument (e.g. MFD) paging/control structures
- To provide an interface with the Equipment Models.

5.1.3.3 Interfaces

CDE defines a simple data format for the import of the CAD files. A converter will be implemented for the MultiGen™ format used internally. A "linearized" IGES converter will also be available. The CDE communicates with the Equipment model to design/modify both the symbolic and graphic portions of a piece of equipment, and to check the consistency.

5.1.3.4 Capabilities

CDE is a mouse-driven Mac-like software. It supports multi-windows, and uses dials and buttons to set up the 3-D window views. It allows a user to select polygons and spawns child windows for the purpose of editing the instruments.

CDE is designed for two levels of users. For a non-programming user, s/he loads the instrument libraries, and creates the cockpit displays based on those instrument fields available in the libraries. This imposes the limitation that when a graphical entity/animation is not in a library, it is not

available for a new instrument. A graphical entity in the libraries is usually "normalized" in the X-Y plane. The user moves the mouse to scale the 2-D image and map it onto the a 3-D polygon.

For a C programmer, s/he can code any type of instrument with the animation method written in C. A text editor can be invoked through the CDE. When the file is loaded into the computer memory, the CDE will parse the C code and execute it. The programmer may change the C code and reload it at any time. If there is no error in the execution, the instrument file can be put into a library and made available for the end users. To allow an end user to access these external instruments, the CDE will provide the programmer a way to control the mouse and the opening of the dialog boxes. The "look" and "input" of the dialog boxes will be defined by the programmer through text files. The user's input through the dialog box will be passed, as arguments, to the C-code entry function of the corresponding instrument.

In addition to the graphics, the CDE will provide the end user certain ways to animate the instruments in the CDE stand-alone animation mode. In this mode, a user can use a text editor to specify data in a disk file for the purpose of animating a selected instrument. Another program will open this file and send the data, through a network interface, to CDE for the animation. The program will allow the user to stop/pause the animation at any tick. With this capability, a user can easily observe and evaluate an instrument's behavior and appearance under differing data input conditions.

5.1.3.5 Sample Operational Scenario

Through the menus and dialog boxes, a user opens a CAD file which defines the 3-D geometry, loads external libraries of instruments, creates an MFD page, selects a polygon as the MFD screen, builds the instrument fields in the MFD page, and maps it onto the MFD screen. If the Equipment model is also running, the user may move the mouse to select an instrument field in the CDE window so that the Equipment model is informed of the selection process. The Equipment model pops up a window to take the user's input for the attributes of the selected instrument. The user also may choose to provide parameter data for a selected instrument and set up an animation of that instrument to evaluate its behavior and appearance.

5.1.3.6 Issues

1) If an instrument field is defined in a library, the animation method is normally provided and transparent to the user. However, for new instruments not supported in the libraries, there is no easy way to allow a non-programming user to define the animation method.

2) Since the external libraries are written in C and the CDE does not care or know how an instrument is displayed, other applications like Jack and the Equipment model will not be able to access the geometry of the instrument. An external "instrument" converter will be required to perform this function.

3) Since it is not possible to define all the instruments in the libraries, the CDE will, at least, provide the instruments needed for the AH-64, LONGBOW and MH-47A helicopters. In addition, a MacDraw-type library will be provided so that a user can use it to define and sketch new instruments.

5.1.4 INFORMATION SOURCE LAYOUT EDITOR (ISLE)

This section was written by Carolyn Banda and Mike Prevost.

5.1.4.1 Description

A prototype of the Information Source Layout Editor (ISLE) was developed in Phase IV. This tool provides users with the capability to visualize, manipulate, and optimize information topology in the design of human-machine interfaces. It uses human factors guidelines to assist the designer in the spatial layout of the information required by machine operators to perform their tasks effectively. The tool contains techniques for visualizing the relative "goodness" of a configuration, as well as mechanisms such as optimization vectors to provide guidance toward a more optimal design. Also available is a rule-based design checker to determine compliance with selected human factors guidelines.

5.1.4.2 Requirements

For Phase V, the requirements for ISLE are to expanded it from its Phase IV prototype as follows:

1) Review and assess current research in the area of display layout design metrics. This includes work by Wickens, Yufik and other researchers.
2) Incorporate new human factors design metrics into ISLE from various sources as appropriate:
   a) MIL-STD 1472 guidelines and criteria for display layouts
   b) MultiFunction Display (MFD) design metrics
   c) Others
3) Allow user manipulation of existing rules and weighting functions.
4) Investigate new optimization algorithms.
5) Integrate ISLE with MIDAS Phase V user interface to support easy transitioning between MIDAS components.
6) Utilize task, equipment and simulation data for layout optimization and evaluation.

5.1.4.3 Interfaces

The user's interface to ISLE will be an interactive mouse/menu type interface still to be defined. The user will create a set of information sources. Using existing relations, relation strengths can then be defined by the user, extracted from simulation data or be assigned default values. ISLE will be very closely linked with the Cockpit Design Editor (CDE) so that changes to one representation can be reflected in the other. A mechanism will be implemented in which information can be represented abstractly in ISLE and later associated with a representation in CDE. There will be a method for the user to toggle between the CDE window and an ISLE overlay. The user will also have the ability to define representations for the information in the CDE before it has been entered into ISLE. If no information has been previously defined, a default information icon will be generated to represent it in ISLE.

5.1.4.4 Capabilities

The human factors guidelines currently available in ISLE were obtained from suggestions by Dr. Christopher Wickens from the University of Illinois. These guidelines are briefly described below. For more information on the design guidelines and on how ISLE uses them, the reader is referred to Annex F of the MIDAS Phase IV Documentation, or Prevost and Banda (1991).

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, physical, and correlational proximities for the information sources. In computing an overall task proximity strength for each information pair, functional proximity is weighted more heavily than physical or correlational proximity.

2) According to symmetrical location compatibility, which is one form of stimulus-response compatibility, displays should generally be located on the same side as the hand controlling the parameter being shown.

3) Displays which have high frequency of use are strongly attracted to prime areas on the display panel, which may be set by the designer by placement of "attractor" bars and regions.

ISLE will continue to have two main components, one analytic and the other rule-based. For the analytic component, the set of human factors design guidelines described above form networks of relations among the information sources and between the information sources and other entities such as controls or attractor/repeller regions of the display surfaces. Arcs act like springs, with arc thickness indicating the strength of the attractive force (that is, the strength of the relation). The designer can show relations for a single information source, a set of sources or all sources. Within the analytic component, both manual and automatic optimization modes are available. In both cases, the effects of moving the display icons are seen graphically. A global network tension measure indicates the degree of overall layout optimization according to the task proximity metric. The rule-based component consists of an advisor which can be invoked to issue warnings for any detected violations of display layout guidelines.

The focus for Phase V will be to make ISLE an integrated tool of MIDAS that can be used to analyze the design of MFD menu spaces and page layouts. A large part of the effort this phase will involve reviewing the results of MFD experiments performed by students of Dr. Wickens, the MIL-STD 1472 and other literature for an implementable set of applicable and general MFD design metrics.

In addition to the new metrics, a large effort will be made to integrate ISLE into the rest of MIDAS so that it can be sensitive to simulation data, equipment data and CDE design changes.

5.1.4.5 Sample Operational Scenario

Before investing time in building a detailed graphical representation of a crewstation, the user can first define, manipulate, cluster, and analyze the information sources in ISLE.

The first step in using ISLE is to create an icon to represent each atomic information source or select an information source from a predefined library. If ISLE is being applied to analyze an existing crewstation, then the existing information sources layout, groupings, page sets, etc., can be entered. Next the relations between information sources and the context in which they are applicable must be entered into the system. If the crewstation information display space is non-homogeneous, the designer can create forces that tend to attract or repel information sources to or from nearby areas depending on the desirability for information location. The designer is then guided by a variety of mechanisms to cluster information into groups. As information groupings are identified, they can be mapped to a coordinate system, then to a page, and finally, one or more pages can be mapped to a display device. The display device is then placed into the crewstation environment, thereby fixing the location of the information sources.

Figure 3 below shows how the ISLE screen would look to the user.

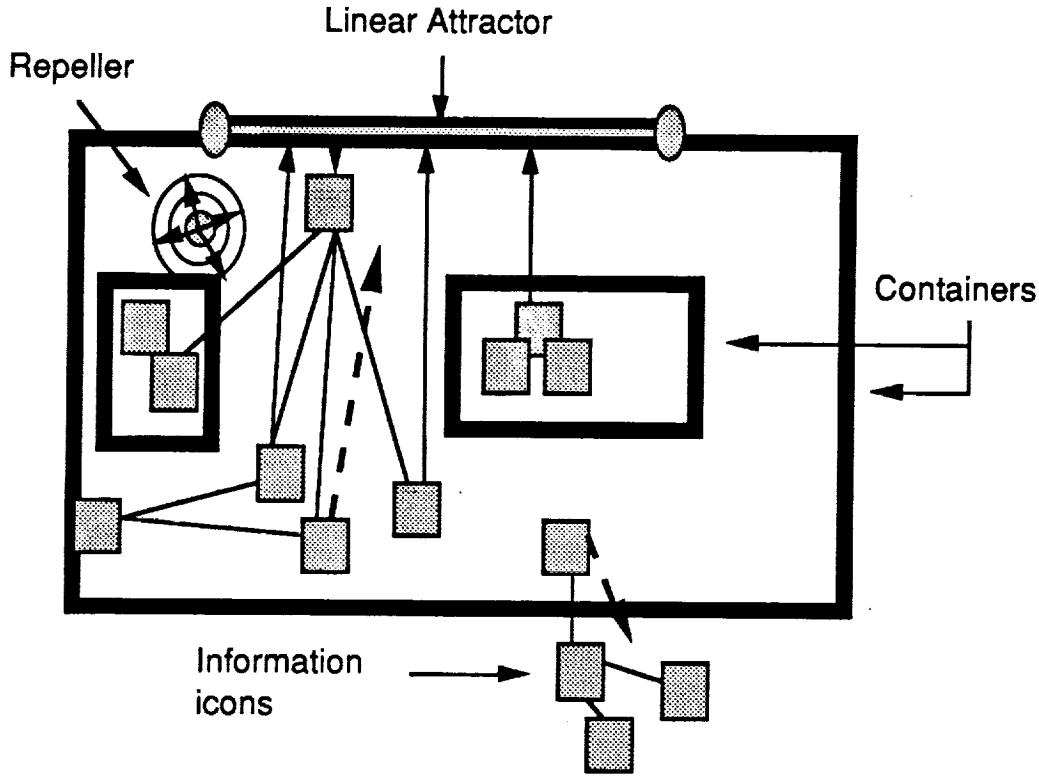

Figure 3. User's View of ISLE

In making a modification such as replacing numerous displays with one Multi-Function Display (MFD), the WHAT information is already known but the organization is not. When the user has a baseline, s/he then flips over to the CDE to define a representation and display device. At this point the user may find that representation-sensitive features of information layout compel him/her to modify the information layout in ISLE again. This time the sizes of at least some of the display devices are known and are shown as bounding boxes, or containers. In this way, the physical representation is linked to the information icons. When the user wanted to manipulate the physical characteristics, s/he would work in CDE. If a display were moved in CDE, the information associated with that display would also be moved without input from the user. The jumping between ISLE mode and CDE mode will be accomplished by menu selection and/or a hot function key.

5.1.4.6 Issues

- Use object-oriented C++ for the programming language to implement ISLE to provide a better mapping from the object-oriented concepts of information icons and their relations. ISLE is one of the first applications to use C++ which places a new requirement on users who wish to modify MIDAS software. It will also increase the complexity of integrating with the CDE.

- Upgrade to CLIPS 5.0, which has more features than CLIPS 4.3, is object-oriented and would therefore integrate well with the future ISLE.

- For checkout and maintenance of the Human Factors guidelines encoded as CLIPS rules, some support work should be done to enhance the interface between the developer and CLIPS. This includes providing an interactive window for communicating with CLIPS directly.

- Explore the possibility of working with Dr. Steve Casner (Code FLT) on utilizing some of BOZ. It might be possible with a small investment in time to integrate Casner's tool as another tool in ISLE. This would allow users to automate the selection of representation for information as well as the layout.

- The current optimizer and rule set do not address the conflict resolution problem. As more metrics are simultaneously applied, this will become a bigger problem. Without new optimization algorithm/rules, there may be no way to automate the layout even when metrics are available.

- Many of the support functions for ISLE were stubbed off. Capabilities such as saving configuration files, relation weights, matrices, etc., are not part of the requirements but need to be there. Some capabilities might be subsumed by the CDE as a result of integration but many will still need to be implemented.

5.2 MISSION SIMULATION SYSTEM

5.2.1 MISSION SIMULATION SYSTEM ARCHITECTURE

The simulation capability will be implemented using an object-oriented agent architecture in which all communicating components are agents that adhere to specifications, developed during the design cycle, for agent communication, message passing and biographers. Agents can be composed of other agents (subagents) so that for complex entities in the system, subfunctions are defined that can be developed, tested, and queried on an individual basis. Strict system semantics for fully concurrent computing potential would demand that all entities in the simulation have an agent status. In this phase of development that strict adherence is not required. Agents will be defined on the basis of their requirements to be analyzed and to contribute to the system performance analysis. In so far as possible "psychologically real" functions e.g., memory and world-knowledge representations, will have an agent-level status. Agents are defined to model the human operator performance as well as vehicle and crewstation performance.

The simulation will be tick-based, that is, controlled by the sending of a scaleable "tick" or time interval to agents in the simulation system. The simulation agents reside on both the Symbolics machine and the SGI machine. The 3D graphics capability of the SGI machine will provide graphical displays to visualize and monitor the progress of the simulation (e.g., aircraft position, control/display state, human movement or visual activity, etc).

Figure 4 shows the agents comprising the simulation system, the types of messages that are passed between agents and the machine on which each agent resides. The communication links between agents are labeled with the type of message: update, execute or query (Q/R). These message types are described in section 5.2.1.1.2. The tick sent from the Sim Exec is described in section 5.2.3.1.4.

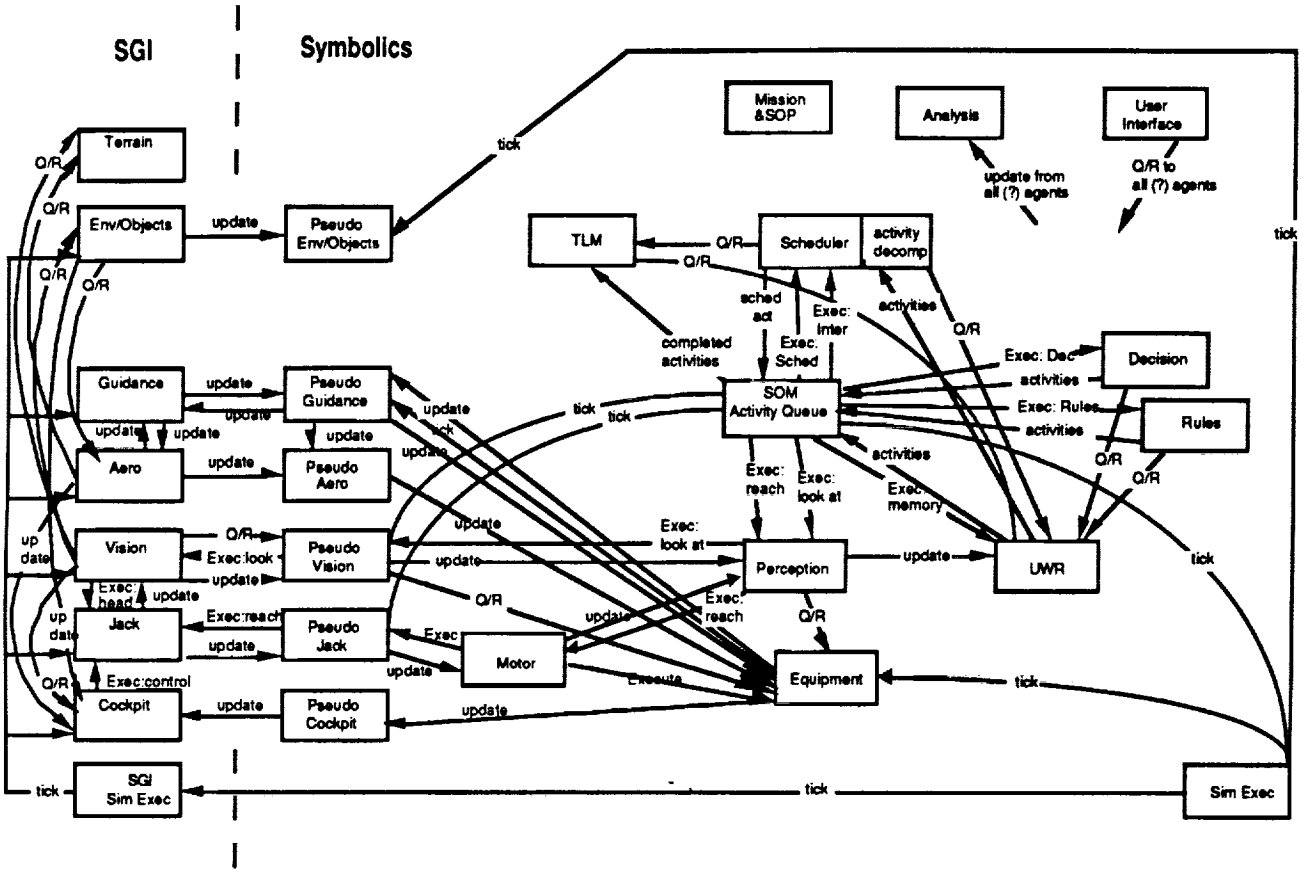

Figure 4. Agents and Communication Links

The Simulation Executive (Sim Exec), which starts/stops the simulation and sends out tick messages, is implemented in Lisp on the Symbolics machine, with a subagent SGI Sim Exec implemented in C on the SGI machine. The Sim Exec on the SGI machine receives the tick message from the Sim Exec on the Symbolics and redistributes it to agents on the SGI machine.

The Symbolic Operator Model (SOM) agent provides models of several aspects of human performance. The SOM has the following subagents, each representing a model of a certain aspect of human performance:

- Decision (by Algorithm), which models decision-making on the basis of algorithms
- Decision by Rules, which models decision-making on the basis of heuristics or rules
- Updateable World Representation (UWR), which represents working memory
- Perception, which is a place holder for a future model of visual attention which will map stimulus input to perception
- Motor, which models certain aspects of physical movements, such as duration
- Scheduler, and its subagent, the Activity/Goal Decomposer, which work together to decompose activities into their components and schedule them in accordance with resource and other constraints
- Task Loading Model (TLM), which computes operator load in four dimensions, visual, auditory, cognitive and motor
- Vision, which provides information about the field of view and the objects therein.

All subagents of the SOM are implemented in Lisp on the Symbolics machine, except the Vision agent, which is implemented in C and C++ on the SGI machine.

The Equipment agent has numerous subagents (not depicted in Figure 4) which represent the specific equipment elements included in the crewstation, such as altimeter, FM radio or MultiFunction Display, for example. The Equipment agent and its subagents are implemented in Lisp on the Symbolics machine.

The Mission & Standard Operating Procedure (SOP) agent holds data describing the mission activities and flightpath which are uploaded into the UWR during initialization.

"Pseudo-agents" are defined to serve as the message interfaces for intermachine exchanges during the simulation to keep the communication functions modular and eliminate the need for each agent to manage the actual intermachine communications. They are implemented in Lisp on the Symbolics machine. See section 5.2.1.3 for further discussion.

On the SGI machine, the Terrain and Environment/Other Objects agents provide the environment in which the simulated mission plays out. The Aerodynamics and Guidance agents model the behavior of the aircraft during the simulated mission. The Cockpit agent displays animated views of the instruments/controls in the crewstation, while the Jack agent provides a visual representation of the human operators. Both the Jack agent and the Cockpit agent are components of the Visual Editor and Simulation Tool (VEST). As stated above, the Vision agent is conceptually a subagent of the SOM, although it is implemented in C and C++ on the SGI machine. All agents on the SGI machine share a common C++ library of agent data structures and functions for communication with other agents in the simulation system, including those on the Symbolics machine. The specific functions of each SGI-resident agent are, for the most part, implemented in C.

5.2.1.1 AGENTS AND COMMUNICATION METHODS

This section was written by Sayuri Nishimura.

5.2.1.1.1 Description

An entity that performs a special and more-or-less autonomous task in the MIDAS simulation is represented by an agent. Communication between agents takes place by message passing.

5.2.1.1.2 Capabilities

An agent is defined as an object which knows where its acquaintances are (or knows how to get to them) which knows its superior and inferior agents in the agent-subagent hierarchy, and which is capable of storing local information about its operation, through the action of a biographer function.

The protocol defined for an agent consists of the following three methods, each of which takes a single argument, namely the agent itself.

1. Reset: This method is called whenever the agent needs to be reset. At the simulation start, reset is called on every agent in the simulation by propagating it top-down through the agent hierarchy tree.

2. Setup-acquaintances: This method is used to initialize the acquaintances for an agent. After all the agents are created at the beginning of the simulation, the setup-acquaintance method is invoked on every agent in the simulation by propagating it top-down through the agent hierarchy tree.

3. Terminate: This method is propagated to every agent just before the simulation is to terminate.

4. Abort: This method is propagated to every agent just before the simulation is aborted due to some abnormal conditions.

Aside from the above methods, if an agent is to respond to each simulation tick, that is, if an agent is a tickable object, the agent should have an appropriate tick method defined for it. The tick method expects the current simulation time to be passed as the second argument.

Each agent can communicate with its acquaintances (agents) by sending one of the following three types of messages:

1. Query : This message contains as its argument a reader function that is used to fetch a value from the receiver agent.

2. Update: This message is used to mutate a slot of the receiver agent. The parameters passed are an identifier of the slot to be updated, and its new value.

3. Execute: This message is used to force the receiver agent to execute some (arbitrary) procedure. The body of this type of message contains the procedure and its list of arguments.

In the MIDAS simulation, almost all the entities involved are agents including activities. The exceptions are messages used for communication.

Since the components of this system are so highly integrated, an early-integration approach has been taken during this phase of development. For Milestone 1, the agent data structures and message passing functions for all agents on both machines were implemented and tested. This provided a test bed of essentially "empty" agents, which lacked any actual functionality but which communicated according to the established message passing protocols.

5.2.1.2 PSEUDO AGENTS

This section was written by David Bushnell.

5.2.1.2.1 Description

Pseudo-agents are Symbolics-based agents that help other Symbolics-based agents to communicate with SGI-based agents when MIDAS is running in its integrated mode. Each pseudo-agent is the representative of a single remote agent. When MIDAS is running stand alone on the Symbolics, pseudo-agents act as stand-ins for the SGI based agents. The idea is that no other agents should have to know which agents are local and which are remote, how the communications protocols between local and remote agents differ, or when the system is running stand alone or integrated. (Here and in the rest of this section, local agents are ones running on a Symbolics machine, while remote agents are ones running on an SGI machine.) In addition, the implementation of pseudo-agents is designed to hide the details of *how* network communication is implemented, so that the builders of new pseudo-agents need only worry about *what* should be communicated.

It should be noted that when a simulation is running stand alone on the Symbolics, pseudo-agents may not provide as accurate or as complete a response as their corresponding remote agents might have. Whether or not they do is up to their implementers. Running the simulation stand alone on the Symbolics is not intended to support end users, but is merely an accommodation for software developers.

5.2.1.2.2 Requirements

The three requirements of pseudo-agents are:

- They must isolate local agents from knowledge of MIDAS' network structure.
- They must isolate local agents from knowledge of the network communications protocols.
- They must isolate the developers of new pseudo-agents from details of how the network communication is implemented.

5.2.1.2.3 Interfaces

5.2.1.2.3.1 Inputs

Pseudo-agents receive inputs from three sources:

- from local agents wishing to communicate with remote agents,
- from remote agents wishing to communicate with local agents, and
- from data files describing a remote agent's behavior.

Local agents needing to communicate with a remote agent send messages and message arguments to the remote agent's pseudo-agent. Remote agents needing to communicate with a local agent send messages via the network to their pseudo-agent. Pseudo-agents running in standalone mode may respond to ticks and messages with data from their associated files.

5.2.1.2.3.2 Outputs

Pseudo-agents send outputs to two destinations:

- to their corresponding remote agents and
- to various local agents.

Upon receiving messages destined for their remote agents, pseudo-agents reformat the messages into a format suitable for network communication and send them along to their destinations. Upon receiving messages for other local agents, pseudo-agents reformat the messages into the proper form for those local agents.

5.2.1.2.4 Capabilities

When a pseudo-agent receives a message from a local agent to a remote agent, it uses the message, its arguments, and their data types to determine which communications protocol to use. The pseudo-agent then reformats the arguments into the form needed by that communications protocol and sends the message to the remote agent. When the remote agent is done processing the message, it will return an error code and, depending on the original message, may return a response. The pseudo agent then converts the error code to an error status description that the local agent can handle and returns the error status and (if necessary) the response to the local agent.

When pseudo-agents receive a message from a remote agent that is destined for a local agent, they do the reverse of the above. They convert the remote message into a local message and pass it to the specified local agent. Any response or error code returned by the local agent is sent back to the remote agent.

When a simulation is running in stand alone mode, pseudo-agents remain functionally the same as far as other local agents are concerned. They accept messages in the same way and continue to return responses.

5.2.1.2.5 Sample Operational Scenario

As an example of how pseudo-agents operate, consider the case of the Aero-Model agent running on an SGI machine. It needs to communicate its aerodynamics data to the Equipment Model agent running on a Symbolics machine. Its communications path looks like:

Figure 5. Pseudo Agent Communications

So, on each tick the Aero-Model agent will invoke MIDAS' remote communications protocol to send its new data to the Pseudo-Aero pseudo-agent on the Symbolics machine. This pseudo-agent will extract the message's destination (the Equipment Model) and data from the communications packet and reformat the data into the form specified when this communications path was defined for this pseudo-agent. Pseudo-Aero will then send an update message to the Equipment Model with Pseudo-Aero's data in it. Finally, Pseudo-Aero will return a "Success" error code to the Aero-Model agent.

5.2.1.2.6 Issues

Pseudo-Agents were implemented as part of the first milestone of Phase V. The primary unresolved issues for the pseudo-agents are how to improve performance and how to handle errors. The first milestone's implementation used Sun's RPC protocol to handle the inter-machine communication. This has proved to be rather slow (with only 25 to 30 RPC calls per second possible between a Symbolics 3675 and an SGI Iris machine). We are looking into improving performance by working with Symbolics to improve the speed of their RPC code, by converting to another protocol, or by getting a faster Symbolics machine.

Error handling is an issue for the simulation as a whole, but it is especially important with the pseudo-agents because they are the interface between two machines. This means that they must know the details of the error handling protocols on both machines, as well as being a special source of errors (i.e. network or communications errors). This problem is being addressed as part of the effort to develop an overall approach to error handling in MIDAS.

5.2.2 SPECIFICATION OF SIMULATION ELEMENTS

Prior to running a simulated mission, the user must specify the flightpath and other operational aspects of the mission, activities that the operator must perform during the mission, the functional and physical elements of the crewstation equipment and certain operator characteristics. The user enters these specifications through various MIDAS editors. The Cockpit Design Editor (CDE), with which the user specifies the visual appearance, location and animation characteristics of

cockpit instruments, was described in section 5.1.3 above. Flightpath planning and specification of other scenario objects is performed with the assistance of the Route Editor, described below in section 5.2.2.1. The functional characteristics of cockpit equipment are specified with the assistance of the Equipment Editor, described in section 5.2.2.2 below. The Activity Editor, described below in section 5.2.2.3, makes it possible to specify new activities and assemble the desired mission scenario for the simulation. The Parametric Analysis and Statistics components, described in section 5.2.2.4, assist the user in specifying what data to collect and in setting initial conditions and making multiple simulation runs.

5.2.2.1 ROUTE EDITOR

This section was written by Alex Chiu.

5.2.2.1.1 Description

This component is mainly used as the graphical route planning front-end for MIDAS. It will be designed and implemented to possess capabilities equivalent to those in the Mission Editor developed at Boeing Helicopters in Philadelphia, with whom this program participates in a Technical Exchange Agreement. It will be integrated into VEST and will therefore be an integral part of the MIDAS system. To achieve consistency with the VEST user interface and data structures, certain modifications to the Boeing-developed Mission Editor will be required, although some portions of the code will be directly incorporated into MIDAS.

5.2.2.1.2 Requirements

To meet Phase V MIDAS needs, it is required to allow the user to

- select a DMA terrain gaming area,
- define route of flight,
- characterize each route waypoint, e.g., x, y, agl, airspeed,
- place environmental objects on terrain.

5.2.2.1.3 Interfaces

After user has selected the route of flight, the Route Editor needs to send the selected flight to both the Guidance agent and Aerodynamics agent.

5.2.2.1.4 Capabilities

The route editor will have two modes, edit mode and view mode. The edit mode allows the user to select a point on terrain and to enter desired parameters for each selected point through a dialog box. The view mode allows the user to position and view the terrain from different perspectives. A set of control functions providing means to manipulate three dimensional objects will be defined and displayed in the control panel and will be accessible to the user. Capabilities of magnifying terrain elevation and varying waterline level will also be developed. The waterline may be used to create lakes and rivers.

5.2.2.1.5 Sample Operational Scenario

This route editor is what the user needs for defining route of flight. It will be included as part of VEST so that the user-defined flight path can be transmitted to the Guidance and Aerodynamics agents by VEST, based on the RPC protocol.

5.2.2.1.6 Issues

None.

5.2.2.2 EQUIPMENT EDITOR

This section was written by David Bushnell.

5.2.2.2.1 Description

Designers will use the equipment editor to create, edit, and delete the functional equipment components required by their scenarios. They will be able to draw on existing equipment models to rapidly create new (related) functional equipment models. In designing equipment models, the Equipment Editor complements the CDE with the Equipment Editor editing the functional aspects of equipment and the CDE editing the graphical aspects. The two can be used together to simultaneously edit both aspects of the models.

5.2.2.2.2 Requirements

The Phase V requirements for the Equipment Editor are:

- It shall allow designers to rapidly develop and use equipment models similar to existing ones.
- It shall work with the CDE to permit simultaneous design of symbolic and graphic aspects of equipment.
- It shall enforce some degree of consistency between symbolic and graphic equipment definitions.

5.2.2.2.3 Interfaces

5.2.2.2.3.1 Inputs

The Equipment Editor will get inputs from existing data files, the CDE, and its users. The data files will describe previously created equipment components. If the CDE is being used simultaneously to edit the graphical equipment model, then it will supply inputs to the Equipment Editor to keep the two synchronized. And of course the users will be there directing the whole production.

More specifically, the inputs to the equipment model are:

- Component names and types
- Whether components will also exist in the CDE
- Component attributes and values
- Which attribute values will be sent to the CDE on each tick
- Which attributes are updateable by other agents
- Which attributes are queryable by other agents
- CDE standalone log files that record changes made using the CDE in standalone mode
- Component description files that define existing components
- "Edit Component" messages from the CDE that describe components being edited there

5.2.2.2.3.2 Outputs

The Equipment Editor will send outputs to data files, the CDE, and its users. The data files will be of two types: component description files describing the newly edited components and standalone log files that are created for the CDE when the Equipment Editor is run standalone. When the Equipment Editor is run with the CDE, it will send messages to the CDE to keep them both synchronized. Finally, the editor will maintain a display for the user showing its current state.

More specifically, the outputs of the Equipment Editor will be:

- Component description files describing the components that were just edited
- Equipment Editor standalone log files recording changes made when running standalone
- "Edit Component" messages to the CDE describing components being edited

5.2.2.2.4 Capabilities

As a practical matter, the Equipment Editor is completely new in Phase V. The only way to create equipment in Phase IV was to type macro calls into Zmacs editor buffers. The goal in Phase V is to develop a set of tools that will make creating and editing equipment models easier. To this end, several kinds of editors will be developed: a component editor for editing components and their attributes, several kinds of "functionality editors" for describing how components react to inputs or change over time, and an activity editor for editing standard operating procedures associated with equipment components. The activity editor is not actually part of the equipment editor and will not be further described here. It is an independent module and is described in section 5.2.2.3.

5.2.2.2.4.1 Component Editor

The component editor is the users' entry point to the Equipment Editor. With it, users select components for editing and make their desired changes. An important requirement on the component editor is that it must interact with the CDE to allow the users to edit components' functional and graphic aspects at the same time. Three aspects of this interaction will be implemented this phase:

- The CDE and component editor will keep each other informed of which components the other is editing. This will allow them to maintain consistent displays.
- The CDE will send components' "animation attributes" to the component editor whenever they are updated. (The "animation attributes" are those that the cockpit module will accept as inputs for display during a simulation run.)
- When they are run standalone, the CDE and component editors will maintain "standalone log files" that describe the changes made to the equipment during the editing session. These logs will be used to resynchronize the equipment models when they are next run together.

5.2.2.2.4.2 Functionality Editors

As currently envisioned, equipment components' functionalities can be described in four ways:

- by a finite state machine with a set of states and a set of legal transitions between states,
- by a script that specifies messages to send on each tick,
- by a script that specifies how to generate output messages in response to input messages, and
- by a set of CLOS methods that explicitly code behavior.

Each of these will have its own editor that users can invoke from the component editor.

5.2.2.2.5 Sample Operational Scenario

Users can edit components with either the Equipment Editor or the CDE. When MIDAS is running as an integrated whole, users can switch back and forth between editing the graphical and the symbolic models. As an example of editing the symbolic model, the following figure shows a user editing an MFD. The functionality will be modeled with both a script and a finite state machine. It contains a brightness control, a screen, a navigation page, and a communications page.

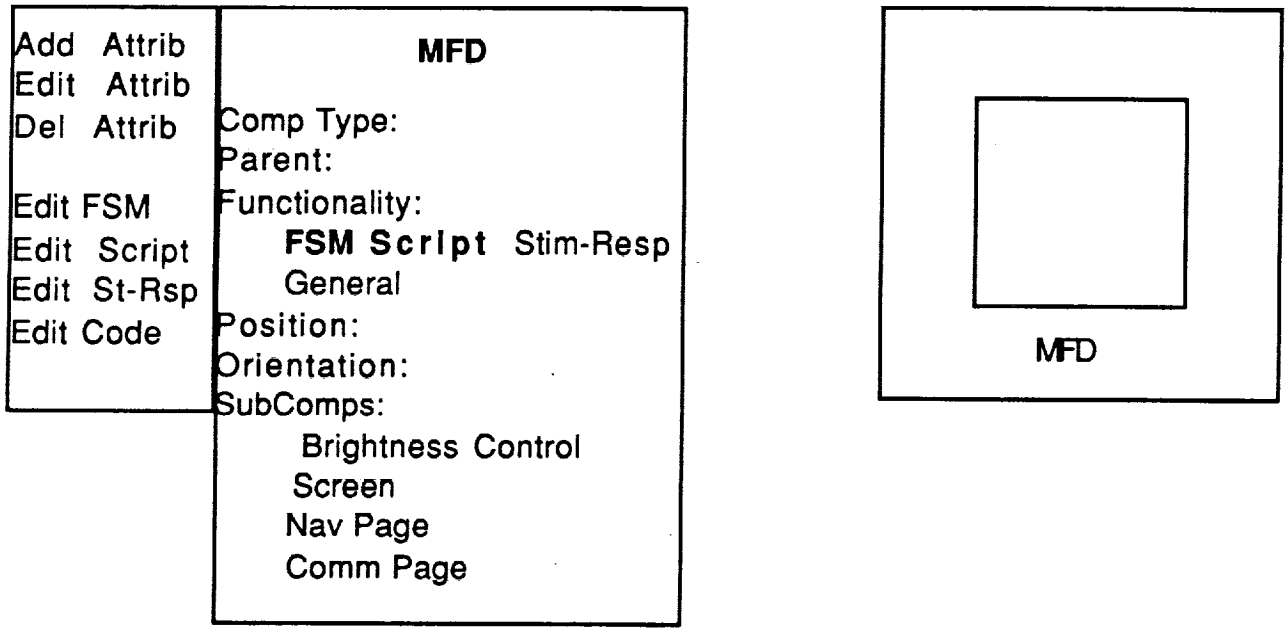

Figure 6. User's View of Equipment Editor

5.2.2.2.6 Issues

The major issue with the Equipment Editor is how it should be integrated with the CDE. Among the questions that need to be answered in order to integrate them are:

- How should the two editors be kept synchronized?
- How much consistency between the two models can be automatically enforced?
- If one of the editors has been used to edit its model standalone, how should the other model be updated to be consistent with the new changes?

Complete integration will not be possible this phase, but some approaches to these questions have been outlined in the preceding sections.

5.2.2.3 ACTIVITY EDITOR

This section was written by David Bushnell.

5.2.2.3.1 Description

The Activity Editor will be used to enter, update, and browse through activities stored in the Equipment Model and the Mission & SOP agents. It can be invoked by either the Equipment Editor or the Mission & SOP. It is therefore designed as a utility that either of these agents can

use. They will call the Activity Editor when the user needs it, passing in their existing sets of activities and it will return to them the updated sets of activities when the user is done. It is not responsible for keeping track of where these activity sets come from or where they go to. Those are the jobs of its callers. The Activity Editor is new to Phase V.

5.2.2.3.2 Requirements

The requirements of the Activity Editor are to:

- allow designers to create, edit, and delete activities;
- allow designers to browse through trees of activities;
- allow designers to decompose higher level activities into lower level ones.

5.2.2.3.3 Interfaces

5.2.2.3.3.1 Inputs

The inputs to the Activity Editor are a set of trees of activities that will be edited and the designers' commands. The activity trees represent the activities that need to be changed or browsed through. This set will be empty in the case where no previous activities exist. The editor will support the commands

- Create new activity
- Edit activity attributes
- Delete activity
- Scroll activity trees left or right, up or down

5.2.2.3.3.2 Outputs

The output of the Activity Editor is the set of updated activity trees.

5.2.2.3.4 Capabilities

The Activity Editor can be used in two ways, as a browser or as an editor of activities. As a browser, the editor supports visualization of the structure of sets of activity trees. As an editor, it supports the creation, deletion, and updating of individual activities.

5.2.2.3.5 Sample Operational Scenario

Upon entry into the editor, the users are shown the tree(s) of existing activities, as in Figure 7.

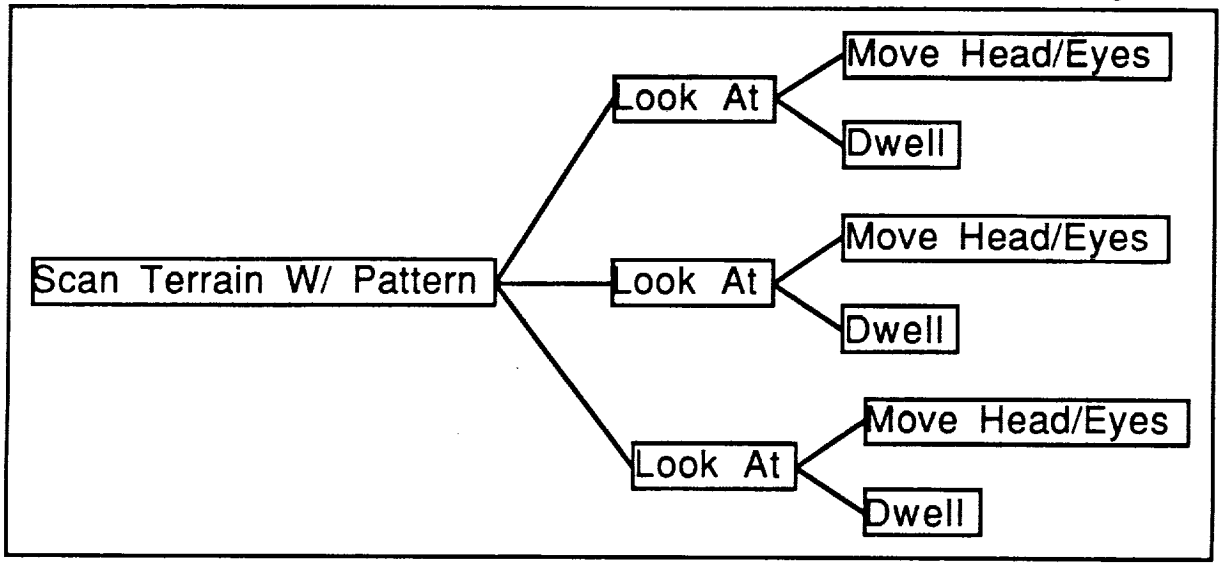

Figure 7. An Activity Tree

In this diagram, each box represents an activity and each link represents a parent-child decomposition. The users can browse through these trees by scrolling the editor window and by examining individual activities. They can issue commands to create new activities and delete old ones, as well as to edit the attributes of existing activities. Activities are edited by creating, deleting, or filling in their attributes. The editor interface is shown in Figure 8.

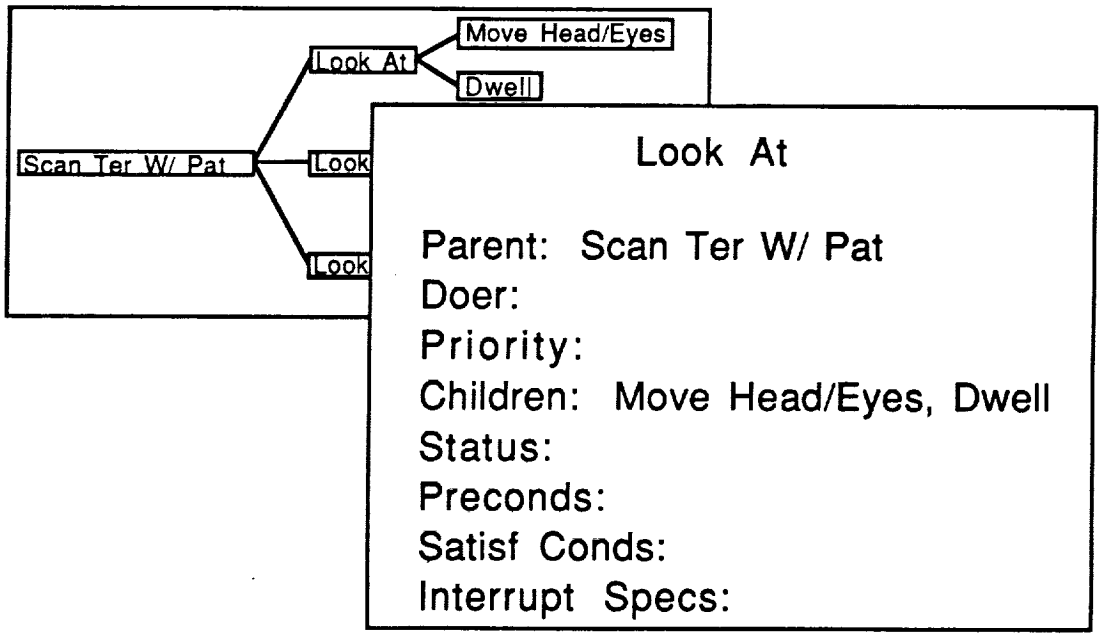

Figure 8. Example of an Activity Editor Screen

5.2.2.3.6 Issues

The principle issue with the Activity Editor is what its user interface looks like. Since there are several similar editors in MIDAS' interface, care should be taken that they have similar interfaces.

5.2.2.4 PARAMETRIC ANALYSIS AND STATISTICS

This section was written by Greg Smith of Select Systems Analysis.

5.2.2.4.1 PARAMETRIC ANALYSIS

5.2.2.4.1.1 Description

The Parametric Analysis agent provides the MIDAS user with a structured facility to define and perform trade-off studies. The Parametric Analysis agent is designed to raise the productivity of the user by providing a special facility to collect and organize trade-off study definitions without requiring the user to establish separate scenario specifications for each experiment trial. The Parametric Analysis agent will also aid the user in study results analysis, output data management, and report writing.

5.2.2.4.1.2 Requirements

The following requirements are relevant to the Phase V implementation of the Parametric Analysis agent:

- The Parametric Analysis agent will provide the MIDAS user with a convenient mechanism to collect trade-off study specifications.
- The Parametric Analysis agent will contain a batch facility to perform a series of simulation runs within the context of a single trade-off study definition.
- The Parametric Analysis agent will provide tools to assess changes in simulated operator performance across different parametric experiment trials.

5.2.2.4.1.3 Interfaces

The Parametric Analysis agent has interfaces with all MIDAS simulation agents. These interfaces are exercised to alter MIDAS agent data (e.g., Jack manikin size or sex) prior to the start of an experiment trial and restore the agent's data back to its initial specification following the simulation run. Additional interfaces are defined between the Parametric Analysis agent and the User Interface and Statistics agents. The User Interface agent communicates trade study specification data to the Parametric Analysis agent and the Statistics agent provides hypothesis testing and report writing utilities.

5.2.2.4.1.4 Capabilities

The Parametric Analysis agent interacts with the MIDAS user through the User Interface to collect trade-off study definitions. The Parametric Analysis agent restricts which MIDAS agent object slots are available to users for inclusion in trade-off studies. The Parametric Analysis agent will store these study specifications on files for retrieval upon initialization of a particular study.

The Parametric Analysis agent will control the processing of multiple MIDAS simulation scenario experiments. The Parametric Analysis agent retrieves user-defined parametric experiment characteristics prior to the start of each simulation trial. The Parametric Analysis agent transmits

these characteristics to the appropriate simulation agents. The Parametric Analysis agent caches the original value of all agent slots that are modified by the current experiment trial. The Parametric Analysis agent invokes the MIDAS simulation after all simulation agents have been updated to reflect the experiment trial definition. The Parametric Analysis agent dumps each agent's biographer and statistics information into a file after the trial completes. The Parametric Analysis agent also sends a series of messages following the trial to restore all modified object slots back to their initial pre-trial values.

The Parametric Analysis agent will be capable of comparing simulation performance across different experiment trials. The Parametric Analysis agent will make these comparisons using the statistical hypothesis testing methods defined in the MIDAS Statistics agent. Finally, the user will have the option of printing any parametric experiment output/report data.

Figure 9 illustrates the basic processing loop of the Parametric Analysis agent.

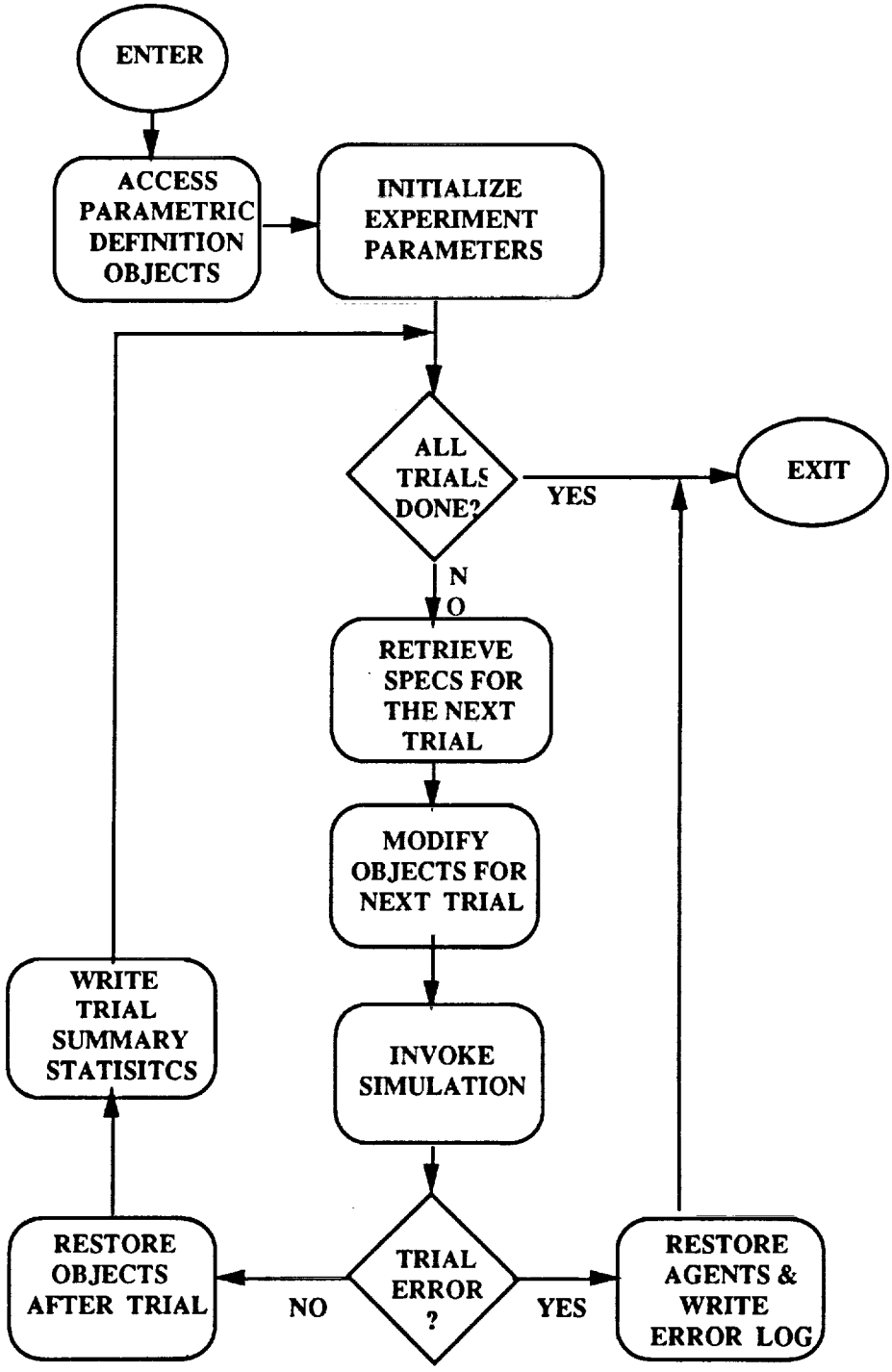

Figure 9. Parametric Analysis Agent Processing

5.2.2.4.1.5 Sample Operational Scenario

The Parametric Analysis agent was not exercised in Milestone 1. One candidate trade-off study experiment might involve the changing the baseline Jack manikin size to reflect the 5, 50, and 95 percentile operators.

5.2.2.4.1.6 Issues

The major outstanding Parametric Analysis issues are:

- What level of error correction and data consistency checking will be provided to the user while defining parametric experiments? The current assumption is that the all data checking will be provided by the agent who's object slots are being modified by the experiment. It is also assumed that all communication between the Parametric Analysis agent and other MIDAS agents will be implemented using the standard agent communication methods.

- How will the Parametric Analysis agent and the User Interface communicate? The major issue to be resolved is whether or not the Parametric Analysis agent should be subsumed entirely by the User Interface agent given the Parametric Analysis agent's extensive requirements for dialogue with the MIDAS user.

5.2.2.4.2 STATISTICS

5.2.2.4.2.1 Description

The Statistics agent is designed to provide MIDAS with generic object class definitions and methods capable of collecting and summarizing most simulation performance information. Since these methods are generic in nature, both from data collection and presentation points of view, they are intended to be inherited by all other MIDAS simulation agents.

5.2.2.4.2.2 Requirements

The following requirements are relevant to the Phase V implementation of the Statistics agent:

- The Statistics agent will permit the conditional collection of key MIDAS agent performance measures.

- The Statistics agent will be capable of collecting observation, counting, time series, and frequency information.

- The Statistics agent will be capable of supporting simple hypothesis tests (e.g. F-test, T-test) to determine if significant changes in simulation performance result from changes in baseline operator, mission, and equipment specifications.

- The Statistics agent can be inherited by any MIDAS agent and should not influence the agent's basic operation or methods.

5.2.2.4.2.3 Interfaces

The Statistics agent has interfaces to all agents through object inheritance. The specific agent statistics collection requirements are specified via the User Interface. Likewise, requests for statistics summary data are sent to the Statistics agent via the User Interface.

5.2.2.4.2.4 Capabilities

The Statistics agent will collect summary performance information for all MIDAS agents. Tables 5.2.2.4-1 through 5.2.2.4-23 document statistics categories envisioned for Phase V implementation. Each table defines a relevant set of agent statistics categories, when they should be displayed (R=> run time, A=> post-simulation analysis), and their relevance to the Milestone 1 demonstration scenario.

TABLE 5.2.2.4-1. AERODYNAMICS.

Output Category	Display Time	Milestone 1
Agent Message Traffic	R	Y
Aircraft State Vector	R	Y
Aircraft State Distributions	A	N

TABLE 5.2.2.4-2. ATTENTION.

Statistics Category	Display Time	Milestone 1
Attention Requests	R&A	Y
Attention Responses	R&A	Y
Attention Activity Failures	R&A	Y
Current Activities	R	Y
Attention State Distributions	A	N

TABLE 5.2.2.4-3. COCKPIT.

Statistics Category	Display Time	Milestone 1
Agent Message Traffic	R	Y

TABLE 5.2.2.4-4. DECISION.

Statistics Category	Display Time	Milestone 1
Agent Message Traffic	R	Y
Utility Function Values	R&A	Y
Utility Attribute Values	R&A	Y
Decision Selection	R&A	Y
Decision Selection Dist.	A	N

TABLE 5.2.2.4-5. ENVIRONMENT.

Statistics Category	Display Time	Milestone 1
Agent Message Traffic	R	Y
Object CLOS Data	R&A	Y

TABLE 5.2.2.4-6. EQUIPMENT.

Statistics Category	Display Time	Milestone 1
Agent Message Traffic	R	Y
Equipment State Distribution (display & control)	A	N

TABLE 5.2.2.4-7. GOAL DECOMPOSER.

Statistics Category	Display Time	Milestone 1
Agent Message Traffic	R	Y

TABLE 5.2.2.4-8. GUIDANCE

Statistics Category	Display Time	Milestone 1
Agent Message Traffic	R	Y
Guidance Command Dist.	A	N

TABLE 5.2.2.4-9. JACK.

Statistics Category	Display Time	Milestone 1
Agent Message Traffic	R	Y
Appendage Use Dist.	A	N
Appendage Slew Limits	A	N

TABLE 5.2.2.4-10. MISSION & STANDARD OPERATING PROCEDURES.

Statistics Category	Display Time	Milestone 1
Agent Message Traffic	R	Y
Goal Activation / Deletion	R	Y
Goal Activation / Deletion Distributions	A	N

TABLE 5.2.2.4-11. OWNSHIP

Statistics Category	Display Time	Milestone 1
Agent Message Traffic	R	Y

TABLE 5.2.2.4-12. PERCEPTION.

Statistics Category	Display Time	Milestone 1
Agent Message Traffic	R	Y
Perceived States	R&A	N
Perception Errors	R&A	N

TABLE 5.2.2.4-13. ROUTE.

Statistics Category	Display Time	Milestone 1
Agent Message Traffic	R	Y

TABLE 5.2.2.4-14. RULES.

Statistics Category	Display Time	Milestone 1
Agent Message Traffic	R	Y
Rule Activations	R	Y
State Values Bound to Rules	R	Y
Rule Traceback	R	Y
Rule Firing Distributions	A	N
State Utilization Distributions	A	N
Decision Distributions	A	N

TABLE 5.2.2.4-15. SCHEDULER.

Statistics Category	Display Time	Milestone 1
Agent Message Traffic	R	Y
Binding Scheduling Constraints	R	Y
Scheduler Strategy	R	Y
Schedule Stability	A	N
Input Activities	R	Y
Resultant Schedule	R	Y
Schedule Time Horizon	R&A	Y
Scheduling Activity Load	R&A	Y
Scheduling Summary Dist.	A	N

TABLE 5.2.2.4-16. SIMULATION EXECUTIVE.

Statistics Category	Display Time	Milestone 1
Agent Message Traffic	R	Y

TABLE 5.2.2.4-17. SYMBOLIC OPERATOR MODEL.

Statistics Category	Display Time	Milestone 1
Agent Message Traffic	R	Y
Active Goals	R&A	Y
Current Activities	R&A	Y
Scheduled Activities	R&A	Y
Pending Activities	R&A	Y
Created Activities	R&A	Y
Interrupted Activities	R&A	Y
Removed Activities	R&A	Y
Resumed Activities	R&A	Y
Completed Activities	R&A	Y
Pending Times	R&A	Y
Service Times	R&A	Y

TABLE 5.2.2.4-18. TASK LOAD MODEL.

Statistics Category	Display Time	Milestone 1
Agent Message Traffic	R	Y
Load Values	R	Y
Load Distributions	A	N
Taxonomy Element Demands	R	Y
Taxonomy Access Dist.	A	N
Clash Pair Activations	R	Y
Clash Pair Dist.	A	N

TABLE 5.2.2.4-19. TERRAIN.

Statistics Category	Display Time	Milestone 1
Agent Message Traffic	R	Y

TABLE 5.2.2.4-20. UPDATEABLE WORLD REPRESENTATION.

Statistics Category	Display Time	Milestone 1
Agent Message Traffic	R	Y
State Values	R	Y
State Value Dist.	A	N
State Updates	R	Y
State Update Dist.	A	N
State Errors	R&A	N
State Uncertainties	R&A	N

TABLE 5.2.2.4-21. VISION

Statistics Category	Display Time	Milestone 1
Agent Message Traffic	R	Y
Background Characterization	R&A	N
Lighting Conditions	R&A	N
Activity Success/Failure (including rationale)	R&A	Y
Time Requirements	R&A	Y
Head/Eye Movements	R&A	Y
Eye Fixation Dist.	A	N

Statistics are collected during the simulation through the use of SETF AFTER-METHODS attached to specific agent object slot accessors. These statistics collection methods are created during the simulation RESET process and are removed during the TERMINATE process. The statistics collection methods will only be created for those agent statistics categories that are selected by the MIDAS user. Statistics collection requirements will be transmitted to the Statistics agent from the User Interface.

5.2.2.4.2.5 Sample Operational Scenario

In the Milestone 1 demonstration, statistics categories were defined for the Mission & SOP agent. These statistics were used to develop summary information characterizing inter-agent message traffic. In particular, statistics objects were defined to develop frequency information for Mission & SOP message sources and destinations. Statistics were also defined which estimated the mean time and standard deviation between successive messages sent to the Mission & SOP agent by the Equipment and UWR agents.

5.2.2.4.2.6 Issues

The major outstanding Statistics issues are:

- What is the preferred method of Statistics Agent object and method inheritance? The default assumption is that each MIDAS developer will define an agent statistics class containing a set of slots to record all relevant agent statistics data. These agent statistics object definitions will in turn be created from basic Statistics agent object class definitions.

- How should statistics information be stored for later analysis and report writing? It is assumed that the user will be able to dump all MIDAS statistics and biographer data to a file after the simulation run completes. The generation of this output file will be controlled by the User Interface agent. Likewise, the User Interface agent will control the loading of previously generated simulation output data for analysis.

- What is the final set of statistics categories that should be collected for each agent? This issue will be resolved by the individual MIDAS agent developers as the simulation capabilities mature.

5.2.3 SIMULATION SYSTEM COMPONENTS

5.2.3.1 SIMULATION EXECUTIVE (SIM EXEC)

This section was written by Sayuri Nishimura.

5.2.3.1.1 Description

The Simulation Executive (Sim Exec) is the highest level of abstraction in the hierarchy of the MIDAS simulation system. The execution of the entire MIDAS simulation is controlled through this agent.

5.2.3.1.2 Requirements

To provide a clean interface to control the simulation execution.

5.2.3.1.3 Interfaces

A Sim Exec object is created for each simulation run through a toplevel user interface which invokes the simulation run. The user interface is a call to a function called "start-midas".

Input:	A reset message sent by the function start-midas
	A setup-acquaintances message sent by the function start-midas
	A run message from the function start-midas
Output:	None

5.2.3.1.4 Capabilities

The Sim Exec is created by the function "start-midas" and it causes all the toplevel agents to be created. Each of the toplevel agents is then responsible for creating its own subagents. The Sim Exec also deals with the handshake with the SGI machine if the simulation is to run in the integrated mode, that is, the SGI machine is to be used. After its creation, the Sim Exec agent can respond to the following methods:

1. Reset: This is called by the toplevel interface, "start-midas", after a Sim Exec agent is created. (It can, however, also be called during a simulation run.) It invokes the simulation user interface module to let the user adjust simulation parameters. It then calls the Reset function on each of the toplevel agents, which in turn call the Reset function on their subagents.

2. Run: This is called by the toplevel interface "start-midas" after the Sim Exec agent has been reset. This causes the simulation run to start and keeps it running by repeatedly invoking a tick until it is appropriate to end the simulation run.

3. Tick: This is called during a simulation run to initiate the particular tick (i.e. to keep consistent time among all the agents in the simulation). It will propagate the tick to each of the toplevel agents, which in turn will propagate it to their appropriate subagents. If a simulation is running in the integrated mode, then the tick function is called on the SGI Sim Exec first to allow the parallel execution between the Symbolics machine and the SGI machine.

4. Terminate: This is called when the simulation is to terminate. It calls the Terminate function on each of the toplevel agents, which in turn call the Terminate function on their subagents.

5. Abort: This is called when the simulation is aborted. It calls the Abort function on each of the toplevel agents, which in turn call the Abort function on their subagents.

5.2.3.1.5 Sample Operational Scenario

In the milestone 1 scenario, a Sim Exec agent is created at the start of the simulation, a reset call is made on it, and the simulation starts by executing the run function. A terminate call is made on it when the mission goal is satisfied.

5.2.3.1.6 Issues

None.

5.2.3.2 SIMULATION INITIALIZATION

This section was written by Sayuri Nishimura.

5.2.3.2.1 Description

This is the toplevel interface to invoke a MIDAS simulation, via the function called "start-midas".

5.2.3.2.2 Requirements

To provide a user with a clear toplevel interface to invoke a MIDAS simulation.

5.2.3.2.3 Interfaces

The function "start-midas" takes a number of keyword arguments which are used to control the creation of the agents in the simulation. These keyword arguments are directly passed to each function call that creates an agent.

5.2.3.2.4 Capabilities

The functionality of the "start-midas" is realized by calling appropriate functions on a newly-created Sim Exec agent which represents the particular simulation run. The simulation is initialized via a call to Reset on the agent, initiated via a call to Run on it and terminated via a call to Terminate on it.

5.2.3.2.5 Sample Operational Scenario

In the milestone 1 simulation, "start-midas" is called as follows:

```
(start-midas    :mode :integrated
                :mission-goal (make-instance 'scan-terrain-with-pattern)
                :scripts `((midas-hank::nav-page ,midas-hank::nav-page-script1)))
```

where :mode specifies whether the simulation is to run in the integrated mode or in the standalone mode, :mission-goal specifies the goal of the mission, and :scripts initializes the equipment.

5.2.3.2.6 Issues

None

5.2.3.3 ACTIVITY REPRESENTATION AND DECOMPOSITION

This section was written by Sayuri Nishimura.

5.2.3.3.1 Description

An activity is an agent which represents a certain task that a human (or automatic) operator performs. An activity is organized in a hierarchical fashion from the top level mission goal to the lowest level activity, and the relationship between a parent activity and child activities are specified by the "decompose" method of the parent activity.

5.2.3.3.2 Requirements

Activities will be characterized by:

1. Preconditions that define the allowable conditions for their spawning and decomposition
2. Satisfaction condition, which define their successful completion.
3. Spawning specification, which detail the temporal and logical constrains on any "child activities" that might be needed for activity performance.
4. Decomposition methods that describe the partitioning of high level activities into the agent-specific requirements for sub-activity operation.
5. Interruption status and interruption specifications which detail the interruption and resumption policies for that activity.
6. Load, which indicates task performance requirements in the Aldrich, Szabo and McCracken resources of visual, auditory, cognitive and psychomotor requirements.
7. Duration, either estimated or calculated by an activity-specific function.
8. Priority, which, in this implementation, will be provided as a fixed table of relative priorities assigned to that activity and constrained by operator-type.

5.2.3.3.3 Interfaces

An activity is created as a mission goal which is uploaded from the Mission & SOP agent or is created in the Updateable World Representation (UWR) through a daemon function. The decomposition of an activity is performed by the Scheduler agent and the execution of an agent is controlled by the Symbolic Operator Model (SOM) agent.

An activity can be defined through the Activity Editor.

Input:	Decompose messages sent by the Scheduler Tick messages sent by the SOM
Output:	None

5.2.3.3.4 Capabilities

The decomposition of an activity is defined by way of the decompose method for the activity. The decompose method of an activity generates a list of the child activities for the activity. An activity may be interrupted (suspended) in the midst of its execution if the Scheduler decides that other more important activities should run in place of it. At the time of interruption, the "interrupt" method for the activity is called and generates a continuation activity which is to be executed when it gets resumed.

There are several types of control activities defined that impose certain constraints on their children's execution. Among them are:

1. Sequential-activity whose children are to be executed one after another.
2. Parallel-any-activity whose children can be executed in parallel and whose execution should terminate if any of the children completes.
3. Parallel-all-activity whose children can be executed in parallel and whose execution should terminate only if all the children complete.

5.2.3.3.5 Sample Operational Scenario

In the Milestone 1 scenario, the demonstration starts with a sequential activity of three look-at activities. The second look-at activity recognizes a downed helicopter and a decide-by-rule activity is spawned via a daemon in the UWR. The decision made by the decide-by-rule activity spawns a navigate-to-target-position activity which spawns a decide-by-algorithm activity. The decide-by-algorithm activity spawns a look-at-hdi activity and a call-new-course-to-pilot activity.

5.2.3.3.6 Issues

1. Some aspects of activity precondition can be realized by using a control activity. For example, the use of a sequential activity can force the order of two activities, the first of which may serve as meeting the precondition. Precondition checking both at the activity definition level and at the individual activity precondition level would be expensive and there should be criteria on what kind of conditions should be expressed in which format. Dr. Corker has stated that preconditions are not representative of cognitive control at the level of execution, but there still remains an issue of what should happen when they are not met.

2. The satisfaction condition may not be necessary. If an activity included a satisfaction condition, then what would the human (or automatic) operator be supposed to do were the condition not met? Would the action to be taken in case of failure also have to be specified in the activity? Such an activity might instead be encoded so that it is decomposed into

child activities: one would check the satisfaction condition and the other would take an appropriate action in case it failed. Dr. Corker has stated that preconditions are not representative of cognitive control at the level of execution, but there still remains an issue of what should happen when they are not met.

3. The resolution of the lowest level activities for modeling purposes should be the level of primitive activities that a human (the Scheduler) would think about. The set of the primitive activities should be determined through discussion with Dr. Corker. Decomposition is defined to stop at the level where the agents execute the activity.

5.2.3.4 MISSION AND STANDARD OPERATING PROCEDURES (SOP)

This section was written by Greg Smith of Select Systems Analysis.

5.2.3.4.1 Description

The Mission & SOP agent provides MIDAS users with methods and objects for storing:
1) equipment independent operator activity specifications,
2) aircraft mission scenario characteristics, and
3) operator tactics.

The Mission & SOP agent will distribute these activity and mission specification data to the relevant MIDAS agents at simulation initialization time. The Mission & SOP agent will also provide a God's-eye-view assessment of operator task requirements during simulation runs to determine any shortfalls in crew performance.

5.2.3.4.2 Requirements

The following requirements are relevant to the Phase V implementation of the Mission & SOP agent:

- Operator activities will be specified within the context of a mission decomposition structure.
- The Mission & SOP agent will permit mission decomposition and operator procedural data to be specified at any user-preferred level of detail.
- The requirements (i.e., pre-conditions) for invoking particular Mission & SOP activities will be specified in terms of rules. These rules will refer to previously defined operator mission (e.g., Ingress) and equipment (e.g., hydraulics system failure) states.
- The Mission & SOP agent will be sufficiently flexible to permit the specification of commercial aircraft mission data.
- All Mission & SOP activities must ultimately decompose into simulated operator- or equipment-based activities.

5.2.3.4.3 Interfaces

The primary Mission & SOP interfaces are with the User Interface, Mission Editor, Equipment, Updateable World Representation (UWR), Symbolic Operator Model, and Rules agents. Table 5.2.3.4-1 documents Mission & SOP interfaces with other MIDAS agents.

TABLE 5.2.3.4-1. MISSION & SOP INTERFACES.

USER INTERFACE		
Name	**Direc.**	**Description**
Activity Specifications	Input	Mission Decomposition data in activity-based format
Equipment Settings	Input	Default Equipment settings
ROUTE EDITOR		
Name	**Direc.**	**Description**
Waypoints	Input	Specification of vehicle route
Navigation Reference Points	Input	Location of Phase lines, land marks, etc.
Object Locations	Input	Location of known objects in mission environment (e.g. refueling point, tank column, downed airman)
EQUIPMENT		
Name	**Direc.**	**Description**
Equipment Activities	Input	Names and requirements of equipment-based operator activities
Equipment States	Input	Current value of equipment settings that relate to mission-based operator activity requirements
UPDATEABLE WORLD REPRESENTATION		
Name	**Direc.**	**Description**
Mission Concepts / States	Output	Required world states that the operator maintains to determine activity requirements
Operator States	Input	Current value of key operator states
SYMBOLIC OPERATOR MODEL		
Name	**Direc.**	**Description**
Activity Specifications	Output	Operator Mission decomposed in terms of activity specifications. Activity specifications include decomposition methods.
RULES		
Name	**Direc.**	**Description**
Activity Rules	Output	Rules specified in terms of valid UWR and equipment states that determine operator activity requirements.

In addition to the interfaces documented above, secondary Mission & SOP interfaces are also defined with the Statistics and Parametric Analysis Agents. The interface with the Statistics agent is used to control the collection of simulation summary performance measures and the interface

with the Parametric Analysis agent is used to modify and restore Mission & SOP data values during trade-off study experiments.

5.2.3.4.4 Capabilities

The Mission & SOP agent will collect user mission decomposition data via the Activity Editor component of the User Interface. The collected activity data will be checked to ensure that valid equipment states are specified in activity spawning and termination specifications. The Mission & SOP agent will be capable of defining new UWR state concepts (e.g., mission phase, equipment failed) as it develops the mission decomposition hierarchy. These state concepts will be tested for validity while the user defines mission activity rule specifications.

The Mission & SOP agent will access Route Editor data to retrieve additional aircraft mission information. These data will be obtained from the Route Editor during simulation initialization.

The Mission & SOP agent will operate at run time to determine operator tasking requirements. The agent will not be ticked by the Simulation Executive, but execute via send-execute messages from the Equipment and Updateable World Representation agents.

The primary data structure used by the Mission & SOP agent is the activity object. The Mission & SOP agent will also use additional objects to store the aircraft route, location of navigation reference points and locations of other simulation players.

5.2.3.4.5 Sample Operational Scenario

The Mission & SOP agent received a message from the Updateable World Representation during the Milestone 1 scenario when the operator detected the target. This message was used to determine if a change in the Operator's mission responsibilities (i.e., tasking) should occur. Likewise, the Equipment navigation page sent messages to the Mission & SOP agent regarding the state of aircraft equipment. These messages were also used to determine if changes in the operator's mission were required.

5.2.3.4.6 Issues

The major outstanding Mission & SOP issues are:

- How will the Activity Editor interact with the Mission & SOP agent in the generation of aircrew mission activities?
- What is the required level of activity / mission state estimation that should be performed by the Mission & SOP agent at run time for analysis purposes?

5.2.3.5 SYMBOLIC OPERATOR MODEL (SOM)

This section was written by Sayuri Nishimura.

5.2.3.5.1 Description

The SOM is the highest level of abstraction of the human (or automated) operator model in the MIDAS simulation system. All the psychologically interesting components such as the Scheduler, Updateable World Representation (UWR), Decide by Rules, Decide by Algorithm, Task Loading Model (TLM), Perception, Vision, and Motor are its subagents.

5.2.3.5.2 Requirements

Support the development and the implementation of at least two symbolic operators, pilot and copilot, interacting in scenarios developed and coordinated with the Program Office. The SOM, or its subagents, should be capable of representing the behavior of automated systems as well as human operators.

5.2.3.5.3 Interfaces

A SOM agent, whether it represents the pilot or the copilot, is a toplevel agent positioned directly beneath the Sim Exec agent. It gets created, reset, and ticked through the Sim Exec agent. At each simulation tick, a SOM agent sends the tick to each of its currently active activities, if any, or invokes the Scheduler to determine a new set of scheduled activities.

Input:	Scheduled activities which are output by the Scheduler Tick messages sent by the sim-exec agent. Reset messages sent by the sim-exec agent. A terminate message sent by the sim-exec agent. An abort message sent by the sim-exec agent. A setup-acquaintances message sent by the sim-exec agent.
Output:	Updated current-activities, suspended-activities, working-goals and pending-activities. Reset messages to all the subagents of the SOM. A terminate message to all the subagents of the SOM. An abort message to all the subagents of the SOM. A setup-acquaintances message to all the subagents of the SOM.
Global data:	Five activity queues which are explained in the next section.

5.2.3.5.4 Capabilities

A SOM agent keeps track of activities so that the appropriate activities are performed or interrupted at the right times, based on the scheduled activities which are output by the Scheduler. In order to perform this task, a SOM keeps the following five activity lists which are shared with the Scheduler.

1. Current-activities: This is a list of activities currently being executed. This list is updated by the SOM after the Scheduler has scheduled new activities. At each simulation tick, the SOM propagates the tick to each activity on this list.

2. Working-goals: This is a list of high-level activities which the human (or automatic) operator has begun working on but has not completed. This is one of the inputs to the Scheduler to determine scheduled activities.

3. Pending-activities: This is a list of activities which are generated during the simulation in a contingent way. This is one of the inputs to the Scheduler. This sort of activity is most likely to be generated by a daemon attached to a slot in the UWR.

4. Suspended-activities: This is a list of activities which are suspended in order to perform a newly generated activity of higher priority. This list is modified by the SOM based on the output (scheduled-activities) from the Scheduler.

5. Scheduled-activities: This is a list of activities produced by the Scheduler . The SOM examines this list and updates the current-activities list and the suspended-activities list if necessary.

The SOM invokes the Scheduler when:

- the Scheduled-activities list is empty,
- an activity is added to the Pending-activity list,
- an activity is completed and the Pending-activity list is non-empty, or
- a contingent activity produced an activity.

5.2.3.5.5 Sample Operational Scenario

In the Milestone 1 scenario, the Current-activities list of the copilot never gets empty and at each simulation tick, the SOM sends the tick to each of its current activities. SOM invoked the Scheduler when a decide-by-rule activity and a decide-by-algorithm activity generated their result activities.

5.2.3.5.6 Issues

None.

5.2.3.6 SCHEDULER

This section was written by Renuka Shankar.

5.2.3.6.1 Description

The Scheduler agent represents a computational model of human scheduling behavior. The hypothesis guiding the Scheduler agent is that human operators interacting with Human Machine Interface (HMI) designs - such as helicopter crewstations - perform scheduling using specific scheduling strategies. The principle aim of the Scheduler is to integrate such scheduling strategies into a computational model.

5.2.3.6.2 Requirements

The Z-Scheduler was developed in Phase IV of MIDAS development as a stand-alone module that interacted with the Task Loading Model but not with the simulation system. For details, see the MIDAS Phase IV Documentation. For Phase V, the principal requirement is that the the Scheduler be converted into an agent that is fully integrated with the other agents in the simulation system, communicating via the established message passing protocols. Any interaction with the user will take place through the top level MIDAS User Interface.

5.2.3.6.3 Interfaces

User Settable Parameters of the Scheduler Agent

As the users of MIDAS get ready to run simulations, they can choose the set of allowable scheduling strategies: The users will be provided a list of possible scheduling strategies that can be applied by the Scheduler and they get to select a subset of this list that they think that the operator of the HMI system would use. Currently (7/30/91) we are certain that there will be two strategies: the *minimize-time strategy* and the *balance-load* strategy. However, as stated in the Statement of Requirements, new strategies will be added as and when we learn of them from interactions with other groups in Code FL.

Initialization of the Scheduler Agent

The Scheduler agent does not contain a resource model within itself; rather, it consults with an external resource model and does resource allocation based on the external resource model's constraints. Currently, the Task Loading Model (TLM) is the resource model that is connected to the Scheduler agent. However, it is conceivable that, in the future, other resource models could be for the TLM agent, following which the Scheduler agent would then allocate resources to tasks based on the new resource model. Thus, one of the initialization functions of the Scheduler agent is to find out the nature of the resource model that it will use. Doing so keeps the Scheduler modular and not dependent on any single resource model.

Initializing the Scheduler includes the following functions:

1. Setting the scheduling strategies selected by the user.

2. Setting the expected structure for the 'resources-required' slot on every activity. Consider, for example, the structure is a vector of four (4) elements, then all the resource allocation procedures will be set up to expect four elements in the 'resources-required' slot for the activities that are to be scheduled.

3. For each resource dimension, setting the cutoff values. This information is received from the resource model. The set of cut-off values includes values for maximum capacity of each resource dimension, values for the minimum capacity, a value for the maximum comfort limit and a value for the minimum comfort limit.

4. Cleaning up the blackboard of any remaining tasks, constraints and so on.

5. Initializing the knowledge sources which includes cleaning up the local blackboards.

6. Clearing up any displays (windows etc.) that have old values in them. It is assumed that all the windows and display functionality of the Scheduler Agent will be channeled uniformly through the MIDAS User Interface agent.

Communication during simulation to/from the Scheduler Agent

Communication with the TLM

The Scheduler will communicate with the TLM to learn about resource requirements for the given activities.

Method	: Send-Query
Query	: Calculate-resource-requirement
Arguments	: Task-names
Return values	: A vector having the same structure as specified during initialization.

Communication with the SOM

The Scheduler agent will be invoked from generic function *execute* defined for the activity of type *schedule*. This will be invoked by the SOM as a function call:

(execute #<schedule>#)

The Scheduler agent will also have direct access to the following information from the SOM:

Pending-activities	: List of activities that are just created and yet to be scheduled.
Suspended-activities	: List of activities that are on hold because other more important activities took precedence.
Scheduled-activities	: List of activities that are ready to go once their place in the scheduled order becomes current.
Current-activities	: Leaf (lowest level) activities that are being executed at the current moment.
Working-goals	: Immediate parents (one-step higher-level) activities of the activities in the *current-activities* list.

5.2.3.6.4 Capabilities

The Scheduler agent is invoked through *execute* method defined for activities of type *schedule*. In the current design for the Scheduler agent, executing a schedule activity includes the following procedures:

1. Calculate the temporal horizon. Currently, we calculate the temporal horizon using the velocity of the aircraft as the single influencing factor; the greater the velocity smaller the temporal horizon and vice versa. The details of this dependency remain to be worked out.

2. Decompose the activities present in the *Pending-activities* queue until the leaf-activities are created (or the hierarchies bottom out for each of the activities in the *Pending-activities* queue).

3. Form the list of leaf-activities to be scheduled by accepting only those leaf-activities (generated from the above step) that fall into the temporal horizon.

4. Find a total order for performing the leaf-activities.

5. Translate the total order generated by the scheduling process back into constraints and cache these constraints in the respective leaf-activities.

6. Push all the scheduled leaf-activities in the *Scheduled-activities* queue of the agent.

A leaf-activity should have the following slots specified in order for the Scheduler agent to schedule it.

NAME
ESTIMATED-DURATION
PRIORITY
RESOURCE REQUIRED
TEMPORAL CONSTRAINTS

The *temporal-constraints* slot contains a list of constraints that can belong to any of the following categories:

After
During
Overlaps
Start-at
Start-by
End-at

End-by

As you can notice the above set is considerably smaller than the set that was allowed in the Phase IV version of the Scheduler agent. We have limited this set in order to fold in more psychological theories on human scheduling behavior [ref: Dr. Kevin Corker].

Further information to the Scheduler is specified in the instance of the *schedule* activity which is created for invoking the Scheduler agent. The slots for the *schedule* activity will be as shown below:

```
Strategy    ; is a vector of 2 elements— the first specifying the stage it is applicable in and the
            ; second specifying the name of the strategy. For example, to indicate that the
            ; minimize-time strategy should be applied, we need to represent that
            ; as '(resource-allocation-stage minimize-time).
Deliberation-time  ; time available for scheduling. At the present time this will be the same
                   ; as the estimated-duration for the schedule activity itself. At some later
                   ; date, we can use this time to limit the number of alternative schedules
                   ; considered by the Scheduler agent.
Estimated-duration      ; Since the schedule activity is a leaf activity an estimated duration
                        ; needs to be specified. However, this estimated duration can
                        ; be some function of the number of activities to be scheduled etc.
Resource-required       ; This is in step with the requirements of a leaf-activity. We need
                        ; some basic estimate of the amount of resources required by the
                        ; schedule activity.
```

Note: The amount of resource-required for the *schedule* activity will have an amount set aside so that a *schedule* activity can be invoked at any time during the simulation.

The Scheduler agent models the assumption that the scheduling process always keeps aside enough resources to allow for interruption. The interruption can either disrupt the scheduling process itself or can invalidate the prepared schedule depending on when it occurs in the simulation. If an interrupting activity gets created at the same time when the *schedule* activity is active the Scheduler then tries to fit this new activity in the schedule generated thus far. However, if the interrupting activity invalidates the prepared schedule, a new *schedule* activity is created for handling the interrupting activity. The Scheduler agent when invoked by this activity will then do a two-step filtering to see if this interrupting activity needs resources immediately:

1. If the priority of the interrupting activity is less than the priorities of the *scheduled-activities* then do nothing.

2. Else if the priority of the interrupting activity is less than the *current-activities* then remove the prepared schedule and schedule only the activities on the *scheduled-activities* queue leaving the currently executing activities undisturbed.

3. Else interrupt the current activities, remove the prepared schedule and start afresh all over again.

Note: Looking at the members of the *current-activities* queue at the time when a *schedule* activity gets created will indicate if the *schedule* activity will check for interruption or not. In other words, the difference between a *schedule* activity invoked under normal circumstances and a *schedule* activity invoked for handling interrupting activities is that in the first case there will be no activities on the current activities queue and not so for the latter.

5.2.3.6.5 Sample Operational Scenario

Consider that we are viewing the operations of the Scheduler agent at a moment in time when the values in the SOM are as follows:

Pending-Activities	: Act-1, Act-2
Suspended-activities	: Nil
Scheduled-activities	: Nil
Current-activities	: schedule
Working-goals	: Nil

The Scheduler agent when activated by the execution of the activity of type *schedule*, will first calculate the temporal horizon. This implies that the Scheduler agent will ask the UWR agent for the current velocity of the aircraft and depending on it, will calculate a value for the temporal horizon. The Scheduler will then take the activities on the *pending-activities* list and decompose each of them, each time checking if the decomposed activity is relevant in the temporal horizon (i.e., if the activity is likely to performed in the temporal horizon). The decomposition would proceed recursively, until the leaf activities are collected for scheduling. The Scheduler will also look to the *working-goals* list to see of there are any children that have not been scheduled and will gather such children for scheduling. Once all the candidates for scheduling have been collected, the Scheduler will then perform resource allocation for each activity based on the specified scheduling strategy.

Assume that Act-1 and Act-2 are leaf activities and hence decomposition has no effect on them. Also, assume that the temporal horizon is such that both the activities are relevant in it. Hence, once the Scheduler comes up with the total order for the activities, say Act-1 is before Act-2, this information is then stored in each of the activities and later both these activities get posted to the *scheduled-activities* queue. The execution monitoring function will then decide which activities from the *scheduled-activities* are ready to be executed at the current moment. Hence, if the SOM values are seen immediately after the *schedule* activity gets completed, they would be as follows:

Pending-Activities	: Nil
Suspended-activities	: Nil
Scheduled-activities	: Act-2
Current-activities	: Act-1
Working-goals	: Nil

5.2.3.6.6 Issues

1. Scheduling Strategy Specification: The Scheduler agent schedules a given set of activities based on a given scheduling strategy. Currently, we do not have a mechanism for setting the strategy every time scheduling needs to be performed. There are couple of solutions to this issue. The strategy could be the output of a function whose factors are user-defined priority of the activities and probably even the time horizon. Or, it could be set by the user during high level activity definition. The advantage of the second method is that in a single scheduling activity, multiple strategies may be applied.

2. Should *schedule* activity be introduced for situations where the human does not consciously engage in scheduling, but rather just follows a learned behavior?

3. How often should the Scheduler attempt to fill in the residual resources with the activities in the *suspended-activities* list? Our approach will be to check the *suspended-activities* list immediately after the current *working-goal* has been executed and before beginning to schedule other new goals. The assumption is that people usually try to complete the unfinished tasks (which were interrupted by other more important tasks) right after

completing their current important tasks but before venturing on to start some new tasks [Ref: Kevin Corker].

4. Do we have a firm decision to stay with GEST for Phase V? The management has decided that using GEST would be appropriate for Phase V.

5.2.3.7 UPDATEABLE WORLD REPRESENTATION (UWR)

This section was written by Sayuri Nishimura.

5.2.3.7.1 Description

The UWR is a subagent of a SOM agent which represents human (or automated) operator's internal representation of world knowledge.

5.2.3.7.2 Requirements

1. The UWR must provide a mechanism whereby multiple independent human agents representing flight crew, wingmen, ground control elements,etc. can access tailored or personalized information about the operational world.

2. It makes no assumption that the human operators' representations of the world are consonant with the state of the simulated world.

3. The structure that will organize the individual world representation is a semantic net in which the nodes of the net represent object structures in the simulation world and mission relevant information.

5.2.3.7.3 Interfaces

The fundamental operations of the UWR agent, such as retrieving a slot value or updating a slot value, are invoked by messages passed by other SOM subagents such as the Decide by Rules agent, the Perception agent, the Decide by Algorithm agent, the TLM agent and the Motor agent.

At the start of the simulation, the data from the Equipment agent and the Mission & SOP agent, consisting of mission goals and a collection of standard operating procedures, are uploaded into the UWR.

Input:	A message send-query from any subagent of the SOM agent A message send-update from the Perception agent
Output:	Values queried.

5.2.3.7.4 Capabilities

The UWR agent can generate a new activity through a daemon function attached to a certain memory slot. Such a daemon function may produce a new activity which is posted on to the Pending-activity list of the SOM. The daemon functions can be divided into the following two categories.

1. Slot-filling daemon: Upon receiving a message to retrieve a slot value, if the UWR does not have any value in the slot, then this daemon is invoked to fill it.
2. If-updated daemon: Upon receiving a message to update a slot value, this daemon is invoked to see whether any action should need to be taken.

5.2.3.7.5 Sample Operational Scenario

In the Milestone 1 scenario, one of the "dwell" activities recognizes a downed helicopter and stores it in a slot in the UWR. A daemon associated with the slot then gets invoked and posts a new activity, a decide-by-rule activity, which decides on how to handle target sighting.

5.2.3.7.6 Issues

The possibility of a deadlock situation arising has been discussed. For example, if a rules agent finds out that a value of an UWR slot is needed to proceed with the computation, then it sends a message to the UWR agent to retrieve the value. If the value of the slot has never been filled, then the slot-filling daemon fires and posts a new activity to fetch the value from the world. If the VACM value of the original rules activity is high, then the newly created activity does not have a chance to get executed, thus a deadlock. For Phase V, this issue will be avoided by manipulating activity priorities.

5.2.3.8 DECIDE BY RULES

This section was written by Carolyn Banda.

5.2.3.8.1 Description

The Decide by Rules agent, in combination with the activities model and other models comprising the Symbolic Operator Model (SOM), produces a model of human behavior, with the rules providing one of the operator's methods for making decisions. Broadly stated, the Decide by Rules agent uses the knowledge encoded in its rules and relevant context data from the Updateable World Representation (UWR) to decide on a course of action in a situation requiring a decision.

Daemon rules will be attached to selected attributes in the UWR, where they will detect significant changes in attribute values as the UWR is updated during the simulation. The daemons can also be used to detect anomalous conditions during queries to the UWR, such as missing attribute values.

Much of the time the Decide by Rules agent will be invoked by the daemons; at other times, rules will be invoked by normal planned decision points in the scheduled activities of the mission. Note that the daemons can also cause initiation of non decision-oriented activities, such as procedures.

5.2.3.8.2 Requirements

1. The Decide by Rules agent will model contingent behavior by deciding on a course of action based on current simulation context. [Contextual parameters to be used in the decision process will be defined by the project cognitive scientist. That is, the context to be considered in rule-based decision-making will be bounded.]

2. Rules will obtain current state and context information from the UWR.

3. There will be daemon rules which are activated by a change in the UWR state. Daemons will decide whether the information is significant enough to trigger cognitive action (i.e., decide what to do) or insignificant and therefore should be ignored.

4. The taxonomy of the rule base will start with partitions of rules relating to equipment function, mission and strategy (SOP), and external events. Further, rules within these partitions will be organized into packets of related rules.

5. The Decide by Rules agent must handle the case where it cannot decide on an activity due to lack of data or lack of knowledge.

6. A method will be developed to ensure that rule-based (and decision algorithm-generated) activities that have been scheduled to be performed are still appropriate at the time of their performance.

7. Rules will be viewable by the user through a User Interface component designed for this purpose. Selected parameter values in the rules may be modifiable by the user in a manner to be determined.

8. The decide-by-rule activity will have an estimated cognitive load and an estimated duration. A method must be developed to make these estimates, probably based on complexity of the decision to be made.

5.2.3.8.3 Interfaces

Internal (simulation-based) interaction of Decide by Rules with other simulation agents:

Inputs to the Decide by Rules agent during the simulation:

- Name of rules packet to be applied. This is determined by the caller, which is often the daemon rules.
- State information from the UWR.

Outputs from the Decide by Rules agent during the simulation:

- Rules agent places the decided-on activity in the result slot of the "decide-by-rule" activity (with which it was invoked). If the Decide by Rules agent cannot decide because of insufficient knowledge in its rules, it will pass responsibility to the decide-by-algorithm agent by returning the decide-by-algorithm activity, along with the available options (which the decide-by-algorithm agent requires as input).

Initialization: The rule base will be uploaded from the Mission & SOP agent, along with other mission activities. In addition, the Decide by Rules agent will do the normal initialization procedures done by simulation agents, such as setup acquaintances. Any local, rules-related attributes, such as statistics on rules operation, will also be initialized.

For the daemons, during initialization thresholds of significance will be set up for selected UWR attributes.

User interaction with the rules and daemons:

The rules will be viewable by the user through an interface developed for this purpose. Values of selected parameters in the rules may be modifiable by the user.

5.2.3.8.4 Capabilities

The Decide by Rules agent consists of a collection of rule packets and is invoked during the simulation by the "decide-by-rule" activity for selected types of decisions. Rules are of the form:

IF <condition-list> THEN <action-list>

When activated, Decide by Rules will obtain current context information from the UWR by sending it a query message.

GEST is proposed as the inference engine to provide rule-based decision making. With GEST, the rule base can be partitioned and organized hierarchically according to subject.

In the case where the Decide by Rules agent can't decide on an activity, two possible reasons are:
1) Missing information in the UWR; that is, the desired UWR slot does not contain a value.
2) Rules contain insufficient knowledge for current situation.

In case 1), the Decide by Rules agent will wait for the UWR to return a value. UWR daemons will detect request for missing information and generate necessary activities to obtain the information. Rules will continue when information is returned. See the UWR description for more details on this issue. In case 2), the Decide by Rules agent will transfer responsibility for choosing the next activity to the Decide by Algorithm agent.

For the daemons, thresholds of significance for various attributes in the UWR will be maintained. During UWR update, daemon rules will check new values and trends against these thresholds to decide if the change is judged to be significant enough to require action.

5.2.3.8.5 Sample Operational Scenario

A sample scenario showing decisions by the Decide by Rules agent is the following. (The details of the example are for illustration only and will change as more knowledge is gained about what pilots REALLY do.)

Situation: While enroute to a waypoint during a mission, the aircraft experiences a problem with hydraulic pressure. The pilot (or copilot) first detects an advisory, then a caution, and a warning that hydraulic pressure is too low. Some applicable rules, belonging to the rule packet for the hydraulic system, are:

IF hydraulic-low-pressure-advisory
THEN increase-monitor-rate-to-advisory-hydraulic-monitoring-rate

IF hydraulic-pressure-low-caution
THEN increase-monitor-rate-to-caution-hydraulic-monitoring-rate

IF (AND hydraulic-pressure-low-warning
in-friendly-territory)
THEN initiate-hydraulic-pressure-low-emergency-procedure

IF (AND hydraulic-pressure-low-warning
in-hostile-territory
probability-of-safe-return-to-friendly-territory is high)
THEN (AND return-to-friendly-territory
initiate-hydraulic-pressure-low-emergency-procedure)

IF (AND hydraulic-pressure-low-warning
in-hostile-territory
probability-of-safe-return-to-friendly-territory is not high)
THEN (AND tell-copilot-(or-pilot)-to-notify-commander
initiate-hydraulic-pressure-low-emergency-procedure)

Daemons attached to the hydraulic system attributes in the UWR would detect the advisory, the caution, and warning conditions upon updates to the UWR. The daemons would create a decide-by-rule activity with the appropriate rule packet ID (here, "hydraulic system") and an appropriate priority and give this activity to the Scheduler.

When the decide-by-rule activity comes up on the current-activities queue and is performed, the Decide by Rules agent is activated. The Decide by Rules agent then invokes GEST to make the decision (assuming we use GEST). The resultant activity or activities are created and placed in the result slot of the decide-by-rule activity, which will cause them to be scheduled in the same way as the result of a Decide by Algorithm activity is scheduled(see next section).

5.2.3.8.6 Issues

1) Rule-based decisions are made at varying levels of abstraction, ranging from tactical decisions at a high level to low level decisions, such as choosing which of the alternate paths to take to carry out a procedure. We may not want to use a full-blown inference engine for low level decisions; a contingent or choice activity type may serve better in this case. It is proposed that the inference engine GEST be used for the high level decisions and that low level decisions, such as alternate paths through a procedure, be handled by a conditional activity type, to be defined. If this approach is adopted, we need to develop criteria for deciding when to model rule-based decision-making using rules and when to use regular activities of type conditional for this purpose.

2) Assuming GEST is used, the inference engine must be separated from the GEST interface; this work is also be required by the Scheduler, and has been completed.

3) With respect to item (6) in the Rules Requirement Section, a method needs to be developed to check to see if scheduled activities are still appropriate at time of performance.

4) A method to estimate duration for decide-by-rule activities has to be worked out. A way to do this should become more clear as we progress into more detailed design.

5.2.3.9 DECIDE BY ALGORITHM

This section was written by Renuka Shankar.

5.2.3.9.1 Description

In general, decisions are made when deviations from the expected state (triggered by anomalous situations) are noticed. The main goal behind making such decisions is to bring the situation back to an expected state. Although decision making can be performed in many ways, in this document we describe the prescriptive decision making process which is a computationally deterministic process.

Typically, the prescriptive decision making process is a two-step process. First, all the options that can address the state transition (from the unexpected state to the expected state) are generated and next, an appropriate option is selected from these generated options based on a decision strategy. However, for the purposes of this paper we will be concentrating on modeling the second step of the prescriptive decision making process.

Research in the past indicates that individuals apply different strategies while selecting a course of action among a whole range of possible courses of actions [J.R. Bettman, E.J. Johnson, and J.W. Payne, "A Componential Analysis of Cognitive Effort in Choice," ONR - Technical Report 88-1]. Modeling this result is one of the major aims of the prescriptive decision making agent., which has

come to be called the Decide by Algorithm agent. Strategies applied to select the best option are primarily deterministic algorithms that are used to compare the options based on their attribute values. Given a set of options, the time needed to select the best option and the selected option itself might vary with the algorithm applied for selecting the best option.

The Decide by Algorithm agent models six algorithms each of which can be used for selecting the best option. These are the:

1. *weighted additive algorithm*
2. *equal weighted additive algorithm*
3. *lexicographic algorithm*
4. *elimination by aspect algorithm*
5. *satisficing conjunctive algorithm*
6. *majority of confirming dimensions algorithm.*

All these algorithms differ in the way in which the "goodness" value for an option is calculated; each uses a different combination of the attribute values, attribute weights and attribute cut-off values for calculating the "goodness" value.

5.2.3.9.2 Requirements

1. Attach a cost factor for the decision making process — We know that deciding a course of action from many alternatives takes time and effort and we would like to capture this effect in the MIDAS system. In our design, the decision agent is invoked through a specific activity called *decide-by-algorithm*. Immediately such an activity is spawned (created), the decision agent is invoked to calculate the time required to perform the decision. The time required is calculated by looking up the data tables indexed by number of attributes and number of options.

2. Model algorithmic decision making process — From the Statement of Requirements document, we learn that in the event there are no rules to trigger a recognition-primed decision process, "a more computationally intensive decision process" needs to be invoked. As stated above, we model such decision process using algorithms that have been identified in individuals during decision making.

3. Output of the decision making process should be an activity —The decision agent is invoked using a *decide-by-algorithm* activity which has a slot containing the available options. Each option has attribute values associated with it and each attribute has weights and cut-off values associated with it. In our current design, once the best option is chosen (by applying the specified algorithm), it is transformed into an activity and placed in the *'result'* slot of the *'decide-by-algorithm'* activity; thus meeting requirement 3.

5.2.3.9.3 Interfaces

SOM----> Decision

The decision agent will be invoked from generic function *execute* defined for the activity of type *decide-by-algorithm*. This will be invoked by the SOM as a function call:

(execute #<decide-by-algorithm>#)

The above function will be invoked only during the last tick of the decide activity because we do not have a handle on what parts of the decide process constitute one-tick worth of processing. Hence, although in reality all of the decision making is done in the last tick of the decide-by-

algorithm activity, from the simulation standpoint it will be as though the decision making lasted for the whole "n" ticks (where "n" is the estimated duration of the activity expressed in tick) where for the first "n-1" ticks no significant work is performed and only at the "n"th tick the complete decision is made.

Decision----> SOM

The decision agent will update a slot *result* of the #<decide-by-algorithm># to point to the chosen activity instance. The SOM will then add this chosen activity into the *pending-activities* queue.

5.2.3.9.4 Capabilities

The Decide-by-Algorithm agent is activated whenever the activity of type *decide-by-algorithm* begins execution. The definition of the 'decide-by-algorithm is shown below:

Decide-by-algorithm
- Possible-options
- Available-options
- Algorithm-chosen
- Estimated-duration
- Attribute-names
- Attribute-weights
- Attribute-cut-off
- Result

Since we do not have any process or domain models to generate the possible options during run-time, these options need to be specified by the user *a priori*. However, we provide means for creating only a subset of the possible options by evaluating user-defined functions at run-time, thus making the generation of the options context-dependent.

The slot *Possible-options* will contain a list of options, each option being a symbol for which a class definition exists. This class definition should inherit from the class *option-for-decision* which is defined as below:

Option-for-decision
- Attribute-values
- Applicability-function

Every time a *decide-by-algorithm* activity is created, its initialization involves two steps:

1. Generating *available-options*: This is done by instantiating every possible option and only if its *applicability-function* holds will it be entered in the *available-options* list.

2. Setting the *estimated-duration*: Since the number of *available-options* is known and since the *algorithm-chosen* is known, a table look-up will give the value for the *estimated-duration*.

On invoking the *execute* method on the *decide-by-algorithm* activity, the *algorithm-chosen* is applied and the resulting selected option is then instantiated as an activity which is inserted in the *result* slot.

The activity in the *result* slot has to be scheduled before getting executed. Unlike the normal way in which all activities get into the *pending-activities* queue to be recognized by the Scheduler, the

activity in the *result* slot is handled in a slightly different manner. The *result* slot activity will be added on as a child of the parent of the *decision-by-algorithm* activity. Hence, typically a *decide-by-algorithm* activity will have a parent of type *sequential-activity*. When the *sequential-activity* was first decomposed by the Scheduler, the only child would have been the *decision-by-algorithm* activity. The Scheduler would have scheduled the *decide-by-algorithm* activity, which would have subsequently been executed and while being executed, its parent (*sequential-activity)* would have been on the *working-goals* list. However, the parent activity does not get out of the *working-goals* list even after the *decide-by-algorithm* activity is complete, since at the time of completion of the *decide-by-algorithm* activity, the parent would then beget a new child (activity in the *result* slot) which will then be subsequently scheduled by the Scheduler agent (since the Scheduler agent also checks the remaining unscheduled children of the working goals as candidates for scheduling).

5.2.3.9.5 Sample Operational Scenario

In the Milestone 1 snippette, a *decide-by-algorithm* activity is created when the copilot wants to decide whether to navigate with or without assistance. Hence, the *options-for-decisions* are *navigate-with-assistance* and *navigate-without-assistance*. The *weighted-additive* algorithm when applied, selected the *navigate-without-assistance* option.

5.2.3.9.6 Issues

1. Lack of knowledge-rich models to generate the options for decision making: Currently, in the MIDAS system we do not have any process models that have enough domain knowledge for generating the options. Hence, all the possible options for the decide-by-algorithm activity must be known a priori (before the simulation). However, we do have a simple model for instantiating a subset of the possible options based on some applicability conditions to give rise to the available options.

2. Lack of data beyond decision making with six (6) alternatives: The tables for retrieving the time required for decision making apply only to decision making with two to six alternatives. I am not certain how to extrapolate these times to decision making with more than six alternatives. Hence, we might have to limit the alternatives (options) of decide-by-algorithm activities to six.

3. Lack of data beyond four attributes: Here again we might have to limit decision making across only four attributes since this is the limitation imposed by our current data.

4. User-definable *decide-by-algorithm* activities: I believe that the user should also be given the freedom to specify whether a decide activity in the activity decomposition belongs to the category of *decide-by-rule* or *decide-by-algorithm*. This would be advisable for two reasons. First, we do not have the capacity in the Decide by Rules agent to generate the *options-for-decision* which is necessary for the Decide by Algorithm agent. Second, it is reasonable to assume that, since it is the user who inputs the rules in the Decide by Rules agent, s/he is aware of the limitations in the rule knowledge and hence, while adding a decision activity knows *a priori* that no rules can make this decision and that this decision would eventually be handed to the Decide by Algorithm agent for the final answer. Hence, s/he can decide up front that the decision activity should be of type *decide-by-algorithm.*

5. Default decision algorithm? If no algorithm can be specified for a *decide-by-algorithm* activity, can one of the algorithms be considered as the default? If yes, should we use the algorithm that takes the minimum time, maximum time or some intermediate time? Also, if there is to be a default algorithm, we would also require default values for the attribute values and their weights.

5.2.3.10 TASK LOADING MODEL (TLM)

This section was written by Lowell Staveland.

5.2.3.10.1 Description

The Task Loading Model (TLM) is a MIDAS agent that interacts with other MIDAS agents and Activities in order to classify the activities with attributes of four psychological dimensions (Visual, Auditory, Cognitive, Motor). The classifications are used to generate estimates of the psychological loads in each of the dimensions that would be imposed on an operator of the simulated design. Designers interact with the TLM via a MIDAS interface to classify tasks and predict task loads. The TLM adjusts the classifications and loads based on simulation state values that are obtained by accessing the slot values of specific agents in the simulation.

5.2.3.10.2 Requirements

The current TLM will be incorporated to function in-line in the simulation and interact with the specific agents.

1) Task loads will be applied to individual activities in a pre-simulation initialization.
2) A classification of the loading imposed by the performance of simulated concurrent activities will be provided.
3) The classification methods will be applied in-line in the simulation and will be responsive to the task ensemble and to the context in which those tasks are to be performed.
4) Interface for user modification and/or examination of task loads will be provided.
5) The mechanisms for providing load estimates at multiple levels of resolution will be provided.

5.2.3.10.3 Interfaces

The following sequence delineates the communications between the TLM and its acquaintances that provide the classification data. The term slot-value in the sequence stands for information requests that haven't yet been defined.

TLM Communications:

1) Scheduler sends send-query to TLM
2) TLM requests slot-value in UWR
3) TLM requests slot-value in Decide by Rules
4) TLM requests slot-value in Decide by Algorithm
5) TLM requests slot-value in Copilot-SOM
6) TLM requests slot-value in Pilot-SOM
7) TLM requests slot-value in Activity
8) TLM requests slot-value in Perception
9) Analysis to TLM
10) User Interface to TLM

Scheduler to TLM

Scheduler sends TLM a list of pending or current activities whenever the list needs to be scheduled.

message-type: query
arguments: (current-activities pending-activities)

TLM to Decide by Rules

-SGI and Symbolics connected, Symbolics Standalone

message-type: get slot-value
Slot-names: Number-of-rules-invoked, decision-time-required

TLM to Decide by Algorithm
-SGI and Symbolics connected, Symbolics Standalone
message-type: get slot-value
Slot-names: decision-time-required, algorithms-available, current-decision.
Potential slot-names: algorithm-used

TLM to Copilot SOM
-SGI and Symbolics connected, Symbolics Standalone
message-type: get slot-value
Slot-names: pending-activities, current-activities, scheduled-activities

TLM to Pilot SOM
message-type: get slot-value
Slot-names: pending-activities, current-activities, scheduled-activities

TLM to Activity
message-type: get slot-value
Slot-names: Priority, continuation, object-in-view.

TLM to Perception
message-type: get slot-value
Slot-names: fixation-site, in-cockpit, out-of-cockpit

Analysis to TLM
-requests the primary output

The primary output of TLM is information that describes the simulated operator's imposed task load through time:

1) Distribution of Visual, Auditory, Cognitive, and Motor loadings
2) Distribution data regarding active clash pairs and/or operator component resource attribute demands (e.g., visual scan, right hand.
3) Set of activities generating the current load and resource demands (potentially just during overload conditions)

If activity classification becomes context sensitive, then the resource channel assignments (e.g., left hand) and state conditions leading to the relevant assignments (e.g., right hand busy) will be collected.

Interface to TLM
-requests the primary output
The interface will be used by the design analyst to classify the activities during initialization of the simulation. It will also be used to sample the primary output on an as needed basis.

5.2.3.10.4 Capabilities

The primary input to the TLM is an array of information provided by both a design analyst and by simulation agents. During initialization, activity classifications will be attached by the design analyst to the leaf-node activities residing in the Mission & SOP. The Scheduler will query the TLM for the list of resources and their structure. This allows the output of any resource model to be used by the Scheduler in the event that the TLM is replaced or radically altered. Loads will be

calculated for a single activity during initialization and they will be used by the Scheduler for scheduling.

Based on the element definitions list and the Boeing Human Engineering Design and Analysis Document - Operator (HEDAD-O), templates will be defined in the equipment model that will store information that the TLM requires.

The classifications for each activity will currently be stored in a hash table and not in a slot in the activity. Future developments will encode the necessary slots. The load values will be calculated during initialization after the classifications by invoking the TLM as an initialization procedure. An activity's load value will be recorded in its load slot.

During the simulation, the classifications will be adjusted to reflect the simulated environment in which the activity is executed. The classifications for concurrent activities will be adjusted through methods that access the slot-values for agents listed in section 5.2.3.10.3. These methods that access the necessary slots will be invoked at each tick by a tick-message from the Sim-Exec. Other methods and functions will operate on the slot-values that are returned to adjust the classifications. Concurrent task loads will be calculated during scheduling and during execution. The adjusted classifications, the concurrent load values, the names of the concurrent activities and the time of execution will be recorded as separate defstructs at each tick. All of the defstructs for concurrent activities will be collected and biographed in a concurrent load history slot in the TLM agent.

5.2.3.10.5 Sample Operational Scenario

To illustrate the TLM, a snippette of a scenario with two high level activities will be used. *Scan-terrain-with-pattern* and *navigate-to-target-position* are decomposed and their initialization procedures are called. The load initialization procedures checks the leaf-activity (e.g, *look-at* and *reach)* load slots for load values. If there are no load values, then the procedures check the hash tables for their classifications. If there are no classifications then either the activities are submitted to an design analyst through the interface for classification or the classification methods are invoked that access the slots in the simulation agents containing classification data. The classifications for the activities *look-at* and *reach* are used to generate the load values that are written to their load slots. When *look-at* and *reach* are posted to the Pending-activity queue, the Scheduler collects them for scheduling. The Scheduler then invokes the TLM on *look-at* and *reach* to calculate their concurrent loads. If the loads are acceptable (within the ranges set by the Scheduler's load balancing strategy), then *look-at* and *reach* are posted to the Scheduled-activity queue. When *look-at* and *reach* are executed they are posted to the Current-activity list. At every tick the TLM collects the list of Current-activities and adjusts their classifications to reflect the simulated environment. The classifications are adjusted by invoking the classification methods that access the slots in the simulation agents containing classification data.. *Look-at* and *reach* are re-classified to reflect that they are being performed concurrently. The new classification is used to generate their concurrent load values. The adjusted classifications, the concurrent load values, the names of the concurrent activities and the time of execution are recorded as a defstructs. This defstruct is biographed in the concurrent load history slot in the TLM.

5.2.3.10.6 Issues

The TLM data collected for the set of activities currently being ticked must be partitioned from that computed while generating constraint information for the Scheduler.

How should scheduling and planning activities that take time and load be classified? TLM was not designed for this kind of classification. It's probable that the two activities would be equal given the current implementation of the TLM. However, maybe several types of scheduling and planning activities could be defined, each with a fixed load, which would capture some variability

among various planning and scheduling activities. Additionally, a few simple rules that modified their classifications based on limited contexts could be implemented to increase the amount of variability captured.

Another issue is that of classifying activities at multiple levels of resolution. Even though activities are loaded at the leaf node level, the resolution of the leaf nodes differs. To overcome this problem, the differing levels of activities could be roughly classified according to the number of dimensions sufficient to classify the demands imposed by that level. For example, four levels of resolution corresponding to the number of dimensions could be implemented. The basic level level 1, would require only one dimension for classification, level 2 would require two dimensions, level 3 would require three dimensions and level 4 would require four dimensions. For example, a *dwell* activity could be classified as only a visual task, because it primarily imposes visual demands, a *decide-by-rule* activity as only a cognitive task because it primarily imposes cognitive demands. However, a *call-new-course-to-pilot* activity may impose demands on all four dimensions requiring classifications and load estimates on all four dimensions.

5.2.3.11 PERCEPTION

This section was written by Renuka Shankar.

5.2.3.11.1 Description

The Perception agent is the SOM agent's mechanism for being aware of the elements of the environment through physical sensation. It is planned that a Visual Attention Model developed by Dr. Roger Remington and Dr. Jim Johnston of Code FLM will be incorporated into MIDAS in the near future. At present the Perception agent serves as a place holder for integrating this or other theories of attention. The attention model to be included in the Perception agent acts as a filter for the information streaming into the Perception agent from the outside environment via the physical agents. Hence, the Perception agent can be viewed as an interface between the agents modeling physical sensations (like vision and motor agents) and the agents modeling cognitive processing (like the UWR).

5.2.3.11.2 Requirements

- Generate commands for the relevant agents based on the information in the activity.
- Update the UWR every time the Perception agent is updated.
- Other requirements regarding what types of attention mechanisms influence perception need to be worked out.

5.2.3.11.3 Interfaces

SOM----> Perception

- Perception is activated by sending *send-execute* message to the perception agent with the *perform* method and the activity instance as arguments.

Perception----> UWR

- Updates the slot values of objects in the UWR agent using *send-update* to the UWR with the following as the arguments
  #<equipment-object#>
  slot-name

slot-value

Perception----> Pseudo-Vision

- Based on the information regarding fixation site present in the activities, the Perception agent sends out commands to the Pseudo-vision agent to fixate on the specified site.
- Communicates with Pseudo-vision agent using *send-execute* message with *fixate* as the procedure to be executed and with fixation site as the argument.

5.2.3.11.4 Capabilities

Most of the capabilities in the perception agent comes from integrating the attention model into it. Although the interface design for integrating the attention into perception agent is currently in progress, I propose that the attention model can be used in two modes in our simulation:

1. Given the time available for performing a leaf-activity, the attention model can specify the information that is gleaned in that time frame.
2. Given that a particular type of information is needed for a leaf-activity, the attention model can specify the time taken to glean such information.

5.2.3.11.5 Sample Operational Scenario

Currently, the interface between Visual Attention model and the Perception agent is being worked out and once this is completed a sample operational scenario can be formed to explain the workings of the Perception agent.

5.2.3.11.6 Issues

- Attention Model: Since the Visual Attention model will be serving as a filter for the Perception agent, all the inputs required for this model must also be made available to the Perception agent. However, the exact specifications for these inputs to the Visual Attention model remains to be worked out.
- Communication from Perception to UWR: Currently, information flows from the Perception agent to the UWR agent informing the UWR of the state changes in the environment. However, when the Visual Attention model is integrated in the MIDAS system, it also needs information from the UWR. Since the attention model is a part of the perception agent we should allow for this communication link in the Perception agent.

5.2.3.12 VISION AGENT

This section was written by Mike Prevost.

5.2.3.12.1 Description

The Vision agent performs the functions that one would normally be associate with the human eye. It can be thought of as a sensor model that prefilters visual information, similar to the eye, before any cognitive processing takes place. It does not determine where to look only how to move to achieve the a given fixation and what can be seen given the current situation.

5.2.3.12.2 Requirements

The Vision agent is required to:

1) respond to fixation requests and compute new eye and head coordinates that would accomplished the fixation.

2) be capable of calculating what is in the field of view at a given tick.

3) initially provide unfiltered vision information to the Psuedo Vision agent, possibly moving to a probabilistic model of target detection.

4) allow for the modification of Vision agent parameters such as, saccade speed and pattern or fixation duration.

5.2.3.12.3 Interfaces

At **initialization** time the vision agent will query Jack to set the starting eye and head coordinates and then update the UWR via Pseudo Vision and Perception with these new coordinates.

Inputs to the Vision agent are requests to fixate sites or objects, requests for field of view information and requests for setting dwell times. The requests are all sent by Pseudo Vision. Input also comes from Jack in the form of current fixation site updates. The Vision agent also receives a tick message from the SGI Simulation Executive.

Outputs from the Vision agent are new eye and head coordinates which are sent to the Pseudo Vision agent when passing through an update from Jack, or to Jack when requesting a new fixation site. Field of view information is calculated by requesting data from Terrain, Cockpit and Environmental objects. By knowing the head and eye coordinates and attaining the data from the above agents it is possible to build a list of the objects that are in the field of view. This list would usually be sent to Pseudo Vision.

Also output from the Vision model will be error , warning and status messages. They will be via the system defined report methods. Finally, output data for biographical purposes will be sent to the biographer.

5.2.3.12.4 Capabilities

Fixate request:
The Vision agent will be expecting either a 3 D coordinate or a symbolic name of an object to fixate, and the dwell time. The model will then compute the transformation that will move the visual axis to the desired site. If the angular size of the move is greater than some predetermined value, e.g., 17 degrees, the model will compute tick coordinated, interpolated, intermediate fixation points. A list of these fixation points and head movements will be maintained internal to the agent. As Jack reports the reaching of the commanded position the Vision agent will take the next point on the list and update Jack's fixation point. This will continue until the list is empty. The vision model must be given a new fixation point before Jack's fixation point will be changed so Jack will continue to fixate the last position until given a new one. If a new fixation point is sent while still processing a previous fixation point list then the list is cleared and a new list is created.

Field of view request:
As part of any field of view (FOV) query the Vision agent filter settings and mode must be sent. Filter settings, such as radius of FOV, depth of field filtering, or minimal pass value for

size/contrast target can be programmatic changes to effect what is returned from the FOV query. The filter mode has three values: ALL, FILTER, CLIP. ALL will return everything within the field of view without processing. FILTER will still return every object in the FOV but will modify attributes of the objects based on current filter settings. CLIP, will only return objects that pass the filter setting and can result in a considerable performance improvement. For example, if the fixation point is local and depth of field filtering is true and CLIP is true, then the Vision Model will query only the cockpit model and return only those objects within the FOV and depth of field. The objects returned will be sent over as a list data structure. The exact information such as, angular size or distance from the fovea, can be added as the need by other models in the system becomes defined.

5.2.3.12.5 Sample Operational Scenario

A simple example of how the vision information would flow in the simulation follows:

1) Perception makes a request to fixate the MultiFunction Display (MFD).

2) A request is sent to Vision via Pseudo Vision to change the fixation site with dwell time set for 300 ms.

3) Vision looks up the name (MFD) in the symbol table and calculates the distance between the current fixation site and commanded fixation site and computes the number of intermediate fixation points and head positions. A list of these points is built and the old list is then set to this list, there by interrupting any on going fixation processes.

4) When Vision is ticked it sends the next site in the fixation list to Jack unless it had been previously sent (multi tick dwell time). If the dwell time is greater than one tick then its dwell is decremented by one tick else it is removed from the list. If there are no sites left in the list then the next site in the default scan list is sent to Jack.

5) When Vision's fixation site is updated by Jack it is compared to the last commanded site to be sure they match. If they don't match a warning is sent via the system error reporting facilities.

5.2.3.12.6 Issues

1) Standardized coordinate space. Many of the MIDAS components will be specifying positions in geometric coordinates. In order for the simulation to work correctly there must be a standardized coordinate system.

2) In order for the Vision model to determine if an object is in the FOV, a line of site calculation must be performed. To avoid computational inefficiencies and communication bottlenecks , the terrain model will be given two 3 D points and will return true if there is a line of sight between then and false otherwise.

3) Filter settings and modes. It would be preferable not to determine FOV and line of sight for every object in the simulation every time a FOV is computed. The filter settings and modes are a way of allowing programmatic modification in a way that can be more in tune with the informational needs of the requester. At first FOV information will only be given when requested. If it turns out that this information is requested at least once per tick then the UWR will be updated once per tick. The fixation point in the current design can only change once per tick. When the attention model get implemented this will require the FOV computation for all objects. The filters will then only specify perceptual attributes for the objects.

4) Symbolic names must be coordinated. In order for a remote machine to ask the vision agent to fixate an object by name, there must be a correspondence between that name and the symbol that represents that object.

5) Linear interpolation for intermediate fixation points. Humans coordinate head and eye movements in complex ways. The first approximation to this behavior is to simply move the head and eyes together for fixations and interpolate for moves too great to achieve in one tick interval. Later a more accurate model can be substituted.

5.2.3.13 MOTOR

This section was written by Lowell Staveland. Further design and implementation will be done by Greg Smith of Select Systems Analysis.

5.2.3.13.1 Description

The Motor agent will direct and record motor movements when integrated with the MIDAS anthropometric model, route motor response commands to the appropriate agent(s), and implement Fitts Law on the Symbolics. The Motor model currently is synonymous with the Pseudo-Jack agent; the motor model's functionality is encoded within Pseudo-Jack

5.2.3.13.2 Requirements

The current requirements will guide implementation for a model of discrete motor response movements:

1) Fitts law algorithm describing reach time/accuracy tradeoffs in human motor response will be developed and encoded in a Symbolics-based motor response model. The coefficients that are used will be the same kind and number as those used in Jack. The reach sites and distances will come from Jack. The target widths will come from a query sent to the the piece of equipment at the reach site.

2) Reach times will be calculated to support activity scheduling and performance timeline development.

3) Reach times will be estimated using a Symbolics-based model for conceptual, gross level simulation.

Phase VI plans are to include continuous motor response control models.

5.2.3.13.3 Interfaces

At the present time, all of the functionality of the Motor agent is imbedded within the Pseudo-Jack agent. Consequently, all of the motor model's interfaces are discussed in terms of communications between Pseudo-Jack and its acquaintances.

Motor and Pseudo-Jack (PJ) Communications:

1) PJ sends execute movement to Equipment agent
2) PJ sends execute movement to Jack
3) Jack sends update movement to PJ
4) PJ sends update movement to Motor model

PJ sends execute movement to Equipment agent

message-type: send-execute method
arguments: a motor movement activity

PJ sends execute movement to Jack
PJ sends Jack the information that Jack needs to execute reaches.
message-type: send-execute method
data-structure: (figure rpc:c-string)
(reachgoal rpc:c-string)
(endeffector rpc:c-string)
(side rpc:c-string)

Jack sends update movement to PJ
Jack updates PJ with the new x, y, z positions for the hands and feet which PJ then stuffs into the motor model.
message-type: send-update
data-structure: (xhandpos single-float)
(yhandpos single-float)
(zhandpos single-float)
(noofticks rpc:integer-32)
(side rpc:c-string)
(figure rpc:c-string)

PJ sends update movement to Motor
-Motor is inside PJ
-PJ stuffs the new x, y, z positions for the hands and feet into the slots in PJ agent
message-type: (setf slot-name new-slot-value)
data-structure: (figure rpc:c-string)
(reachgoal rpc:c-string)
(endeffector rpc:c-string)
(xhandpos single-float)
(yhandpos single-float)
(zhandpos single-float)
(noofticks rpc:integer-32)
(side rpc:c-string)

5.2.3.13.4 Capabilities

All of the functionality of the Motor response model is embedded within the Pseudo-Jack agent. Consequently, all of the Motor agent's capabilities are discussed in terms of flow of control and data between Pseudo-Jack and its acquaintances.

Pseudo-Jack contains all of the slots necessary to record the motor response activity. The positions of the hands and feet are used in conjunction with the equipment reach sites to calculate reach times. Both position and site information is stored in the slots. There are methods associated with the slots that encode the Fitts Law algorithms and do the calculations. These slots and their associated methods and functions comprise the motor model.

Pseudo-Jack acts as the routing agent that receives and sends the execute movement messages to agents that require them. PJ's acquaintance list names the agents that send and receive messages. An activity will have a classification assigned by analyst before initialization, but after an activity editor has been invoked (TLM needs this data)

The Scheduler will resolve motor conflicts between activities.

5.2.3.13.5 Sample Operational Scenario

The following numeric sequence delineates the flow of the data-structures involving Pseudo-Jack and the Motor agent, from initialization to completion for an activity-object. Execute-activity-push-button is used as an example.

1) Push-button originates in the Mission & Sop agent and is uploaded into the UWR during initialization. When the push-button activity is spawned, its movement parameters (figure reachgoal endeffector side) are specified. Pseudo-Jack is sent send-execute activity-push-button.

2) Pseudo-Jack creates the data-structure that Jack requires:
jack-execute-reach-params (structure (figure rpc:c-string)
(reachgoal rpc:c-string)
(endeffector rpc:c-string)
(side rpc:c-string)

PJ then sends this structure to Jack on the SGI via a send-execute method.

3) Jack updates PJ by sending a send-update data-structure:
jack-update-touching-at-params (structure (xhandpos single-float)
(yhandpos single-float)
(zhandpos single-float)
(noofticks rpc:integer-32)
(side rpc:c-string)
(figure rpc:c-string)

4) PJ updates the acting Motor agent by stuffing the values returned by Jack into the appropriate slots in PJ: (figure reachgoal endeffector side xhandpos yhandpos zhandpos noofticks). The methods that calculate Fitts Law will always access current slot values to determine reach times whenever PJ receives the appropriate send-execute method.

5) Pseudo-Jack sends a send-execute activity-push-button to the Equipment agent. The equipment agent then updates itself.

5.2.3.13.6 Issues

None.

5.2.3.14 ANTHROPOMETRIC MODEL FOR SIMULATION (JACK AGENT)

This section was written by Christian Neukom.

5.2.3.14.1 Description

Jack is an anthropometric modeling system developed through grants with Dr. Norman Badler at the University of Pennsylvania. The newest version of Jack, 5.1, was released in May 1991. This software performs two separate functions in Phase V MIDAS:

1) The latest version of Jack with enhancements developed by A^3I, as described in this document, will be available to the MIDAS user as an interactive tool for performing analyses of reach and fit and certain other MIL-STD-1472 geometry-based design checks.
2) An earlier version of Jack is incorporated into VEST and animated to provide a graphical representation of the human operator during the simulation.

The enhancements to Jack as an interactive design tool to be developed during Phase V of the A^{3}I Program are described in section 5.1.1. The integration of Jack software into the MIDAS simulation system as the Jack agent is described in this section.

5.2.3.14.2 Requirements

In Phase V, the simulation will be tick-based, i.e., the state of the simulation will be evaluated at each time interval. The requirement for the Jack agent is to provide a 3-D representation of the human figure(s) during the simulation and to keep movements within the joint limits of the particular human figure(s) in use.

Jack version 4.2 satisfies the anthropometric requirements for the simulation in Phase V and will be used in order to be independent of the continuous upgrades. The older version has also the advantage of being simpler and thus provides faster processing speed - an important factor in the simulation. However, it will be possible to incorporate a newer version if our requirements change.

5.2.3.14.3 Interfaces

Jack will interact directly with three other agents during the simulation: SGISimExec, Motor and Vision. The Motor agent controls hand and foot movements from the Symbolics side and the Vision and SGISimExec agents reside on the SGI machine. The communication with these agents takes place via RPC calls.

Communication with SGISimExec

Input:	tick number, reset
Output:	none

Communication with Pseudo-Jack

- Reach:

Input:	figure name, reach goal (x,y,z), endeffector (palm,index finger, foot), side of endeffector (left,right), reach class
Output:	figure, success/fail for each endeffector

- Report Posture:

Input:	figure
Output:	figure, position (x,y,z) of palm, index finger and foot for each side,

Communication with Vision

Input:	fixation point
Output:	head position/orientation

Initialization

Initialization of the pilot figure will occur through VEST and via the representative agents before the start of the simulation. During initialization, the figures are created and the user has a chance to reposition the figures if necessary. Various parameters such as figure percentile and color of the figure can be selected.

5.2.3.14.4 Capabilities

An inverse kinematic algorithm is used to do various reaches and changing posture. The graphical display of the environments and the Jack figures will be through VEST.

5.2.3.14.5 Sample Operational Scenario

A typical Phase V scenario will employ two human figures, a pilot and a copilot. Both figures will be driven by an instance of a Motor agent. In this phase, the focus will be on the Copilot. The Motor agents will calculate reach timing using Fitts' law and break up the movement into a number of steps. The Jack agent will perform the reach and report to the Motor agent if the reach was successful. If the reach could not be accomplished, the Motor agent queries Jack agent for the present posture.

Vision will send requests to the Copilot to fixate his eyes on a site. The message to Jack will be a new fixation site. Jack will move the figure's head in the direction of the site and then return the new head position /orientation to Vision. A trace from the figure's eyes to the fixation point will be displayed. Initially, fixation will be instantaneous and only involve head movement but later extensions may include eye - head movement.

5.2.3.14.6 Issues

None

5.2.3.15 EQUIPMENT MODEL

This section was written by David Bushnell.

5.2.3.15.1 Description

The Equipment Model defines the way equipment components in a MIDAS simulation operate. It is designed to let a user easily develop specific components from more generic ones as well as build up complex components from simpler ones. It is what drives the graphics displays during a simulation run. In addition, the Equipment Model stores a component's standard operating procedure activities with that component.

5.2.3.15.2 Requirements

The requirements for the Equipment Model are:

- It shall represent the functionality of a scenario's equipment.
- All equipment components shall be agents.
- It shall permit equipment representation at different levels of detail.
- The symbolic and graphical equipment models shall be tied together so that during a simulation run, the graphical displays show the current status of the equipment.

5.2.3.15.3 Interfaces

5.2.3.15.3.1 Input

The Equipment Model receives the following inputs:

- Aero-guidance data

- Environment data
- Pseudo-Jack commands
- Perception queries
- Tick messages

The aero-guidance data comes from the Aerodynamics agent and gives the helicopter's current airspeed, altitude above the terrain (AGL), and position. The environment data comes from the Environment agent and supplies such parameters as the results of using radar or IFF equipment. The exact data received from the environment will depend on the defined equipment suite. Pseudo-Jack commands are commands from Pseudo-Jack that originate as the result of activities in the SOM requiring operations on the equipment. Again, the exact commands will depend on the chosen scenario. Common Pseudo-Jack commands might be to push a button or turn a switch on a control panel. Tick messages are required by equipment components whose behavior is time dependent.

5.2.3.15.3.2 Outputs

The Equipment Model produces the following outputs:

- Guidance updates
- Cockpit updates
- Perception responses
- UWR updates

The guidance updates are messages to the guidance model controlling the flight of the helicopter. (These are not currently generated, but may be in some future scenario.) The Perception responses are responses to queries generated by Perception. The UWR updates are direct updates to the UWR's equipment model that allow parts of it to be tied directly to the corresponding equipment so that the actual memory or perception processes do not have to be modeled.

5.2.3.15.4 Capabilities

Equipment components can have their operation expressed in four different ways. These four ways are not mutually exclusive -- a single component could use a mixture of any or all of them. The different ways are:

- By a Finite State Machine (FSM),
- By a time script,
- By a stimulus-response script, or
- By a CLOS method and associated functions.

When a component's operation is specified by an FSM, it can respond to either tick or communications messages. At all times, the FSM is in a given state and has an associated set of transitions. In response to a tick or message that matches one of these transitions, the FSM will execute that transition's "Transition Function" and change to its new state.

When a component's operation is specified by a time script, it will produce output messages of the types and on the ticks specified by its script.

A stimulus response script has three parts:

- an input message pattern,
- a delay, and
- an output message.

When a component's operation is specified by a stimulus-response script, the component goes through the following steps:

- It matches each received message with the input message patterns in its script.
- When it finds a matching one, it checks the delay.
- If the delay is zero, then the output message is sent immediately.
- If the output delay is non-zero, then the output message is queued up for later transmission
- On each tick, the message queue is checked and any messages queued up for that tick are sent.

When a component's operation is specified by methods and functions, then that code is executed either in response to ticks or incoming messages, as appropriate.

5.2.3.15.5 Sample Operational Scenario

As an example of how an equipment component's operation would be specified, consider a simple model of how a helicopter engine's torque is derived from the Aerodynamics agent's airspeed. The data flow is shown below.

In this model, on each tick the Aero agent computes the helicopter's new airspeed and sends it to the Engine component by way of the Pseudo-Aero agent. Pseudo-Aero derives the engine's torque from the airspeed and updates the torque of the engine. The relevant part of the Pseudo-Aero agent's script is:

PSEUDO-AERO SCRIPT

Stimulus-Response 1
Stimulus Message:
From: AERO
Type: Update
Slot: Air-Speed
New-Value: AIRSPEED

Delay: 0

Response Message(s):
To: Engine
Type: Update
Slot: Torque
New-Value: (LOOK-UP-TORQUE AIRSPEED)

This script says that in response to an "UPDATE Air-Speed" message from the AERO model agent, Pseudo-Aero should immediately send out an "UPDATE Torque" message to the ENGINE component. The new torque will be derived from the AIRSPEED in AERO's message by the function LOOK-UP-TORQUE.

5.2.3.15.6 Issues

There are three major issues with the Equipment Model:

1) How to upload activities into the Mission & SOP agent.
2) How to upload the equipment model into the UWR.
3) How to integrate the FSM, time script, stimulus response, and method-based functionality specifications so that a single agent can use any or all of them at the same time.

Uploading activities into the Mission & SOP agent is straightforward. The only work that needs to be done is in meshing the highest levels of the equipment components' standard operating procedure activities with the lowest levels of the Mission & SOP's activities. Uploading the equipment model into the UWR will be done along the lines of the prototype code developed for A^3I by Dr. Lawrence Davis. Work still remains to be done on integrating the four classes of functionality.

5.2.3.16 AERODYNAMICS

This section was written by Alex Chiu.

5.2.3.16.1 Description

The Aerodynamics agent represents a helicopter flight dynamics model, based on TMAN, to support MIDAS in carrying out flight simulations.

5.2.3.16.2 Requirements

The existing TMAN helicopter aerodynamics model shall continue to be used for Phase V. To meet Phase V requirements, it needs to be modified to
(1) fit into the MIDAS agent structure,
(2) accommodate a more sophisticated guidance model.

5.2.3.16.3 Interfaces

This component requires the controls (collective, cyclic, pedals) computed by the Guidance agent as inputs. The output of this component is the new helicopter state: x, y, z, yaw, pitch, roll, speed, and altitude above the ground level (agl).

This component needs to communicate directly with SGI Sim Exec agent, Guidance agent, Pseudo Aerodynamics agent, Terrain agent, and Cockpit agent.

This component is initially positioned at the first waypoint as defined by the user with the Route Editor with three angular orientations all zero.

5.2.3.16.4 Capabilities

This component takes the controls computed by the Guidance agent as forcing functions and integrates the dynamics equations over the interval of a tick size to yield new helicopter state (x, y, z, yaw, pitch, roll, speed, agl). For more detail, refer to the MIDAS Phase IV Documentation or NASA TM86686, "Piloted Simulation of One-on-One Helicopter Air Combat at NOE Flight Levels", Lewis and Aiken, April 1985.

5.2.3.16.5 Sample Operational Scenario

Because the MIDAS simulation uses a tick-based approach, every SGI agent will not perform its task until it is ticked by the SGI Sim Exec. To ensure proper data flow in the MIDAS simulations a ticking sequence is set up at the beginning of each simulation run and needs to be followed through the entire simulation. In this sequence the Guidance agent is ticked before the Aerodynamics agent. When the Guidance agent is ticked, it computes control demands and updates the Aerodynamics agent with the control demands. Therefore, when the Aerodynamics agent gets ticked, the controls stored in its data structure are already current. It performs the integration operation, with the current helicopter state as the initial condition and the controls updated by the Guidance agent as the forcing function, to yield a new helicopter state (x, y, z, yaw, pitch, roll, speed). To compute agl, the Aerodynamics agent queries the Terrain agent for the terrain elevation at the newly computed (x,y). The Terrain agent responds with the elevation upon successful calculation. The helicopter state is then augmented to include agl, obtained by subtracting the terrain elevation from z. The Guidance agent, Cockpit agent, and Pseudo Aerodynamics agent are then updated with the new helicopter state.

5.2.3.16.6 Issues

None

5.2.3.17 GUIDANCE

This section was written by Alex Chiu.

5.2.3.17.1 Description

The Guidance agent is represented by a model based on the work developed for NOE flight by Dr. Victor Cheng (NASA/Ames Code FSN) to support MIDAS in carrying out flight simulations.

5.2.3.17.2 Requirements

To meet Phase V requirements, the Guidance component needs to
(1) be separated from the Aerodynamics portion,
(2) be converted into an agent to fit in the agent MIDAS structure,
(3) fly a route approximating the contour-hugging characteristics typical of NOE flight.

5.2.3.17.3 Interfaces

This agent requires the helicopter state (x, y, z, yaw, pitch, roll, speed, agl) computed by the Aerodynamics agent as input. It also requires the flight path defined by the user with the Route Editor. A look-ahead capability needs to be developed to predict the next desired position. This process requires the terrain information from the Terrain agent. Based on the current helicopter position and the estimated future position, the Guidance agent computes the optimal control demands. The outputs of this component are the control demands (collective, cyclic, pedals).

This agent needs to communicate with the SGI Sim Exec agent, Aerodynamics agent, and Pseudo Guidance agent, in addition to the Terrain agent mentioned earlier.

This agent will be initialized with collective, cyclic, and pedals controls all zero.

5.2.3.17.4 Capabilities

This agent uses a simple method to estimate roughly where the helicopter will be on the nominal trajectory a short period of time (3 seconds, say) from now. Based on the estimated position and current position this agent computes the control demands (collective, cyclic, pedals) and updates Aerodynamics agent and Pseudo Guidance agent.

5.2.3.17.5 Sample Operational Scenario

Because the MIDAS simulation uses a tick-based approach, every SGI agent will not perform its task until it is ticked by the SGI Sim Exec. To ensure proper data flow in the MIDAS simulations a ticking sequence is established at the beginning of each simulation run and observed through the entire simulation. When the Guidance agent gets ticked, it computes the control demands based on the current helicopter position and the estimated future position as mentioned earlier. The Aerodynamics agent and Pseudo Guidance agent are then updated with the new control demands.

At initialization, the flight path needs to be passed to this component through an appropriate means.

5.2.3.17.6 Issues

None

5.2.3.18 ENVIRONMENT AND TERRAIN

This section was written by Alex Chiu.

5.2.3.18.1 Description

This component contains a piece of Defense Mapping Agency (DMA) level 1 terrain. The capability that allows the user to choose a different cell of DMA terrain as a gaming area will be developed. This piece of terrain needs to be integrated with the Route Editor so that the user can define a flight path. Furthermore, this piece of terrain needs to be integrated with VEST so that the terrain gets displayed in the views. It also allows the user to place and control up to six environmental objects, e.g., other aircraft, ground vehicles, surface-to-air missiles, etc.

5.2.3.18.2 Requirements

To meet Phase V requirements, the Terrain component has to be converted into an agent and is required to provide the following capabilities.

(1) Calculate terrain for given (x, y).
(2) Check line of sight for a pair of points.

5.2.3.18.3 Interfaces

To calculate terrain elevation, the Terrain agent requires a two dimensional point (x, y) as input. The output will be the elevation at the given (x, y), expressed as a floating point. To check line of sight, two points (floating points) are required. The output of this will be an integer indicating clear or not. The former capability is needed by the Aerodynamics agent, and the latter is needed by the Vision agent.

This agent needs to communicate directly with SGI Sim Exec agent, Aerodynamics agent, and Vision agent.

5.2.3.18.4 Capabilities

To compute terrain elevation, it first searches for the polygon on which the given 2d point (x,y) is located. Once the polygon is identified, the plane equation is formed from the vertices of the polygon. The equation is then solved for the given (x, y) to yield the elevation.

To check line of sight for a given pair of points, the Terrain agent first identifies the polygons intersected with the projected line segment formed by the given points on the terrain. The intersected polygons are then checked sequentially to see if any polygon lies above or intersects the line segment connected by the two given points.

5.2.3.18.5 Sample Operational Scenario

For each tick in a MIDAS simulation run, the Terrain agent gets queried by the Aerodynamics agent for the terrain elevation. It might get queried by the Vision agent for the line of sight calculation when needed.

5.2.3.18.6 Issues

None

5.2.3.19 VISUAL EDITOR AND SIMULATION TOOL (VEST)

This section was written by Scott Chen.

5.2.3.19.1 Description

The Visual Editor and Simulation Tool (VEST) is a module to provide graphical views of the progression of the simulation. It also generates the terrain surfaces and defines the flight path.

5.2.3.19.2 Requirements

The requirements for VEST are:

- To make VEST independent of the MultiGen™ commercial application previously used.
- To provide a "god's eye" view of the flight simulation.
- To provide an "over-the shoulder" view of the cockpit simulation.
- To provide a "snap-to-visual-fixation" zoom view of the selected instrument.
- To display and animate the Jack figures.
- To select and extract DMA level 1 digital terrain data.
- To choose the environment objects.
- To develop a route editor.

5.2.3.19.3 Interfaces

VEST interacts, through RPC calls, with the following agents: the SGI Simulation Executive, Aerodynamics agent, Environment agent, Pseudo cockpit agent and Vision agent. The Vision agent queries the locations of the instruments. The rest of the interacting agents update the positions, orientations or equipment states. A file containing the cockpit geometry and attributes will be prepared by VEST for the Equipment agent to process at initialization.

In the simulation, the graphical display of VEST is always one tick behind. Besides the graphical objects of the cockpit, environment objects and terrain, VEST also displays the figures of the Jack agent.

5.2.3.19.4 Capabilities

VEST shows the graphical display of the MIDAS simulation. It will use the CDE data structures and support multi-windows. To reduce the overhead checking, the terrain and environment objects will have different data structures from the cockpit.

The window views may be set up by dials and buttons, as in the CDE windows. For static views as in the "god's eye" view, this process is intuitive. However, it appeared in Phase IV that it was very difficult to adjust the camera position for the "over-the-shoulder" view in the flight simulation. Part of the reason was due to the limitation of MultiGen™ that it can open only one single database at a time. The new VEST will be designed to distinguish the type of graphical object being displayed and thus control the views. Based on this, it will take a new approach to allow the user to use the same CDE controls to set up the camera positions. Instead of moving the ownship over the terrain, it remains static and the terrain moves toward it. VEST will also allow a user to select an instrument and do a "blown-up" view.

VEST will be linked to the display and reach modules of Jack. It will allow a user to scale, rotate, and translate the Jack figures. It will also provide an interface with the Jack modules to animate the aviator's reaches.

5.2.3.19.5 Sample Operational Scenario

The users may invoke the DMA function to extract the terrain data and select a piece of terrain for the gaming area. The Route Editor may also be invoked to define the flight path. Through different steps, they can load the geometry files of the ownship and environment objects, and define the associated attributes such as the gravity center, scale and orientation. The CDE database can be opened to animate the instruments. The Jack environment file can also be parsed and loaded if there are reaches to be animated. Through dials and buttons, the window views can be set up. (The above steps can be assembled in a script file). The users may then put VEST in the network mode to display the MIDAS simulation or in a stand-alone mode to show the simulation from a history file.

5.2.3.19.6 Issues

None

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	June 1992	Contractor Report

4. TITLE AND SUBTITLE
Army-NASA Aircrew/Aircraft Integration Program: Phase V A^{3}I Man-Machine Integration Design and Analysis System (MIDAS) Software Concept Document

5. FUNDING NUMBERS
NAS2-13210

6. AUTHOR(S)
Carolyn Banda, David Bushnell, Scott Chen, Alex Chiu, Christian Neukom, Sayuri Nishimura, Michael Prevost, Renuka Shankar, Lowell Staveland, and Greg Smith of Select Systems Analysis

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-92137

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-177596

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 is the Software Concept Document for the Man-machine Integration Design and Analysis System (MIDAS) being developed as part of Phase V of the Army-NASA Aircraft/Aircraft Integration (A^{3}I) Program. The approach taken in this program since its inception in 1984 is that of incremental development. Each phase of development consists of planning, setting requirements, preliminary design, detailed design, implementation, testing, demonstration and documentation. This document, produced during the preliminary design period of Phase V, is intended to record the top level design concept for MIDAS as it is currently conceived. This document has two main objectives 1) to inform interested readers of the goals of the MIDAS Phase V development period, and 2) to serve as the initial version of the MIDAS design document which will be continuously updated as the design evolves. It is expected that Phase V development will be complete and ready for demonstration to invited visitors from industry, government and academia in May 1992. The final design of MIDAS will be documented in a Detailed Design Document published after the demonstrations.

MIDAS is an integrated suite of software components that constitutes a prototype workstation to aid designers in applying human factors principles and human performance models to the design of complex human-machine systems. MIDAS is intended to be used at the 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. Phase V MIDAS will include the capability to run selected segments of a simulated mission in which complex interactions among the various models of the environment, mission and crew are propagated to show illuminating details of the operator task performance, loading, and sequences/duration based on the specific equipment and context of the simulated mission segment.

14. SUBJECT TERMS
Computer-aided engineering, Human performance modelling, Crewstation design, Man-machine interface, Human factors engineering

15. NUMBER OF PAGES
85

16. PRICE CODE
A05

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified		