1986

NASA/ASEE SUMMER FACULTY FELLOWSHIP PROGRAM

MARSHALL SPACE FLIGHT CENTER
THE UNIVERSITY OF ALABAMA

AN EVALUATION OF THE DOCUMENTED REQUIREMENTS OF THE SSP UIL AND A REVIEW OF COMMERCIAL SOFTWARE PACKAGES FOR THE DEVELOPMENT AND TESTING OF UIL PROTOTYPES

Prepared By:	Esther Naomi Gill, Ed.D.
Academic Rank:	Associate Professor
University and Department:	Oakwood College Department of Business and Information Systems Management
NASA/MSFC:	
Laboratory:	Information & Electronic Systems
Division:	Software & Data Management
Branch:	Systems Software
MSFC Counterpart(s):	John W. Wolfsberger and Robert L. Stevens
Date:	August 8, 1986
Contract No.:	NGT 01-002-099 The University of Alabama

2.0 ACKNOWLEDGEMENTS

Thanks go again to the Information and Electronics Systems Laboratory, W. C. Bradford, Director, Gabe Wallace, Assistant Lab Director; Software & Data Management Division, Jack Lucas, Chief; Systems Software Branch, Walter Mitchell, Chief; and Counterparts, John Wolfsberger and Robert Stevens who for the second year opened their laboratory, shared their facilities, and gave of their expertise and time which another year enabled the successful completion of this research effort.

Thanks also go to Cathy White, who shared her library and reviewed my efforts at various stages. To Mike Kynard and Ken Williamson who aided with computer equipment and salvaged this document when I thought all was lost--"You saved my sanity."

My appreciation to Ernestine Cothran, F. A. Speer, Mike Freeman, and Dina Conrad, NASA/ASEE and University of Alabama Summer Faculty Fellowship Program Coordinators and Secretary for giving opportunities for teachers to get into the dynamic work environment so their teaching can be relevant, their students might have employment opportunities, and the dynamic work environment might be shared in the classroom through grants proposed and sponsored. The informative seminars and tours were also appreciated.

Last, but not least, highest praise goes to the ASEE organization whose vision it was to design this worthwhile NASA/ASEE Summer Faculty Fellowship Program, and also to NASA for sharing in its implementation.

Esther Naomi Gill

AN EVALUATION OF THE DOCUMENTED REQUIREMENTS OF THE SSP UIL AND A REVIEW OF COMMERCIAL SOFTWARE PACKAGES FOR THE DEVELOPMENT AND TESTING OF UIL PROTOTYPES

BY

Esther Naomi Gill, Ed.D.
Associate Professor of Computer Science
Oakwood College
Huntsville, AL 35896

A_B_S_T_R_A_C_T

This study will report the progress toward the development of the User Interface Language (UIL) for the NASA Space Station.

First, an examination was made of each Center's role in this development; namely, Goddard Space Flight Center (GSFC), Johnson Space Center (JSC), Kennedy Space Center (KSC), and Marshall Space Flight Center (MSFC).

Secondly, a review was conducted of software packages currently on the market which might be integrated with the interface language and aid in reaching the objectives of customization, standardization, transparency, reliability, maintainability, language substitutions, expandability, portability, flexibility; and recommendations are given for best choices in hardware and software acquisition for in-house testing of these possible integrations.

Finally, software acquisition in the line of tools to aid expert-system development and/or novice program development, artificial intelligent voice technology and touch screen or joystick or mouse utilization as well as networking were recommended. Other recommendations concerned using the language Ada for the UIL shell because of its high level of standardization, structure, and ability to accept & execute programs written in other programming languages, its DOD ownership and control (already written and owned by US Government), and keeping the user interface language simple--(something that is truly USER_FRIENDLY) so that multiples of users (both national and international) will find the commercialization of space (at non-prohibitive costs) within their realm of possibility which is, after all, the purpose of the Space Station.

AN EVALUATION OF THE
DOCUMENTED REQUIREMENTS OF THE SSP UIL
AND A REVIEW OF COMMERCIAL SOFTWARE PACKAGES
FOR THE DEVELOPMENT AND TESTING OF UIL PROTOTYPES

4.0 TABLE OF CONTENTS (OUTLINE)

5.0 INTRODUCTION

5.1 PURPOSE STATEMENT

While the U.S. Space Program is recovering from the shock of the Challenger accident, life and the challenge of a permanently based, manned Space Station go on. NASA likewise can, will, and must lick its wounds, overcome its difficulties, face its critics, surmount its obstacles, learn from its mistakes and proceed with cautious optimism and expertise to meet this challenging goal in the next decade. A part of this goal, which is the purpose of this paper, includes the development of a Space Station common module User Interface Language (UIL) that is effective and efficient.

Some words to NASA leaders would be to hang in there and see what the end will be--don't give up the ship! A team doesn't need to finish the ball game with a new manager, a relief pitcher, and all pinch hitters--that's like going to the locker room before the inning is over, throwing in the towel, starting from scratch when you're ahead in the game, abandoning ship when there's no danger of it sinking. Once you've made the team and are in the game, you have to play until the game is over; and unless you're thrown out, you don't have the option to quit.

5.2 STATUS

The development of the SS offers NASA a chance both to advance the technology of automation and robotics and to put that technology to use. It was at this point that the wand was picked up and a progress report and evaluation of NASA's SS software UIL development, and operational activities were made.

Finally a review of existing (commercial) and/or created (off-the-shelf) software packages were discussed in the light of the possibility of integrating their technology for analysts and programmers productivity increases, purchasing feasibility, and checking out various UILs. The UIL will support software development, onboard and ground operations, tests and diagnostics, and a family of languages (Assembly, COBOL, FORTRAN, BASIC, C, Forth, Pascal, Ada, etc.). It will be device independent and transparent (friendly) to the user.

5.3 USERS

Interface users have been described as being

requirement/specification engineers
software designers
coding and testing engineers
maintenance engineers
experimenters
managers
crews (astronauts, science, flight control & ground control)
payload operators
application customers
developers

These users will be using the UIL to support such SS operations as:

Performing, Planning and Scheduling
Interacting with Real-Time Module/Payload Satellite Processes
Performing In-orbit Checkout, Repair and Servicing
Controlling and Monitoring Space Environmental Control and Life Cycle Support System (ECLSS)
Communicating with Other Languages
Detecting and Diagnosing Equipment Malfunctions
Interacting with Configuration Management Malfunctions
Monitoring On-going Systems and Making Adjustments or Trouble-shooting Where Needed
Examining and Updating Databases
Monitoring and Reporting Observations
Interacting and Communicating with Ground Systems
Interpreting, Recording and Presenting of Data
Communicating with Payloads
Forecasting Potential Conditions and Making Contingency Analyses
Validating Systems Performance
Creating and Describing Graphic Displays
Examining Programs' Performance
Monitoring User's Use of UIL

6.0 NASA CENTERS SS UIL RESPONSIBILITY AND ACTIVITY

The operations perspective and requirements responsibilities for the UIL were divided among three NASA Centers: Goddard Space Flight Center (GSFC) - Scientific Community, Commercial Customers and Customer Data and Operations Language (CDOL), Johnson Space Center (JSC) - Mission Control and Crew Operations and the compilation of the requirements from the three centers to form the User Interface Language (UIL), Kennedy Space Center (KSC) - System Integration and Test Function Support and Space Station Operating Language (SSOL). Marshall Space Flight Center (MSFC) - Aid Software Development and Operational Support Activities. Ground Operations and Support will be a joint-Center effort.

A review of each Center's effort follows with the idea of recognizing similarities and differences and attempting to determine to what degree each is or is not important to the successful development of the SS UIL:

6.1 GODDARD SPACE FLIGHT CENTER (GSFC) -SCIENTIFIC COMMUNITY AND COMMERCIAL CUSTOMERS

Goddard Space Flight Center (GSFC) described the requirements for Commercial Customers via the Customer Data and Operations Language (CDOL). This language provides access to support functions required by both internal and external SS operators, commercial customers, science researchers, science operators, mission operators, test and Integration engineers, scheduling, and flight controllers, etc.

Some services related to payload development, test, integration scheduling operations are made available by CDOL to both internal and external users.

CDOL is a user-oriented, easy-to-use, standardized interface to the non-mission-unique ground functions provided by the Customer Data and Operations System (CDOS). CDOL represents the platform, and payload operations perspective of the Space Station User Interface Language (SS UIL). CDOL is either equivalent to UIL or is a proper subset of UIL. UIL provides the standard access to all Space Station Information System (SSIS) services and facilities including CDOS, onboard Data Management System (DMS), and Science Application Information System (SAIS) segments.

CDOL represents one of the three operations perspectives for which user interface requirements have been defined. The requirements associated with these varied operations perspectives will be integrated to form the requirements for the NASA-wide UI language.

The requirements definition effort of the Customer Data and Operations Language lead by GSFC include payload and platform operations and payload test support.

The goal of CDOS is to support user and customer needs in all phases of mission planning, scheduling, and operations. CDOL supports the following modes of operation: Immediate execution, background execution, and batch execution.

The major functions provided by CDOS are planning and scheduling of platform and payload activity, real-time monitoring, control and verification for platform and payloads, access to data management services in support of platforms, payloads, and customer applications, system integration, test, simulation, and training support and platform and payload operations support. These functions will be provided and/or supported by a variety of support services (including SSIS services, user applications software, operating systems, and database management systems (DBMS) which execute at various locations in a distributed processing environment.

CDOL provides the user interface to the mission planning, scheduling, operations and data handling capabilities proposed by CDOS. CDOL provides the mechanism for user access to both local and distributed support services.

The availability of sophisticated, low-cost terminals, personal computers, and workstations argues strongly that several, widely available, common configurations should be chosen as a basis for the initial core and operations CDOL releases.

CDOL General Requirements:

1. CDOL must provide a standarized user interface to payload platform operations and data analysis functions provided by CDOS.
2. CDOL shall be responsible for limited validation of user inputs.

Command Procedure Requirements: CDOL shall support

1. the definition of structured command procedures.
2. the creation of libraries of frequently used command procedures.

Language Element Requirements:

1. The language shall provide a mechanism for assigning values to variables.
2. The language shall provide a branching or selection control mechanism.
3. The language shall provide an iteration control (looping) mechanism.
4. The language shall provide for command procedure activation or suspension at a user-specified time or after a specified time interval.

Human-Machine Requirements:

Interactive Dialog Style

1. The language shall support advisory, warning, and alarm messages that alert the user to problems.
2. The system feedback message shall utilize various visual display attributes actuated by system or user-defined.
3. The language shall support HELP messages in support of all defined alert messages.

User Assistance:

1. Three levels of HELP shall be available on the basis of user experience level; very concise assistance (for experts), brief assistance (for intermediate users), and extended assistance (for novices).

Optional Capabilities:

1. Depending on availability of appropriate workstation support and on the software system chosen to support graphics services, there are a variety of optional capabilities that should be provided via the CDOL interface to graphics services; such as, window outlining; low resolution color graphics, high resolution bit-mapped graphics; icons; and a variety of character attributes, fonts, and styles.
2. To the degree provided by CDOS, and as authorized by local management, CDOL should permit full interaction with Electronic Mail functions.

6.2 JOHNSON SPACE CENTER (JSC) - MISSION CONTROL AND CREW OPERATIONS

The Tri-Center Committee focused on a command and control language (UIL) having keywords, arguments, syntax, rules, conditional statements, etc. Other groups have

focused on I/O devices (e.g., keyboards, cathode ray Tube, mouse, or touch screen) and the human factors involved that allow the user to interface with the computer in an efficient manner. All of these considerations are important in building a good user interface, which is really user-machine communications. JSC was also responsible for crew interfaces, and putting together data from other centers and writing the final UIL.

The user interface language (UIL) crew and mission control operations support; requirements definition effort was led by JSC.

The real requirements have to do with making it easy to do the job, so that interfacing with the computer is not another difficult task that has to be conquered. The goal then is to have a standard user interface where similar functions should be done in similar ways across all applications and system services.

There will be a SSIS (Space Station Information System) software service called the User Interface (UI) which will interpret human inputs and route them to the appropriate application/service to be performed and will interpret application/outputs and present them in an understandable form to the human user. The User Interface forms a shell around the SSIS and its services.

The user interface provides the means whereby the user can effectively and efficiently communicate with a supporting computer system. The interface consists of three parts: UIL defined syntax and semantics; man-machine interface (MMI) techniques; and a set of supporting services for the language and interface techniques. Associated with these three parts are standards, guidelines and managerial constraints.

UIL syntactic and semantic definitions specify the characters, words and structures used for communication between users and computer system. It permits the users to precisely and concisely define their processes in the terminology that is best suited to their job.

The UIL is required for the following reasons: familiarity, documentation (self-documenting or readable languages provide records of instructions that are easy to understand and maintain), portability of procedures and command sequences (a common UIL supported at all or most SS program sites--a solution to the portability problem), and commonality. As long as the diverse, specific needs of various users can be translated into a common UIL syntax and command structure, this level of standardization can be met.

Users can select the interactive method (menus, commands, graphics, etc.) that best meet their needs.

Standards:

When coordination and transfer of information between user groups occurs, then standardization of the language is clearly needed. However, when language statements support needs that are unique to a particular group of users or specific function, standards that guide the language subsets may be useful.

Another reason for standardization that involves UIL is the concept of the format for recording (or logging) the user's actions that were specified through other MMI techniques--language becomes a readable "history" of choices and selections made by the user.

Capabilities--The basic capabilities that the UIL needs to support are Systems Operations, Simulations, and Queries.

Execution Modes--Several modes of operation for the UIL are Batch (the process of requesting processing for a sequence of statements and awaiting the results); the other operating modes apply to an Interactive Environment or a dialog between man and machine (is most useful for systems operations and query capabilities); and predefined sequences (in this case, the user has the ability to interface to the sequence through the UIL, and predefined sequences could be partially processed ahead of time).

Man-Machine Interface Techniques--MMI techniques refer to the different methods and devices that can be used for the machine (computer system) to present (output) information to the users, and for the users to present (input) information (data and commands) to the machine. Regardless of the MMI technique chosen to interface by the user, all information presented to the user must be clear, concise, and precise. The reason for having a wide variety of MMI techniques (displays, Texts, Graphics, Menus, Forms, Prompts/Responses) is to provide the users with (the natural language, voice recognition/actuation, interface devices; such as, lights and alarms, buttons, programmable keys; pointing devices; such as, mice, joystsicks, trackballs, touch screens, light pens, and cursor keys; and hardcopy devices; such as, printers, plotters, and cameras) their choice for most effective and efficient interaction with each system function.

The following list specifies the currently identified requirements criteria for the SSIS user interface:

1. Provide all users with a friendly, automated, easy-to-use means of accomplishing the functions of developing, testing, operating, managing, and maintaining the various elements of the SSP.
2. Support all users regardless of their level of expertise.
3. Place support of the users' performance with respect to speed, accuracy, and ease of use above ease of implementation considerations.
4. Meet the objectives of upgradeability, maintainability, readability, writeability, and learnability.
5. Be easily managed, maintained and transported to the users' site with the assurance of function repeatability and no degradation of system integrity.
6. Be independent of the host and target computer system.
7. Easily accommodate growth and technological advances with minimal or no impact on the user or the interface products developed by the user.
8. Not preclude other languages and systems from co-existing with it.
9. Be standardized as well as all of its lower-level interfaces to SSIS applications and data.
10. Be common across all elements of the SSIS.

6.3 KENNEDY SPACE CENTER (KSC)--UTILIZE THE UIL AS AN INTEGRATION, TEST, AND LAUNCH TOOL

The systems integration and test function support requirements definition effort was led by Kennedy Space Center (KSC). The Space Station Operating Language (SSOL) is an automated environment, where SS Integration and Test (I&T) activities can be designed, developed, tested, and performed. The SSOL will be responsive to users, independent of their, location and it will promote standardization and transportability of user developed procedures from site to site, capitalizing on the functional commonality of the I&T activities at these sites. SSOL system will take full advantage of advances in the state-of-the-art in both software and hardware development. Advances in the user friendliness of commercial real-time operating systems, data base management systems, supporting software development tools, processing speed, and memory capacity are only a few of the anticipated technological improvements which may enhance the SSOL system.

KSC logically derived and established a set of requirements and concepts; the acceptance, design, and implementation of the same will result in the development of an evolutionary user environment in which a diversified user group may accomplish all phases of Space Station I&T.

The requirements provide a functional description of the SSOL System that can be applied or adapted to any computer system architecture. The concepts and requirements are presented in three basic parts: the SSOL System; the SSOL Language; and the SSOL Support Environment.

The SSOL System provides the Space Station I&T community with a consistent, automated, user-friendly means to accomplish integration and testing activities. It supports all phases of I&T activities from factory through launch, and on-orbit, maintaining system integrity and ensuring test repeatability at all I&T sites. The SSOL System is designed to meet the objectives of readability, writeability, learnability, maintainability, transportability, and standardization.

The SSOL coexists with other Space Station application software. Standard interfaces allow other Space Station software to access and make use of the SSOL System, and the test products developed on the SSOL System. These standard interfaces also permit the SSOL System and test products to access and make use of other Space Station software and data.

The SSOL System consists of two parts, the SSOL language and the SSOL Support Environment (SSOL SE). The SSOL Language provides the means for user communication and control of his test articles, support equipment and real-time computer systems. The SSOL SE provides the environment which supports the use of the SSOL language, allowing the users to develop, verify and perform their I&T activities.

The SSOL System provides a consistent, friendly user interface to its Support Environment functions. The primary purpose of the Support Environment is to allow the I&T users to develop and verify their own "user interface" to the Test System for the performance of their specific test activities.

The SSOL System maintains commonality between the user interfaces to the SE functions and the user developed interfaces to the Test System. The basic capabilities provided by the SE user interface can be applied to the interface developed by the user. These capabilities include interactive graphic displays, menus, prompts, and an interfacing/command language. The language used to interface to the SSOL SE functions is, in essence, the User Interface Language (UIL). The language that interfaces with the Test System is the SSOL language.

The SSOL language is an English-oriented, user-friendly tool, that establishes standard terminology and methods for the performance of I&T activities for the Space Station. The use of a standard I&T language ensures a common set of I&T capabilities independent of specific implementation, test articles, and host and target computer systems.

The basic function of the SSOL language is to provide a consistent interface for control and monitoring of the Test System.

The SSOL SE provides the users with an integrated, automated environment to design, develop, maintain, and execute their test products, including the SSOL procedures/statements. In addition the SSOL language, the user can create graphic displays for his interface to the Test System. These displays have the potential to provide friendly, easy-to-understand interfaces. They can be used to initiate action, provide response data, monitor the Test System, interact with SSOL procedures, and provide operator notification of critical conditions.

SSOL SYSTEM CONCEPTUAL REQUIREMENTS

The conceptual requirements for the SSOL System provide a set of goals or objectives for which the eventual implementation of the SSOL System should strive. The conceptual requirements will provide a qualitative means of judging system implementation effectiveness:

1. The SSOL System shall be user-friendly, in all aspects.
2. The SSOL System shall strive to meet the objectives of readability, writeability, learnability, and maintainability in all aspects, with emphasis placed in areas of user interfaces.
3. The SSOL System shall provide an automated environment in which SS I&T activities can be designed, developed, tested, and performed.
4. The SSOL System shall effectively & efficiently support I&T activities from factory test through on-orbit.
5. The SSOL System shall support I&T users in their environments with no degradation of system integrity and with the assurance of test repeatability.
6. The SSOL System shall be host, target, and Test System independent, allowing the SSOL System to be easily transported from site to site.
7. The SSOL System shall be designed to produce SSOL products that are easily transported from site to site.
8. The SSOL System design and implementation shall be configuration manageable
9. The SSOL System shall be designed to easily accommodate technological changes and advances with minimal or no impact on existing SSOL products and on itself.
10. The SSOL System design shall incorporate functional modularity to allow subsets of the SSOL and/or SSOL SE to be used.
11. The SSOL System and its interfaces shall be standardized.

12. The SSOL System shall not preclude other systems, languages, or software utilities from co-existing or interfacing with the SSOL System and SSOL products.

The basic function of the SSOL language is to provide a means of communication and control within the framework of I&T. The I&T users need not be highly skilled in computer systems programmming techniques, but will have the capability to perform I&T functions using familiar terminology and uniform test notations.

The SSOL is adaptable to the I&T users' environments ranging from development testing to integrated systems testing. It is technically capable of performing the I&T functions required by the various user disciplines, including (but not limited to):

Command and monitoring
Data sampling
Data comparison
Data manipulation
Data organization
Data presentation
Data recording
Time controlled events
Workstation interaction
Communications with other languages
Communications between procedures
Test article identification
Exceptions processing

The language will maintain a functional modularity enabling efficient, easy upgrades and modifications, the SSOL qualities of readability, learnability, changeability, and writeability allow the language to be self-documenting. These are key features in achieving reduced I&T life cycle costs and a high degree of user-friendliness.

The SSOL will support interfaces to other languages or routines. This will make available to any user, the capabilities of any language supported by the SSOL SE. This will enable a user who desires to use other programming environments, or who desires to program in languages like LISP, PASCAL, Ada, or Assembly language to access the capabilities of that environment.

6.4 MARSHALL SPACE FLIGHT CENTER (MSFC)--SOFTWARE DEVELOPMENT AND OPERATIONAL SUPPORT

NASA/MSFC Information and Electronics Systems Laboratory, Software and Data Management Division, Systems Software Branch reviewed all the specifications and requirements documents of GSFC, JSC, and KSC and raised questions needing answers and made comments and

recommendations. They also held an open forum in which some 300 plus software vendors, prime contractors, MSFC, GSFC, NASA Headquarters, JSC, KSC, LeRC, and foreign participants from Canada, Japan, the Netherlands, ESA, and ESTEC were in attendance entitled, "Space Station Software Recommendations (April 25, 1985)."

Some recommendations of note which came out of that meeting were:

1. Using existing (inherited) software as an alternative to a totally new development.
2. Developing policies and procedures to accommodate modern, appropriate software development methodology.
3. Developing policies and procedures for the acquisition of software rather than the development of software.
4. Developing policies and procedures for insuring non-loss of software and continuous operations due to inadvertent and/or catastrophic loss of operational or support hardware.

UIL is a set of signs and symbols whereby a user communicates with a system (universal). It is a formal language consciously constructed for definite and restricted purposes (functional). The SS UIL will consist of the user interface standard and a family of UILs built around that standard.

According to Peter Prun, the UIL is an English-like, user-friendly tool used to communicate with and control the Space Station Environment. The UIL selected must provide a standard interface to the various computerized functions of the Space Station. The language will be a user-oriented, high-level language using English-like commands. It will be a tool by which a user will communicate with and control his environ in a simple manner without being skilled in programming techniques.

Commonality - UIL supports standardization of user interface to multiple subsystems. It will provide a common and consistent user interface throughout all stages of tests and operations both on ground and in-orbit.

User Friendliness - UIL will be user friendly in all man-machine interfaces. UIL will be a high-order language requiring little skill in programming techniques and will be straight-forward, English-like commands.

Reduce Life Cycle Costs - In order to minimize costs in time and resources, the UIL will make use of the similarity

of operations requirements at different stages of test, integration, and operations. It will reduce costs by meeting the goals mentioned in the UIL purposes.

Portability - The UIL should be easily transported to different computer systems and locations assuring consistency and independence from system hardware and also assure system integrity.

Maintenance - The UIL should be easily maintained throughout its like cycle and provide a toolset to assist in maintaining its products and interfaces.

Flexibility - The UIL should be adaptable to growth in capability and new technology, it should meet the needs of the users. It should be flexible enough to easily adapt to different situations and user needs.

The language structure should support real-time commanding and monitoring in a simple manner as well as lead to automation of many tasks. This would make it adaptable to the user's environment from testing to operations. It should provide a short form for real-time interactive control. Functional modularity will allow language subsets and enable efficient easy upgrades and modifications. It must be technically adequate in fulfilling the requirements of users. The language should provide meaningful, accurate, and easy-to-understand communication between the user and the subsystem.

Characteristics - The user must be able to interrogate the system to obtain answers, to monitor the system to determine performance, to create, modify and interact with graphic displays, to obtain readouts, to indicate status and diagnose failures, to examine databases for updates, retrieval, and management, to exert manual control over applications for real-time management or for overriding system software.

The UIL will provide an extensive control and monitor capability which will interact with space station's systems, payloads, and platforms both on the ground and in-orbit. The UIL will have the capability to communicate with other processes as well as concurrent processing. It should have the ability to handle real-time functions in an efficient, simple manner. This includes fault and warning detection and handling. The UIL should support individual commanding of SS systems, payloads, and platforms. It should also provide a multifunction scan with error reporting capabilities.

It should allow simultaneous viewing of displays. It should perform distributed protocal data handling, control and display functions.

The language must provide a flexible data storage analysis, retrieval, manipulation, and recording capability. It should provide capability task control such as initiation, termination, suspension, and resumption.

The UIL should provide an interface to:

--minimal operator attendance
--software development tools
--free/flexible access to Resources
--program generation and debug
--minimul skill requirements
--health and status monitoring
--subsystem performance and trend data
--network operating systsem
--high level of information
--visual imaging for proximity operations
--payloads
--audio devices
--electronic mail
--models/simulations
--other languages
--display graphics
--common databases

The language implementation should stress ease of use. UIL should provide an easy interactive mechanism for entry of commands. It should make use of menus techniques, function keys, icons, mouse, track balls, and other manual computer devices or techniques. It should make good use of graphic displays.

The UIL should support different levels of interaction to accommodate infrequent users as well as skilled users. It should use menus for tutored input and also allow command inputs for fast, efficient communication needed for real-time applications. The UIL should support two modes of operation, interactive and precompiled and linked programs.

The UIL should support a comprehensive "HELP" system giving multiple levels of user self help for all software processes and combinations of software processes. The UIL should support automatic error exception/fault detection notification and processing for Space Station System.

Definitions - A User Interface Language is a language that is invoked to communicate with various processes on a computer. It contains a set of command words, the rules for combining them (syntax), and their meaning (semantics).

General Form - Imperative statement, conditional prefix, direct object, prepositional phrase.

Syntax of a UIL Statement - Contains at most 4 terms:

COND (Conditional Term) - IF/THEN.
CMD (Command Term) - VERB
ELEM (Element Term) - Direct Object
ARG (Argument Term) - Prepositional Phrase

Help Capacity

--May be invoked at any time
--May be embedded in user definitions

Input Capability

--keyboard
--Touchscreen
--Voice
--etc.

The user interface is the set of system interfaces, guidelines, tools, features, and the interface language that support the user's ability to communicate with the processes.

Dr. Thomas Tullis and Glen Love in their writing concerning SSP UIL gave a description of current work being done in the human-computer interface field that appears to have application to UIL and the Space Station.

M. Guillebeau, F. Nixon, S. Owens, and M. Ulehla sponsored by Martin Marietta Denver Aerospace and prepared by TRW System Development Division, Defense System Group, Huntsville Operations describe MSFC UIL requirements perspective and can be used as MSFC input to the NASA inter-Center UIL specification activity.

UIL Tests--It was MSFC's inter-Center activity to:

--Evaluate Tools and Provide Experience
--Prepare UIL Prototype and Detail Requirements
--Share Lesson Learned from Space Lab Experience
--Describe Life Cycle Management Function
--Aid in Software Development
--Evaluate Documented Requirements
--Research Commercial Packages for Possible UI Incorporation
--Act as Checkout Tool Facility

6.5 GROUND OPERATIONS AND SUPPORT

Support for Ground Operations was a joint-Centers' (GSFC, JSC, KSC, and MSFC) effort. Therefore, my research did not cover this phase although a couple of centers' views on the subject have been included from my literature review.

6.6 NASA CENTERS SS UIL REQUIREMENTS ACTIVITY & EVALUATION

With every center utilizing the objectives of:

Portability - Hardware & Software independence which assures consistency and integrity.

Maintainability - Upkeep throughout life cycle.

Flexibility - Easily adapted to different situations and users' needs.

Userfriendliness - High-level, English-like, transparent to User, interface which works for different levels of interaction (unskilled, semi-skilled, skilled users) and utilizes (prompts, touch screens, mice, keyboards, function keys, menus, etc.) for interactive, real-time or batch processing. The terms readability, writeability, and learnability which have been used by some centers as separate objectives are synonymous with userfriendliness.

Commonality - Standardization assures same (consistent) system for ground and in-orbit operations.

It is felt that if the common objectives (listed above) are strictly adhered to, a consistent automated user-friendly means of communicating with ground and in-orbit users and crew members can be achieved within the time frame of the Space Station launch (1992).

7.0 REVIEW AND IDENTITY OF UI AVAILABLE TOOLS, SYSTEMS AND/OR LANGUAGES

7.1 MANAGEMENT TOOLS

Libraries have finally become sufficiently refined so that few programmers can realistically object to them on purely technical terms. The libraries save so much time and effort that no software developer--excepting those working in demanding environments--can afford to do without them. Someday they may even take on the traditional roles of 4GLs and application generators.

Libraries of subroutines and functions that developers can incorporate into their own products dramatically reduce software development time and cost. There are two types of programmers' tools sold today. The first, and by far the most common, comprises source code libraries of subroutines and functions that developers can incorporate into their own software. The second is of programs that assist a developer in the writing, management, and maintenance of code.

Vendors maintain collections of subroutines for jobs such as screen handling, mathematical operations, database-search methods, and graphics and string functions that address common programming tasks. Developers can purchase them in source code so that they can be modified and further specialized to fit their particular application. Most source code libraries are written in BASIC or C.

Phoenix Computer Products Corporation (PCP) markets a broad line of programming tools in the microcomputer-based MS-DOS world. Their current offerings include Pmate, a programmer's editor; Pre-C., a C program analyzer for the MS-DOS environment; Plink, a linking editor; Pfix, a high-level debugger; and Pfinish, a program which allows user to indentify inefficient areas of code. A particularly unusual programmer's tool, Pfinish allows MS-DOS programmers to find problem areas, not only in application programs, but also in their compilers and operating systems. Polytron Corp. markets a very broad line of programming aids; such as, PolyMake, which the company describes as "an intelligent program builder and maintenance tool." Essentially, PolyMate is an MS-DOS version of utility in the UNIX operating system known as "make." If a programmer makes in one module of code a change that requires other modules to be likewise modified, PolyMake will automatically go through the program and update the files that have not been changed. It costs $99.

Keyboards are the most commonly used devices for getting data into and out of computers; however, there are some easier-to-use devices that are catching on including mice, trackballs, touchscreens, graphic tablets, touchpads and even voice input devices. All of them appeal to a class of users who may be intimidated by keyboards.

Pencept Inc.'s Penpad 320, for the IBM PC and compatible systems, combines the capabilities of a keyboard, a mouse, a touchpad, and a digitizing tablet in one device. Penpad simplifies the production of business-presentation graphics by permitting the drawing, coloring, labeling, and positioning of elements directly through the pad. Making transparencies for business presentations is facilitated by the ability to integrate text and labels into the graphics. Application interfacing is available for LOTUS 1-2-3 and templates. Interfaces for WordStar, dBaseII, and others are under development. Penware software is also available. A software tool kit provides the ability to customize the Penpad for unique software and programming applications for the IBM PC.

VOICE/SPEECH SYSTEMS

Voice recognition has been like the pretty girl everyone admires but is afraid to ask out. However, with tremendous price reductions, the influence of personal computers, and the gathering interest of original-equipment manufacturers, voice recognition is being "asked out" increasingly to manufacturing applications and sized up by the business software and computer community.

The big change is price: Voice recognition, data-input method similar to a keyboard or a mouse is inexpensive. A few years ago, a voice-input device would run $10,000 or more. Now new plug-in voice boards for the IBM PC go for as little as $1,000, voice recognition chip sets are $100, and sophisticated algorithms go for $10 a copy.

Speech synthesis, a natural complement to voice recognition, completes the I/O loop. This technology has been around for a while and is inexpensive enough for many firms to justify its use. Voice vendors have begun to incorporate the technology so that PCs input terminals, and other devices can "listen" and "talk." Many of these applications consist of the hands-and-eyes-busy scenario, which vendors currently consider the mosts appropriate use for voice. SpeechLink Access lets Lotus 1-2-3 receive vocal commands, & an attendant program provides speech-synthesized

prompts. Key Tronics' speech-recognition package, 5152V, includes a speech-recognition unit housed in a keyboard, a noise-cancelling microphone, a footswitch, and vocabulary-management software. The speech-recognition board has a 160-word vocabulary. It is speaker dependent: The unit recognizes the unique sound patterns of persons for which it is trained. Speech & key input can be performed concurrently. The 5152V is compatible with the IBM PC and PC/XT.

7.2 SOFTWARE DEVELOPMENT TOOLS

7.2.1 4GLS

Fourth-generation languages are new languages different and presumably better than their third-generation "cousins" such as BASIC, COBOL, FORTRAN, or PL/1. According to Dr. George Schussel, president of Digital Consulting Associates, Inc., a fourth-generation language is a single, integrated system development tool that offers 10:1 productivity improvement over third-generation languages and has a kernal (subset) which can be learned and used in two days.

An ideal 4GL supports both a procedural language and a nonprocedural facility. Access to COBOL or other third-generation facilities is an alternative to the self-contained language.

The nonprocedural language allows the user to state his or her needs in business or logical phrases, leaving creation of code to meet those needs up to the language processor. A screen-painting facility is characteristic of 4GLs, allowing the user to create systems by defining the form, sequence and content of input and output screens, and the storage of data received through input screens. A word processing facility should be part of a 4GL. Ideally such a facility should be complete, but, minimally, it should be an editor.

Summing up the benefits of using a 4GL: Systems can be built more quickly, systems work faster, program development process speeded, and quality improvement, fine-tuned to users' needs, and coding is logically the best.

7.2.1.1 End-User Tools

Developers who want to use traditional programming languages; such as, COBOL, C, or BASIC, and want to control much of the coding, but still want to reduce their own effort, can exploit the libraries of subroutines already on the market. These subroutines involve window managers, terminal

emulators, report generators, string manipulators, communications functions, database-search software; and device, printer, screen, mouse, and terminal drivers. The Programmer's Shop, a software-supply house that specializes in tools, carries over 100 versions in the MS-DOS environment alone. So if developers want to minimize coding--or even do no coding at all--they can exploit a host of fourth-generation languages (4GLS) and application generators that have recently come on the scene. Informix-4GL and Accell make it extremely easy either to link their UNIX-based DBMS with other applications or to build new products on top of them.

One new wrinkle in the 4GL market is machine-specific 4GLs. Integrated-software developers working in the IBM area, for example, can take advantage of such languages as Mach 2 from Tominy, Inc. Capable of producing code for the entire spectrum of IBM machines from mainframes to microcomputers, Mach 2 uses an English-like structure, menus, and forms-based visual programming. A similar product in the Digital Equipment Corp. sphere is SmartStar from Signal Technologies, Inc. In addition to performing some office-automation functions, SmartStar contains a 4GL that allows developers to link code in third-generation languages (3GLs), such as, COBOL and C, with SmartStar applications into one integrated product.

7.2.1.2 Productivity Gains

A 4GL can cut programming time by upwards of 90 per cent but requires the developer to buy into somebody else's product. Tools can vastly reduce programming time and even make for better products by allowing developers to incorporate other vendor's standalone applications into their packages.

7.2.2 ARTIFICIAL INTELLIGENCE (AI)

Artificial Intelligence (AI) has been defined as the computer program that is knowledge based (logically) rather than digital in its handling of the data (numeric or text). Several programming tasks could be written in AI structure and format. These programs if they use knowledge and reasoning techniques to solve problems which normally require the abilities of human experts would be described as expert systems.

The growth of this industry, however, has not made it any easier to decide what to do about artificial intelligence. Basic questions--such as, what it is and what it's good for--remain unanswered. One reason for such questions is that

artificial-intelligence systems and conventional data processing systems are dramatically different. Conventional data processing solves problems by performing rote functions, using unambiguous data presented according to a strick format. On the other hand artificial intelligence, at least in the theory, solves problems by applying a kind of reasoning, which can accommodate uncertainty and incomplete information. It is conceivable that someday artificial intelligence might be able to perform automatic programming, guide intelligent robots, and allow machines to talk to people.

7.2.2.1 Expert Systems

Expert systems draw on a body of knowledge and make decisions more or less like people do--using fragmentary information, adjusting for uncertainty, applying informal rules of thumb where no hard and fast formulas exist. They store and use not only large amounts of data, but also patterns, rules, and strategies that mimic human reasoning.

The working parts of an expert system are a knowledge base and an inference "engine." The knowledge base stores data, rules about how to deal with the data, and statements about how they are related. The inference engine draws on the knowledge base to come up with answers.

Teknowledge, Intellicorp, Carnegie Group, and Inference all sell expert systems & expert-system development software.

7.2.2.1.1 Teknowledge

Teknowledge, Inc. of Palo Alto, California, is the second-largest artificial intelligence software vendor. The company sells two standard packages that allows users to develop their own expert systems, but the bulk of its business lies in using products itself to tailor systems for individual customers.

Its packages are not written in LISP or PROLOG, the languages used by most artificial-intelligence programmers, but in the C language. That allows them to run on any UNIX-based system which make it easier and less expensive to use Teknowledge software, compared with packages that can run only on specified LISP or PROLOG machines.

The company's standard products are S.1, which is used by programming teams to develop large-scale systems, and M.1, which is used by individuals or small groups to develop small systems. Teknowledge offers two versions of the software: The "development environment: is used by programmers to write expert systems. Once the system is developed, users can run

it on a less-expensive version of the product, usually called a "delivery system."

7.2.2.1.1.1 S.1--A development system for large knowledge-based systems, is Teknowledge's premier product. The first version, written in LISP, runs on LISP machines from Xerox Corp. and Symbolics Corp., as well as on VAX minicomputers from Digital Equipment Corp. equipped with DEC's LISP compiler. Version 2, written in the C language, runs under the UNIX operating system on the AT&T PC 7300, the DEC MicroVAX, the NCR Corp. Tower 32, and workstations from Sun Microsystems and Apollo Computer, Inc. Teknowledge plans to offer versions that run on other large UNIX systems. The price of the S.1 package that is used to develop expert systems ranges from $18,000 to $45,000, depending on the quantity ordered. The package used to run the expert systems after they have been developed costs from $1,500 to $9,000.

7.2.2.1.1.2 M.1--Version 2 of M.1 is written in C, runs on the full IBM line of microcomputers and its clones. It requires 512K bytes of memory and can store as many as 1,000 rules. The compiled C language produced by Version 2 runs several times faster than Version 1, which produces interpreted programs in PROLOG. Teknowledge Inc. announced a price cut in the form of a new package price for its existing Quick Start products and services. The Quick Start package consists of one copy of the firm's M.1 product for expert systems development on personal computers, 10 copies of the M.1 version that allows end users to run finished M.1 programs, and training and maintenance. The total cost of the package is now $7,500 down from the $14,000 charged for the separate items.

7.2.2.1.2 Intellicorp (KEE)

Intellicorp, in Mountain View, California, is the largest of the four leading vendors of expert-systems software. Knowledge Engineering Environment, or KEE, introduced in 1983, is an integrated package of software tools, which allows programmers to develop expert systems without extensive training in AI. It is based on LISP; applications developed under KEE must run under it. Using only LISP restricts the number of processors KEE runs on, most of KEE installations run on computers from Symbolics, Inc. and Xerox Corp. The price of the first KEE license is $30,000. Intellicorp won't sell that license, however, unless the customer also buys a minimum support package that costs $25,000. The package includes one year of customer support and product up-dates, seven days training for two and four days on-site consulting.

7.2.2.1.3 Carnegie Group, Inc., Los Angeles, California

To sell more products, the Carnegie Group is porting its software from LISP to the C language, which makes its software products compatible with UNIX and IBM's RT Personal Computers.

7.2.2.1.3.1 Language Craft--is a $25,000 package for developing natural-language interfaces to expert systems, data bases, operating systems, and applications.

7.2.2.1.3.2 Knowledge Craft--is designed for building large, complex industrial expert systems. The $50,000 package "is very rich in knowledge representation, and the more complex the knowledge base, the better the fit it is." Both are written in Common LISP. They run on Symbolics Inc. 3600 processors, the Texas Instrument Inc. Explorer, and the DEC VAXstation.

7.2.2.1.4 Inference Corp. (ART)

Inference Corp. located in Los Angeles, California is the oldest of the major expert-system software vendors (October, 1979). Inference's package, the Automated Reasoning Tool (ART), is the fastest, best integrated product on the market. The $65,000 package was developed from scratch which is why it is the only major expert-system development software that includes a knowledge-base compiler. ART has the most powerful "inference engine" of the leading products, meaning that it can deal with its knowledge base in more sophisticated ways. ART is written in LISP, so current users run it primarily on computers from Symbolics, Inc., Xerox Corp., LISP Machines Inc., and Texas Instrument Inc. That will soon change, however. A version of ART written in the C language will be available by the end of 1986 for the IBM RT PC and the DEC VAX minicomputer. Along with the University of Houston, Inference Corp. is working on a version of ART written in Ada which probably will be the first expert-system development software available in that language.

7.2.2.2 ADA

Ada developed to the specifications of the Department of Defense in the late 1970s for military and commercial applications may be the most standardized language in the world. The Government has reserved all rights to Ada and to market an Ada compiler that does not meet exacting Federal requirements is to risk prosecution under the law. As a result, Ada is so standardized that binary code produced from one company's compiler should look exactly the same as binary code generated by another company's compiler.

If Ada ever becomes a popular commercial language--and Ada compilers for IBM PCs and compatibles have recently appeared (You can't do large, serious expert systems on a PC yet, but as PC memory grows toward 4 Mbytes, PC expert systems are becomming more than toys)--it could land itself in a library market larger than anything yet envisioned.

The ultimate objective of aritificial intelligence (AI) software development is the embedment of AI capabilities in target computing devices. Ada programming language appears to be suitable as both a development and target language for AI embedment.

7.3 USER INTERFACES

7.3.1 TAE (TRANSPORTABLE APPLICATIONS EXECUTIVE)

TAE was developed at Goddard Space Flight Center (GSFC) to provide a standard interface to users for application program control, data analysis, user interaction and operating system services. It was developed for VAX under VMS, VAX under UNIX, PDP 11 under RSX-11M, and Data General under RDOS. Its interface modes are Menus, Command Mode, and User Created Procedures. It appears to be an effective user interface for infrequent as well as expert users. Its highly portable (87% of code), supports system extendability, and provides common interactive user interface to applications programs. It does not have a graphics interface and is not in use for real-time operations (some speed penalty) or Integration and Test which are primary user interface functions.

7.3.2 TEN/PLUS

An easy-to-use, easy-to-learn user interface, TEN/PLUS supports editing, file manipulation, mail, and networking functions. TEN/PLUS also can be extended to include new applications as they are developed by virtually any vendor. Interactive Systems Corp. has developed a family of products that can build powerful networks connecting VMS-based computer systems with UNIX- and PC DOS-based computers.

TEN/PLUS provides a consistent interface for computer users. It aims at organizations seeking a standard, easy-to-use environment that can run on different UNIX systems and on various classes of computing equipment: personal computers, multiuser microcomputers, minicomputers, and mainframes.

7.4 CASE (COMPUTER-AIDED SOFTWARE ENGINEERING)

Vendors claim their products help bring order to the process of designing software. The products help only in the beginning of the software-development process; they automate the designing of software. Actually, generating code and then debugging and maintaining--true automation or programming--is still beyond their grasp. The widespread availability of the personal computer (See illustration XIX-30) provided systems analysts an economical machine that could provide easy-to-learn, high-resolution graphics and fast response time, and facile graphics capabilities. CASE software runs on IBM PC/XTs and ATs. The system offers pull-down menus, mice or other graphics facilities that let system analysts design on screens instead of with pencils and templates.

Users may look forward to the day when the CASE products can add some AI. Having an embedded expert system on board could help an analyst find his way--and perhaps eventually enable a non-data processor to design and generate his own system and resulting code. As it becomes more sophisticated, CASE will be called upon to automate as much of the analyst's and the programmer's job as possible.

8.0 UIL REQUIREMENTS EVALUATION, ISSUES AND COMMENTS

Pre-existing Software--An important issue which remains to be addressed is how commercial off-the-shelf products (software) can be incorporated into the user interface.

Design Review by Users--The users should have an opportunity for hands-on testing of prototypes of key portions of the UI. This would enable users to provide feedback to the developers on the potential design problems before final development and delivery of the product.

The complexities of the system hardware and software should be user transparent.

There is no other language so structured or standardized and portable as Ada. Ada is capable of satisfying any practical algorithmic requirements for AI applications at speeds 10 to 200 times faster than the equivalent LISP programs. Ada was developed to the specifications of the Department of Defense in the late 1970s for military and commercial applications, and the Government has reserved the rights to Ada and its exacting requirements which must be meet by all Ada compiler vendors.

Given the wide range of users and user functions, it may be difficult to satisfy adequately all users and their functions with one language that still meets the objectives of being easy to learn and easy to use. All of the language requirements noted by the three NASA Centers are oriented toward making the language intuitive, readable, writeable, learnable, easy to use, easy to remember, system independent, adaptable to future SSP growth and expansion, and flexible. These features are intended to reduce the cognitive load on the users, allowing them to concentrate on their particular applications, and not on the details of interfacing with the system.

Therefore, for these reasons, this researcher highly recommends the use of Ada (the "Sleeping Beauty") to write the UIL with off-the-shelf software packages integrated where feasible and to have users access applications through user-friendly menus and/or touch screens, etc. in the language of their choice.

9.0 SOURCES USED

Documents

KSC SS Operations Language System (SSOL) Interactive Prototype.

SSIS User Interface Definition and Integration Task, April 28, 1986, Kennedy Space Center (KSC).

Space Station Software Recommendations. Report of an open forum held at NASA/MSFC, Huntsville, AL, April 24-25, 1985.

SS UIL Customer Data and Operations Requirements, Data Systems Technology Division, NASA/GSFC, Greenbelt, MD, April, 1986.

User Interface Language (UIL) for Space Station Crew Control.

User Interface Requirements Document (Problems Overview), NASA/MSFC UI Activity, Prepared by TRW.

Botten, Leroy, Curtis Emerson, and Karen Moe. Space Station User Interface Language Conference, April 28, 1986, GSFC.

Dickerson, Larry R. Space Station Operations Language (SSOL) System Level A Requirements Specification. Prepared by KSC Engineering Development, March 28, 1986.

Dupree, Holly and Peter L. Prun. Scoping the UIL A Total Systems Concept, MSFC, Martin Marietta, General Digital Industries. 4/28/86.

Elia, Christine. Preliminary Design and Integration of the User Interface. Presented by SS UIL Coordination Meeting at KSC, April 28, 29, 1986.

Guillebeau, M., F. Nixon, S. Owens, and Martin Ulehla. Space Station User Interface Language Requirements Specifications, Sponsored by Marietta Denver Aerospace. Prepared by TRW Systems Development Division Defense Systems Group, Huntsville Operations, MSFC.

Snyder, J. and G. Weisskopf. SSIS User Interface Definition and Integration Task. JSC, April 28, 1986.

Snyder, Joseph & George Weisskopf. Space Station Information System User Interface Concepts. JSC, March, 1986.

Snyder, Joseph. Status of SSE Procurement. JSC, April 28, 1986.

Tullis, Tom Dr. and Glen Love. Space Station Program (SSP) User Interface. Analysis Performed Under MDAC IRAD, Project 238.

Articles

"AI Is Becomming Just Another Form of DP," InformationWEEK, May 5, 1986, Page 16.

"Linking VMS with UNIX and PC DOS," SYSTEMS & SOFTWARE, February, 1986, P. 1.

"New Products Microcomputer Software - Ada Offerings," SOFTWARE NEWS, January, 1986, P. 64.

"The Current State-of-the-Art in User Interfaces," SYSTEMS & SOFTWARE, March, 1985, Pp. 131-140.

"Uniform '86: The UNIX Connections," MINI-MICRO SYSTEMS, January, 1986, Pp. 117, 118.

"Voice Recognition Starts Sounding Off," SYSTEMS & SOFTWARE, March, 1985, Pp. 55-59.

ben-Aaron, Diana, "Getting into AI: Proceed with Care," InformationWEEK, April 28, 1986, P. 40.

Briday, Robert, "Making A CASE for Computer-Aided Software Engineering," InformationWEEK, June 23, 1986, Pp. 33, 34.

Keller, Eric L., "Voice Board for IBM PC Recognizes Continuous Speech," SYSTEMS & SOFTWARE, March, 1985.

Leavitt, Don, "More Than End-User Tools, 4GLs Add to Productivity," SOFTWARE NEWS, April, 1986, Pp. 70, 72, 74, and 77.

Naedel, Dick, "Ada & AI," DEFENSE ELECTRONICS, April, 1986, Pp. 90-100.

Schindler, Jr., Paul E., "At Teknowledge, Knowledge is First," InformationWEEK, April 28, 1986, Pp. 26, 27.

_____, "Carnegie: Schooled in Expert Systems," InformationWEEK, April 28, 1986, Pp. 35-57.

_____, "Expert Systsems: AI Goes Commercial," InformationWEEK, April 28, 1986, Pp. 55-59.

_____, "Intellicorp Holds KEE to the AI Market," InformationWEEK, April 28, 1986, Pp. 30, 32.

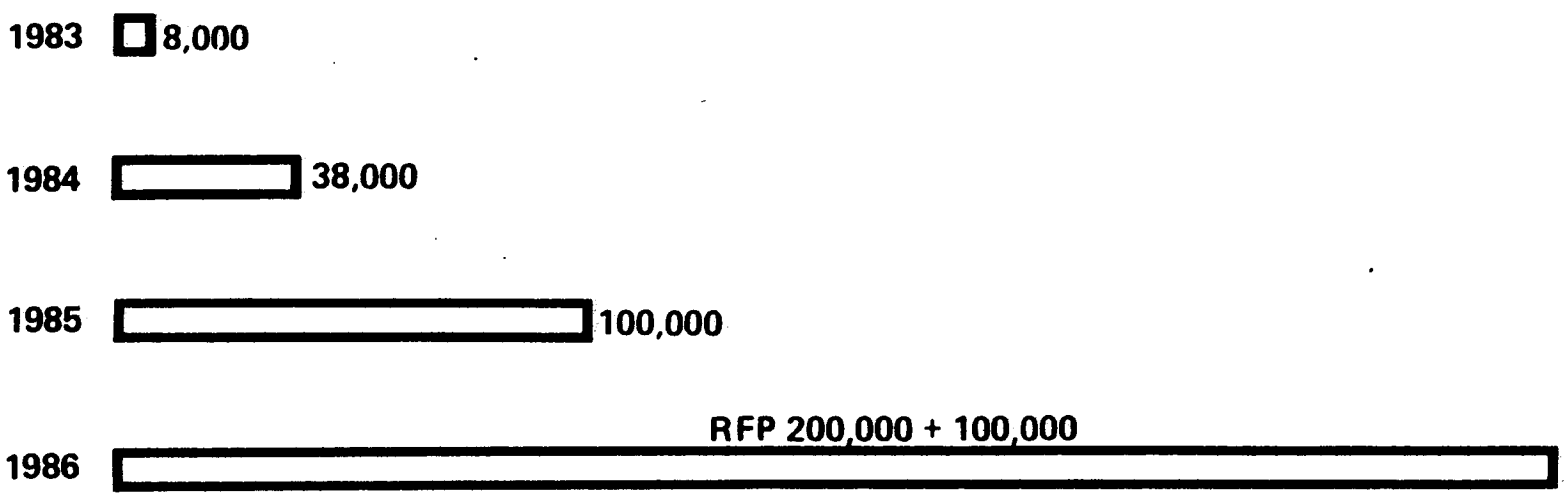
_____, "The ART of Inference: Power and Flexibility," InformationWEEK, April 28, 1986, Pp. 38-39.

Tucker, Michael (Associate Editor), "Software Tools Slash Development Time," MINI-MICRO SYSTEMS, June, 1986, Pp. 95-100.

_____, "Tools Speed Software Integration, MINI-MICRO SYSTEMS, February, 1986, Pp. 55, 56, 59, 60, and 63.

ORGANIZATION	MARSHALL SPACE FLIGHT CENTER	NAME
EB42	AN EVALUATION OF THE DOCUMENTED REQUIREMENTS OF THE SSP UIL & A REVIEW OF COMMERCIAL PKGS. FOR DEVELOPMENT & TESTING OF THE UIL PROTOTYPES	GILL
CHART NO. 12		DATE JULY 23, 1986

MILITARY/GOVERNMENT USERS
NATIONS SINGLE LARGEST CUSTOMERS FOR PCS



1990 EVERY WHITE-COLLAR WORKER IN GSA WILL HAVE A PC OR AN INTELLIGENT TERMINAL TO USE.

SYSTEMS & SOFTWARE 1/86, P.24

10.0 GLOSSARY OF ACRONYMS USED

AI	- Artificial Intelligence
ART	- Automated Reasoning Tool
ECLSS	- Environmental Control & Life Cycle Support System
ESA	- European Space Administration
ESTEC	- European Space Technology
CASE	- Computer Assisted Software Engineering
CDOL	- Customer Data & Operations Language
CDOS	- Customer Data & Operations System
DBMS	- Data Base Management System
DMS	- Data Management System
GSFC	- Goddard Space Flight Center
I&T	- Integration and Test
JSC	- Johnson Space Center
KEE	- Knowledge Engineering Environment
KSC	- Kennedy Space Center
LeRC	- Lewis Research Center
MMI	- Man-Machine Integration
MS DOS	- MicroSoft Disk Operating System
MSFC	- Marshall Space Flight Center
PCP	- Phoenix Computer Products
SAIS	- Science Application Information System
SE	- Support Environment
SS	- Space Station
SSD	- Space Station Data
SSIS	- Space Station Information System
SSOL	- Space Station Operations Language
SSP	- Space Station Program
TAE	- Transportable Applications Executive
UI	- User Interface
UIL	- User Interface Language