N93-32102

AUTOMATED ASSEMBLY OF LARGE SPACE STRUCTURES USING AN EXPERT SYSTEM EXECUTIVE

Cheryl L. Allen; NASA Langley Research Center; MS 152D; Hampton Va 23665-5225

ABSTRACT

NASA Langley Research Center has developed a unique testbed for investigating the practical problems associated with the assembly of large space structures using robotic manipulators. The testbed is an interdisciplinary effort which considers the full spectrum of assembly problems from the design of mechanisms to the development of software. This paper will describe the automated structures assembly testbed and its operation, detail the expert system executive and its development, and discuss the planned system evolution. Emphasis will be placed on the expert system development of the program executive.

The executive program must be capable of directing and reliably performing complex assembly tasks with the flexibility to recover from realistic system errors. By employing an expert system, information pertaining to the operation of the system was encapsulated concisely within a knowledge base. This lead to a substantial reduction in code, increased flexibility, eased software upgrades, and realized a savings in software maintenance costs.

INTRODUCTION

Projected crewed missions to the moon and Mars represent a departure from previous space endeavors in that the large vehicles involved will have to be assembled and checked out on orbit. The construction of these vehicles will require extensive in-space operations calling for enhanced capabilities in the areas of assembly and servicing. In order to perform these functions with the limited crew resources available, a much higher level of automation must be realized than is currently obtainable. NASA Langley Research Center has developed a unique testbed to investigate the practical problems associated with the automated assembly of large space structures using robotic manipulators. The research program is an interdisciplinary effort which considers the full spectrum of assembly problems from the design of mechanisms compatible with automated operations to the definition of software structures and algorithms required for their support.

The LARC research program adheres to several principles and ground rules: (1) all system development, testing, and demonstration is performed using realistic test hardware, which is felt to be the only way to identify all the problems associated with automated assembly; (2) system design and automation are considered integrated and complimentary technologies with solutions developed cooperatively; and (3) the program is targeted towards a fully automated system that utilizes either an astronaut or earth based operator as a monitor who is called upon only when the robotic system encounters a problem requiring intervention or assistance. The third principle describes a mode of operation known as supervised autonomy which holds the most promise for the accomplishment of large construction tasks with the limited crew resources available on orbit.

The purpose of this paper is to briefly describe the automated structures assembly testbed and its operation, to detail the expert system executive and its development, and to discuss the system expansion currently underway. The emphasis of the paper is on the expert system implementation of the program executive, however the system components are described and a narrative of the assembly process is given to serve as a basis for the description of the software and its functions.

FACILITY DESCRIPTION

The Automated Structures Assembly Laboratory (ASAL) is shown in figure 1. Figure 1a shows a schematic of the assembly system with the major components labeled, and figure 1b is an actual photo of the facility in operation. The assembly system consists of a robot arm, a motion base system, two specialized end effectors, components for a truss assembly, and storage canisters for those components. The ASAL utilizes commercially available equipment to minimize cost and ease modification as research needs dictate. The hardware system is a ground-based research tool designed to permit evaluation of assembly techniques, strut and end effector components, computer software architecture and algorithms, and operator interface requirements.

The structure selected for assembly is a planar tetrahedral truss which supports hexagonal reflector-type panels (see figure 2). The completed structure consists of 102 two-meter-long strut members and 12 panels measuring approximately 2.3 meters across the vertices. The structure was designed to be a laboratory prototype representative of the type of structures which support the functional surfaces of a number of planned or proposed missions, such as antennas and aerobrakes.

A brief description of the major components will follow; however, the details of the facility hardware, performance characteristics, and assembly procedures can be found in references 1-3.

Robot Arm

The robot arm is an electronically driven six-degree-of-freedom industrial manipulator selected for its reach envelope, payload capacity, positioning repeatability, and reliability. The robot arm computer is based on a 68000 microprocessor and all robot motions are programmed in a modified BASIC programming language. No modifications have been made to the manipulator other than those available from the manufacturer.

Motion Base System

The motion base system includes a linear translational x-y Cartesian carriage and a rotating turntable. The robot arm is mounted on the carriage, and the truss is assembled on the rotating turntable located at one end of the x carriage. Motion base drive motors on all three axes are commanded by a 80286 micro-processor based indexer.

End Effectors

The end effectors, as shown in figure 3, are specialized tools mounted on the robot arm which perform the strut and panel installation and removal operations. Figure 3a shows the strut end effector and figure 3b shows the panel end

effector. All end effector operations are controlled by an onboard microprocessor mounted near the robot wrist. Typical microprocessor operations are detailed in reference 4. All end effector mechanisms are equipped with simple sensors such as microswitches and linear potentiometers to monitor operations and notify the operator if a problem occurs. The processor is programmed in ANSI compatible C and includes sufficient I/O to monitor the sensors associated with the end effector mechanism operations. A commercial force/torque load cell is mounted between the end effector and the robot arm to provide compliant move capability during both strut pick-up and installation operations.

Truss/Panel Elements

The truss joints and nodes designed for this assembly are shown in figure 4. The joint is composed of two parts, a connector section which is bonded to the graphite-epoxy tube to form a strut and the receptacle section which is mechanically attached to the node. The truss members are connected by specially designed connector joints located near the nodes. The strut end effector grasps and holds the joint receptacle to provide stability during strut installation and removal (ref. 5). The strut end effector uses pneumatically actuated receptacle fingers to grasp passive guidance v-grooves on the node receptacles. After the end effector inserts the strut into the receptacle, locking nuts are turned by a small electric gear-head motor, securing the strut into place. Assembly begins by connecting struts to three nodes that are premounted on the motion base turntable.

As the truss assembly progresses, the panels are placed on the nodes at the top of the truss using the panel end effector (ref. 6). The panel is an aluminium hexagonal frame with a reflective mylar covering. Once in position the panels are locked into place using end effector actuator pins.

Storage Canisters

The truss struts are stored in nine trays which are stacked in the working canister directly behind the robot arm. Each tray is fitted with handles which allow the strut end effector to pick up empty trays from the working canister and transfer them to the storage canister located to one side of the robot arm.

The panels are stored vertically in a large canister at one end of the y carriage. The same pins that are used to attach the panels to the truss structure are also used to latch the panels in the canister.

ASSEMBLY PROCEDURE

The assembly process begins when the strut end effector acquires the first strut from the top tray in the working canister. Once acquired, the strut is carried above the working canister and the motion bases are positioned so that the robot arm can reach the required installation position. The robot arm then moves through a sequence of predetermined points, arriving at an approach point located approximately 12 inches from the intended installation point in the structure. At the approach point, control is turned over to a machine vision system.

The machine vision system uses two small video cameras located on the end effector to view targets made of reflective material mounted on the node receptacles as shown in figure 5. A special five dot domino pattern is used as the target. The video image of the target is processed to discriminate the target from the background and the centroids of the dots are determined. The position of the centroids are defined with respect to the camera using a pose estimation routine. The pose information is used to direct robot arm moves toward the target location for strut installation. Details of the vision system can be found in reference 7. Once the arm reaches the installation point, the vision system relinquishes control and the end effector grapples the strut receptacles in the structure, repositions the robot arm to reduce forces and torques at the end effector that are caused be minor positioning errors, and inserts and locks the strut joint. The robot arm then returns to the working canister for another strut.

Once a specified number of struts have been installed, panels can be secured to the top of the structure. This involves stowing the strut end effector by latching it to the tray in the top of the storage canister and picking up the panel end effector stored at one end of the panel canister. This end effector change is accomplished by a commercially available pneumatic quick-change mechanism. Panels are retrieved using y-carriage motion base moves, and installed at predetermined points atop the structure. Machine vision is not used for the placement of panels in the structure.

Combinations of strut and panel installation sequences are currently followed until a platform with 102 struts and 12 panels is completed.

SYSTEM CONTROL AND COMMUNICATIONS

The ASAL facility is managed by several digital computers serially connected through RS232 communication lines as depicted in figure 6. The system executive and operator interface functions are performed on a micro-VAX workstation. The robot motions, carriage movements, and end effector operations are executed on individual processors, as are the computations required by the vision system.

Software Design

The design layout for the assembly system software is illustrated in figure 7 and detailed in reference 8. The software is arranged into four hierarchical levels of commands (Administrative, Assembly, Device and Component) each of which decompose into a sequence of commands for the next lower level. The highest or administrative level performs the preliminary setup of the system. The operator can examine and modify data and system options. Command and assembly sequence files can be selected, created and modified. It is this level that is intended to interface with a goal-directed task sequence planner. Currently the assembly sequence is manually determined and maintained in a file. Each entry in the assembly sequence file represents an appropriate assembly level command which specifies the operations to be performed on a given element (ie. strut or panel). The standard operating mode is centered at the assembly level and reflects the automated aspect of the system. At this level the software manages all the devices, data verification, and error recovery. The assembly level commands decompose into a series of commands for each of the three devices; the motion bases, the robot arm, and the end effector.

Although the assembly software system is intended to operate in a fully automated mode, it is imperative that the operator be provided with sufficient internal information and have command access and authority at all levels to deal effectively with assembly errors. The operator has complete control of error recovery and final decision on error resolution. The operator may decide that an error is not severe and command the system to proceed anyway. Also, if none of the recovery options presented are successful, the

operator may instruct the system to abort the failed operation and automatically roll the assembly process back to a known, successful condition. During assembly operations, the operator has the capability to pause the assembly process at any point and observe some detail using a video display before either continuing or reversing the sequence. This intervention capability imposes a significant burden on the system software.

The system executive directs and monitors assembly operations across the various processors, and reports current status information to the operator. The executive maintains the conditions and constraints of the assembly operations, including details of the geometry of both the structure and the storage canisters. During an assembly, the executive makes decisions about what end effector to use and the procedures required for it's use. Finally, the executive keeps track of possible problems and recovery techniques for all assembly scenarios. In order to do this effectively, the executive has full access to the current status of the assembly operation and the system hardware including complete, detailed descriptions of the state of the assembled structure, the motion base, the robot arm, and the end effector hardware. This information is continuously updated based upon verification by sensors.

Initially, the assembly executive software was written in FORTRAN. The procedural language was already familiar to developers in ASAL and therefore could be used to verify and refine assembly system operations in a relatively short period of time. The initial task was to construct a simplified structure of 102 struts, using a single, premounted end effector who's functions were commanded via the robot arm computer. The robot arm moved to predefined installation positions without the use of machine vision. As the scope of the research project grew (with the addition of panels, a second end effector, and distributed processors) the complexity of the knowledge to be managed by the assembly software increased. Because traditional programming languages proved to be cumbersome in keeping pace with system upgrades, the decision was made to rewrite portions of the software using an expert system. The first level of code targeted for this transition was the decision-intensive assembly executive. The following sections describe the application of expert system techniques to the assembly executive, giving examples of their use.

Expert System Assembly Executive

An expert system is a computer program that uses knowledge and reasoning techniques to solve problems that normally require the services of a human expert. Like conventional programs, expert systems usually perform well defined tasks; however, unlike conventional programs, expert systems also explain their actions, justify their conclusions, and provide details of the knowledge they contain. Expert systems are ideal for capturing and utilizing the "rules of thumb"-type logic that evolves from the experience gained in ASAL.

The assembly executive is responsible for making decisions about the actions to take (and the order to take them) during the construction of a given structure. To make informed decisions, the executive has to have access to all current system information, and the knowledge to evaluate that information in light of the desired task. It is this decision-making component of the assembly executive that was best suited to implementation using expert system techniques.

Methodology

A subset of the general area of expert systems concentrates on explicitly representing an expert's knowledge about a class of problems and then providing a separate reasoning mechanism (called an inference engine) that operates on this knowledge to produce a solution. These kinds of systems are known as knowledge-based expert systems. The knowledge base is a file containing the facts which make up the human expert's knowledge about a specific domain. An inference engine is a program that applies reasoning techniques to the facts, as defined by the knowledge base, to draw conclusions. Inference engines vary according to the representation of the knowledge and the strategy for applying the knowledge.

There are a variety of expert system development tools available to assist programmers in building powerful systems capable of solving a wide range of problems. The commercially available Knowledge Engineering System (KES (ref. 9 and 10)) was selected for use in ASAL. The KES tool provides the inference engine, knowledge representation schemes, and facilities for creating an operator interface. KES provides an embedding technique for integrating expert systems with existing software by allowing procedural language code to send, receive, and modify data from a knowledge base through the use of special data types and run-time functions.

The KES inference engine uses rules to represent knowledge. This knowledge representation scheme is particularly well-suited to applications such as automated assembly where the facts can be organized in the form of branching logic or if-then constructs. KES uses deductive reasoning as the technique for problem solving, where certain outcomes follow directly from certain inputs.

The pursuit of a solution (or goal) drives the reasoning methodology used by KES. This goal-driven inferencing technique is known as backward chaining. Implicit subgoals are set up to determine values for attributes that appear in the antecedent of a rule that infers a value for some other attribute, and so on, until a value for the goal attribute has been determined. In addition to goal-driven inferencing, KES also performs event-driven inferencing through the use of demons. Event-driven (forward chaining) inferencing takes place when the expert system responds to the occurrence of an event rather than the pursuit of a goal.

The following section will describe how the fore-mentioned methodologies have been applied to the automated assembly system software in ASAL using KES.

Implementation

As mentioned previously, the executive portion of the assembly system software was the first to be implemented as a expert system. The executive is responsible for managing all the devices (the motion bases, the robot arm, the end effectors, and the vision system), data verification, and error recovery. Figure 8 illustrates where the knowledge base fits into the overall software system architecture. By embedding the knowledge base in the automated assembly system , the executive has access to expert system methodologies for decision-making while leaving the already familiar operator interface and existing database management schemes intact.

The operator gains access to the executive through a menu-driven interface. By implementing a menu-driven interface, the operator is only presented with the commands he needs at any given time. As shown in figure 8, a layer of procedural code (FORTRAN and C routines) surrounds the knowledge base and handles the menuing functions and information exchange between the knowledge base and the

hardware. Database information is also transferred through this surrounding code. The knowledge base contains the data constructs (attributes and classes), rules, and demons necessary to make informed decisions about assembly actions.

The expert system uses the knowledge base as the primary source for determining the command sent to a particular device at any given time. Commands are sent to the individual processors associated with the specific hardware device for interpretation and execution. When up-to-date information about a piece of hardware is needed, sensors are polled through the device interfaces and the information is passed back to the knowledge base. After a device-specific processor has completed processing a command, a return status is forwarded to the knowledge base so the next action can be sent. In the case of a successful return, the database is updated and the next command in the sequence of assembly actions is determined. In the case of an error, instructions to return to the last known successful state may be issued. Information about all system functions is constantly updated and reported to the operator via status windows.

The structuring and content of the knowledge base lies at the heart of the expert system, and therefore warrants further consideration. The next sections will detail the more important components of the knowledge base, and present examples of their application.

Classes-

KES uses a structure called a class to describe a group of objects having the same set of characteristics. Each object is referred to as a member of the class, and each characteristic is maintained in a class construct known as an attribute. Two classes are defined in the current automated assembly knowledge base; one for struts and one for panels.

There are 102 unique members in the strut class; one for each strut in the truss assembly. An example of the attribute declarations for the strut members is shown in figure 9. The values associated with these attributes are stored in a database and are associated with some physical aspect of the strut and the way it is stored in the canister or installed in the structure. As indicated in the figure, there are 13 attributes identified for struts: three associated with naming conventions (OBSERVER NAME, ALTERNATE NAME, and ROBOT NAME); two identifying the canister storage location (TRAY, SLOT); five containing information about the physical characteristics of the strut (NODE END) and any special conditions for installation (CAP END, FLIP, CAN_FLIP, and NODE DIRECT); one to track the current location of the strut (WHERE); and two that define carriage positions of the robot during installation (MB_INDEX1 and MB_INDEX2). Additional information regarding these attributes can be found in reference 8.

A class has also been defined for panels and contains information pertaining to the installed location for the panel and the whether or not the panel is installed.

Rules-

Rules are the most powerful knowledge source available to the inference engine. They represent the expert's knowledge, and they direct the actions of the expert system towards a desired goal. The general format of a rule is

if antecedent **then** consequent **endif.**

The antecedent is a logical comparison which evaluates to either true or false. The antecedent must be true for the consequent to be performed. The consequent consists of KES commands which contribute, or drive, the system toward a goal. For the assembly executive, the rules formulated require an intimate knowledge of the physical operations, potential system states, and capabilities of the various hardware. Rules have been defined for capturing information pertaining to tray transfer operations, and path segment selection for strut and panel installation/removal operations.

The path the robot arm travels from a rest position above the storage canister to the installation point in the structure is divided into segments or states. Figure 10 presents two rules that are used to determine the next segment (next_state) in the installation path for a strut. For this illustration the robot arm is poised above the supply canister awaiting direction to proceed to the grasp point of the canister. The current location of the robot arm (current_state) and the direction of the robot arm's motion (phase) determine the next segment in the robot's path. The robot's phase (either into or out of the structure) is determined from the current location of the robot (current_state), the current location of the strut (current_strut>where), and the task or goal specified by the operator (target_state). The current location of the robot is maintained in a database, and the location of the strut is held within the class member for that strut. To determine whether or not the consequence of the *state* rule is performed, the *phase* rule must be evaluated. The execution (called firing) of a rule often depends upon other rules being satisfied. It is this backward chaining technique that makes rules so powerful.

The strut installation path from the pickup point of the strut at the canister through the installation point at the structure and return requires 22 rules. These 22 simplified rules replaced approximately 850 lines of FORTRAN code. The total knowledge base currently contains 59 rules; twenty-two rules for determining strut assembly paths as previously indicated, twenty-two for panel paths, and fifteen for transferring trays from the supply canister to the storage canister and vice versa.

Demons-

Demons provide a method for event-driven inferencing within KES. Where rules actively seek additional information in an attempt to satisfy a specific goal, demons remain passive until an event occurs which initiates their execution. Adding event-driven inferencing (demons) to backward-chaining inferencing (rules) makes for a more dynamic expert system by providing a natural way of expressing some types of knowledge. Demons are useful for monitoring attributes for new or changed values in an attempt to modularize the procedural portions of the knowledge base.

A demon is composed of two parts; a guard and a body. A guard is similar to the antecedent of a rule, and contains conditional statements to be evaluated. The body contains a list of commands that KES executes sequentially. Assigning a new value to an attribute in the guard constitutes an event, causing all associated demons (ie. demons with that attribute in their guard) to be evaluated. If the guard evaluates to true, then KES executes the commands in the body of the demon. In the assembly executive knowledge base, when a value is assigned to an attribute in the consequent of certian rules, a demon is activated, initiating event-driven inferencing.

Suppose the *state* rule of figure 10 evaluates to true, and the next segment in the strut installation path is determined to be the canister grasp point (GP_CAN). The demon in figure 11 is used to generate the command strings necessary to move the robot arm to GP_CAN. Following the example, first some preliminary flags are set and the end effector conditions are checked. By assigning a value of true to the attribute *check scar* (a), another demon is activated which

makes sure the end effector is in the configuration necessary for making a safe approach to the canister. The value returned by the end effector is stored in the attribute *ee response*, which is examined before continuing (b). An uncorrectable error during the end effector operation would cause a roll back of the system to the last successful state (c). A successful return from the end effector allows the expert system to send a command to the processor associated with the robot arm to reset the force/torque sensor (d). Installation conditions for the current strut are ascertained (e) before the command to send to the robot is synthesized (f and g). The slot and tray numbers are appended to the base command string (h), and the command is sent (i). The assignment of true to the *send merlin* attribute constitutes an event which activates yet another demon. The send merlin demon sends the command and evaluates the robot response. If the device operated successfully, the current state is updated (j). The *message* command (j) is the means for sending the new value for the robot state to the database through the embedded interface. An unsuccessful robot operation results in a reverse (k and l).

A demon can change the value of the attribute that triggered its execution resulting in recursive behavior. The body of a demon can also determine the value of another attribute which itself may have associated demons. These demons can be triggered, invoking forward chaining. By blending both forward and backward chaining in a recursive environment, the assembly executive knowledge base has evolved into a concise and powerful mechanism for representing assembly knowledge.

Benefits

The concise representation afforded by the rule-based system reduced the lines of code significantly over the procedural (FORTRAN) version. A number of additional capabilities have been added to the system (panel operations, end effector changes, and machine vision), and the number of lines of code is still far below that of the original FORTRAN version. This reduction has lead to increased maintainability, and modifications and upgrades have been performed rapidly. The knowledge base is easier to debug and modify because the knowledge is separate from the algorithms and is readily accessible at run time.

This structural assembly project is relatively simple compared to many of the in-space check-out and servicing tasks currently being proposed. Expert system techniques have already proven to be mandatory for effective system management in ASAL. Such knowledge-based methodologies are a requirement for the timely development and maintenance of these types of complex systems.

RESEARCH OPPORTUNITIES

The overall goal of the ASAL research is to develop a complete integrated assembly system which incorporates on-line, automated planning and scheduling functions. The expert system executive described in this paper represents a first step in an evolution toward such advanced capabilities.

A baseline automated assembly system for space structures has been successful in assembling and disassembling a 102-member tetrahedral truss and demonstrating the utility of a supervised autonomy mode of operation. Complete assembly of the truss with the 12 attached panels using machine vision and the microprocessor controlled end effectors, all under the control of the expert system executive is being initiated. This test will demonstrate the capabilities of both the hardware and the software. In addition, performance data will be gathered which will help direct the evolution of the system. An attempt will be made to quantify error recovery actions taken by the operator with the goal of automating many error recovery procedures.

Currently when an error occurs, a menu of potential solutions is presented to the operator. The operator must then assess the error by visually verifying sensor data and select one or more options from an error recovery menu. By recording and studying the operators choices, the state of the system when the error occurred, the order in which error recovery actions are attempted, and the successful actions as well as failures, it is hoped that many processes can be automated. The final decision on error resolution will still rest with the operator, but a number of historically successful error recovery actions can be attempted before operator intervention is requested.

The enhancement with the largest software impact within ASAL will be the change over from the current system architecture (as seen in figure 6) to the highly distributed architecture as depicted in figure 12. Under this new architecture, all the devices will have their own processors, and will be controlled by an expert system scheduler. Maintaining separate devices for the individual processors will allow for concurrency among many assembly operations.

A number of advanced planners, each with their own knowledge base, are also included in the design of the new architecture. Knowledge bases will exist for: (1) a tray storage planner so that a fixed tray and slot assignment per strut will no longer be necessary, (2) a task planner for developing assembly scenarios based upon a definition of truss geometry and stiffness characteristics, (3) a path planner for determining a collision free path to the structure without having to rely on pre-determined approach points, and (4) a planner for combining necessary operations as a logical sequence and determining what actions can take place concurrently.

To manage the increased number of knowledge bases and individual processors, the KES software has been upgraded to an applications development tool known as the Strategic Networked Applications Platform (SNAP). SNAP supports the development of applications that operate in a distributed hardware environment. SNAP is made up of five components of pre-built software, the most important of which is the object model, containing the information driving the application. SNAP supports an object-oriented model of application data providing a direct mapping of the real world objects associated with the application to objects in the object model. Objects (classes) defined in the object model can be processed using either a rule-based knowledge source (backward chained rules), event-driven procedures (demons), or functions. Other components for building end-user interfaces using windows, mapping to permanent storage such as databases or files, integrating new or existing code, and communicating with other devices combine with the object model to make a complete application. Existing KES applications can be directly converted into SNAP compatible applications.

CONCLUDING REMARKS

The research conducted in ASAL has successfully demonstrated the viability of using robotic manipulators to automatically assemble and disassemble large truss structures. During the construction of a given structure, the system software assembly executive is responsible for making decisions about the actions to take, and the order in which to take them. To make informed decisions the

executive has to have access to all current system information, and the knowledge to evaluate that information in the light of the desired task. Due to the complexity of the software, continued implementation in traditional programming languages (ie. FORTRAN) became prohibitive. Traditional programming languages are not well suited for encapsulating the knowledge required for intricate assembly sequences. Preliminary investigations into the application of expert system technologies to perform the decision-making portions of the assembly software have been very encouraging.

Planned enhancements include implementation of a distributed architecture and several advanced planners. Multiple devices, each with their own processors, will be controlled by an expert system scheduler. The addition of a number of advanced planners, each with their own knowledge base, will make for a robust and reliable assembly system.

REFERENCES

1. Rhodes, Marvin D.; Will, Ralph W.; and Wise, Marion A.: *A Telerobotic System for Automated Assembly of Large Space Structures,* NASA TM-101581, March 1989.

2. Rhodes, Marvin D.; and Will, Ralph W.: *Automated Assembly of Large Space Structures,* 41st. International Astronautical Congress, Dresden, GDR, October 6-13, 1990.

3. Will, Ralph W.; Rhodes, Marvin D.; Doggett, William R.; Herstrom, Catherine L.; Grantham, Carolyn; Allen, Cheryl L.; Sydow, P. Daniel; Cooper, Eric G.; Quach, Cuong C.; and Wise, Marion A.: *An Automated Assembly System for Large Space Structures.* Intelligent Robotic Systems for Space Exploration. edited by Alan Desrochers, Academic Publishers, 1992.

4. Doggett, William R.; Rhodes, Marvin d.; Wise, Marion A.: and Armistead, Maurice F.: *A Smart End-Effector for Assembly of Space Truss Structures,* Fifth Annual Space Operations, Applications, and Research Symposium, Houston, TX, July 9-11, 1991.

5. Wu, K. Chauncey; Adams, Richard R.; and Rhodes, Marvin D.: *Analytical and Photogrammetric Characterization of a Planar Tetrahedral Truss.*, NASA TM-4231, 1990.

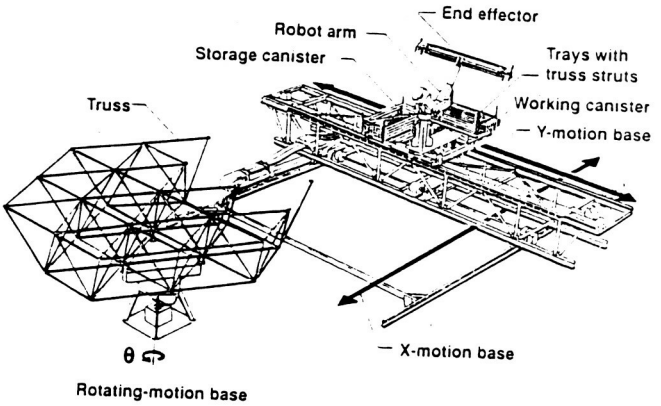
6. Kenner, Scott W.; Rhodes, Marvin D.; and Fichter, W.B.: *Component Count and Preliminary Assembly Considerations for Large Space Structures,* NASA TM-102604, Feburary 1990.

7. Sydow, P. Daniel ; and Cooper, Eric G.: *Development of a Machine Vision System for Automated Structural Assembly*, NASA TM-4366, May 1992.

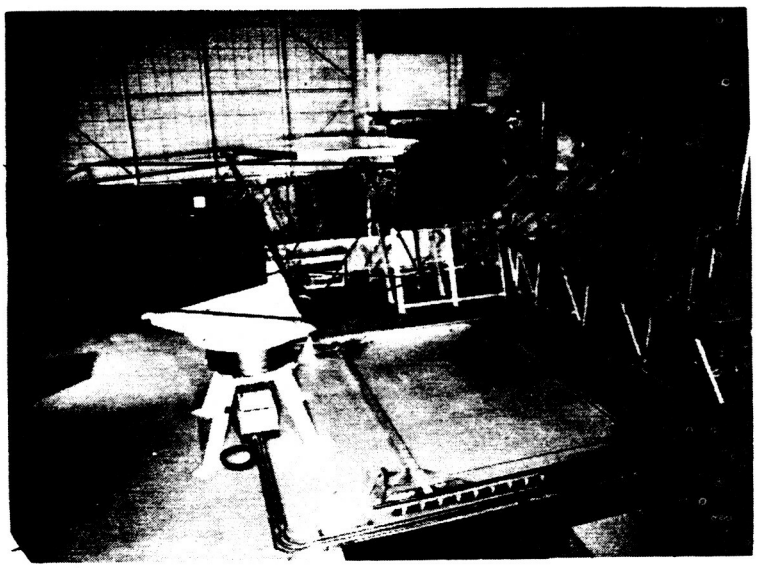
8. Herstrom, Catherine L.; Grantham, Carolyn; Allen, Cheryl L.; Doggett, William R.; and Will, Ralph W: *Software Design for Automated Assembly of Truss Structures,* NASA TP-3198, June 1992.

9. *Knowledge Base Author's Manual - KES PS.* Software Architecture & Engineering, Inc., c.1990.

10. *Knowledge Base Author's Reference Manual - KES PS.* Software Architecture & Engineering, Inc., c.1990.



(a) Schematic



(b) Photograph

Figure 1. Automated Structures Assembly Laboratory

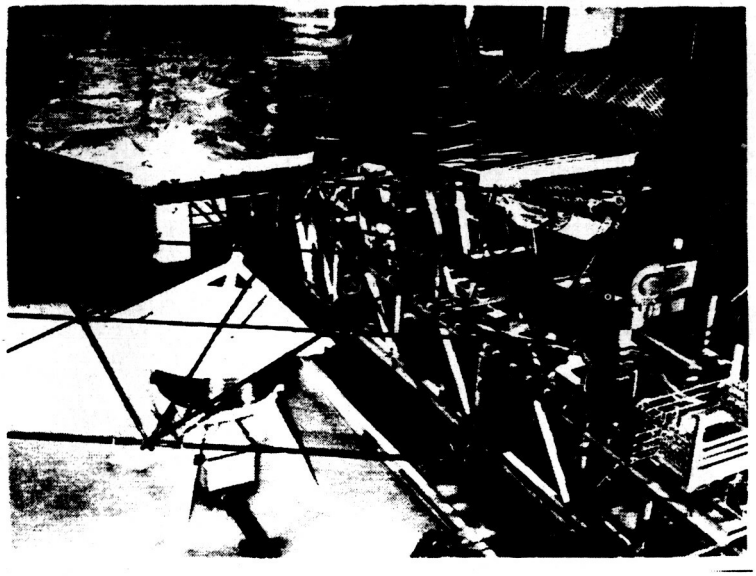
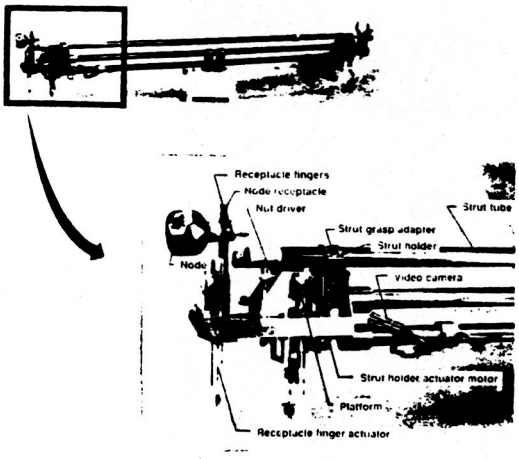
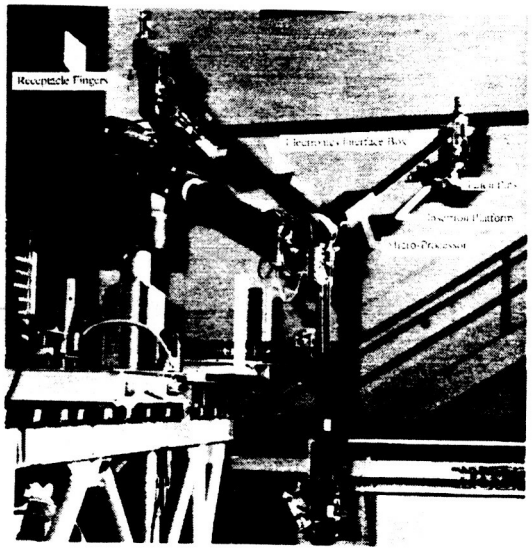


Figure 2. Tetrahedral Truss with Hexagonal Panels



(a) Strut End Effector



(b) Panel End Effector

Figure 3. ASAL End Effectors

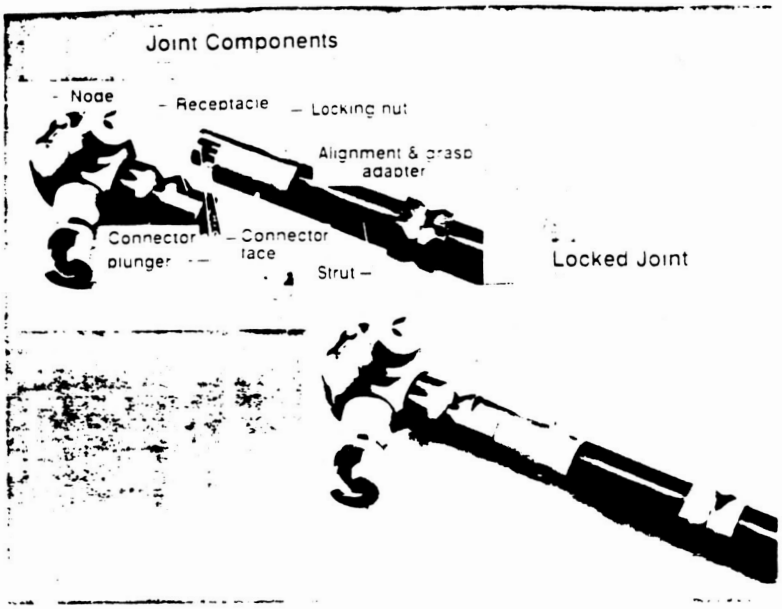


Figure 4. Strut/Node Joint Connector Hardware

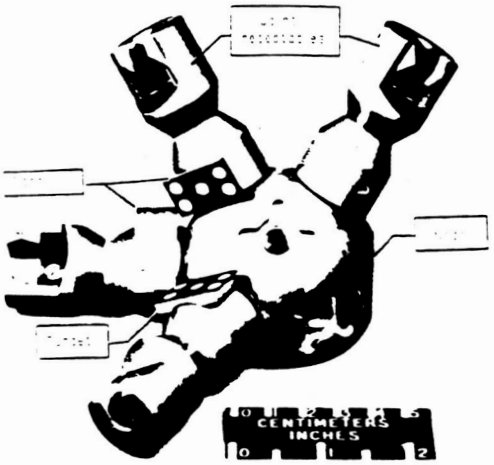


Figure 5. Truss Node with Joint Receptacle Targets

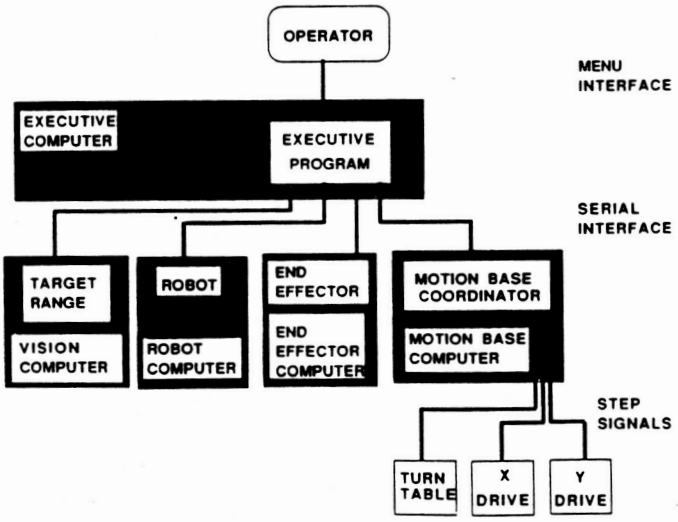


Figure 6. ASAL Computer Control System

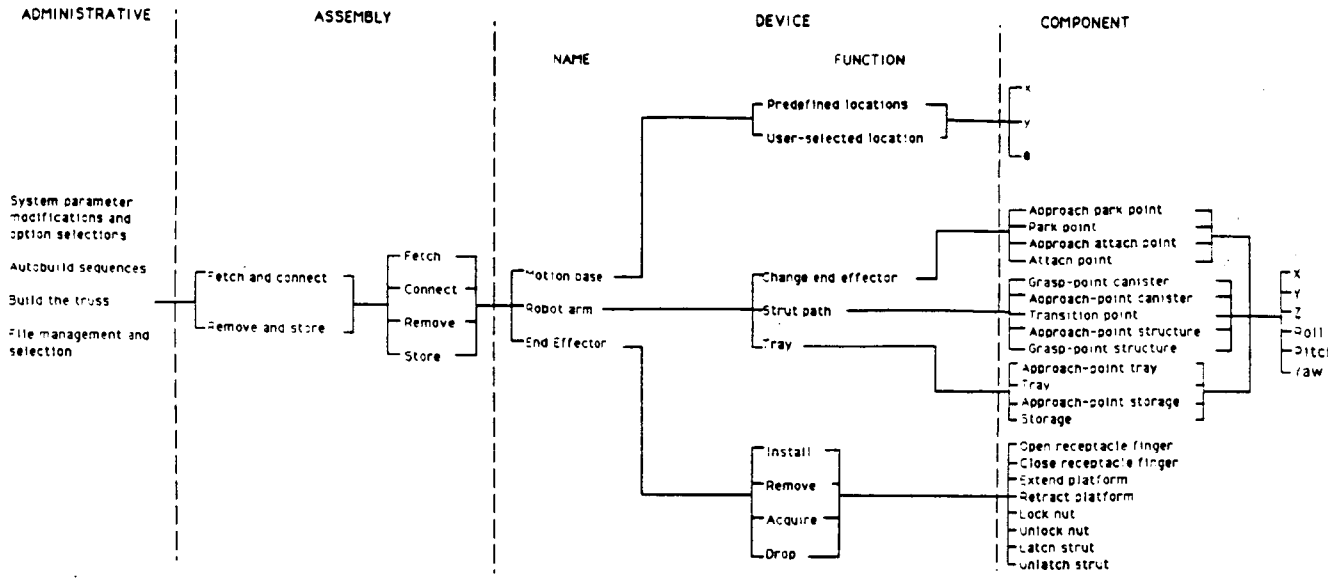


Figure 7. Software Design Layout

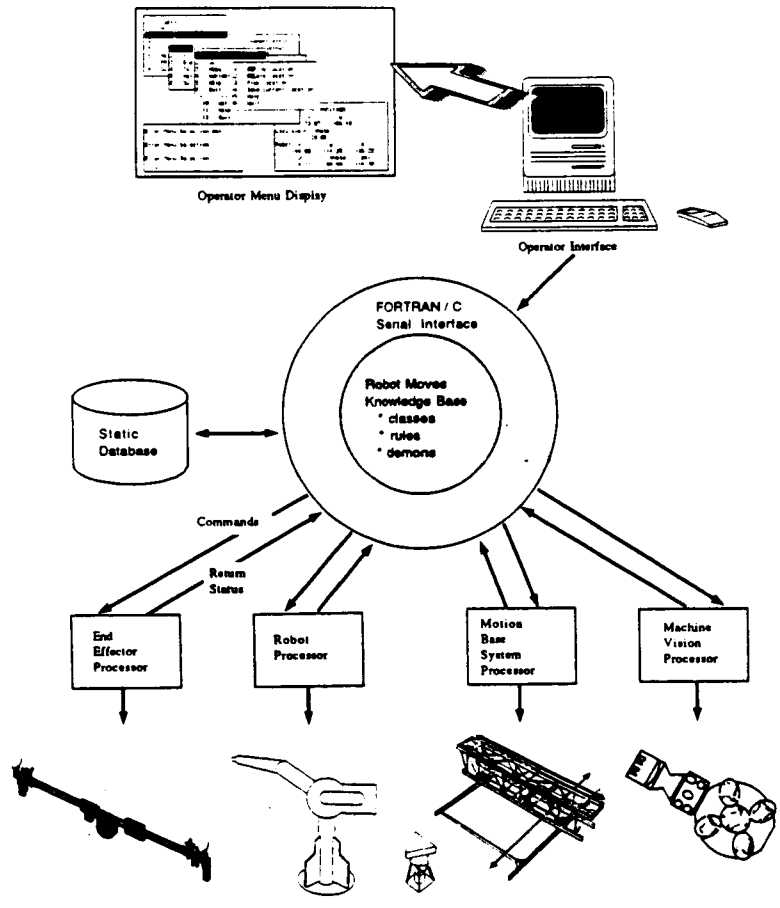


Figure 8. ASAL Software Architecture

```
Classes:

 STRUTS:
      attributes:
            OBSERVER NAME: str.
            ALTERNATE NAME: str.
            ROBOT NAME: str.
            TRAY: int.
            SLOT: int.
            NODE END: str.
            CAP END: str.
            WHERE: sgl (CANISTER, INSTALLED, ARM).
            FLIP: str.
            CAN_FLIP: truth.
            NODE DIRECT: str.
            MB_INDEX1: int.
            MB_INDEX2: int.
      %
endclass.
```

Figure 9. Class Defination for Struts

```
State:
 if
       current_state = AP_CAN* and
       phase = out
 then
       next_state = GP_CAN**
 endif.


Phase:
 if
       current_state = AP_CAN and
       target_state = GP_CAN and
       current_strut>where = CANISTER | ARM
 then
       phase = out
 endif.

*      AP_CAN : Canister approach point
**     GP_CAN: Canister grasp point
```

Figure 10. Example Rules for Strut Path Determination

```
State GP_CAN:
      when
            next_state = GP_CAN
      then
            reassert rule_flag = false.
            erase status_mode.
(a)         reassert check_scar = true.
(b)         if ee response = reversed then
(c)               reassert return = true.
            else        \ee response = worked
                  if ((init = false and restart = false) or override) and
                  status_mode = false then
(d)                     message "COMMAND$reset fts".
                  endif.
(e)               if current strut>CAN_FLIP then
(f)                     reassert tomerl = "GOTO GP_FLIP_CAN*"
                  else
(g)                     reassert tomerl = "GOTO GP_CAN*"
                  endif.
                  if determined (current strut) then
(h)                     reassert tomerl = combine(tomerl,current strut>SLOT,"*",
                                                  current strut>TRAY).
                  endif.
(i)               reassert send merlin = true.
                  if halt_op = false then
                        if robot success then
(j)                           message "UPDATE$charstate,GP_CAN".
                              reassert current_state = GP_CAN.
                        else  \return to calling state
                              if current strut>CAN_FLIP then
(k)                                 reassert tomerl = combine(
                                                "GOTO REV_GP_FLIP",
                                                current strut>SLOT,"*",
                                                current  strut>TRAY).
                                    reassert send merlin = true.
                              endif.
(l)                           reassert return = true.
                        endif.
                  endif.
            endif.
      endwhen.
```

Figure 11. Demon for Moving Robot to Canister Grasp Point

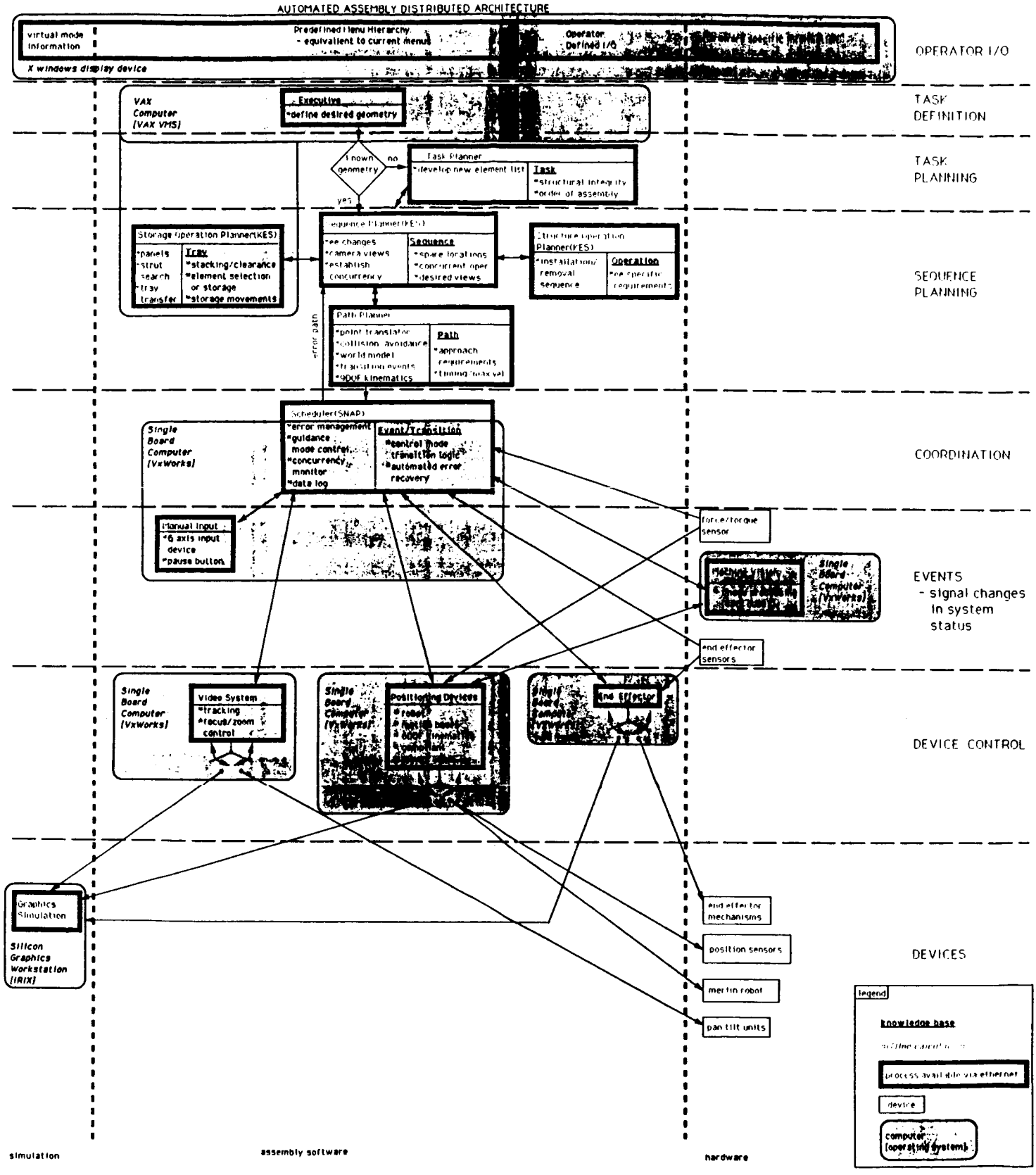


Figure 12. ASAL Distributed Architecture