Advanced Data Management Design For Autonomous Telerobotic Systems in Space Using Spaceborne Symbolic Processors.

Andre Goforth
Information Sciences Office
Intelligent Systems Technology Branch, Code RII
NASA Ames Research Center
Moffett Field, California 94035

ABSTRACT:
The use of computers in autonomous telerobots is reaching the point where advanced distributed processing concepts and techniques are needed to support the functioning of Space Station era telerobotic systems. This paper covers three major issues that have impact on the design of data management functions in a telerobot. It also presents a design concept that incorporates an intelligent systems manager (ISM) running on a spaceborne symbolic processor, (SSP), to address these issues.

The first issue is the support of a system-wide control architecture or control philosophy. Salient features of two candidates are presented that impose constraints on data management design. The second issue is the role of data managements in terms of system integration. This refers to providing shared or coordinated data processing and storage resources to a variety of telerobotic components such as vision, mechanical sensing, real-time coordinated multiple limb and end effector control, and planning and reasoning. The third issue is hardware that supports symbolic processing in conjunction with standard data I/O and numeric processing. A spaceborne symbolic processor, (SSP), that currently is seen to be technologically feasible and is being developed is described and used as a baseline in the design concept.

INTRODUCTION

The objective of this paper is to introduce, informally and largely by examples and comparison, an advanced design concept to data management in autonomous telerobots. The motivation for introducing advanced data management techniques in such systems is to address the system-wide complexity problem in general and the system level issues of evolvability and modularity in particular.

Data Management is a broad term. Nonetheless, it is fair to say that it is at the core of most system integration efforts. For NASA, one possible and logical approach to developing telerobotic systems is to view building them as another satellite or space craft. In this scenario, the contractor is largely responsible for system integration. As a consequence, Data Management is tucked away in the last development activity preceding the operations phase. Fortunately, there are several major efforts within NASA to elevate systems architecture and integration to an anticipatory design process[1,2].

---

[1] Albus, James A., McCain, H.G., Lumia, R., NASA/NBS Standard Reference Model For Telerobot Control System Architecture(NASREM), December 4, 1986, National Bureau Of Standards, Robot Systems Division.

CONCEPT DESIGN

In this paper, we are interested in Data Management as an integral collection of support services that satisfy the system-wide information(data) processing needs of a telerobotic system. Therefore, we recommend that these services be viewed as a subsystem that is on an equal footing with all the other subsystems in a telerobotic system.

For example, a popular view of computers is that they "think" or are the "brains" in a feature-laden appliance. In a like manner, data management in telerobots may be viewed as the nervous system as well as the brains. Given this idea, the possibility of incorporating higher levels of intelligence in a primitive information broker such as a Data Management Subsystem allows for an evolutionary approach to increasing autonomy and intelligence of telerobots.

To further this objective, we recommend that data management have its own powerful organization conventions or architectural model. An Intelligent Systems Manager, (ISM), would embody the principles of the architecture in a particular design. Among the many possible goals for an ISM, as an intelligent information (data) broker, two important ones are to reduce the complexity of interaction among multiple intelligent subsystems, and to oversee top level safety of a telerobotic system with respect to these subsystems.

Currently, there are several telerobotic system architectures and models that partition the high level telerobotic functions in different ways. We now discuss how the ISM concept fits in with these and how it aids in inserting advanced distributed processing concepts into a telerobotic control design.

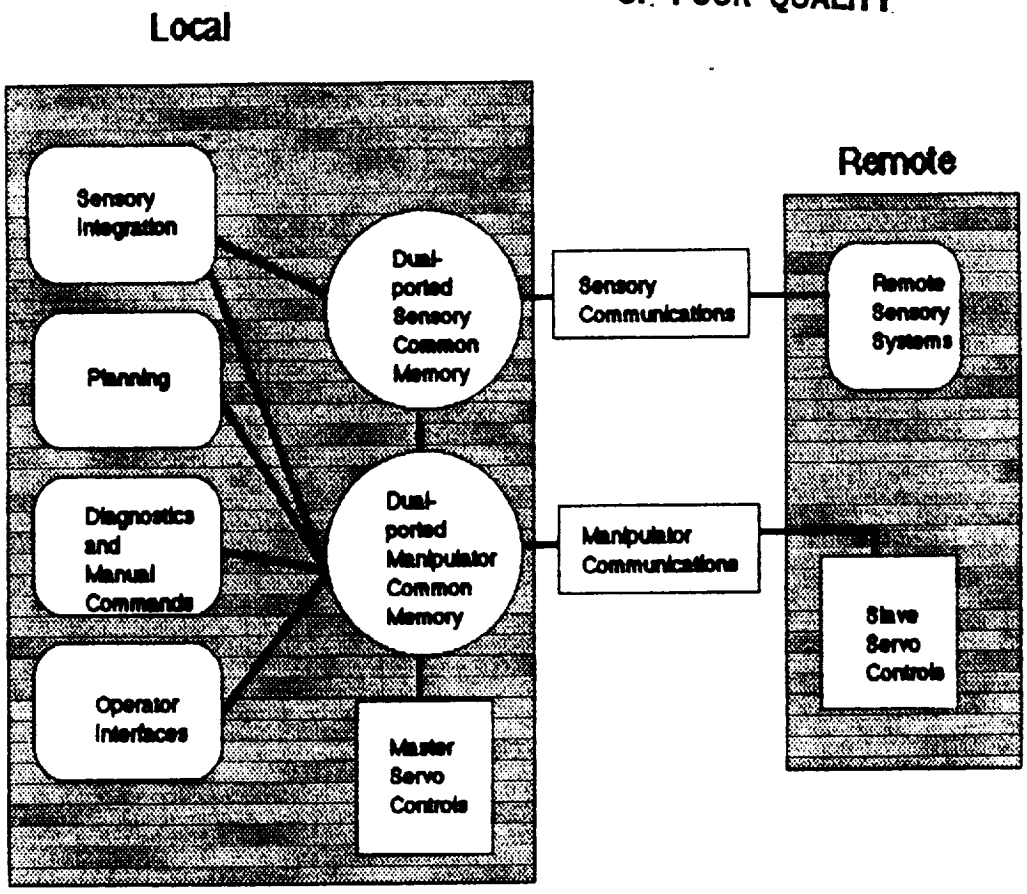
A common starting point for architecture definition of a telerobot is the specification of a controller(s). At one end of the scale we have a 'point design'. For example, requirements are immediately mapped to a specific collection of "off the shelf" components that are hardwired together to function as a controller. Of course, what is a 'point design' is a matter of degree and depends on one's systems engineering criteria.

A more general purpose approach involves defining a set of generic activities that require services of a controller(s). This is particularly difficult because the field of autonomous telerobotics is so new. As a consequence, the subject of design criteria for partitioning a telerobotic system into subsystems is evolving. One partitioning, given by Martin, et al[3], is illustrated in the conceptual layout shown below. Martin, et al, propose the use of powerful microprocessors and outline an architecture to interconnect and interface them to support teleoperator control of mechanical manipulators. In this example and in general, seperating out what is integral to a Data Management Subsystem from data processing elements indigeous to other subsystems in a telerobot is no easy task.. For example, in Martin, et al's, design there are data management functions in all activity areas.

---

[2]Functional Requirements For The 1988 Telerobotic Testbed, JPL D-3693, October 1986.

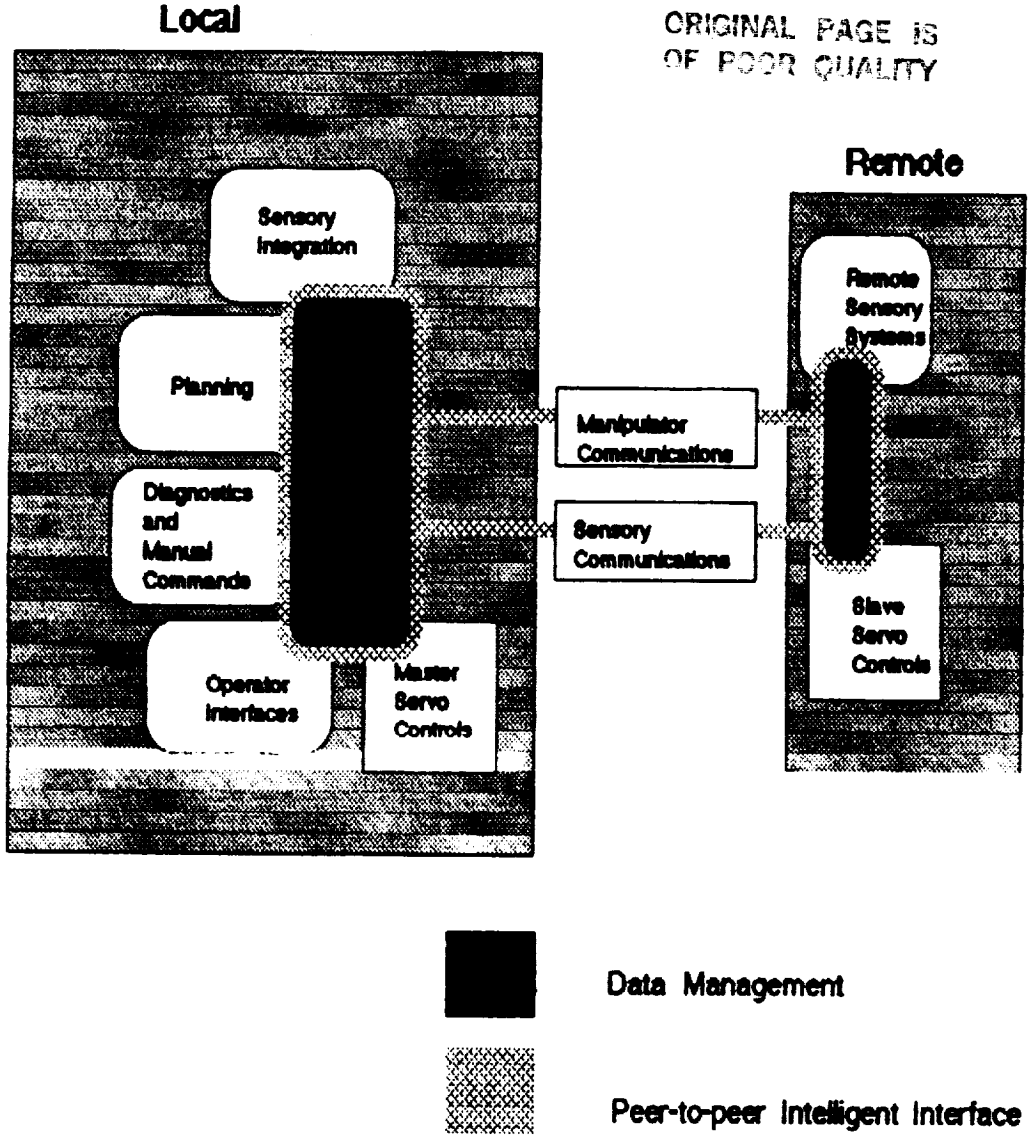
[3]Martin, Lee H., Paul E.S. Salterlee Jr., and Richard F. Spille, "Distributed Control Architecture For Real-Time Telerobotic Operation", JPL Space Telerobotics Workshop, January 20-22, 1987 [in publication].



Conceptual layout of the principal centers for telerobotic control(Martin et al)

Figure 1

What we propose in this example is that another activity area - data management - be added to the top level list of major control activities as shown in the following figure:



Example of Data Management As A Principal Center For Control.

Figure 2

Since, we do not know the implementation cost, the above conceptual layout is not yet recommended as a better way to solve the specific class of telerobotic functions that Martin, et al, are addressing. We use it to illustrate an important shift in design. Each subsystem has its own information processing needs and, therefore, will have internal data management facilities. The crux of the matter is figuring out what the interfaces are between the Data Management subsystem and the other subsystems.

INTELLIGENT INTERFACE

On one hand, the layout proposed above may seem to be equivalent to the systems integration design already derivable from the existing architecture as proposed by Martin, et al. On the other hand, suppose a uniform and common interface is available

that any subsystem must use in order to coordinate end-to-end activities with other subsystems in the telerobot. This interface is between a subsystem and the Data Management subsystem. This is represented in the above figure by the overlap of the Data Management subsystem over the others and the stripped band.

Let's assume such an arrangement for the moment. In order to add another subsystem, this subsystem must be built to work with that interface. In the case that the new subsystem can obtain all nonindigenous resources and end-to-end services only from the data management subsystem, we then say that system evolvability or scalability in linear in the architecture with respect to this interface.

The issue now facing us is whether such an interface is definable in telerobotic systems and, if so, what will it take for it to be effective? Another way to look at the question is to ask what is the best way to organize the Data Management subsystem so that its impact on subsystem dependencies is minimized? The more flexibility the Data Subsystem has the better.

Existing interfaces occurring in different levels of computer technology reflect a form of architectural linearity or scalability for a variety of modules. For example, at the hardware level, standard buses such as the VME and Multibus support a number of controllers up to a standard configuration limit. At the local area network level, Ethernet supports the linear insertion of nodes up to a configuration-dependent maximum.

At the operating systems level, there are distributed operating systems running on multiprocessor configurations that allow, up to a limit, processes or jobs which have no interdependencies to be assigned to available processors at a fixed per process/job overhead. A commercial operating system exhibiting such a capability is Dynix[1]. It is a proprietary design and runs on a multiprocessor system, the Balance 8000, which supports a configuration of up to 24 32-bit microprocessors. In fact, performance measurements show an efficient linear performance curve for jobs with certain types of dependencies.

The point we want to emphasize is that was only a few years ago it seemed impractical to even try to organize[2] large scale systems such as computers to operate in parallel on a collection of jobs and processes that were reasonably independent and maintain a linear performance curve across a useful work range. Today this is becoming routine. Likewise, with something as complex as a telerobot , we can make headway in achieving efficient linearity for a practical set of parallel and independent tasks. Therefore, we wee instances of linearity in performance and scalability in a variety of computer technology levels.

OBJECTS

The success of achieving this sort of linearity and modularity of performance from the Data Management subsystem in a telerobotic system depends initially on our ability to

---

[1]The Balance 8000 Technical Reference Manual, Sequent Computer Systems, Inc., Beaverton Oregon.

[2]Amdahl's Law circa 1970: For the same amount of money one big computer will provide more throughput that a collection of smaller ones.

represent the concept of an intelligent interface in software. The representation approach we recommend is through the use of abstract objects.

The use of abstract objects is one of several key concepts in advanced distributed processing[1]. Many programming languages support objects either directly by syntactic conventions or indirectly by the programmer's use of an object-oriented methodology. For example, work is being done in object-oriented design using Ada[2,3]. Strongly related to the object oriented design approach is the concept of layered design. The layers of an onion are often used as an analogy for the layering of objects in the whole design.

Is is beyond the scope and purpose of this paper to do more than introduce the salient aspects of this subject. We are interested in provoking thought along these lines in development people working in a variety of telerobotic disciplines. In particular, those R & D engineers who must deal with the system engineering problems of data management or are greatly affected by it s presence or absence.

The sort of telerobotic systems we are anticipating will have requirements that are at least as computationally complex and challenging as those envisioned in DARPA's research[4] in autonomous vehicles. NASA's Flight Telerobotic Servicer(FTS) will easily have the same complexity in data management and computational requirements as these systems are envisioned to have if it is to support a major autonomous mode of operation for space stations maintainence.

Object oriented design methodology and programming at the most general or abstract level spans at least two different data/information management disciplines. Whereas, computer science is often associated with procedural languages such as Ada and with numerical algorithms, Artificial Intelligence(AI), is associated with functional languages such as Lisp and with symbolic processing. These differences are mentioned to illustrate that, in fact, this concept and methodology is an integral part of each of these two disciplines(or styles) and is a strong common point between the two. This commonality in using objects is often lost because of the pigeon-holing of professionals as being either in AI or in computer science. Therefore, we prefer to keep the discussion of objects independent of any language (which is a form of implementation of these concepts) or discipline.

---

[1]Lampson, B.W., Paul, M., and Siegert, H.J., Distributed Systems-Architecture and Implementation, An Advanced Course, 1981, Springer-Verlag, 15-16.

[2]Firesmith, Donald G., Object-Oriented Development, Proceeding: First International Conference on Ada Programming Language Application For the NASA Space Station, June 2-5, 1986, High Technologies Laboratories, University of Houston-Clear Lake, Texas, D.4.1.1., D..4.1.11.

[3]Booch, Grady, Software Engineering With Ada., The Benjamin/Cummings Publishing Company, Inc. 1983.

[4]Torrero, Edward A., Military R&D, The DARPA Program('Strategic Computing', DARPA October 1983), Next-Generation Computers, IEEEE Press, 1985, 153-154.

Another reason to keep the discussion at a generic level is that, except for small subsystem controllers in telerobots or scientific payload instrument control applications, current implementation of programming languages supporting these constructs put major constraints on object representation, manipulation, and performance in a distributed environment. However, significant strides are already being made. Martin, et al, note that the performance capability of the Novix[1] microprocessor is largely due to its direct support of the Forth language.. A relatively new language called Neon[2], which borrows heavily features from Forth[3] and Smalltalk[4] supports objects in its syntax. Consequently, it is an example of a product that straddles the fence between AI and computer science. Such innovations are likely to continue and even accelerate in the future. Oak Ridge National Laboratory, in a report on the Man-Equivalent TeleRobot, mentions that its experience in using Forth as a language for real-time control has been positive[5]. Therefore, it is plausible that an object design environment that meets significant telerobotic requirements in data management could be developed in the near future. Whether it is a 'new' language, such as Neon, or a sophisticated development environment built on top of existing software technology, such as Ada or one of the varieties of Lisp, remains an open question. The key issue in either case is the availability of an appropriate computer architecture(s) and technology that support such a design approach and, at the same time, will embed successfully in telerobots.

We are encouraged to pursue this approach to data management given that there are more and more programming implementations of these concepts available on powerful microprocessors.

Abstract objects(or resources), both active and passive, are such things as files, directories, processes, tasks, virtual I/O devices, databases, and any other item that is useful for the designer to identify as part of the system at a certain level. In contrast, real objects are such things as processors, secondary storage, controllers and any physical item that must be taken into account at a certain level (layer) of detail to perform a function. (The irony is that the use of abstract objects in designing data management architecture is making abstract things appear as "hardware" entities to be software/systems engineer and real things such as scientific instruments and sensors appear as "software" entities.) Each object is specified by its representation and a set of operations or functions and associated parameters that can be performed on the representation. The implementation details of an object representation are contained in an object manager. (Note that an object manager may itself be a object in the system.) In a distributed environment, message passing is needed to exchange information

---

[1] See footnote 3.

[2] The Neon Manual, Kriya System, Inc. 505 N. Lakeshore Drive, Suite 5510 Chicago, Ill. 60611. Runs on the Macintosh computer.

[3] Winfield, A. The Complete Forth, A New Way To Program Microcomputers, Wiley and Sons, 1983.

[4] Goldberg, A., Smalltalk-80: The Language and Its Implementation, Addison-Wesley, 1983.

[5] Recommendation For The Next-Generation Space Telerobot System, Oak Ridge National Laboratory, TM-9951, March 1986.

between object managers and to carry out operations on objects. The mechanism of message passing may be implemented using shared memory and procedure calls or may be implemented by send/receive operations that require a protocol to insure reliability and fault tolerance.

We now describe the conceptual elements of what we mean for an interface to be intelligent, and, correspondingly, refer to an intelligent Data Management subsystem as one that uses such an interface. We use the terminology of objects discussed up to this point in order to establish specifically the properties of this interface.

An interface is defined as a set of conventions for the exchange of information between two object managers. The three components of an interface are:

- A set of visible data objects or modules and the allowed operations and parameters associated with each visible data object or module.

- A set of rules governing the logical or legal sequences of these operations.

- The encoding and formatting conventions required for operations and data.

We say a peer-to-peer intelligent interface exists between the Data Management subsystem and the other subsystems in a telerobot when the following properties hold: First, it is possible for the Data Management subsystem to actually pass or export copies of code that meet the specification of the three components of the interface specification to a subsystem. A extreme example of control is that the Data Management subsystem would have to pass or export all of the actual code needed in a subsystem for that subsystem to be able to use Data Management subsystem services. As an example of negotiated control, a subsystem could likewise pass or export select components of the interface back to the Data Management subsystem for purposes of adaptive configuration of services. Second, the interface itself must be symmetric. By this we mean that two peers use the same specification (with differences limited to addresses and local house-keeping functions) in order to interact. In some computation settings, such as data communications, such an interface would be viewed as a protocol. In this sense, what we see being passed two subsystems in an autonomous telerobot is the protocol itself tailored to allow for special "hand-eye" coordination or the control of dynamic chaining of control loops. However, we see even more complex information being passed in such a fashion.

For example, in DARPA's autonomous vehicles program it is envisioned that these vehicles are characterized by their ability to accept high-level task description[10]. In a like manner, an intelligent data management subsystem would have to be able to take a template of information given to it by a higher level task synthesizer.. Consequently, it would pass templates of information, (in AI parlance, knowledge base facts and rules) to the subsystems in order to set up the coordination of data processing functions within the telerobot.

The Data Management subsystem could support an even more adaptive mode of interaction among subsystems if the overall telerobotic control design allowed a top level task synthesizer to pass information templates directly to subsystems, (A low level Data

Management subsystem service of pass information template to X, Y, and Z would be used and available on a reflex basis to a high level task synthesizer.). The subsystems, in turn, would synthesize their information needs based on what was requested of them and would then pass their information templates up to the Data Management subsystem.

The object oriented, peer-to-peer, intelligent interface envisioned as the Data Management subsystem boundary discussed in this paper, has a built-in conceptual adaptability to integrate the following two autonomous telerobotic data processing requirements : to support what appears to be the top down flow of data, i.e., the cognitive and more offline type of activities of planning and reasoning; and, to support the bottom up or reactive and more real-time activities such as run-time control of physical processes and processing of sensory information.

For example, some telerobotic operator controlled operations may require on demand a large portion of the data management subsystem's resources to handle real-time interrupts and to process a large quantity of data, for example, integrating multi-sensor data). For a given cost/performance profile a fixed or nonadaptive data management subsystem design may be easily overloaded by real-time operations. Similarly, a static design may be overwhelmed by large amounts of planning and reasoning due to critical and abrupt changes in task objectives.

INTELLIGENT AGENTS

From the point of view of a subsystem, what are the bare minimum or necessary and sufficient conditions to support the peer-to-peer intelligent interface concept? In order to answer this question, we introduce the concept of the Intelligent Agent that, by our definition, resides in each subsystem of an autonomous telerobot. The Intelligent Agent has potentially several roles within the context of of a subsystem. These roles may be determined by an external knowledge source. In a paper by Sztipanovits, the Multigraph Architecture (MA) is a four layer architecture for intelligent systems that provides "knowledge-level" information for Autonomous Communication Objects (ACO) in its Knowledge Base Layer[1]. For the purposes this paper we focus our concept of an Intelligent Agent as a small compact information broker[2] that is responsible for managing the interface of its host subsystem with respect to the rest of the system. It has to insure the correct use by its host subsystem of the intelligent interface.

All Intelligent Agents in the system adhere to the same intelligent interface in a fair manner. Each IA has the capability to actually pass or export an object from its subsystem to another one and have the Intelligent Agent in the receiving subsystem accept it upon demand, i.e. within a 'reasonable' time frame. The capability for an IA to accept any object upon demand may be impractical. What is more practical and

---

[1] Sztipanovits, J., Execution Environment For Intelligent Real-Time Control Systems, JPL Space Telerobotics Workshop, January 20-22, 1987 (In publication).

[2] An anonymously authored NASA SAIS document uses the term Intelligent Agent as the "eyes and ears" in remote space platforms that take commands and sends information to a master controller located in the Space Station or in a larger system. All instruments and platforms are designed in such a way as to be able to host an Intelligent Agent.

addresses the intent of the conceptual design is that each IA can be unilaterally signaled and required to, at the very minimum, take a "command " object.

The last capability that an Intelligent Agent must have in its role as interface manager is the ability to reset(replace) another Intelligent Agent. Equivalently, any Intelligent Agent automatically accepts any "command" object, and that "command" object may be "replace yourself with me".

INTELLIGENT SYSTEMS MANAGER

Obviously some higher level management functions is required in order not to have Intelligent Agents resetting one another in a hazardous manner. The Intelligent Systems Manager (ISM) is , by our definition, the designated Intelligent Agent that has the authority to give and revoke all other IA's capability to reset peers. Furthermore, it has authority to give and take other resource privileges of IAs. It is now readily possible to design such an organizational scheme to be logical secure through the use of object capabilities [1]. The actual design of incorporating these into a Data Management subsystem still has to be done. What makes the designer's job much easier is that, if properly used, capabilities can insure the logical soundness of an executive resource controller such as an IA/ISM in real time.

The proper use of object capabilities assumes that a logically sound theory and specification of access and control between the ISM and IAs has been developed. As a simple example, only legal sequences of reset capability are ever granted.. What are the rules used by the ISM and conditions maintained by it among all the IAs so as to enforce correctness? The development of a logically sound cooperation mechanism in which only legal sequences are possible and illegal ones created by external corruption of data are contained, is a major area of research in advanced distributed processing. In elementary and not so elementary cases mature theoretical results are available. What yet needs to be done is to investigate the technology and implementation aspects. An autonomous telerobotic system hosting Intelligent Agents is an ideal testbed for capability-based architecture design.

The interface manager function is only one of several for an IA/ISM in a telerobotic system. It happens to be a minimum and the cornerstone of the design concept. The generic role for IA/ISMs is to serve as accretion points. These points are viewed being within the Data Management subsystem and allow for the insertion of more and more "smarts" or intelligence in the whole design of an autonomous telerobot. Note, this is specifically directed to information flow between subsystems. A telerobotic system will gain 'smarts' from advances in sensor and reasoning technologies. In addition, the IA/ISM will allow a telerobotics system to get "smart" from integration of subsystems. The IA is, itself, a place to insert improved reasoning and learning technology. However, for an initial implementation, only the interface manager portion may be done.

Later on, as experience is gained with this approach, more functionality and robustness can be added. The result is that a sequence of IAs and ISMs may be built, each one more advanced that its predecessor and serving more and more autonomous

---

[1] Lampson, B.W., Paul, M., and Siegert, H.J., Distributed Systems-Architecture and Implementation, An Advanced Course, 1981, Springer-Verlag, 202, 235-245.

telerobots.. A demonstration goal of the intelligent interface could be that different generations of IAs may coexist in real-time in a system. Furthermore, insertion of a next generation IA could be made in real-time without having to disable the telerobotic system for any major length of time.

The Intelligent Systems Manager design concept is not meant to be an alternative to a system-wide control architecture or control philosophy. Specifically, we see it providing a conceptual bridge for mapping between the data processing resource space and the overall functional space as described in a model such as the NASA/NBS NASREM model[1] and the target subsystems in a telerobotic system. This approach of introducing another concept design model such as the IA/ISM is an attempt to bridge the concept hierarchy problem discussed by Wolf, et al[2]. The problem is what are the appropriate levels in which to decompose a problem such as building an intelligent supervisory control system. The four given are the functional, resource, knowledge and computer architecture. Each of these may have their own model.

The NASREM model gives an all inclusive functional system model for a telerobot. It is a six layer hierarchical model from top to bottom and has three horizontal partitions for sensory processing, world modeling and task decomposition. In our view, one of the purposes of this model is to be the framework for developing a system effectiveness criteria to be used to evaluate proposed designs. For a specific telerobotic system, this is accomplished through iterations of tradeoff analysis of mission objectives (requirements) and constraints. The logical distribution of functions in the NASREM model is to provide a gauge for a particular design's effectiveness. On the other hand, the distribution of a logical function in an implementation is subject to another effectiveness model that incorporates constraints of the resource function.

For example, Hawker, et al, of Lehigh University, in a paper[3] on multiple robotic manipulators, limited their interpretation of the NBS approach to dual arm control as requiring a triad of controllers. One controller for each arm and a third to control these two. With the ISM design approach to data management, a design goal would be to have the ISM dynamically hand off to two Intelligent Agents(assuming each arm is in a different subsystem and hosts an IA) so that each one could directly communicate in the cooperation of the two arms without the ISM in the loop. For example, ISM send to IA right arm a reset object Q and then tells IA left arm to accept from IA right arm an object that will cause reset of itself.

The reason for all of this is that the dynamic control algorithm is likely to be considerably different from single arm control. Therefore, we have to replace both single arm IAs (or those portions critically related to run-time control) with a version of an IA that effectively does the dual arm operations. The ISM has to fashion out of a higher order information template (calling for dual arm control), all the contextual information about the task.

---

[1]See footnote 1

[2]Wolfe, William J., Raney, Steven D., Distributed Intelligence For Supervisory Control, JPL Space Telerobotics Workshop, January 20-22, 1987,(in publication).

[3]Hawker, Scott J., Nagel, R.N., Robers, Richard, and Odrey, Nicholas G, Multiple Robotic Manipulators, Byte Magazine, January 1986, 203-219.

Note, this most likely will not be doable with only dynamic swapping of to an alternate program due to configuration and linkages overhead. The object Q may contain several other objects in it that pertain to, for example, real-time collision avoidance and other high level world model information condensed down to be appropriate at this level. Consequently, the logical hierarchy of function of the NASREM model is preserved but the real-time flow of information would go according to the criteria of a different model, a system effectiveness data resource model for autonomous telerobots.

An even more interesting problem than that of dual arm control is the change out of an end effector by a robot. It is highly likely that the data management in the telerobot would have to dynamically reconfigure itself to accommodate such changes. Consider, the more extreme case, where a robot has to take itself apart to fit through an aperture or repair itself by swap out.. The run-time requirements on Data Management adaptability will indeed be challenging.

Realizations of the flexible data processing example have not been tried for robotics yet because of the lack of a suitable computer architecture and technology that support the rea-time object-oriented processing described here for telerobots.. However, this is rapidly changing and may already be attainable in some ways. Therefore, with respect to Hawker's conclusion, we feel that the NASREM model is indeed relevant to dual arm control, but that it should not be used as the final system effectiveness model for data flow and processing in a telerobot.

Anyone developing a large scale telerobotic system may wish to partition the data management subsystem in a one-to-one fashion according to the logical hierarchy in a model such as the NASREM as a first cut to understanding functional and logical relationships. But, as these are understood and clearly identified, another model that is responsible for data management resources should be used to refine the design and the ultimate realization of the Data Management subsystem.

For the reasons present above, we recommend that system engineers working in telerobotics carefully look at how the models are used. Some of the debate of the applicability of using a global framework such as the NASREM is due, in our opinion, to trying to use one model to solve a problem that actually needs four separate ones.

Currently, there is a need to develop a resource model for data management in the context of autonomous telerobots. This model could then be used to gauge the effectiveness of proposed designs for data management. The IA/ISM would be one of them when it is sufficiently developed. For now, our criteria for effectiveness is limited to nominal data management functionality, adaptability, and dependability.

SPACE BORNE SYMBOLIC PROCESSOR

At NASA Ames Research Center, the purpose of the Space Borne Symbolic Processor project is to advance the application of revolutionary computer architectures that combine both numeric and symbolic processing for space and aeronautical flight.

Currently in the AI research community, a great deal of experimentation and prototyping of architectures and technology is underway which is specifically aimed at improving the performance of AI-based systems. For a recent survey, see the January 1987 issue of

Computer. The goal of all this research within roughly the next five years is to improve the performance of symbolic processing applications by at least two to three orders of magnitude.over what can be done today. One of the application areas to benefit from this performance improvement is expert systems and expert system building tools.

Significant advances in computer technology and architecture are needed to support the IA/ISM design concept. in space borne autonomous telerobots. This is true particularly in large scale distributed environments where objects must persist for long periods of time, span physically over disjoint memories, and have real-time interaction with other subsystems composed of sensors, dynamic control loops, human operators, and planning databases.

Two critical requirements for supporting an Intelligent Agent are the efficient representation of objects and the high performance run-time support of object handling. We see the dynamic research activity in the AI discipline as being the most promising long term source of technology to support Intelligent Agents. Our two critical requirements may be stated in the following manner: What software and hardware elements must exist in a subsystem in order to host an Intelligent Agent?

In software, the ability to represent object abstraction is required. This is greatly

In hardware, there must be an efficient architecture to support the movement and run-time behavior of active objects. Note, the implicit condition that this be supported transparently in either a loosely or tightly coupled environment of multiple processors This is a vast subject area and, since the sixties, has often been referred to as the semantic gap between the software and hardware. Research is underway wherein new hardware units are being proposed and tried out to support more directly the movement of variably structured objects in a distributed environment.

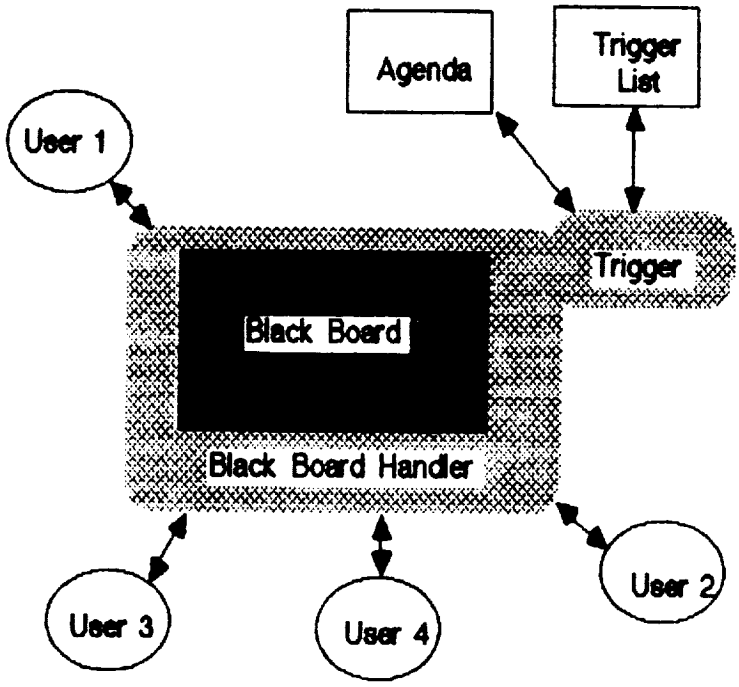
Another crucial hardware element is the specific ability of one subsystem to fork an object into another subsystem. We see rudimentary parallels to this in some programmable interfaces in today's microprocessor-based controllers and in large mainframe computer systems. The IBM 370 series mainframe software would assemble a channel program, send it down the I/O channel to the channel controller, and then hand over control to the controller by a command sequence that said: "execute this program".

The details of realizing this crucial capability of object forking may be implemented in a variety of ways. In terms of a telerobotic system, for example, do we design and build a special hardware backplane that runs through all the subsystems, or glue together existing hardware components, or use a local area network? The most relevant approach to the specifics of doing this is to include this as a systems engineering requirement for subsystems in an autonomous telerobot. Only in the context of a specific set of telerobotic missions requirements can such trade-offs be usefully done. One candidate is the technology being proposed for a Space Borne Symbolic Processor, (SSP).

Integral to the ability to fork an object onto another subsystem is the hardware that allows the resetting of that part of a subsystem.(internal to the Intelligent Agent) that is running a

forked object. As an example, a mainframe would be able to unilaterally reset a controller regardless of what it is doing.

The induced run-time design requirement to support IAs is nontrivial. A sophisticated trigger mechanism that uses a dynamically prioritizable vectored interrupt scheme is one possibility. Work is being proposed for designing and building off-the-shelf hardware components to support the triggering mechanism needed at the hardware level to support the Blackboard model. Simply put, the model is an AI paradigm of individuals communicating by free association by writing on a blackboard their knowledge of a problem(or task) and reading from it as they please. At the nitty gritty hardware level, requirements are much more constrained and strict if serious real-time applications are to be supported. The following figure is one illustration of an architecture of a blackboard at the subsystem/component level.



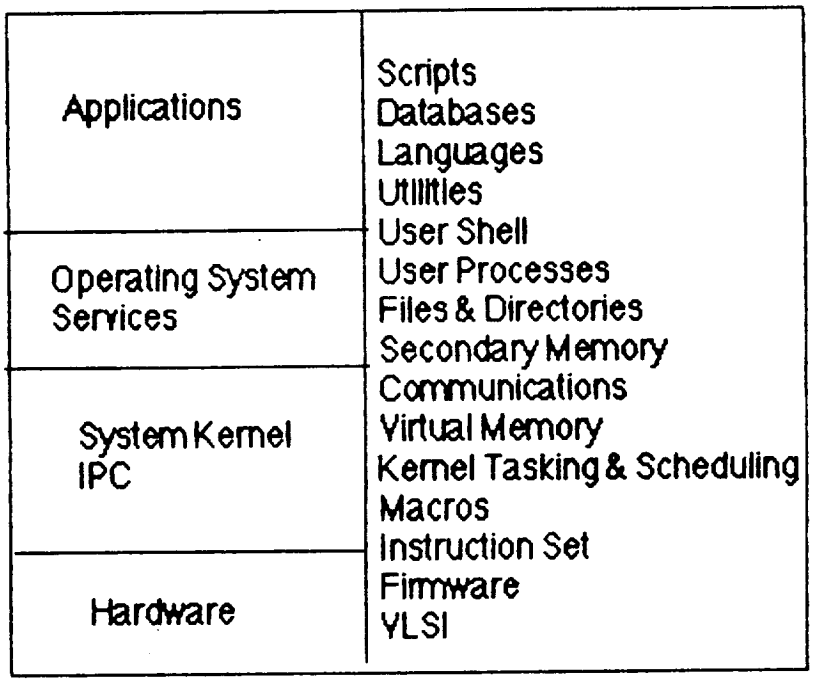
Block Diagram of Black Board Model

Figure 3

The hardware trigger mechanism for real-time black boards may be used in a design to support the Intelligent Agents embedded in a telerobot within the next five years. However, without such devices available it is a crucial tradeoff, depending on certain

telerobotic constraints, whether to even design the Data Management subsystem as hosting an Intelligent Agent. There is a compromise case that the Data Management can afford one and another subsystem cannot. In this instance, this subsystem could multiplex its access to the Data Management subsystem through a shared IA hosted on another subsystem to remain true to the design concept and meet cost constraints.

The current research thrust in architectures and technology for AI applications questions the very basic tenets of computer system design. For example, the boundary between what is software and hardware and the usual conventions of layering an architecture are being re-explored. The following figure is an example of logical layers (or levels) that are used to organize and understand various functions in an computer design.



Computer Architecture Hierarchy Model
Figure 4

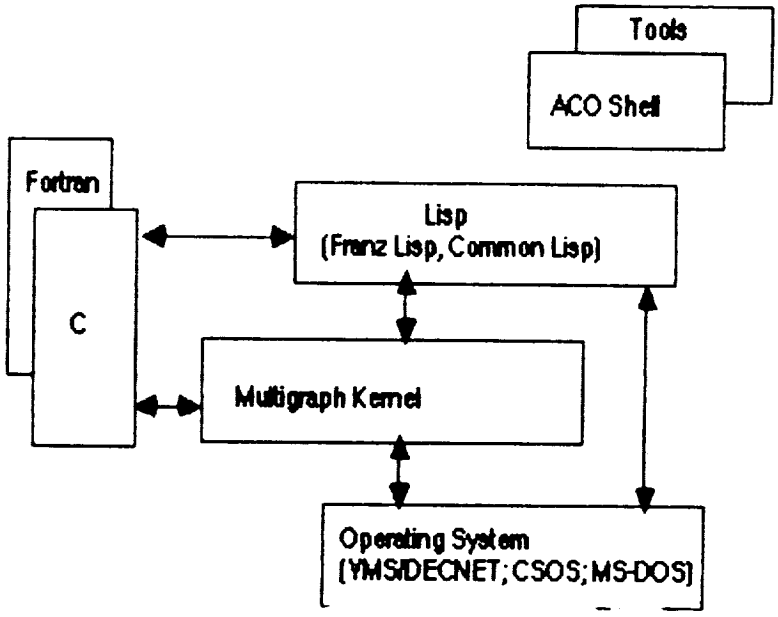
Note that we use this hierarchy only as a model and not as a representation of the design of an architecture . The use of a hierarchy is a powerful and commonly used tool to aid in the understanding of a design. For a particular system design, the layering will be unique to that design.

It is this variability in layer definition that makes evaluating and understanding the impact of innovations in AI architecture and technology difficult to gauge. The only recourse,in many cases, is to actually build prototypes and run experiments. In contrast, the hierarchical layering in numerically oriented architectures is relatively more stable. Increments in performance and functionality are easier to gauge when adding a function in a particular layer or by speeding an existing one up.

A major shortcoming of this model is that the structure or microarchitecture of the execution environment is not obvious. The reason for this is that the execution environment in conventional designs depends on functions at several different layers. Consequently, the structure or microarchitecture of the execution environment is not optimal either in performance or in representation (programming) of real-time object handling.

A major reason for the relatively low performance levels of today's symbolic processors is that they are based on an incremental design approach of hosting AI software on conventional, numerical processing oriented architectures which in turn often suffer from a weak run-time microarchitecture.

An example of what we call a microarchitecture model of the run-time environment is given by Sztipanovits of Vanderbilt University in the following figure. He calls it the structure of the execution environment.



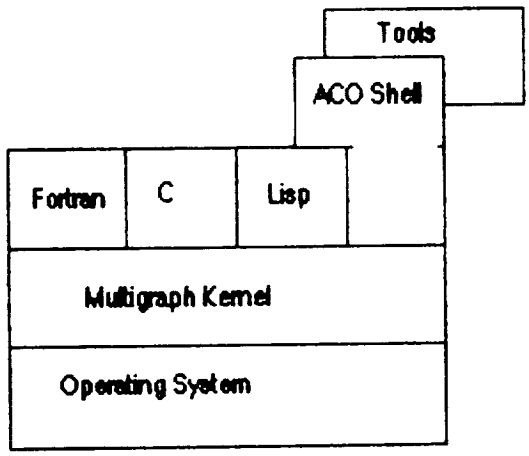
Structure of the Execution Environment(Sztipanovits)

Figure 5

The focus of his research is to use the Multigraph Architecture(MA) to study intelligent systems operating in a real-time, parallel computing environment.. Since, it is a requirement to host this work on a variety of computers he has to include a number of interfaces for facilitating portability and flexibility. We view the following as primarily interfaces of the Multigraph Kernel(MK) being there for these roles: Fortran, C and the Operating System interface.

Let us consider what would happen to the components of this model if we were trying to design a "lean and mean" run-time environment . One step toward this goal would be to minimize the interfaces that the (MK) has. It is interesting to note that many applications using objects and symbolic processing are in runtime environments that are as complicated as this one. This is one example of why existing object based applications do not perform favorably with their numerical counterparts.

Our preliminary design of a very compact microarchitecture for the (MA) is given below.



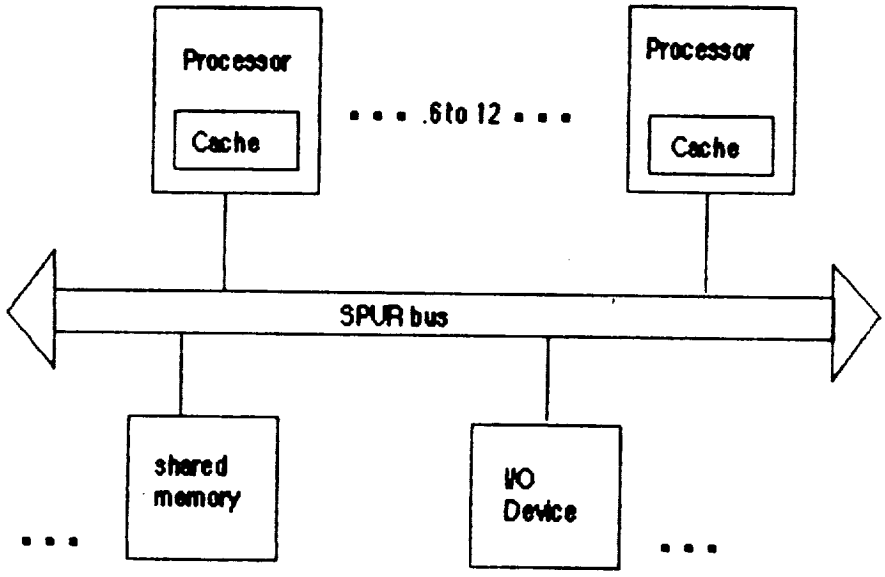
Example of Compact Execution Environment

Figure 6

The interfaces are fewer and simplified. The upper interface of the kernel supports all of the languages equally. The operating system(run-time aspects) is pushed up to where the kernel is. A single object oriented run-time resource manager is part of the kernel. By design, the kernel and hardware interface may be greatly simplified. The (ACO) shell may have direct support in the kernel. This is an optional design tradeoff and depends on how important it is to "hardwire" knowledge base facts directly into the kernel.

The last major architectural issue that remains is how to efficiently mix and match the parallel execution of programs using numeric and symbolic data. The problem is that procedural languages such as C and Fortran and functional languages such as Lisp have some major differences in terms of efficient, high performance data representation and processing.

The SPUR(Symbolic Processing Using RISCs) is a multiprocessor design that does addresses this issue.



Block Diagram of Berkeley SPUR

Figure 7

The design consists of identical processor modules/boards each of which support both symbolic and numeric operations. This commonality is achieved on the board by having special purpose processors that process floating point, and list(symbolic) data separately. An elaborate onboard caching scheme is used to move data to and from the coprocessors and the global memory and to identify the data as to whether what type it

is. Up to 12 processors, a global virtual memory of 256G-bytes , and I/O devices are all connected together by a common bus. The SPURbus is 64 bits wide is based on a modified Texas Instruments NuBus.

Research for the SSP will undoubtably look hard at the features of this design. The tradeoffs of whether to use a bus or network and whether to hid special purpose processors in a module that connects to the bus or network or to hang them directly off the connection media will be a interesting computer engineering trade. One design tradeoff used in the SPUR that is relevant to the flight environment is the size of the onboard caches. In a flight environment, the requirement to power a wide backplane is up against a major power constraint. Since the SPUR is designed to used as a low cost workstation the SPURbus is slow when compared to similar designs using multiprocessors that are aimed at replacing large uniprocessors. The solution was to put in relatively large caches for both instruction and data.

There is one interesting requirement that the designers of the SSP should consider that does not seem to be possible with the SPUR. The development of computer chips and modules that support higher level abstraction above just tagging data is important. Hardware support for Black board functions may soon become a reality. In the future, special purpose hardware for such things as Intelligent Agents and cooperating expert systems may be desirable to support at the computer component and architecture level.

As a consequence, the scope of the SSP in terms of supporting AI technology in flight has the important role of being a pathfinder in how such developments could be configured into a flight system. At one end, we have a stand alone Lisp processor in space running an embedded expert system, and at the other, we have the possibility of multiple blackboards whose knowledge sources are able to share information across disjoint domains.

ACKNOWLEDGEMENTS

I would like to thank the participants in GSFC's Flight Telerobotic Servicer Skunk Works of 1986 for a most simulating and challenging time where I got inspiration for these concepts.

LW 08649

ROBOT HANDS AND EXTRAVEHICULAR ACTIVITY

BY

BETH MARCUS, PH.D.

ARTHUR D. LITTLE,. INC.

MAY 14, 1987