N94-35069

A Distributed Computing Model for Telemetry Data Processing

Matthew R. Barry,*
Kevin L. Scott and Steven P. Weismuller†
Systems Division DF6
NASA/Johnson Space Center
Houston, TX 77058

Abstract

We present a new approach to distributing processed telemetry data among spacecraft flight controllers within the Control Centers at NASA's Johnson Space Center. This approach facilitates the development of application programs which integrate spacecraft-telemetered data and ground-based synthesized data, then distribute this information to flight controllers for analysis and decision-making.

The new approach combines various distributed computing models into one hybrid distributed computing model. This model employs both client-server and peer-to-peer distributed computing models cooperating to provide users with information throughout a diverse operations environment. Specifically, it provides an attractive foundation upon which we are building critical real-time monitoring and control applications, while simultaneously lending itself to peripheral applications in playback operations, mission preparations, flight controller training, and program development and verification.

We have realized the hybrid distributed computing model through an *information sharing protocol*. We shall describe the motivations that inspired us to create this protocol, along with a brief conceptual description of the distributed computing models it employs. We describe the protocol design in more detail, discussing many of the program design considerations and techniques we've adopted. Finally, we describe how this model is especially suitable for supporting the implementation of distributed expert system applications.

1 Introduction

The past approach to spacecraft data distribution in the Mission Control Center (MCC) used a centralized computing model. Mainframe computers processed the incoming telemetry and trajectory data, applied various computations to this data, then distributed this information to hundreds of flight controller displays. This approach worked well for many years, but its cost, inflexibility and resistance to change have become unappealing. Meanwhile, advancing workstation and networked, distributed computing technologies which overcome these limi-

*Rockwell Space Operations Company, 600 Gemini Ave., R20A-4, Houston, TX, 77058.

†Unisys Space Systems, 600 Gemini Ave., U04D, Houston, TX, 77058.

tations grew more compelling. Accompanying these advancements were new software development packages, man-machine interfaces, and artificial intelligence technologies. The application potential of these exciting technologies inspired flight controllers to assume more interactive roles in the development of computational tools which enhance their spacecraft operations capabilities. The desire to implement these enhancements encouraged a revolution in the MCC computing architecture.

The MCC began the transition from centralized to distributed computing several years ago. The current MCC employs *both* configurations simultaneously, running workstation and mainframe applications side-by-side. As the MCC evolves into the Control Center Complex (CCC) for dual Space Shuttle and Space Station operations, the workstations and network will become the principal computing platform.

The current dual-configuration mode of operations in the MCC has afforded flight controllers an opportunity to prototype new operations applications. Until recently these efforts have focused on the development of real-time monitoring and fault detection and diagnosis applications for individual flight control disciplines. These efforts have been successful in demonstrating the enhanced capabilities provided by the distributed computing platform. Moreover, they have encouraged the pursuit of distributed applications which span several flight control disciplines. These distributed applications exploit networking capabilities to provide inter-user information sharing.

The new distributed computing model we describe herein was created to support and encourage this inter-user information sharing. The *information sharing protocol* (ISP) was designed to provide elegant computing techniques which enable workstation applications to communicate with each other. This protocol also supports inter-host communications on a heterogeneous platform complying with industry-standard operating systems and networking protocols. Consequently, the ISP provides users with a consistent communications interface for all of the various operations, training, and development platforms they use in everyday business; lack of such an interface has caused many problems in the past.

There were several motivations that led to the development of the ISP. First, there has been a strong desire to reduce the manpower required for real-time Space Shuttle operations. This reduction in manpower necessitates the development of enhanced computer programs which can perform many of the real-time data monitoring, fault detection and diagnosis, and planning tasks previously performed by flight controllers. These developments have also spawned many projects pursuing "intelligent systems" which meet or exceed the flight controller's reasoning capabilities. Many such systems have already been deployed in the MCC, but there are many more of these intelligent systems appearing on the horizon. The potential for these systems to capture precious spacecraft operations expertise is a clear motivating factor in our work. Second, the increasing number of workstation programs being developed to enhance operations has been accompanied by a significant growth in software development and maintenance costs. The development of the ISP was meant to restrain this growth by providing a package of distributed computing tools which support the basic needs of all flight control disciplines. These tools reduce both development and testing costs. Third, the use of workstations outside of the MCC has increased tremendously over the last few years. Flight controllers now find themselves performing many of their mission preparation, training, and software development tasks in facilities lo-

cated in their offices or laboratories around the center. Since many of the computer programs they use must work in all of these facilities, there is a strong movement toward standardization. This movement is encouraging the use of industry-standard hardware and commercially-developed software to reduce development and maintenance costs. The ISP promotes these standards while providing a layer of customizable telemetry data distribution tools for program development in a heterogeneous facilities environment. Finally, drawing from each of these individual motivations, we envision a tremendous utility in the deployment of distributed expert systems. The ISP was designed to provide a mechanism for these expert systems to communicate with each other.

2 Distributed Computing

A *distributed computing* architecture is a programming model in which computing resources can be allocated in various ways. Typically this involves a network of high-performance workstations providing information to a user. There are two predominant models of how to distribute the available computing resources. A *client-server* computing model refers to a special case of distributed computing in which an application is divided into separate client and server processes. The client process makes requests of the sever process. The server process receives requests from one or more clients and performs some action on their behalf. Servers can provide resource sharing by servicing many clients. Frequently, the client and server pieces are running on different machines; however, this need not be the case. In a *peer-to-peer* computing model, the distributed components of the application act as independent equals, communicating with one another to accomplish a common goal.

The distributed computing model realized in the ISP reaps the benefits of both of these models. At a global level, the goal of the application is to distribute telemetry data to flight controllers. In the client-server role, the primary purpose of the server part of the application is to process telemetry data from a certain data source. The clients request this processed data from the servers. In the peer-to-peer role, the servers allocate computing resources and communicate with one another to provide their clients with data collected from several data sources. The combination of these models provides a system of cooperating agents which supports a wide variety of potential applications.

The user assigns each server to process a portion of data available from the data source. The server provides this data to its clients on a change-only basis; *i.e.* a client will receive data only when it changes from its previous value. Since there are a variety of data sources available, we have written a collection of server programs each of which has been uniquely outfitted to process a particular data source. The data sources include two real-time data streams and various sorts of recorded telemetry files. The server preprocessing includes reading data from the data source, performing noise filtering and other preprocessing on the data, determining what data has changed since the last cycle, and then distributing the changes to the clients. Each server is responsible for any time synchronization and polling tasks.

The user may run clients that receive data from the server and generate new results to supplement the telemetry data. Therefore, the servers also perform the task of reading and distributing this client-synthesized data. Clients that synthesize data for other clients are said to be *publisher* clients, while clients that request any sort of data are said to be *subscriber* clients. Since each server "owns" a

portion of the data source, the servers can interact in the peer-to-peer role to acquire data for clients which subscribe to data owned by other servers. In this situation, one server becomes a subscriber of another server. Along with resource allocation and data distribution, this technique provides a way to enforce data security.

Although there are unique servers for each data source, the data source is transparent to a client. Using the ISP, a client receives data the same way regardless of where the data originated. This feature provides a high degree of portability between computing platforms, and insulates the clients from the nuances of the data source. Essentially, this means that a client program can be deployed against a variety of data sources *without change*. This data-source independence simplifies application program development efforts and reduces the costs associated with software testing and certification.

3 Design Considerations

This section describes a few of the considerations we have made in the design of the ISP. Most of these considerations reflect the motivations established above, while others reflect the specific nature of the telemetry data processing problem. We discuss here only the considerations which we believe to be Space Shuttle-independent. Primarily, we discuss the construction of our hybrid distributed computing model by presenting the attractive features of the two underlying models.

3.1 Client-Server Model

The client-server model is the most influential part of the ISP model. Using this model, we separate the telemetry data source from the client programs. We create server programs to manage the data source and distribute preprocessed data, and we create client programs to do something useful with this information. Usually the client programs perform analysis of the information and display the results to the flight controllers. This separation of tasks is crucial to the hybrid model: it provides the cability to support a variety of data sources and various computing platforms with the minimum number of programs; and it allows user to run various combinations of clients in servers in different contexts, such as playbacks or offline simulations and training cases.

The ISP servers provide information to their clients on a change-only basis. This means that the client programs may operate in an event-driven fashion, instead of having to poll the data stream continuously.[1] When new data is made available to the client, that data is considered an interrupt "event" that the program must handle. The client handles this event by reading the data, processing and perhaps displaying it, then returning to its wait state to receive subsequent events. This provides an efficient allocation of computing resources and reduces the effort necessary to build real-time computer programs. Furthermore, to make better use of the distributed computing platform, we can divide this effort among several functionally-equivalent servers processing a different subset of the full telemetry stream. By not extraneously reprocessing the unchanged values every second, each client realizes a significant computational resource savings and, perhaps more importantly, appears more responsive to user interaction.

We assume that there will normally be many servers running. Each server will process a portion of the full data stream. At initialization time, the server assumes an ownership which identifies those *symbols* from the

[1] Historical Space Shuttle mission data shows that during on-orbit operations only 10-15% of the 25,000 telemetry values change during any second.

symbol dictionary that it becomes responsible for. The server will acquire and process all of the symbols from the symbol dictionary which have the same ownership as the server. For example, the Propulsion (PROP) console server started with ownership `prop` will acquire from the data source all of the symbols in its dictionary with ownership `prop`. All PROP client programs need connect only to a `prop` server; similarly all Guidance, Navigation and Control (GNC) console clients only need connect to a `gnc` server, and so on.

Although the server's primary purpose is to process telemetry data, the server is also capable of processing and redistributing data generated by publisher clients. This feature is vital to distributed client applications, particularly fault detection and diagnosis applications, which intend to share their results with other clients. Under the ISP, this distribution occurs when the publisher client writes its results back to the server, which enqueues the information for any subscriber clients. Since the distribution mechanism is the same for any kind of data, the ultimate subscriber client will not be able to discern where the data originated, *i.e.* from telemetry or from a publisher client.[2]

Finally, since the servers are meant to be background tasks, they do not have user interfaces. To provide server status information and control features, we simply employ a collection of client programs which act only as user interfaces to their server. These user interface clients don't subscribe for any data from the server; instead, they read the server log files for status information, and they send the server control commands through packet messages. Furthermore, this design embodies an additional feature that the separated user interface can converse with *any* of the servers without modification.

3.2 Peer-to-Peer Model

Commonly, clients from one flight control discipline may need to process symbols "owned" by another discipline. We perform this exchange with a server-to-server, or *peer-to-peer*, interconnection. Fundamentally, this means that one server defers requests for symbols that it doesn't own to another server. For example, if a PROP client program needs access to some GNC telemetry symbols, the `prop` server will defer requests for these symbols to the `gnc` server. In this situation, the `prop` server effectively becomes a client of the `gnc` server. The `gnc` server will send the requested data to the `prop` server, which in turn will forward the data to the `prop` client program.[3] This situation is depicted in figure 1. The ownership field in the symbol dictionary provides some of the information needed to establish this connection. Basically, it provides the name of the server which publishes that symbol. Therefore, when a client subscribes with its server for a symbol published by a different server, that client's server looks for the publishing server running somewhere on the network. The ISP network registration service (NRS) is employed to perform this search. Once connected, the deferring server forwards the symbol subscription to the peer server, while the peer server registers the deferring server as its client. Symbol values will flow from the peer server to the deferring server then on to the client. We use this process to perform data security: by requiring configuration management of the symbol dictionaries, we can prevent clients from subscribing to symbols that they aren't authorized to receive.

[2]In fact, we have built a server which does not process a data source at all: it simply redistributes information generated by its publisher clients. We refer to this as a *null data source* server.

[3]Note that although they are independently processing different information, these two server processes are instances of the same program that were given different ownerships at run time.

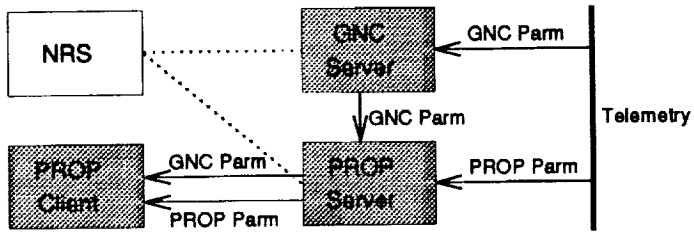

Figure 1: Inter-operator data distribution model. The PROP client can receive both PROP and GNC telemetry data, but it must acquire the GNC-owned data indirectly from the GNC server.

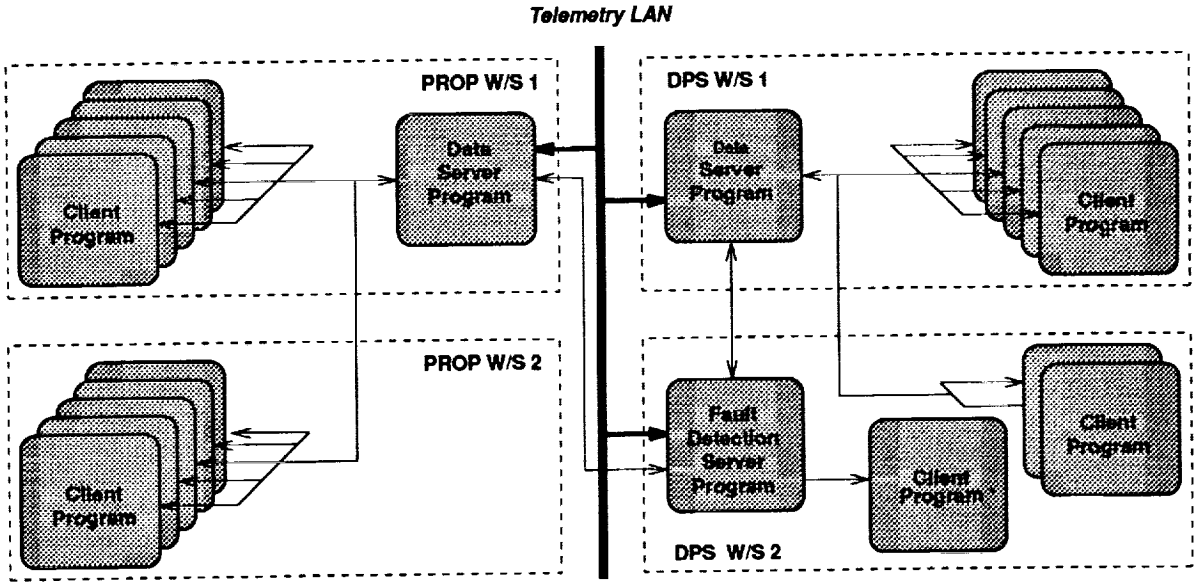

Figure 2: A distributed-resource situation. The DPS flight control discipline allocates a special server to the task of processing fault detection messages generated by the spacecraft software. This server distributes these messages to DPS clients directly, and to the PROP clients through a peer-to-peer connection.

These peer-to-peer connections also are important because they manifest how the model achieves global data processing. All of the processing necessary to make a particular data value "useful" can be performed by the server or client that "owns" that symbol. All other clients which may need the data, but don't own it, do not need to duplicate the processing; they simply request it from the owner. This feature essentially replaces the centralized computing feature which performs all of the data preprocessing before passing the data on to user applications.[4] This enables a wide variety of distributed resource allocation schemes in which each server provides a different data processing function. Figure 2 suggests one such distributed-resource situation.

3.3 Implementation

In realizing this protocol we have employed many industry standards. Primarily, we allow Unix do the things it was meant to do, such as managing resources. We use the TCP/IP protocols for network services, and we use point-to-point connection-oriented stream sockets for interprocess communications. Since the majority of our facilities employ the X-Windows system, we mimic certain facets of the X toolkit. Each ISP *event* has an associated enumerated type definition uniquely identifying that event. Many events also have associated data structures containing event-relevant information, such as a telemetered sensor value and time stamp. All of the events are distributed via message packets. Each message includes a header and a body, and the body can be of arbitrary length. Like the X toolkit, the ISP toolkit provides a *callback* mechanism whereby the programmer can register a series of functions to be called automatically upon arrival of a given event packet. Each registered instance of a callback function can have unique *callback data* to be passed to that function through the argument list. Essentially, this implementation allows programs built around the X protocol to use the same processing logic for "data" events that they use for "display" events.

When a client wishes to establish a session with a server, a variety of information must first be exchanged (figure 3 depicts the event exchanges). The client opens a socket connection with a server, then issues a connection-request event to it. If the server receives and accepts the connection request, it will reply with a connection-accept event. Upon receiving the connection-accept event, the client submits a series of subscription requests for all of the symbols it wants to process. The server validates all of these subscription requests and updates its internal symbol tables. When the client is done submitting subscription requests, it sends an event to enable the data stream. The server will then begin issuing data-change events to the client as its symbol values change. The client will receive these events and process them, possibly sending some synthesized data back to the server for publication. When the session has completed, the two programs exchange disconnection events to gracefully terminate communications and release resources.

4 Distributed Expert Systems

Some of the most exciting applications of the ISP technology lie in the support of distributed expert systems applications. Traditionally, these distributed expert systems have required that each component employ the same inference engine or expert system

[4]This is particularly important for data that is limit-sensed in some manner. The owner of the data applies the proper limits before forwarding the results to subscriber clients.

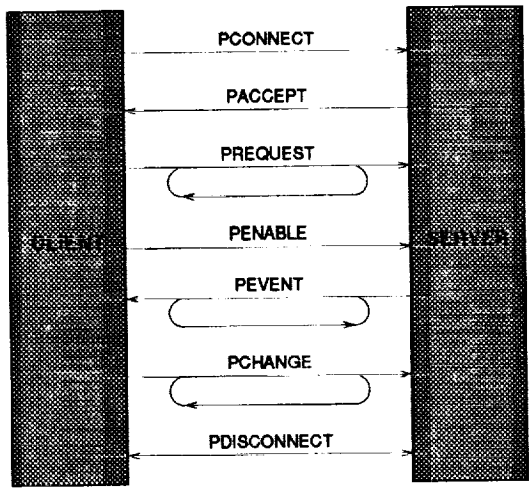

Figure 3: Client-server event exchanges.

shell. Using the ISP, however, we overcome this limitation by establishing a common interface enabling these expert system components to communicate with each other. Furthermore, since the ISP message packets can contain arbitrary data, a variety of information can be communicated. The information can be a sensor reading, a CLIPS fact, the results of a simulation or analysis, a fault message, and so on. These features promote a gradual development of heterogeneous distributed expert systems using the most appropriate inference engine or shell for the task.

Using the ISP, we have already begun to deploy some exciting distributed expert system applications for the CCC. We summarize a few of the highlights below:

PRS We have already deployed the ISP within prototype applications of the Procedural Reasoning System (PRS) [1, 2]. PRS is a real-time, multi-agent reasoning system capable of monitoring and controlling complex physical systems. The PRS development work performed for NASA has focused on the problem of handling spacecraft malfunctions and executing diagnostic procedures, assisting the operator in monitoring execution of these procedures, and adapting them to new situations. The PRS agents converse with each other in a peer-to-peer fashion, distributing the workload and knowledge bases according to the various tasks being performed by the application. These agents exchange information on a change-only basis, updating each other's data bases and contexts in order to monitor and control the behaviors of the physical system. Using the ISP, we empower sophisticated fault detection and analysis programs to provide status information to the PRS malfunction management systems, and provide an interface through which we merge PRS results into conventional telemetry displays.

VISTA Working with the Vista project team at the Rockwell Science Center Palo Alto Laboratory, we are developing an integrated decision-theoretic approach to the problem of managing the complexity of displays used in high-stakes, time-critical decision contexts [3, 4]. As a side effect of this effort, we have also employed the same belief network and utility models we use for display management for the problems of

fault detection, diagnosis, and ideal action selection. By using the ISP to distribute the results of these models among various applications, we are able to build integrated decision-making systems based on probability and utility. For example, we can distribute the results of decision-theoretic models to window manager clients to automatically select the optimum collection and configuration of displays for the human operator, thereby maximizing the presentation of relevant information while minimizing irrelevant clutter. We can also distribute the results of the various action models to the Flight Director, who integrates these actions within the current context into a prioritized list of actions for the astronauts or ground crews.

SELMON We are working with the JPL selective monitoring (SELMON) project team to integrate SELMON techniques within the distributive computing models [5, 6]. SELMON provides a collection of *sensor importance measures* to determine which of the observered sensor values contain the most interesting information. There currently are seven sensor importance measures that can be applied to each value. The four empirical measures, *surprise*, *alarm*, *alarm anticipation*, and *value change*, can be applied directly within the ISP servers as part of the preprocessing, providing a relative information content "score" to each changing sensor value. This score is passed along with the sensor value to the clients. The three model-based measures, *deviation*, *sensitivity*, and *cascading alarms*, are excellent candidates for client-based calculations which can be redistributed through the servers. Since it is then left to the consumer clients to deal with this relative information score, we can gradually deploy this additional information within the clients as they become more "intelligent." For example, we can use color changes to attract attention to sensor readings with high scores, directly influencing the user's decision-making efforts while monitoring component behaviors. We can also accumulate aggregate scores for entire displays, based on the accumulation of individual sensor scores, and prompt the user to concentrate his attention on that display because it contains more interesting information than others. These techniques fall nicely in line with the automated display management ideas of the Vista project. This work also is providing a substrate for the follow-on diagnostic reasoning embedded in monitoring (DREMON) project currently being pursued at JSC.

Beyond these few examples, there are also a variety of other distributed expert system applications, in various stages of development, which assuredly will benefit from a common communications protocol.

5 Information

We encourage questions and comments about the ISP from both inside and outside the spacecraft operations community, as we hope to diversify into other application areas as we continue our refinement of this distributed-computing protocol. For more information, or to obtain the latest ISP source and program distribution, contact the authors at NASA/Johnson Space Center, Mail Code DF6, Houston, TX, 77058, or send electronic mail to `barry@rpal.rockwell.com`, `nunikls@jscprofs.nasa.gov` or `nunispw@jscprofs.nasa.gov`.

References

[1] Georgeff, Kinny, Hodgson, and Lee. *PRS Operational Evaluation.* SRI International, Project ECD 333, Menlo Park, CA, 1993.

[2] Georgeff and Ingrand. *Real-Time Reasoning: The Monitoring and Control of Spacecraft Systems.* Proceedings of the Sixth IEEE Conference on Artificial Intelligence Applications, Santa Barbara, CA, 1990.

[3] Horvitz, Ruokangas, Srinivas, and Barry. *A Decision-Theoretic Approach to the Display of Information for Time-Critical Decisions: The Vista Project.* Proceedings of Fourth Annual Workshop on Space Operations Applications and Research (SOAR '92), NASA/Johnson Space Center, Houston, TX, 1992.

[4] Barry, Horvitz, Ruokangas, and Srinivas. *Vista Goes Online: Decision-Analytic Systems for Real-time Decision Making in the Mission Control Center.* Submitted to the Ninth Annual Goddard Conference on Space Applications of Artificial Intelligence, NASA/Goddard Space Flight Center, Greenbelt, MD, 1994.

[5] Doyle, Chien, Fayyad, and Wyatt. *Focused Real-Time Systems Monitoring Based on Multiple Anomaly Models.* Seventh International Workshop on Qualitative Reasoning, Eastsound, WA, 1993.

[6] Doyle, Charest, Rouquette, Wyatt and Robertson. *Causal Modeling and Event-driven Simulation for Monitoring of Continuous Systems.* AIAA Computers in Aerospace 9, San Diego, CA, 1993.