AN INTELLIGENT MANAGER FOR A DISTRIBUTED TELEMETRY SYSTEM

Arthur N . Rasmussen

ABSTRACT

A number of efforts at NASA's Johnson Space Center are exploring ways of improving operational efficiency and effectiveness of telemetry data distribution. An important component of this is the Real-Time Data System project in the Shuttle Mission Control Center. This project's telemetry system is based on a network of engineering workstations that acquire, distribute, analyze, and display the data. Telemetry data is acquired and partially processed through a commercial programmable telemetry processor. The data is then transferred into workstations where the remaining decommutation, conversion and calibration steps are performed. The results are sent over the network to applications operating within end user workstations. This complex distributed environment is managed by PILOT, an intelligent system that monitors data flow and process integrity with the goal of providing a very high level of availability requiring minimal human involvement. PILOT is a rule-based expert system that oversees the operation of the system. It interacts with agents that operate in the local environment of each workstation and advises the local agents of system status and configuration. This enables each local agent to manage its local environment and provides a resource to which it can come with issues that need a global view for resolution. PILOT is implemented using a commercially available real-time expert system shell and operates in a heterogeneous set of hardware platforms.

KEY WORDS

intelligent manager, distributed data system

INTRODUCTION

The Mission Control Center at NASA's Johnson Space Center has been the focus of manned space flight operations for nearly 30 years. Its purpose is to support all manned space activities including Space Shuttle missions and upcoming missions such as Space Station *Freedom*. The Control Center is manned by a team of flight controllers whose goal is to ensure a safe flight that achieves the highest level of

mission objectives. In addition, this must be performed in an effective and economically efficient manner.

Several activities have been pursuing these goals. The Real Time Data System (RTDS) is a project established to explore the applicability of new technologies and approaches to the task of flight control. The project is examining and producing demonstrations in a number of areas including artificial intelligence, man-machine interfaces and distributed architectures and techniques. The environment in which these have been performed is comprised of a heterogeneous network of engineering workstations. These platforms are based on commonly available commercial software such as the Unix™ operating system, the X-window system and OSF/Motif, and the TCP and UDP networking facilities. Upon this has been constructed a network-based real-time telemetry distribution system which in turn supports flight control applications for real-time analysis and display.

DATA DISTRIBUTION ENVIRONMENT

The telemetry stream downlinked by the Space Shuttle is a relatively complex one. It is a 192 kilobit PCM stream that includes a primary stream and up to 6 embedded asynchronous streams that are in turn mixed with digitized voice channels. Each of the streams is formatted independently and the formats are routinely switched during a typical Shuttle mission; a minimum of seven to more than a dozen is typical. Additionally the embedded streams vary in frame length as needed to accommodate data during different mission phases (e.g., engine data during ascent and payload data while on orbit). There are a large number of parameters in the streams (typically over 10,000 for a given mission) that are sampled at rates that vary from 1 to 100 Hertz.

The RTDS telemetry system deals with this complexity by dividing the work of recovering parameters from the streams into two portions, one performed by a telemetry processor and the other by an engineering workstation. A representative view of the RTDS hardware environment is shown in figure 1. Data from the Shuttle is received as a serial PCM stream that is fed to the telemetry processor. The processor performs bit, word and frame synchronization on the data and then examines the format identifier in the primary stream to determine the frame sizes of the embedded asynchronous streams. The embedded streams are then extracted from the primary stream and frame synchronization is performed on them. The final step within the processor is to label the frames of the streams and deliver them to telemetry workstations using a direct memory access (DMA) channel.

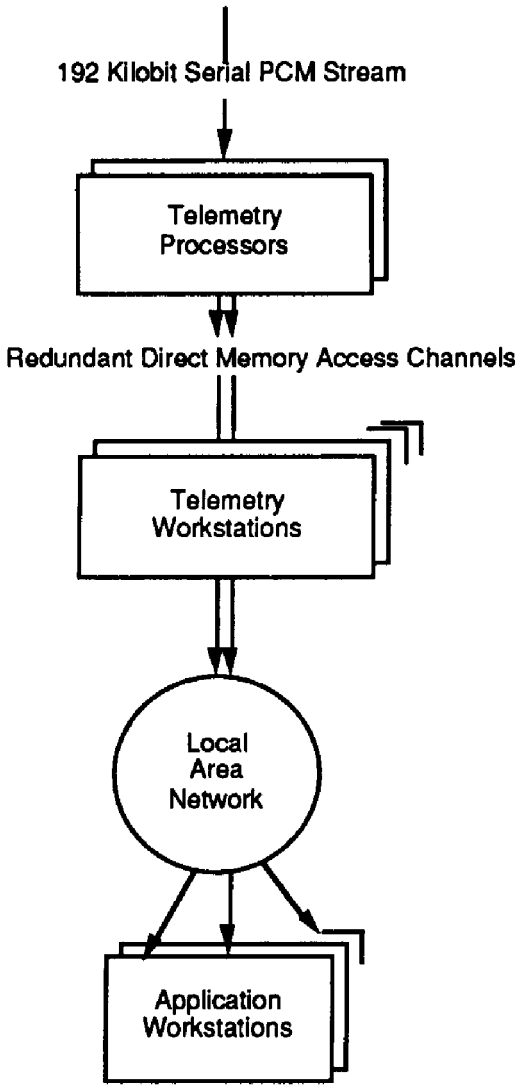

Figure 1. Representative Hardware Environment

The telemetry workstations accept the frame data from the telemetry processor and complete the process. Individual parameters are decommutated based on the current frame format, converted to native workstation data types and then calibrated to engineering units. This puts the data into a state suitable for use by application programs. The data is then transferred over the network to application workstations where it is made available to application processes for analysis and display.

This process is shown in more detail in figure 2. The figure shows a representative data flow from PCM stream form to final form consumed by the application processes with a focus on the activities performed by the software within the workstations. The telemetry frames are received from the telemetry processor by a software process that assembles them into a complete major frame. The frame is built in a shared memory segment to speed communication with the subsequent process. The next process uses

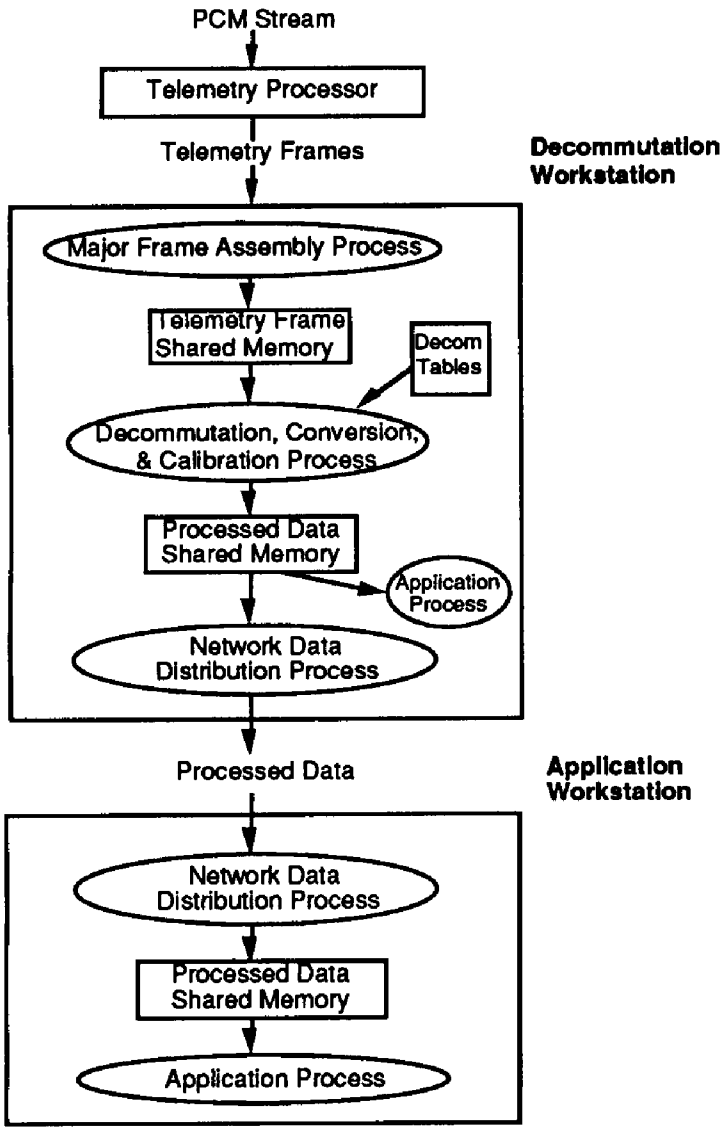

Figure 2. Representative Data Flow

decommutation and calibration information to extract individual parameters from the telemetry frame shared memory; it applies the appropriate conversion and calibration transformations and places the data and associated status information into a shared memory segment. At this point all artifacts of the telemetry streams have been dissociated from the data except for availability status and sample rates, freeing application programs from the need for most format-dependent behaviors.

The application processes obtain the data using an application program interface which delivers to them only the data they request. To distribute the data to other workstations, a distribution process acquires all of the available data (using the same interface as any application), compresses the data based on a change-only algorithm, and transmits the compressed data to receiving processes executing within the

application workstations. The receiving end recreates the process data shared memory in its local workstation environment so that it is identical to that in the sending workstation. This allows the same application interface to be used in both sending and receiving workstations and makes the transfer mechanism transparent to the application processes. The distribution processes are structured so that they can easily be executed within a single workstation and so that a sending workstation can transmit to a large number of receivers. This capability is routinely used within the RTDS environment to establish a hierarchy of data distribution in order to distribute the load of the data distribution more evenly across the environment and to take advantage of local area subnet isolation.

SYSTEM MANAGEMENT APPROACH

The originally purpose of the telemetry system was for exploring issues and approaches to the problems of the real-time distribution of telemetry. Its various capabilities soon made it very popular as both a development environment and as a supplemental flight support facility. (Examples are the system's ability to log telemetry data to file and play back the data in manner identical to the real-time mode, and a tool for controlling this facility that presents the developer with a control panel that looks and functions like a home video recorder.) As the use of the system grew, experience with its operation uncovered data outages caused by a variety of problems, primarily with the software processes and underlying support software (e.g., the operating system and the network software). To obtain the reliability desired for support of simulated and actual Shuttle missions, a person was forced to monitor the system and correct anomalies as they occurred.

As the monitoring of the system became increasingly burdensome, a plan for automating the task was conceived. Several basic points of design philosophy guided this effort. Foremost was the desire to take advantage of the strengths of the distributed nature of the system. This implied abandonment of many previous concepts in use within the Control Center since they were based on a strongly centralized architecture. A second design assumption was that software failures are inevitable, and that reinstatement of the data delivery service was more important than detailed problem diagnosis. This produced a design whose software components are inherently easily removed and replaced when a failure occurs.

The result is a distributed management approach based on a layered architecture. At the lowest layer are the service processes that perform the data delivery tasks. These were instrumented to make their internal state information available to other processes. Once this very necessary visibility was provided, the next layer was created as a process designed to maintain the health of the local workstation environment by

managing the service processes. The final layer is an overall system manager that interacts with the local managers to maximize the health of the whole system.

SYSTEM DESIGN AND IMPLEMENTATION

Instrumentation of the service processes is implemented using simple mechanism that is both effective and efficient. A library of support routines provides each service process with access to a status area within a shared memory segment. The process identifies itself to this facility by name and receives access to the status area. The process then reports its activity and state information in the area; this is done with minimal overhead since the process has direct access to the memory area. The facility allows other processes to examine the area and so gain direct and immediate visibility into the service process. It also allows other processes to make requests of the service process.

Once this mechanism of visibility became available, a process was created to use the information to manage the local workstation environment. It was named Autopilot because of its conceptual mode of operation. The desired set of processes to comprise the local environment is dialed into Autopilot's software "knobs"; it then takes all actions within its range of capabilities to maintain that environment until a new setting is selected.

In normal operation, Autopilot manages all of the service processes executing within the workstation. It starts the processes and monitors their health and execution states using the status area as well as operating system capabilities. Figure 3 shows how Autopilot fits into the workstation's environment. When it detects a problem in a service process, it issues a termination warning to the service using an appropriate mechanism. If the service process is capable, Autopilot issues a "soft" terminate request to allow the process to attempt recovery or cleanup before terminating. After a suitable interval, Autopilot removes the process (if necessary) and replaces it with a new instance. If the service process is capable, Autopilot issues a "soft" terminate request to allow the process to perform cleanup before terminating. The state information of failed processes and all actions taken on processes are logged by Autopilot for later analysis.

The actions taken by Autopilot can be tuned in several ways. Each process managed by an Autopilot has an associated set of management policies and parameters that determine how the Autopilot monitors and takes actions on the process. Examples include the means and criteria by which the Autopilot judges the health of the process, the steps the Autopilot performs while terminating the process, and relationships among related managed processes. A managed process can interact with Autopilot in

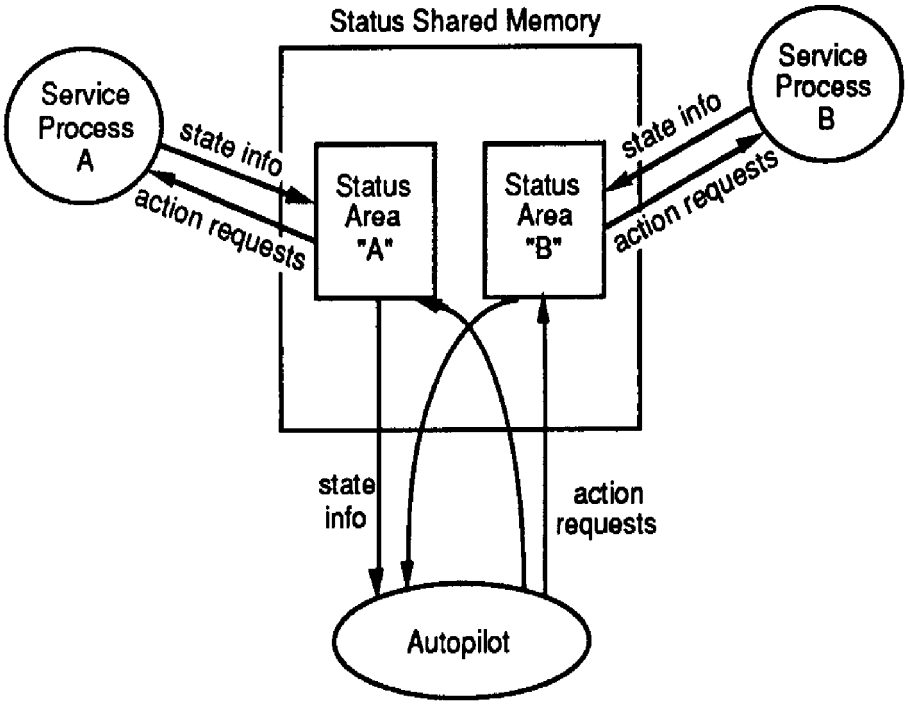

Figure 3. Autopilot Operation

controlled ways, including providing advisory information and requests to Autopilot to adjust the management policies to reflect activities being performed within the process. For example, a managed process can advise Autopilot that it is waiting in an inactive state.

The design of Autopilot calls for it to function as autonomously as possible. This enables it to continue to operate independently of other workstations in the network. By its definition it has a limited view of system operation. To provide system level view, PILOT was created. Its purpose is to interact with a set of workstations, each containing an executing Autopilot, and take the role of the human operator. It is implemented as a rule-based expert system using a commercial real-time shell, G2™. The shell provides a user extensible object class hierarchy, a graphical user interface, rules and procedures. It also has a facility that simplifies communication with other processes.

The ability of PILOT to overlook the entire system gives it the ability to reason using contextual knowledge that is not available to the Autopilots. In turn the ability of Autopilot to manage the details of the local workstation environment frees PILOT from the unnecessary complexity that would result if PILOT were required to do so. PILOT is still in active development; expertise gained from experience with both operations support and PILOT's operation is being used to augment PILOT's knowledge base.

PILOT's current role is primarily that of an advisor. It builds and maintains a dynamic model of the workstations and their data flow interconnections. It provides several displays, including a diagram of the current data flow hierarchy as shown in figure 4; additional displays provide a deeper view into the state of a workstation. PILOT reduces the parameters that describe the workstation environment to a small set of qualitative measures. These are used to characterize the system state for the human operator and to provide a basis for reasoning about nominal and anomalous system operation. The model is used by PILOT to detect and isolate failures and will be used to resolve them. For example, current experimentation includes rules that detect loss of data within a set of workstations and attempt to determine whether a common point of failure can be found. This deals with an observed situation in which the network service software has become deadlocked.

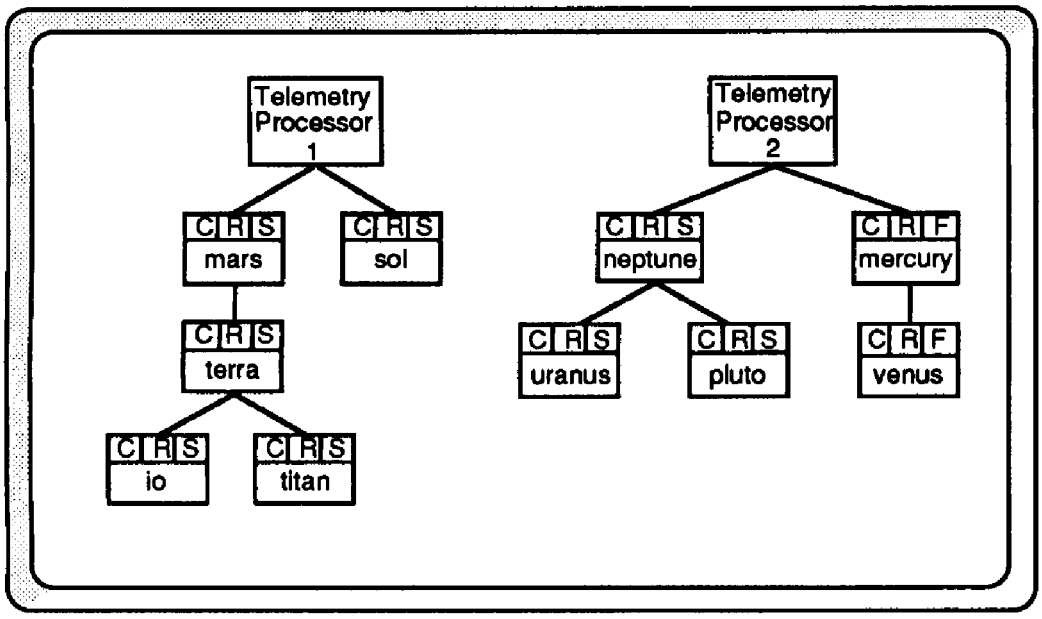

Figure 4. Hierarchical Data Flow Display

As PILOT masters the simpler system management tasks, it will be expanded in two areas. The first is overall system setup and configuration. Currently the human operator provides information about the day's activities and the configurations needed to support them. This process will be automated so that PILOT is aware of the day's schedule and can advise the workstations on proper configuration as well as handle overriding requests from the workstation user. The second major area of work will be in dynamic monitoring and control of the system as a whole. Additional issues at a global level will be addressed such as the policies and procedures used to reroute data flow around failed workstations or network segments. A related area of investigation will be the use of contextual information adjust policies and the ability to learn appropriate policies based on experience. The interaction between PILOT and

Autopilot is also being explored, including means of distributing portions of PILOT's expertise to the Autopilot.

CONCLUSION

The PILOT and autopilot solve many of the needs of a distributed telemetry system. The autopilot's ability to detect and resolve problems with malfunctioning software in the local workstation environment have greatly increased the robustness of the RTDS telemetry system while simultaneously reducing data dropouts due to the delivery system. Experience with the PILOT expert system show that it substantially reduces and simplifies the workload of the data system operator; as additional expertise is captured within it, it is expected it will become able to manage the data system in all but the most catastrophic circumstances.

REFERENCES

Muratore, John F., et al., "Real-time Data Acquisition in Mission Control," *Communications of ACM*, 33, 12: 18-31, 1990.

Heindel, Troy A., et al., "Knowledge-based Systems in Space Shuttle Mission Control," *Aerospace America*, October, 1991, 8-11.

Muratore, John F., et al., "Real-time Data Acquisition for Expert Systems in Unix Workstations at Space Shuttle Mission Control," *Twenty-first Symposium Proceedings, Society of Flight Test Engineers, 1989.*