PILOT: An Intelligent Distributed Operations Support System

Arthur N. Rasmussen

The MITRE Corporation
Houston, Texas, USA

ABSTRACT

The Real-Time Data System (RTDS) project is exploring the application of advanced technologies to the real-time flight operations environment of the Mission Control Centers at NASA's Johnson Space Center. The system, based on a network of engineering workstations, provides services such as delivery of real time telemetry data to flight control applications. To automate the operation of this complex distributed environment, a facility called PILOT (Process Integrity Level and Operation Tracker) is being developed. PILOT comprises a set of distributed agents cooperating with a rule-based expert system; together they monitor process operation and data flows throughout the RTDS network. The goal of PILOT is to provide unattended management and automated operation under user control.

Key Words: Distributed systems, artificial intelligence, automated operations

1. INTRODUCTION

The Mission Control Center at NASA's Johnson Space Center in Houston has been the focus of manned space flight operations for nearly 30 years. The Center's current responsibility is ground support for all Space Shuttle missions; in the near future this will be expanded to other manned space activities including Space Station *Freedom*. The overall goal of the flight controllers in Mission Control is to ensure a safe flight that achieves the highest level of mission objectives.

An important subgoal is performing the ground support role in an economically efficient manner. Several approaches to this are being pursued, one of which is the exploitation of new technologies to increase the level of automation. The Real Time Data System (RTDS) project is chartered with exploring the applicability of new technologies, proving concepts, and demonstrating their practical application to solving problems of real-time operations in Mission Control (Refs. 1-3). An important aspect of this work is the evaluation and integration of distributed system technologies into the unique Mission Control environment.

An area that exemplifies the shift to distributed solutions is the RTDS data acquisition and distribution system. This system ingests the real time telemetry stream received from the Shuttle, decommutates it, applies transformations such as calibration to engineering units, and transfers the data to a set of engineering workstations where the data is analyzed by flight control application software. The complex telemetry stream is composed of a main stream and several embedded asynchronous substreams containing thousands of data being collected at a variety of rates and having hundreds of configuration states.

The RTDS solution incorporates a commercially available programmable telemetry processor which is augmented by custom integration software operating in a workstation. Figure 1 shows an overview of this approach. The approach is representative of architectures being adopted for use in Mission Control and has numerous advantages over previous approaches.

A goal of RTDS is to raise reliability to the highest affordable level. Key to reliability is uninterrupted service from the application program's viewpoint. This allows for failure of components that are providing a service if a compensating action minimizes the impact on the application. Since RTDS is based on commercial hardware and software platforms (including the operating system and base networking software), services such as telemetry data delivery must capitalize on the platforms' strengths and compensate for weaknesses.

The application services provided within RTDS are supplied by custom software components that bridge the gap between the needs of the applications software and the services available as part of the basic platform. Until recently, reliability was obtained by employing a human operator to monitor the health of the system services, diagnose problems and take manual corrective action to correct them. However, the slow operator response to multiple failures and the cost of continuous staffing were not acceptable in the long term.

2. APPROACH

To provide better, faster service and to release the human operators from monitoring dozens of workstations and their software processes, a decision was made to automate with the goal of eliminating the need for a human operator. At the same time it was recognized that the human operators held considerable expertise on efficient operation and that capturing this expertise would be very desirable.

2.1 Design Philosophy

The design philosophy adopted for the task attempts to exploit the strengths of the distributed environment rather than to force the imposition of previous (primarily centralized) architectures. One implication is that problems should be dealt with locally whenever possible so as to provide the quickest response and the least impact on other elements. A second design precept assumes that failures will occur, and that expedient recovery is pragmatically more feasible and desirable than taking extraordinary measures to ensure that a failure will not occur. The third principle states that, wherever possible, the custom software elements should make the needed accommodations since it would be neither sensible nor economically feasible to alter the commercial components of the system.

Within this design framework rests an approach based on three main elements. The first element is in the implementation of the custom software components that provide the application services. These are designed so that they are easily replaced at execution time by a second instance should the first instance fail. In addition, they are instrumented to make their operation externally visible. This allows another process to monitor their states, control their behavior, gather information when failures occur, and diagnose problems.

The second element in the approach is a process that monitors each service delivery process, controlling its behavior, tracking its operation and replacing it should it misbehave. An instance of this process runs in the local environment of each workstation and has the responsibility of maintaining the integrity of services within the environment. The process is called the autopilot because its operation is like that of the autopilot of an aircraft; the software "knobs" on the autopilot are set and the autopilot does its best within its range of control to maintain the requested state.

Whenever there are a set of related entities operating in a distributed environment, issues usually arise that are beyond the scope of a single entity. Means for dealing with such issues are a current area of research; the initial approach for RTDS is the proven technique of having a central coordination entity. As a result the third element in the design is a rule-based expert system that oversees the operation of the autopilots; this is named PILOT (Process Integrity Level and Operation Tracker).

To act as its representative in each workstation's local environment, PILOT employs a process called the "spy". The spy gathers information from the local environment, filters it according to the wishes of PILOT, and transfers it to the PILOT over the network. It also provides PILOT with the mechanism to advise and control local processes.

2.2 The Service Delivery Processes

Nearly all of the processes that provide RTDS services to applications operate as "daemons", executing in a background mode unattached to any specific user interface. In addition, the human user is generally uninterested in (and often unaware of) the details of daemon operation. To provide visibility into daemon behavior and to provide a means to control daemon behavior, a standardized mechanism has been developed and integrated into all RTDS daemons.

This communication mechanism, nicknamed "VIZ", makes it possible to inspect a running process to determine detailed information about its state, configuration and recent error history. This allows a monitoring process such as autopilot to closely watch each daemon and determine whether the daemon is performing useful work or has failed in some way (e.g., has deadlocked or otherwise gotten "stuck"). The monitoring process cannot only quickly detect a failure, but also take actions such as logging the daemon's state information for later analysis or even predicting failure before it actually occurs. Other uses for the visibility features include providing a simple display of daemon operation to users and a more detailed one for development purposes.

The action request portion of VIZ allows an external process to make requests of the daemon. The initial use of this feature provides a simple means to request the process to terminate; using the VIZ capability overcomes the normal operating system restriction that inhibits one user from terminating the processes started by another user. A more important use is the ability for a monitoring process to request status information from a daemon. A planned use by the autopilot process is to provide hints to a new instantiation about the

circumstances of the demise of its predecessor; this may allow the replacement process to avoid the problems encountered by its predecessor.

The VIZ capabilities appear to the daemon to be a set of procedures that allow the daemon to register itself under its name, update its state information and receive action requests. The current implementation is based on transformation of most procedures into direct accesses to a shared memory area. This reduces the overhead to a negligible amount while preserving the ability to make enhancements. The information is stored in a canonical form that permits transporting it over the network in a transparent manner.

2.3 The Autopilot Process

The capabilities of the autopilot form the first level of system management and reflect the basic design tenet that situations arising on a workstation should be handled locally if possible. The purpose of autopilot is to manage the local process environment, track process operation, record anomalous behavior, and replace misbehaving processes with new instances. The normal operation is as a daemon in a "set and forget" mode; one autopilot executes on each workstation, receiving instructions as to the desired process environment and taking action to create and maintain that environment.

In normal operation the autopilot manages all of the daemon processes executing within the workstation. It starts the processes and monitors their health and execution status using both the VIZ and operating system capabilities. Figure 2 shows how the autopilot fits into the workstation's environment. If it detects a problem in a daemon, it issues a warning to the daemon using an appropriate mechanism. After a suitable interval, autopilot removes the process and replaces it with a new instance. If the daemon is capable, the autopilot issues a "soft" terminate request to allow the process to perform cleanup before terminating. If the process fails to terminate, autopilot forces termination. 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 the autopilot in controlled ways, including advisory information and requests to the autopilot to adjust its management policies to reflect activities being performed within the process. For example, a managed process can advise the autopilot that it is entering a phase in which it will appear dormant for an unusually long period.

Like all RTDS daemons, an autopilot reports on its state using the VIZ capabilities. This permits several autopilots to operate and manage independent sets of processes while being managed by a parent autopilot. The state information from VIZ is also used by an interface process (the autopilot "control panel") whose purpose is to control the autopilot. The process has the abilities to determine which processes are being managed by the autopilot, which policies are in effect, and the policy-related state of each process. The interface process also provides a mechanism for making requests such as starting the management of new processes, restarting or removing currently-managed processes, or requesting the termination of the autopilot.

2.4 The PILOT Expert System

Early experience with using the autopilot to manage workstations clearly illuminated which areas were well-addressed and which were not. Once set into operation with an appropriate configuration, an autopilot manages local operations well. As expected, the autopilot easily handles stuck processes and unexpected process terminations. Its operational shortcomings are directly traceable to its lack of contextual and global knowledge; it operates having little information about the expectations of the user and the general status of the network and other workstations.

The role originally envisioned for PILOT, to replace the human system monitor, has been validated by experience with the autopilot. PILOT is still in active development under the strategy of capturing the expertise and reducing the need for human intervention one functional area at a time. Two primary goals are being used to decide the order in which PILOT capabilities are developed. The first is which capabilities will release the maximal amount of the human resource, and the second is which are required as foundations for other capabilities.

The current PILOT is primarily an advisor. It builds and maintains a dynamic model of the workstations, their internal states and their

interrelationships. It creates several basic displays, including a diagram of the current data flow hierarchy as shown in figure 3, and displays of the detailed status of an individual workstation. It reduces the workstation state information to a small set of qualitative parameters. These are now used to characterize system state for the human operator; they are also being used to construct a model-based reasoning capability that will perform diagnostic tasks such as isolating workstations or network segments that are causing loss of data to a subtree.

PILOT is implemented as a rule-based expert system using Gensym Corporation's G2™ real-time expert system shell. The shell provides a user extensible object and class hierarchy, a graphical user interface, procedures and rules. Due to licensing restrictions, PILOT operates on a dedicated workstation. The general strategy implemented in the rules is to detect events and activate rules and procedures to evaluate them.

2.5 The Spy Process

To obtain information and cause actions on a workstation, PILOT uses a spy process as its agent in the local environment of each workstation that PILOT is monitoring. The spy process executes in the local environment and communicates over the network with PILOT. A spy appears to the local environment as a daemon and is managed by the autopilot running in that environment. It attaches to system services, the VIZ service and the program interface provided by autopilot in order to monitor activity in the local environment. Figure 4 shows the role of the spy process.

Each spy gathers information on the local environment and sends it to PILOT; from PILOT it receives requests directing the spy to adjust its behavior or to interact with other processes such as autopilot. PILOT can request that the spy send automatic periodic updates or can place it in a polled send-on-demand mode. A spy attempts to minimize its network traffic by performing local filtering and by sending only changed value events. Current work on the spy agent emphasizes increasing its intelligence; e.g., a spy will know how to initiate and track multi-step operations that are activated by a single request from PILOT.

3. FUTURE DIRECTIONS

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. A user advocate process is currently under development that will allow the local workstation user to make simple requests to perform expected tasks (e.g., "set up my workstation to support the scheduled simulated mission") and to provide for special usage (e.g., "I want to review the data from a specific previous mission"). This process will establish the desired configuration and inform PILOT of the user's desires so that, for example, PILOT does not assume the workstation is improperly configured.

The second major area of work will be in dynamic monitoring and control of the system as a whole. 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. Capabilities will be added to PILOT to perform active troubleshooting and corrective actions. Another area of investigation will be the use of contextual information adjust policies and the ability to learn appropriate policies based on experience.

The planned strategy is to experiment with techniques in PILOT to gain experience quickly, and then to distribute the knowledge gained to the lowest level possible (e.g., to the spy or autopilot). Although the current architecture is strictly hierarchical with PILOT at the apex, plans are to introduce peer-to-peer communication; the autopilots will be first augmented in this way. PILOT will be retained to serve as a central decision resource as well as provide a system-level troubleshooting interface for a human operator. As the network expands and as motivated by the nature of flight control, clusters of workstations will be organized to provide services to flight control areas. This may require several PILOTs operating cooperatively to maintain their clusters and negotiate with neighbors for resources and information.

4. REFERENCES

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

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

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

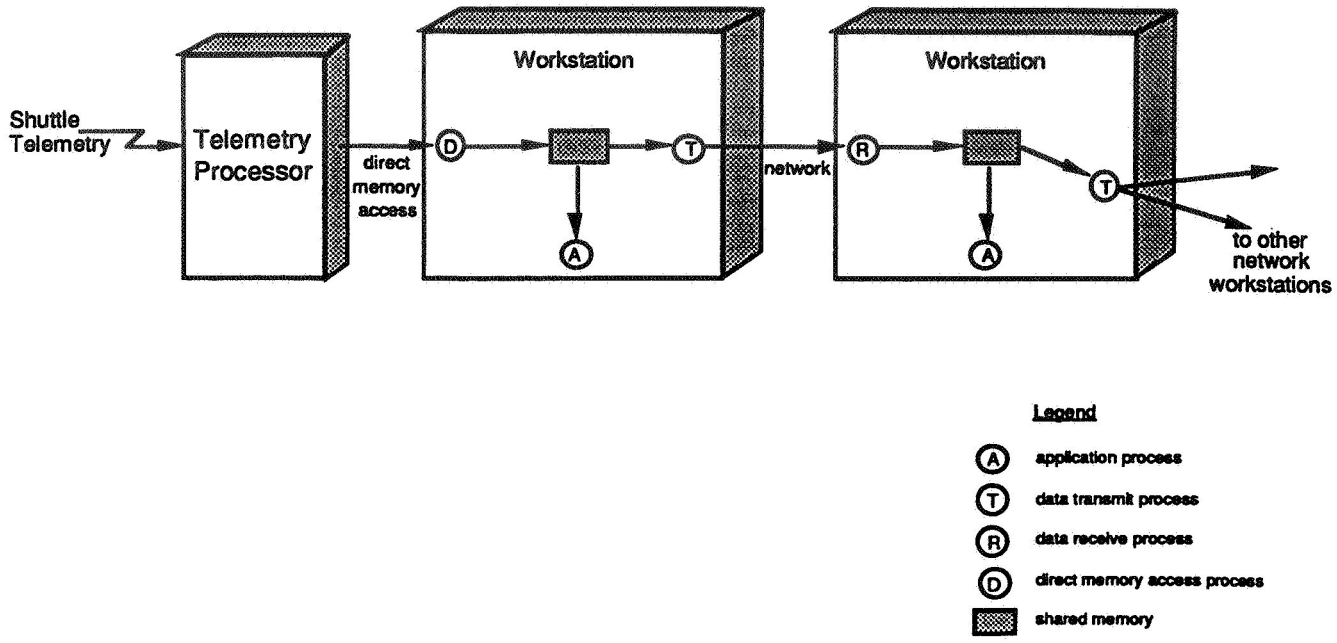

Figure 1. RTDS Telemetry Data Flow

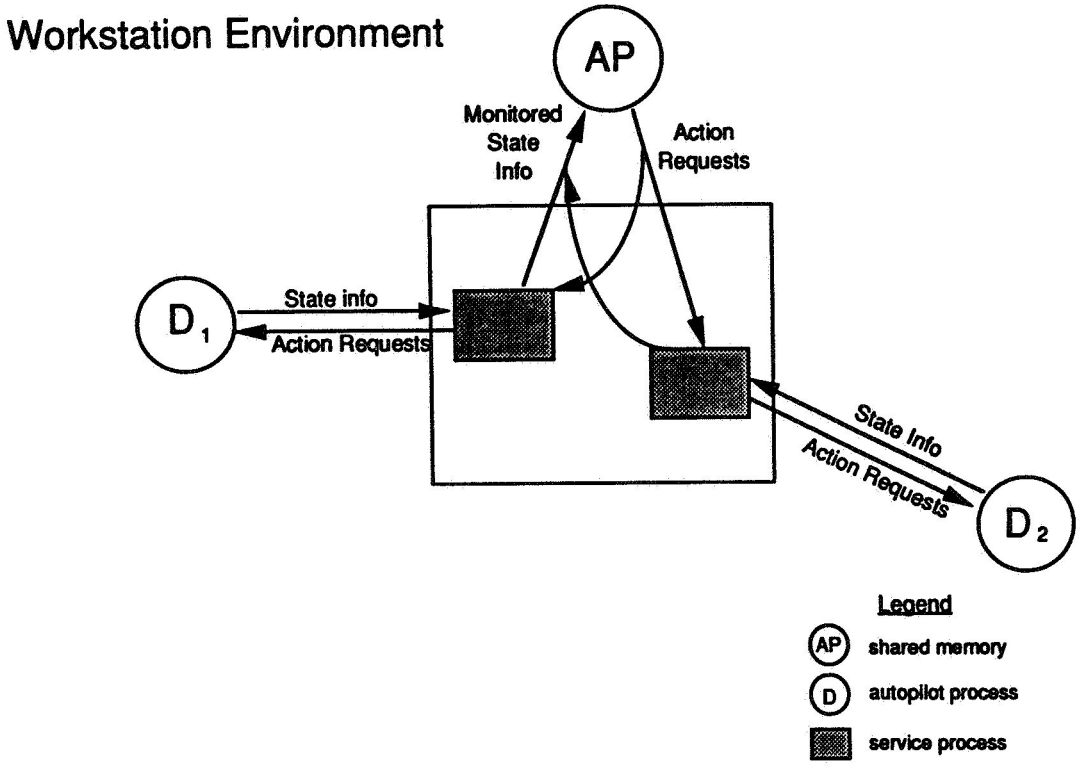

Figure 2. Autopilot Operation

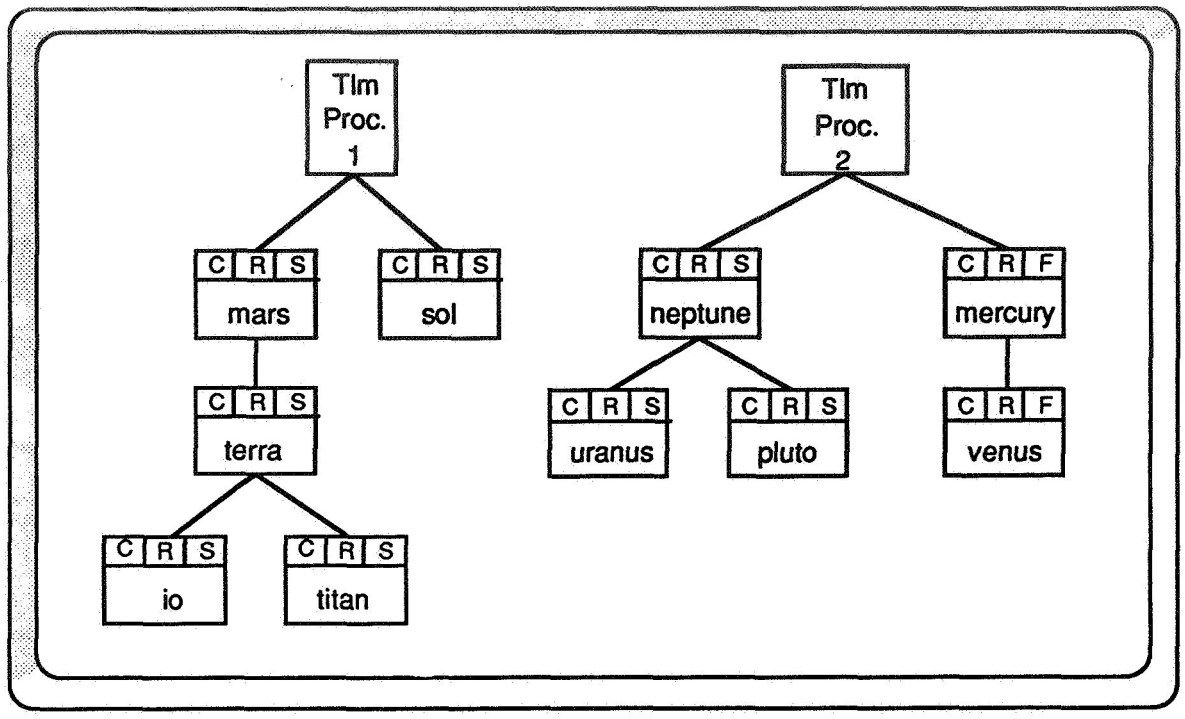

Figure 3. PILOT's Network Hierarchy Display

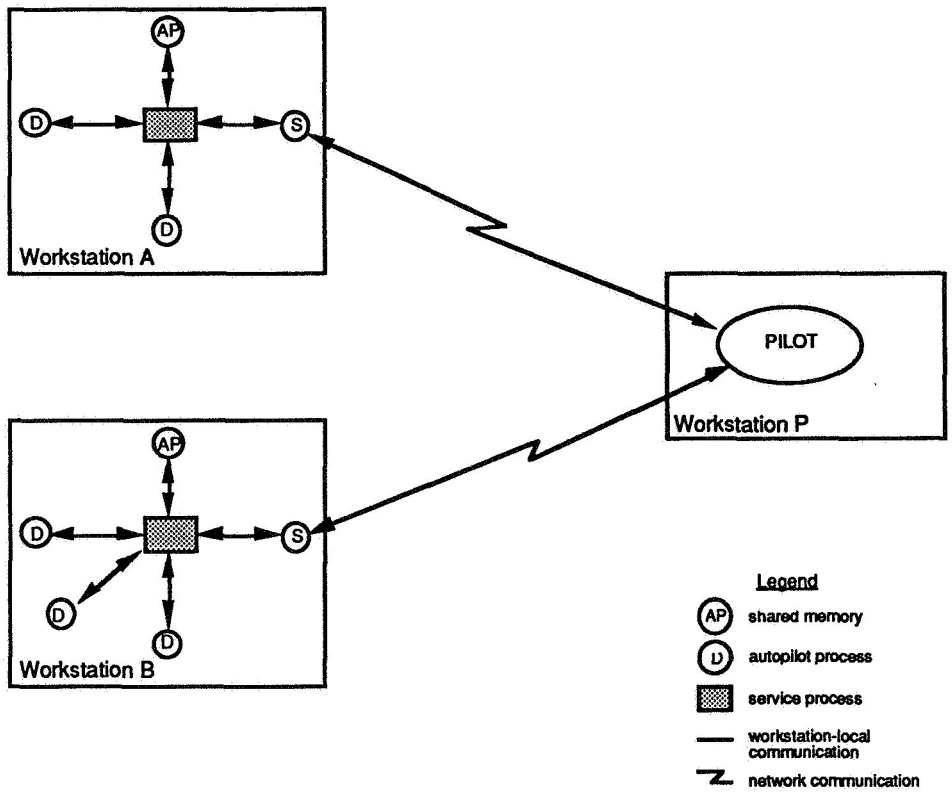

Figure 4. Spy Process Operation