DISTRIBUTED SYSTEMS INTEGRATION AND IMPLEMENTATION TECHNIQUES IN A NETWORK BASED TELEMETRY SYSTEM

Jeffrey D. Kelley
Lead Software Engineer
Loral Data Systems
Sarasota, Florida, 34230

ABSTRACT

This paper addresses the distributed systems implementation techniques used in the development of the EMR O/S90 Open Systems Telemetry System. Specifically, it presents the integration, networked load-balancing, and remote control aspects of the telemetry system which allow it to adapt to differing configurations and availability of resources.

INTRODUCTION

Loral Data Systems has recently developed a workstation-based multi-purpose data acquisition and display system based on Open Systems concepts, standards, and networks. This framework implies an openness of both the applications software and the system architecture. Standards such as POSIX, OSF Motif, SQL, and X Windows, combined with de facto standards such as NFS, TCP/IP, and ANSI C, increase the portability of applications from vendor to vendor and aid with the installation of software onto various platforms. These standards, however, address only the applications software aspect and do nothing to ease the problems of systems integration, system enhancement, load balancing, and remote control that are associated with an integrated networked system. A unique perspective has been applied to these problems to create a system architecture which will allow it to grow, transform, and reallocate its resources with little or no impact on the underlying applications software which combine to create the system as a whole.

This paper addresses specific system architecture designs and implementations which allow the system to resolve problems associated with load balancing, remote control, and systems enhancement and integration. These approaches are centered around the Resource Manager control software. The Resource Manager allows the telemetry system to reallocate and redistribute processing tasks around the network without impacting the software itself. This allows each system to be load-balanced uniquely and independently according to its computing resources. Also, the command-driven nature of the Resource

Manager allows for remote control and batch-oriented processing of virtually any telemetry system feature normally accessed interactively. The entire system can be controlled via commands from remote command lines or remotely located software applications. This allows for the inclusion of the entire telemetry system (or any parts thereof) as a feature (or features) of an existing processing system already in place. Finally, the Resource Manager serves to simplify enhancement and modification of the system. Applications can be added with no impact to the underlying control system, allowing new user interfaces and new applications to be added to the system quickly and easily. These aspects allow the telemetry system to adapt to different processing needs and unique processing environments quickly and efficiently as the telemetry environment evolves.

BACKGROUND

In 1990 Loral Data Systems (LDS) decided to move into the next generation of telemetry acquisition and processing systems. The primary goal of the new system was to shift away from the proprietary world of VAX/VMS and enter into the new forum of standards and Open Systems concepts. In order to accomplish this, the system would be designed to utilize such standards as:

- POSIX-compliant operating systems
- X-Window graphics
- OSF Motif menu presentations
- ANSI C compliant source code
- NFS file management
- TCP/IP network communication
- SQL database queries

Adherence to these standards would open the system to the rest of the computing world and increase portability of the system from vendor to vendor. Although the Open Systems concept was the primary goal of the new architecture and the most visible departure from the past, new design concepts in other areas also merited consideration. Prior experience had shown ways to improve systems integration, enhancement, and customization, while the networked capabilities of Open Systems brought many new factors into play. The new architecture needed to address all of these issues with a cohesive and controlled approach, and a new perspective was needed for the system design to achieve this.

When LDS developed the 8385 Display Station in 1985, networked systems were in their infancy and interoperability among vendors was virtually non-existent. The 8385 was an ASCII-terminal based VAX/VMS telemetry acquisition and display system which allowed for multiple users to access the system via one central computer. Due to the compatibilities of VMS on the various VAX platforms, this computer could range in size from a mainframe to a MicroVAX without affecting system software. However, this approach presented many restrictions. Though the system provided a baseline from which to customize and enhance new systems, the process was tedious and cumbersome due to the architectural design. While attempting to maintain modularity using a "building block" approach, in actuality all of the packages were closely intertwined, creating an entanglement of features which rippled throughout the system when modified. Few graphics were available, and no network capabilities existed. To operate the system, one had to be at an ASCII terminal connected directly to the computer. Portability was restricted to only the various sized DEC VAX computers.

With the advent of System 90 in 1988, graphics and networks made their way into the LDS offerings. System 90 was a single-server, many-client architecture and presented DEC UIS graphics on the workstations. This allowed for some distributed processing, but bottlenecks on the single server still developed. The graphics and the workstations also limited the availability to one person per workstation. To improve system enhancement and customization, an attempt was made to control the entire system as one "wholly contained" entity, rather than as a collection of unrelated packages. While this aided in system installation and integration, it did not alleviate the basic problem of interrelationships between the various software functions. Each function still borrowed from and depended upon other functions, once again causing ripples throughout the system when enhancements or modifications were made. And as before, the system was solely VAX/VMS and, while portable among the various DEC VAX platforms, was completely closed to other vendors.

As work on the latest generation, the new Open Telemetry System (O/S90), began in 1990, a number of design goals were identified. The primary goal was to adhere to the Open Systems concepts and support multiple vendors for the workstation platforms. By following the previously mentioned standards and modularizing those areas which were not covered by standards, this goal could be realized. Other design goals, while not as visible, were just as important. A system architecture had to be developed that would correct some of the inadequacies of past systems, while at the same time allowing the system to address the new capabilities introduced by the Open Systems networking concepts. The architecture of the new system needed to support easy systems integration and customization. Interrelated functions which formed the bedrock of the previous systems had to be disengaged from each other without complicating systems integration.

To resolve the potential bottleneck of a single server system, the design needed to include configurable distribution and reallocation of telemetry functions.

The availability of networks such as Internet at customer sites created another consideration. People not directly located at the range or launch site could benefit from the use of the telemetry system if such access was available. These users may not have X-Window access to the system, or interactive access may not be feasible due to distance and network traffic constraints. Remote controlled access to telemetry system functions would be necessary to address these needs. It was concluded that the three major areas to be addressed by the system architecture design would be:

- Ease of systems integration and customization
- Distributed processing and load balancing
- Remote command and control of telemetry features

All of these factors needed to be addressed in a reasonable manner without unduly straining the system to manage them. As the issues were debated, a common approach to each of these goals formed. The Resource Manager would serve as a distributed, unified, command-driven kernel and address all of these issues together.

RESOURCE MANAGER DESIGN

INTEGRATION AND CUSTOMIZATION

Originally, the driving force behind the Resource Manager was systems integration and customization. Both the 8385 and System 90 had encountered difficulties with customization of the standard software kernel. Interrelationships between software modules and packages could cause severe problems when enhancing a system with customer-specific modifications. Although every system was derived from the same kernel, each customer had unique and specialized requirements. These difficulties were especially evident in the area of User Interfaces. Menu interfaces which gathered input information were often integral components of the telemetry applications they were feeding. Since each customer had different preferences for User Interfaces, modifications often affected major areas of the telemetry sub-system. Faced with the onslaught of X-Windows in the new system, this was a major area of contention. The core functions of the telemetry system needed to be divorced from the User Interfaces. In this way, X-Window interfaces could be modified, or even discarded and written anew, without affecting the kernel system.

To accommodate these concerns, the O/S90 telemetry kernel system is designed to operate in a command-driven manner, with the Resource Manager serving as a command-driven "process broker". Telemetry kernel functions are standalone processes initiated by commands from User Interface applications, rather than being an integral component of the applications. The mechanism that serves as the commanding gateway between these two functional components is the Resource Manager. Figure 1 illustrates the basic data flow in the process of initiating a telemetry kernel function.

Figure 1. Basic Commanding Data Flow

The Resource Manager is initiated at bootup time and begins by reading an ASCII command vs. application file that has been created by the system integrator. This Resource Manager Configuration File uniquely maps each command in the system to an executable image or script file using the pathname of the application or script file. Each major telemetry function of the kernel system is represented in the Configuration File. Each Configuration File entry contains a unique ASCII command for the telemetry function, the pathname of the telemetry function executable or script file, and the target node on which the telemetry function is to be executed. This latter entry will be discussed in detail in the Distributed Processing section of this paper. A sample entry in the Configuration file is shown in Table 1.

Table 1. Resource Manager Configuration File

COMMAND = load_bitsync	# Unique command for telemetry # function.
APPLICATION = /usr/local/bin/bitsync_loader #	Pathname of telemetry function # executable.
NODE = workstation1	# Target node telemetry function # is to be executed on.

At this point, the Resource Manager enters a hibernate state, waiting for commands to arrive from User Interface applications to initiate the various telemetry kernel functions. The User Interface formulates its user input into arguments for the intended telemetry application. The subsequent data flow is detailed below and illustrated by Figure 2:

1. The appropriate command is sent to the Resource Manager in the form of a point-to-point TCP/IP communication, passing along the arguments for the telemetry application.

2. The Resource Manager, upon receipt of this command, looks up the command in its internal table to identify the associated telemetry application.

3. Using the POSIX "fork" and "exec" calls, the Resource Manager then spawns a child process which initiates the telemetry application.

4. When the child process terminates, a signal handler retrieves the exit status of the telemetry application and returns this status to the originating User Interface application.

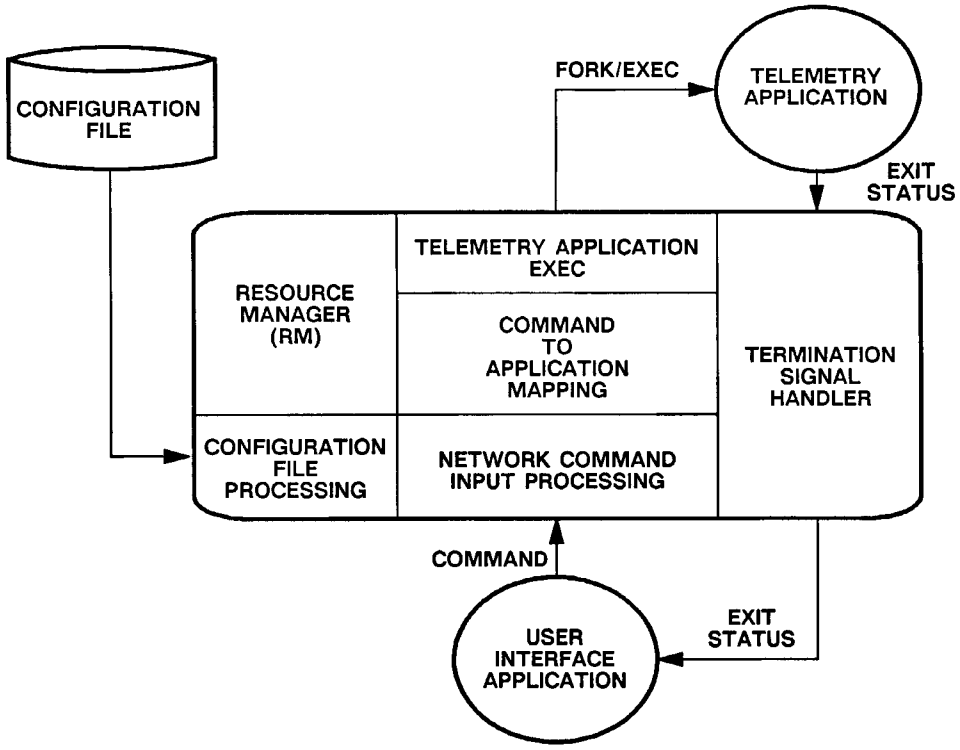

Figure 2. Resource Manager Data Flow

A subroutine library (RSC_SUBS) and a UNIX command-line interface (RSC_RUN) serve as the mechanism by which user interface programs and other processes communicate with the Resource Manager. Using this command-driven approach gives each telemetry function greater independence. One can separate the function of a process from the interface that acquires the inputs. A user interface can be completely replaced with a new interface without rippling through the telemetry kernel. Once the new interface is written (or the old one modified), the proper command is issued to the Resource Manager, and the telemetry function can be initiated without its being aware of the new interface.

For customization, if special processing is required for a given function, the standard kernel telemetry application need not be modified. By modifying the Configuration File to remap the existing command to a new script file or new application, new processing can be inserted as a separate application without affecting either the user interface or the existing telemetry kernel. This script file or new application can then initiate the original telemetry kernel application (via the associated Resource Manager command) after the specialized processing has finished (See Figure 3).

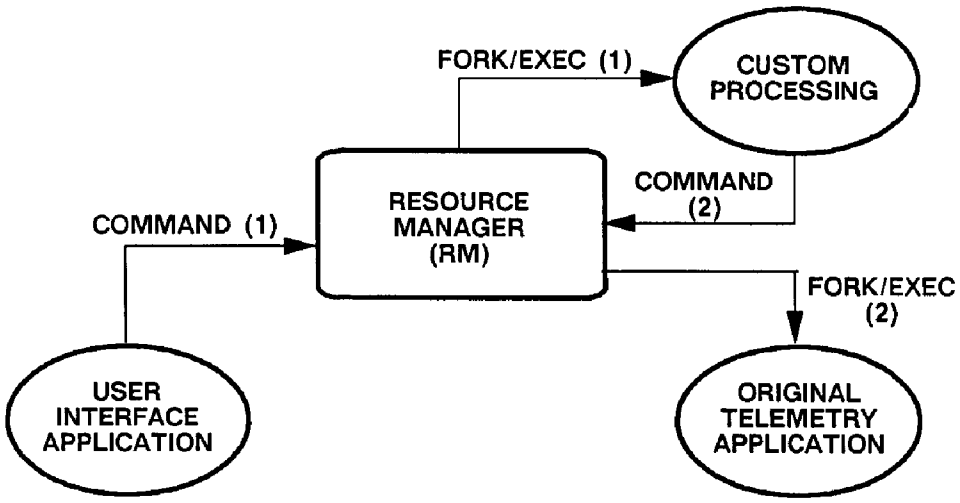

Figure 3. Remap Command to Insert Custom Processing

This functional independence need not apply strictly between the user interfaces and the core telemetry functions. Since the commanding interface to the Resource Manager is implemented via a subroutine library, this modularization can occur at any level in the telemetry kernel function. The farther down the functional hierarchy one carries modularity, the more autonomous each telemetry function becomes. For integration purposes, menu options can be remapped to new telemetry functions by changing the command-to-application mapping in the Configuration File. New software applications can be added to the telemetry kernel system as easily as editing the Configuration File and adding a new command and pathname. These applications need not be related to the

telemetry kernel system, such as third party report generation software or analysis packages. Given the greater autonomy of each software function, additions can be made with little effect on the standard telemetry core system.

LOAD BALANCING AND DISTRIBUTED PROCESSING

The basic command-driven design addresses system integration and customization, but the new Open Systems concepts present additional areas which need to be addressed. Primarily these issues are in the networking and distributed processing areas. Because of the proliferation of multi-workstation systems and the wide range of networking software such as NFS, TCP/IP, OSI and networked SQL databases, the issues of networking and distributed processing are essential to design of any new telemetry system. Today, networked SQL and off-the-shelf databases allow the database to be distributed across many workstations or servers, NFS allows transparent access from many nodes to networked files, and X-Windows allows networked access of graphics information. In a truly distributed system, the concept of a "single-server many- workstation" architecture becomes constrictive. The single server may become a bottleneck, rather than an offloading point. With increasing workstation horsepower, expanding networks, and advances in database and communications software, load distribution is a necessary capability of a telemetry system (See Figure 4). The telemetry system should be able to distribute traditional "server" functions across the spectrum of resources available in the system, and to reallocate them quickly and painlessly.

In order to utilize these features and allow flexibility of system architectures, the Resource Manager allows for distribution of each of the telemetry functions across any of the resources available on a system. No single node need be designated as a "server". On any given system "server" functions may be partitioned and allocated to any of the nodes in the system, dependent upon the resources available. This allows the system integrators to utilize features such as NFS and networked SQL, but does not demand dependency upon them. As an example, NFS will allow each workstation to mount a common disk and share the information on that disk without the applications software being aware of where that information physically resides. This can be extremely useful. However, a common error is to become dependent upon NFS. When this happens, networked files can be misused, and a network overload is induced.

The Resource Manager allows system integrators to optimize features such as NFS and networked SQL when applicable, and to distribute and isolate the same processing on other systems which may have different configurations. This success depends largely upon the knowledge and decisions of the system integrator. Offloading processes properly and allocating resources efficiently is often a judgment call, but the quickness and ease with which the system can be restructured using the Resource Manager allows for trial and error testing during system integration without jeopardizing schedules.

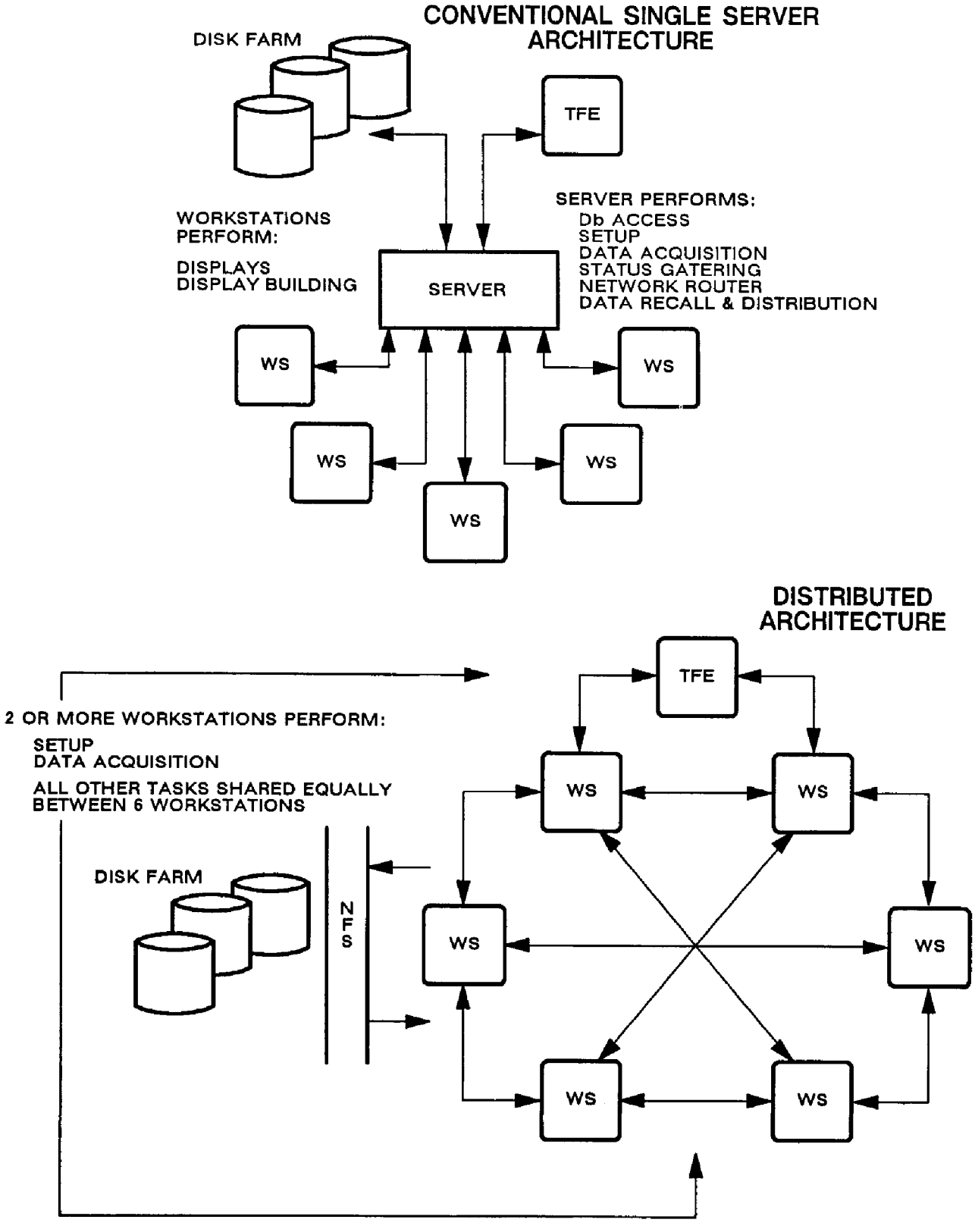

Figure 4. Single Server vs. Distributed Architecture

To support distributed processing, the Resource Manager's original design was enhanced and expanded. Rather than using UNIX-domain IPC as the command path, TCP/IP is utilized to provide the commanding interface. Also, rather than having one Resource Manager on a centralized server, each node in the telemetry system supports a resident Resource Manager. Since the Resource Manager spends the majority of its time hibernating, virtually no overhead is introduced by installing it on each workstation. Applications send commands to their local Resource Manager, who in turn forwards the commands to the Resource Manager on the target node as dictated by the Configuration File (See Figure 5.)

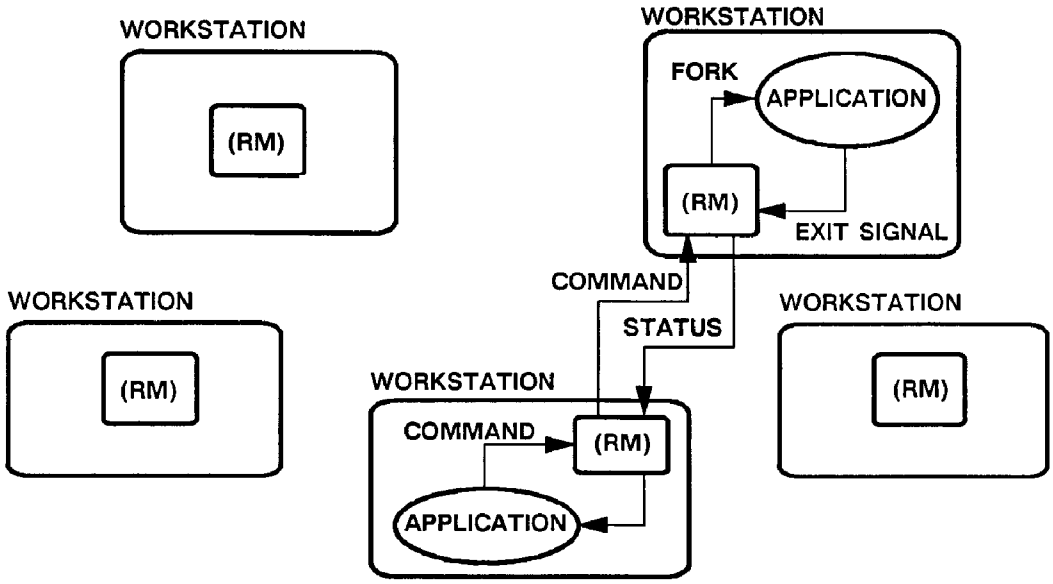

Figure 5. Forwarding Commands to Target Nodes

With this system in place, reconfiguring a system and testing for optimum load-balancing across the telemetry system becomes a relatively pain-free process. Each telemetry function can be reallocated by modifying the Resource Manager Configuration File using any ASCII text editor. Some restrictions apply to specific telemetry functions based on physical resource considerations. A program that loads an external telemetry device via a parallel interface, for example, needs direct access to the parallel interface, restricting the nodes on which it can execute. A data archiving process needs access to the archival disk and a physical interface to the incoming data. These instances are limited, however. By utilizing NFS effectively to allow access to common data files from various nodes, and by using networked SQL to access relational databases, the majority of telemetry functions can be allocated in a myriad of configurations. This integration process allows each system, although maintaining the same basic architecture and functionality, to be load-balanced in accordance with each customer's resources before it leaves the integration floor.

REMOTE PROGRAMMATIC CONTROL

The Resource Manager also allows remote control of the complete range of telemetry system functions. Remote control is needed because of:

- Access to systems via networks such as Internet
- Decreased efficiency of X-Window applications across long networks
- Batch-oriented data analysis at remote sites
- Real-time programmatic control by user software running on customer network backbones

Due to the increasing access of networks such as Internet, people who require access to and derive benefit from the telemetry system may not be in the immediate proximity of the telemetry system. This distance may hinder efficient use of an X-Window menu system, or may dictate remote login access restrictions. Such cases include analysts at remote sites who use data analysis report features of the telemetry system, but wish to work in a batch-oriented mode. These analysts may wish to automatically initiate the data analysis reports late at night during the minimum load hours of system use.

Another interesting trend is the incorporation of telemetry systems into larger customer systems as peripherals on the customer network backbone. In such arenas, customer written software running on the customer network backbone may command and manage the telemetry system functions under program control. Timed execution or programmed responses to data conditions may be the impetus for execution of various telemetry kernel functions. Examples of this would be dynamically loading the Telemetry Preprocessor to incorporate a new data stream at a certain point in a missile test, or turning off processing of a sensor that has become stuck in an out of limits state. While these operations may be accomplished by an operator at a workstation, it may be more desirable to initiate the actions under program control.

One of the factors that allows the O/S90 to accommodate these requirements is in the architecture of the system software itself. As mentioned before, the link between user interface software and telemetry kernel software is the Resource Manager. This independence allows each of the core telemetry kernel functions to be executed without a user interface. Taking this a step further, since each of these functions maps to a unique command in the Configuration File, any program can initiate them via the Resource Manager, not just the User Interface processes. Any software task that wishes to initiate a

telemetry kernel function without the User Interface can use the subroutine library (RSC_SUBS) interface to the Resource Manager to access the kernel function directly.

For software not running on an O/S90 telemetry system node, RSC_SUBS allows for control of telemetry functions without requiring a resident Resource Manager. By explicitly specifying the target node of the requested application when using the RSC_SUBS library, the Configuration File node specification can be overridden. When this feature is invoked, a direct connection is made to the Resource Manager on the target node from the calling program, bypassing the need for a Resource Manager on the local node. The command is then sent, and the process invoked by the Resource Manager on the target node. The decision to make all communication with the Resource Manager network-domain communication rather than use UNIX-domain IPC for local connections was based on this functionality. Since all communication is network- domain TCP/IP, a Resource Manager process does not know nor care whether this request originated locally or from a network source. The processing path remains unchanged.

The command line interface (RSC_RUN), which utilizes RSC_SUBS, allows analysts at remote sites to initiate O/S90 tasks from UNIX command lines or script files in the same manner. These interfaces give analysts and software complete control of the telemetry kernel system without interactive or remote login constraints (See Figure 6). While the main purpose of this capability is for non-interactive control of the system, programmatic control of X-Window applications is also available. This may be useful when an existing processing system would like to include the O/S90 menu system, or parts of the O/S90 menu system, as an option off of the existing system's main menu. If a customer has an existing data processing system based on X-Window interfaces, the RSC_SUBS interface can be used to hook O/S90 menus into the existing system menus using subroutine calls or script file entries.

CONCLUSION

With the influx of new networking technologies creating more diverse systems scenarios and integration schemes, problems associated with these areas are destined to grow. The Resource Manager gives telemetry systems the ability to cope with these areas in a coherent manner, while maintaining maximum flexibility in resource utilization and allocation. Standard telemetry functions can be controlled and integrated easily, interfaces can be expanded, and special processing can be introduced throughout the chain with a minimum impact on standard system software. Networking capabilities allow each system to be load-balanced according to its processing resources without locking the standard system into any one resource allocation architecture. Finally, analysts can initiate batch oriented processing at their discretion, or manipulate the telemetry system as a peripheral tool from their own networks and software, allowing real-time processing to be controlled and modified based on exterior data analysis and processing subsystems.

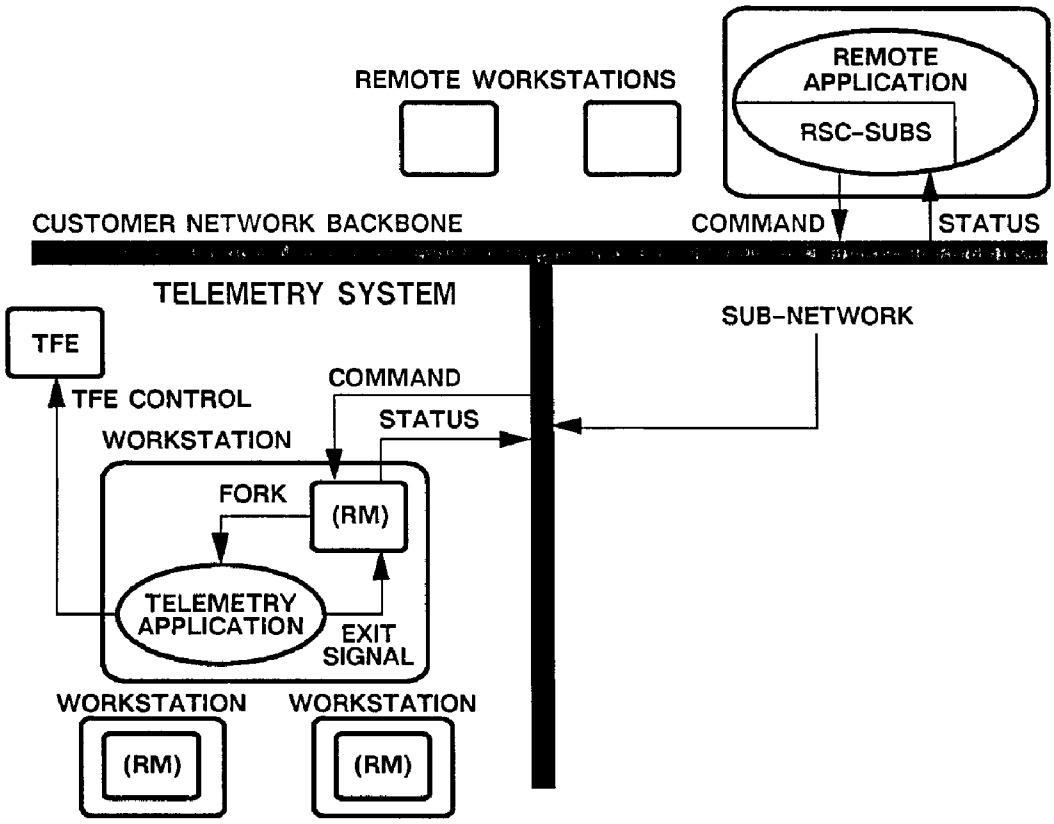

Figure 6. Remote Commanding of Telemetry System