N88-16390

A Multiprocessing Architecture for Real-Time Monitoring

Thomas J. Laffey
James L. Schmidt
Jackson Y. Read
Simon M. Kao

Lockheed Artificial Intelligence Center
2710 Sand Hill Road
Menlo Park, CA 94025
(415)-354-5209

Abstract

This paper describes a multiprocessing architecture environment for performing real-time monitoring and analysis using knowledge-based problem solving techniques. To handle asynchronous inputs and perform in real time, the system consists of three or more separate processes which run concurrently on one or more processors and communicate via a message passing scheme. The Data Management Process gathers, compresses, scales and sends the incoming telemetry data to other tasks. The Inference Process consists of a proprietary high performance inference engine that runs at 1000 rules per second using telemetry data to perform a real-time analysis on the state and health of the Space Telescope. The I/O Process receives telemetry monitors from the Data Management Process and status messages from the Inference Process, updates its graphical displays in real time, and acts as the interface to the console operator. The operator sees a hierarchy of displays which can be traversed using a mouse, and on which the user can display graphs of the monitors. The multiprocessing architecture has been interfaced to a simulator and is able to process the incoming telemetry in "real-time" (i.e., several hundred telemetry monitors per second). In this paper we will also describe why commercial knowledge-based building tools are not well suited for real-time domains, thus forcing us to develop our own proprietary shell. The system has been applied to the real-time monitoring of telemetry data from the NASA Hubble Space Telescope (HST) and the application will be described in another paper at this conference.

Introduction

As the application of knowledge-based systems evolves from an art to an engineering discipline, we can expect more challenging applications to be addressed. Some of the most challenging and interesting environments are found in real-time domains.

A knowledge-based system operating in a real-time situation (e.g., satellite telemetry monitoring) will typically need to respond to a changing task environment involving an asynchronous flow of events and dynamically changing requirements with limitations on time, hardware, and other resources. A flexible software architecture is required to provide the necessary reasoning on rapidly changing data within strict time requirements while accommodating temporal reasoning, non-monotonicity, interrupt handling, and methods for handling noisy input data.

The Problem

Like other existing satellites, the NASA Hubble Space Telescope (HST) has not been designed to to be an autonomous spacecraft. Its engineering telemetry will be monitored for vehicle health and safety 24 hours a day by three shifts of operators in the ST Operations Control Center (STOCC) at the NASA/Goddard Space Flight Center in Greenbelt, Maryland.

Six operator workstations (four to monitor the major subsystems and two for command and supervision) will be used to monitor the incoming telemetry data. Each workstation consists of two color CRTs which display numeric values, updated in real time.

- On one CRT the operator can bring up a page of formatted telemetry data (where a page consists of about 50 different monitor mnemonics and its associated value) or a page consisting of a chronological history of events that have occurred (e.g., a monitor out of limits)
- The other CRT is a slave to any other console and can be used to display what is being shown at another workstation

For the HST there are 4,690 different telemetry monitors in 11 different formats available for interpretation. In normal operating mode, each monitor is sampled at least once every two minutes, with some being sampled many times during that interval. The telemetry format may be changed manually by ground operations or autonomously by the HST under certain situations. The telemetry data is subject to a variety of problems including loss of signal and noise in the transmission channel.

As in any large system, the job of the console operator is difficult because of the complexity of the HST and because it is hard to determine the exact state of the satellite at any time due to the massive amounts of data arriving at such short intervals and the ever present possibility of non-nominal behavior.

Why Commercial Tools are Inadequate for Real-Time Monitoring

Real-time domains present complex, dynamic problems because of their dependence on the time factor. A real-time expert system must satisfy demands that do not exist in conventional domains. Current shells are not generally appropriate for real-time applications for the following reasons:

1. The shells are not fast enough
2. The shells have few or no capabilities for temporal reasoning
3. The shells are difficult to integrate *in an efficient manner* with conventional software
4. The shells have few or no facilities for focusing attention on important events
5. The shells offer no integration with a real-time clock
6. The shells have no facilities for handling asynchronous inputs
7. The shells have no way of handling software/hardware interrupts
8. The shells cannot efficiently take inputs from external stimuli other than a human
9. The shells cannot guarantee response times
10. The shells are not built to run continuously

We next describe a monitoring system called *L*STAR* (for Lockheed Satellite Telemetry Analysis in Real Time) being built to aid the HST console operator in performing the real-time monitoring and analysis of telemetry data from the HST. *L*STAR* runs on a DEC VAXStation II/GPX running under VMS and uses data produced by the BASS Telemetry System at the HST Hardware/Software Integration Facility (HSIF) in Sunnyvale, California.

Solution Method

Three separate processes are used for the real-time analysis of rapidly changing satellite telemetry data. Each of the processes operates independently and communicates information via message passing. (NOTE: We use the terms *process* and *task* interchangeably in this paper.) The different processes are shown in Figure 1:

- INFERENCE PROCESS — used to analyze the dynamic data by means of forward or backward chaining rules
- DATA MANAGEMENT PROCESS — used to gather, scale and compress the incoming telemetry data

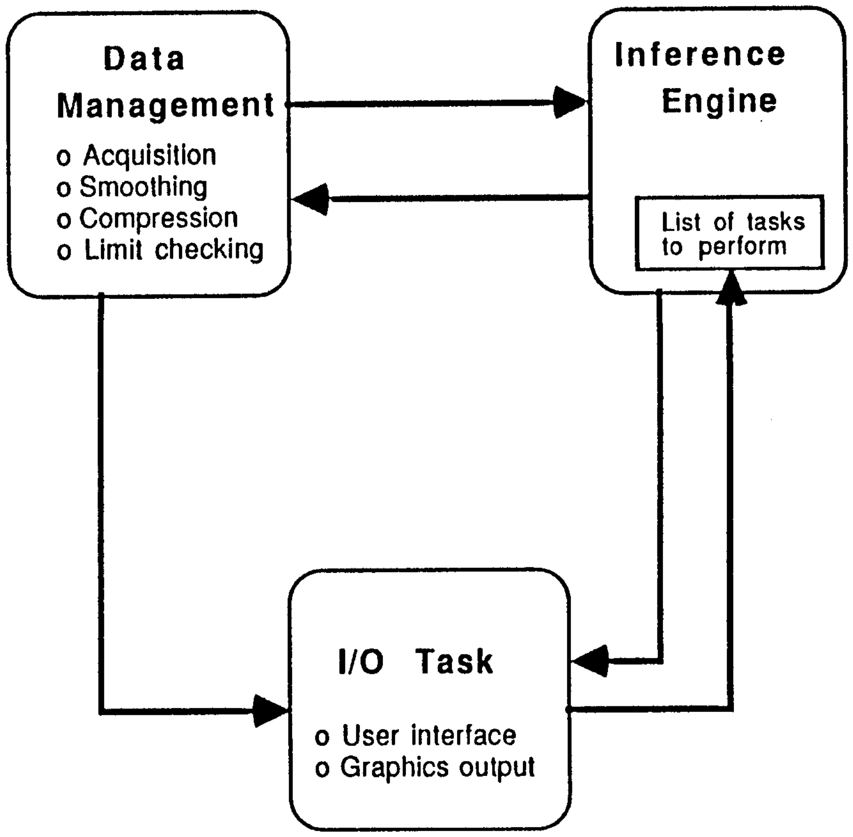

Figure 1: Software Architecture

- I/O PROCESS — used to provide an interface (including real-time graphics) to the operator

Having three independent tasks allows us to distribute the system across different processors. When all three tasks are operating on one processor, the timesharing facilities of the operating system take care of scheduling when each is run.

If all the tasks were done in one process, i.e. sequentially, the Inference Process could not be reasoning with existing data while the Data Management Process was getting new data, or while the I/O Process was performing screen output. The main purpose of having separate tasks was to give the Inference Process complete freedom from input/output worries and let its limiting factor be the processing power of the CPU on which it is resident.

Mailboxes are used for the message passing between tasks. They are used as fast, one-way, in-memory channels for communication of data. Using this mechanism, the Inference Process is only slowed by having to read or write to the mailbox.

A typical scenario follows: The Inference Process examines its knowledge base and sends a set of messages to the Data Management Process indicating which telemetry monitors it needs to perform its analysis. It also sends messages containing other information the Data Management Process needs to know about each telemetry monitor such as the sampling rate, whether it should be smoothed, the scaling factor, alternate names, and to which telemetry set it belongs.

Incoming telemetry data streams are captured from the flight hardware and after initial preprocessing of the raw data by ground computers are fed to the Data Management Process. After some scaling and data compression, this process sends the data of interest to the Inference and I/O processes. The Inference Process can ascertain, using its knowledge base, if the data correspond to nominal vehicle behavior. The I/O Process consists of a data flow diagram of the flight system software and magnitude vs. time plots of the telemetry data. The plots, which are strip charts updated in real-time using data from the Data Management Process, can appear by using mouse clicks when the cursor is over appropriate parts of the diagram. Should the HST change state or non-nominal behavior be detected, messages will be sent from the Inference Process to the I/O Process and subsequently displayed.

The knowledge in this real-time monitoring system is contained within the rules and frames which make up the knowledge base. The knowledge base, used primarily by the Inference Process, contains the critical telemetry items to be monitored and rules to infer the current state and health of the HST. Of the over 4,000 different telemetry monitors, only a small number (about 10%) are used by the operators to determine spacecraft behavior. If, however, non-nominal behavior is detected, other telemetry monitors might be used to diagnose the problem. This would be done by having a rule fire which causes a message to be sent from the Inference Process to the Data Management Process, indicating which new set of telemetry monitors it needs to know about. Rules can also send messages changing the sampling rate of telemetry at which it is already looking.

Discussion

As more and more complex vehicles are put into orbit, it becomes essential that sophisticated methods for evaluating the health of and diagnosing problems within these vehicles be developed. Real-time knowledge-based systems offer promise as an excellent means of dispersing such information. During development, testing is an important part of the cycle. In doing such testing, system designers have to be able to monitor and diagnose the telemetry streams. This kind of expertise should not have to be independently learned by the vehicle operators after launch, when the developers are out of the picture. Thus, for a system such as the one described in this paper, having the expertise saved for the operations aspect is an invaluable step.