Intelligent Real-Time Monitoring*

T. Laffey, S. Weitzenkamp, J. Read, S. Kao, J. Schmidt

Lockheed Artificial Intelligence Center
2710 Sand Hill Road
Menlo Park, CA 94025
(415)-354-5208

Abstract

This paper describes a multi-tasking architecture 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 distributed processes which run concurrently and communicate via a message passing scheme. The Data Management Process acquires, compresses, and routes the incoming sensor data to other processes. The Inference Process consists of a high performance inference engine that performs a real-time analysis on the state and health of the physical system. The I/O Process receives sensor data from the Data Management Process and status messages and recommendations from the Inference Process, updates its graphical displays in real time, and acts as the interface to the console operator. The distributed architecture has been interfaced to an actual spacecraft (NASA's Hubble Space Telescope) and is able to process the incoming telemetry in "real-time" (i.e., several hundred data changes per second).

1 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.

Before going any further we should define precisely what we mean by the term *real-time*. O'Reilly and Cromarty [2] give a detailed discussion on the meaning of real-time and offer a formal definition: "There is a strict time limit by which the system must have produced a response, regardless of the algorithm employed".

We find it useful to categorize tasks of real-time systems into hard and soft real-time as discussed by Stankovic and Zhao [3]. We define a *hard real-time* task as one for which the correctness of the system depends not only on the result of computation, but also on the time at which the results are produced. Furthermore, if these strict timing constraints are not met, there may potentially be disastrous consequences. For such tasks, it is necessary to guarantee that timing constraints are met. In contrast, while soft real-time tasks have timing constraints, there may still be some value for completing the task after its deadline, and disastrous consequences do not result if these tasks miss their deadline. Many applications have both hard and soft real-time requirements. To meet all such deadlines in a system requires sophisticated scheduling algorithms and careful implementation. We will not discuss this topic since it is beyond the scope of this paper.

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. Laffey et. al. [1] give a detailed discussion on the state-of-the-art in using knowledge-based techniques for real-time problems.

2 Problem: Real-Time Telemetry Analysis

Lockheed Missiles and Space Company (LMSC) is the prime contractor for the Support Systems Module (SSM) and Integration Systems Engineering for NASA's Edwin P. Hubble Space Telescope (HST). LMSC has assembled the basic spacecraft structure and integrated the optics and scientific instruments made by contractors from around the world. Additionally, LMSC is the HST Mission Operations Contractor, responsible for the safe and efficient operations of the vehicle.

The telescope, whose electronic sensors could detect a flashlight beam directed at the Earth from the moon, will orbit three hundred miles above the earth's surface after its launch in 1989. The HST will let astronomers peer, unimpeded by the atmosphere, at the edges of the known universe, 14 billion light years away (compared with two billion light years for the best earth-based telescopes).

The HST is a complex, state-of-the-art satellite, a precursor to satellites of the future that will have sensitive missions with precise guidance requirements. With the increasing complexity of the satellites being sent into orbit, it has become clear that a substantial amount of sophisticated expertise is needed at the various ground stations.

Like other existing satellites, the 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. The spacecraft operations will take place in the ST Operations Con-

*This work was supported under Lockheed Independent Research and Development funds

trol 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 close to 5,000 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, noise in the transmission channel, or misconfiguration of the system.

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. A system that would make the monitoring task easier might be one with a better, more visually oriented interface for the operator to monitor; one with some preprocessing of the data to screen out less important information, and one that knew what trends and combinations of data meant non-nominal spacecraft behavior. This paper describes the development of such a system.

3 Why Current Tools are Inadequate

Real-time domains present complex, dynamic problems because of the occurrence of asynchronous events and demanding timing constraints. A real-time expert system must satisfy demands that do not exist in conventional domains. Current shells do not generally offer support 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 events/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
11. The shells lack features to support multi-tasking (e.g., signals and semaphores)
12. Methods do not exist for verifying and validating the shells and the knowledge bases they execute

In the rest of this paper, we describe a monitoring system called *L*STAR* (for Lockheed Satellite Telemetry Analysis in Real Time). It is being built to aid the HST console operator in performing the real-time monitoring, checkout, and analysis of telemetry data from the HST.

4 A Distributed Architecture for Monitoring

*L*STAR* consists of a set of distributed processes which are used in performing real-time analysis of rapidly changing satellite telemetry data. Each of the processes operates independently and communicates information via message passing. The different processes are shown in Figure 1:

- INFERENCE PROCESS — used to analyze the dynamic data by means of frames and time-triggered, forward-chaining, and backward chaining rules
- DATA MANAGEMENT PROCESS — used to gather, scale, compress, and route the incoming telemetry data to the appropriate processes
- I/O PROCESS — used to provide an operator interface (consisting of a hierarchy of schematics with real-time plots)

By having these three independent processes, we can exploit the inherent asynchrony in the overall system to maximize throughput and response. We further gain the advantage of being able to use multiple CPUs if performance requirements call for it. For a typical application, many data inference and I/O modules may exist and be distributed across different processors. For the HST application, analysis and display modules exist for the electrical power system, pointing control system, and others. The various modules pass data/messages to one another over Ethernet.

In the current implementation, there is only a single layer of communicating processes. In the future, we may wish to have a series of ranks or layers, organizing the processes into a lattice of parallel processes.

In the paragraphs which follow, we describe a typical scenario: At initialization, the various Inference Processes examine their knowledge bases and send a set of messages to the Data Management Process indicating which telemetry monitors they need to perform their analysis. They also send messages indicating other information the Data Management Process needs to know about each telemetry monitor such as:

- to which datasets it belongs,
- how often to send it,

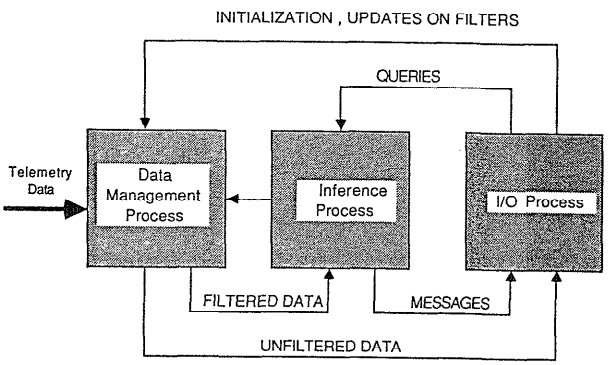

Figure 1: *L*STAR* Process Structure

- whether it should be smoothed,
- aperture setting (the minimum amount it must change before it is reported),
- who to send it to, and
- alternate names.

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, it sends the data *of interest* to the Inference and I/O processes. The Inference Processes infer, using their knowledge base, if the data corresponds to nominal vehicle behavior. These messages are then sent to the I/O Process. The I/O Process consists of interactive displays consisting of schematics from the electrical power system, pointing control system, and the flight software with special windows for interaction with the Inference Process.

In the sections which follow we describe each of the different types of processes in detail.

4.1 Data Management Process

The Data Management Process (DMP) is used to acquire, convert, and compress the incoming telemetry data, and selectively send it to the other modules. It also calculates new measurands from the original telemetry. At initialization, the DMP receives messages from the Inference and I/O processes indicating which datasets they are interested in. As events unfold during the analysis, the Inference Process may command that a new dataset be sent to it or to an I/O Process. An operator may also manually intervene and request a change in datasets. As the system is running, an Inference Process can send a message to the DMP and change characteristics of the data it is analyzing.

For the Space Telescope application, there is a single Data Management Process sending data to multiple I/O and Inference Processes on different processors. All data sent is tagged with the spacecraft's time.

4.2 Inference Process

The Inference Process analyzes the dynamic data by means of frames, rules, and statistical procedures. Rules and procedures can be tested/invoked in three different manners:

1. by a test clock at fixed time intervals (*temporally-driven*),
2. when specified data changes (*data-driven*), and
3. when needed to achieve a goal (*goal-driven*).

The Inference Process performs mission monitoring, anomaly detection, anomaly resolution, and command validation. We now show an actual *L*STAR* rule used in the HST application to check the status of the RGA System:

```
RULE     : "RGA not in high mode"
CONTEXT  : { Maneuver };
PRIORITY : 100;

IF   (decreasing( [value\monitor\QDSTDCP],
                  10 seconds) )
 and ([value\monitor\QDSTDCP] > 0.000043)
 and ([value\monitor\QDFHILO] = 1)
 and ([status\system\RGA] <> abnormal)
THEN [status\system\RGA] := abnormal;
     send(IO, ALERT, "RGA", "RGA not in high
                             mode");
```

The SEND function in the second THEN clause results in a message being sent to the I/O Process which indicates there is an alert for object **RGA**.

Note that the first IF clause checks if the trend of monitor QDSTDCP is *decreasing* over the last 10 seconds. All data in *L*STAR*, either input from the DMP or inferred from the Inference Process, is archived and time-tagged into a ring buffer. The ring buffer consists of a compressed format which keeps track of the last time the datum was updated and each time it changed over a user-specified time period. From the compressed format, the entire signal can be reconstructed, if necessary.

Maintaining a history of selected sensor data allows *L*STAR* to reason about both historical and current data. A number of primitive functions have been written to use this buffered data to calculate trends, correlations, average values, minimums, maximums and standard deviations over varying time periods. Functions also exist to compare current data to historic data (e.g., "if the current value is greater than the value five minutes ago").

It should be noted that not all the rules are continually checked. Some of the rules are triggered by the test clock at regular time intervals. Other rules are checked only when data changes that is used in one of its IF clauses, or when they are needed to achieve a goal. This allows a

single Inference Process to analyze several hundred data changes each second.

Below, we show the actual *L*STAR* syntax of such a temporally driven rule (taken from the electrical power system for the HST). The **TEST INTERVAL** slot specifies that this rule should be tested every 10 seconds.

```
RULE          : "Recharge ratio warning"
CONTEXT       : { eclipse };
PRIORITY      : 1000;
TEST INTERVAL : 10 seconds;

IF   ([value\sensor\csfrrat1] <  0)
THEN [recharge_ratio\battery\b1] := failed;
     send(IO, INFO, "Battery 1 recharge ratio
                     has failed lower limit");
```

Note that the **SEND** function in the conclusion of the rule is used to send an INFO message to the I/O Process.

In order to achieve maximum efficiency, rules are compiled into a efficient intermediate postfix format (and then optionally into C) which does not require any pattern matching to occur while the system is running. All variables used in rules are resolved at compile time by a preprocessor which generate multiple rules from a single rule depending on how many objects the variable binds to. Multiple variables in a single rule can result in a combinatoric increase in the number of rules which are generated. Although this seems theoretically poor, it works quite well in practice. The restriction this puts on the developer, is that any object created during runtime cannot be referenced from a rule with a variable We have found that for real-time monitoring applications, we have not needed this capability. Such may not be the case in a planning or scheduling application where many objects are dynamically created and deleted.

Performance has been further increased by allowing the developer to control how much and which information is collected via selective event recording. During runtime, *L*STAR* has the capability for rules to dynamically turn on/off event recordings such as data assertions and retractions, rule firings, and procedure calls.

Finally, a compact run-time version of the Inference Process was developed which does not carry all the "excess baggage" of the development version. Many of the checking and debugging aids used during development are excluded resulting in increased performance. (Although this practice is certainly not novel, it has seldom been followed by AI systems). The overall result of all these speedup techniques is a system which runs close to 1,000 rules per second on a VAX 8650.

*L*STAR* has the ability to partition the ruleset using a context mechanism and to focus its resources on specified sets of incoming sensor data. Both these abilities free the Inference Process from examining extraneous rules and data that are not relevant to its current task. The **CONTEXT** mechanism increases the speed of the Inference Process by partitioning the ruleset into smaller sets which are valid only in certain contexts. This way, the Inference Process does not always have to examine the entire ruleset, thus saving time. Datasets provide a similar mechanism for speedup. The inference engine does not necessarily need to analyze all the incoming sensor data. Only when some "significant" event has been detected might *L*STAR* look at other input data or change the characteristics (e.g., rate) of the data it is currently using. This important capability is shown in the following *L*STAR* rule:

```
RULE     : "Change to Science mode";
CONTEXT  : { Inertial_Hold,
             FGS_Acq,
             FHST_Acq };
PRIORITY : 100;

IF   ([value\sensor\QSITAKE] = 1)
THEN [context\control\ie] := Science;
     send(DMP, DATASET_CHANGE, "Science");
```

This rule would only be examined if the current context were one of the three listed (i.e., *Inertial_Hold, FGS_Acq*, or *FHST_Acq*). If this rule is executed, the context is changed via the first THEN clause and the inference engine would look at only rules with *SCIENCE* as their context. Additionally, a message would be sent to the Data Management Process to command it to change the dataset to a predefined set containing only science monitors.

4.3 I/O Process

The I/O Process displays the information pertinent to the monitoring task. The objective is to provide visual feedback to focus the attention of the console operator on possible problems and areas of interest for the Space Telescope. The current version of this interface consists of a large number of drawings and schematics of the HST and its related telemetry monitors. There is a hierarchical tree of displays which the user may traverse using a mouse. Each node in the tree corresponds to a system or subsystem of the HST. All nodes contain a schematic of their system/subsystem and an inference process interaction window. The schematic can contain either *permanent* graphs (i.e. part of the schematic itself) or pop-up graphs brought up by mousing pickable items.

The Inference Process interaction window consists of display windows for messages (i.e., **information, alert, warning, cancel_alert, cancel_warning**) and various message queue interaction icons. A typical scenario of interaction between an Inference Process and an I/O Process might involve an ALERT message being sent from the Inference Process to the I/O Process. The I/O Process would then put this message in a priority queue for pending alert messages and inform the operator that a new message has been received. The operator can then display the message by mousing the ALERT icon. The *GOTO* option allows the operator to automatically be sent to the correct display and bring up and highlight the subsystem and any graph(s) pertaining to the ALERT message. After the operator has acted upon the new message it is removed from the pending queue and put in the acknowledged queue where it can be recalled if needed. All messages are time-tagged

and facilities exist which allow the operator to scroll back through the messages as desired.

5 Discussion

The actual utility of the *L*STAR* architecture has been shown through its use since March, 1988 at the HST Test Control Center in Sunnyvale, California. It is being used in the monitoring, checkout, and analysis of telemetry data from the Pointing Control System and Flight Software of the Space Telescope. The actual flow of data from the spacecraft into *L*STAR* is shown in Figure 2. The system runs on a network of DEC VAX computers connected via ethernet, with the I/O Process residing on microVAX II/GPX color workstation. The monitoring and checkout system comfortably handles the *200 data changes which occur each second.* A larger version of the system is currently being developed for the Space Telescope Operations Control Center at NASA Goddard in Greenbelt, Maryland.

The current system can be described as a soft real-time system. It runs under the "illusion" of being *fast enough* to handle any combination of incoming data values. However, we cannot currently guarantee that it could handle a series of catastrophic events. Many of the difficult issues such as guaranteed response times and what to do if a system cannot meet its timing constraints are targets of future research.

6 Acknowledgements

The authors wish to acknowledge the encouragement and support received from Larry Dunham, Joe Rickers, and Wally Whittier.

References

[1] T.J. Laffey, P.A. Cox, J.Y. Read, S.M. Kao, and J.L. Schmidt. Real-time knowledge-based systems. *The AI Magazine*, 9(1):27–45, Spring 1988.

[2] C. A. O'Reilly and A. S. Cromarty. 'Fast' is not 'Real-time' in designing effective real-time AI systems. In *Proceedings of SPIE International Society of Optical Engineering*, pages 249–257, 1985.

[3] J.A. Stankovic and W. Zhao. On real-time transactions. *SIGMOD RECORD*, 17(1):4–18, March 1988.

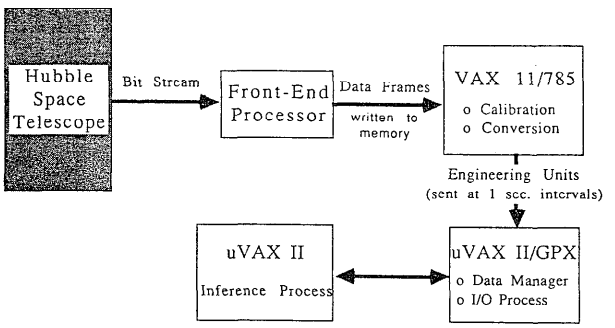

Figure 2: *L*STAR* Implementation at Test Control Center