MARVEL: A Distributed Real-Time Monitoring and Analysis Application

Ursula M. Schwuttke, Alan G. Quan, Robert Angelino, Cynthia L. Childs, John R. Veregge, Raymond Y. Yeung, and Monica B. Rivera, Jet Propulsion Laboratory, California Institute of Technology

Real-time AI is gaining increasing attention for applications in which conventional software methods are unable to meet technology needs. One such application area is the monitoring and analysis of complex systems. MARVEL (multimission automation for real-time verification of spacecraft engineering link), a distributed monitoring and analysis tool with multiple expert systems, was developed and successfully applied to the automation of interplanetary spacecraft operations at the Jet Propulsion Laboratory (JPL) of the National Aeronautics and Space Administration (NASA). In this chapter, we describe MARVEL implementation and validation approaches, the MARVEL architecture, and the specific benefits that were realized by using MARVEL in operations.

MARVEL is an automated system for telemetry monitoring and analysis. It has been actively used for mission operations since 1989. It was first deployed for the *Voyager* spacecraft's encounter with Neptune and has remained under incremental development since this time, with new deliveries occurring every 6 to 10 months. MARVEL combines stan-

Figure 1. The JPL Mission Operations Environment.

dard automation techniques with embedded rule-based expert systems to simultaneously provide real-time monitoring of data from multiple spacecraft subsystems, real-time analysis of anomaly conditions, and both real-time and non–real-time productivity enhancement functions. The primary goal of MARVEL is to combine conventional automation and knowledge-based techniques to provide improved accuracy and efficiency by reducing the need for constant availability of human expertise. A second goal is to demonstrate the benefit that can be realized from incorporating AI techniques into complex real-time applications.

The traditional spacecraft operations environment at JPL, shown in figure 1, has not relied heavily on automation because until fairly recently, software technology was insufficient for meeting many of the complex needs of this application. The traditional approach has involved large teams of highly trained specialists and support personnel for each spacecraft subsystem and each mission. (Each JPL spacecraft has seven subsystems: attitude and articulation control, command and data, power, propulsion, telecommunications, thermal control, and science instruments.) There have been separate teams for uplink (commanding), downlink (monitoring and analysis), and science activities. The downlink teams for the individual spacecraft subsystems include both real-time personnel and non–real-time personnel who are responsible for routine telemetry monitoring and more detailed analysis, re-

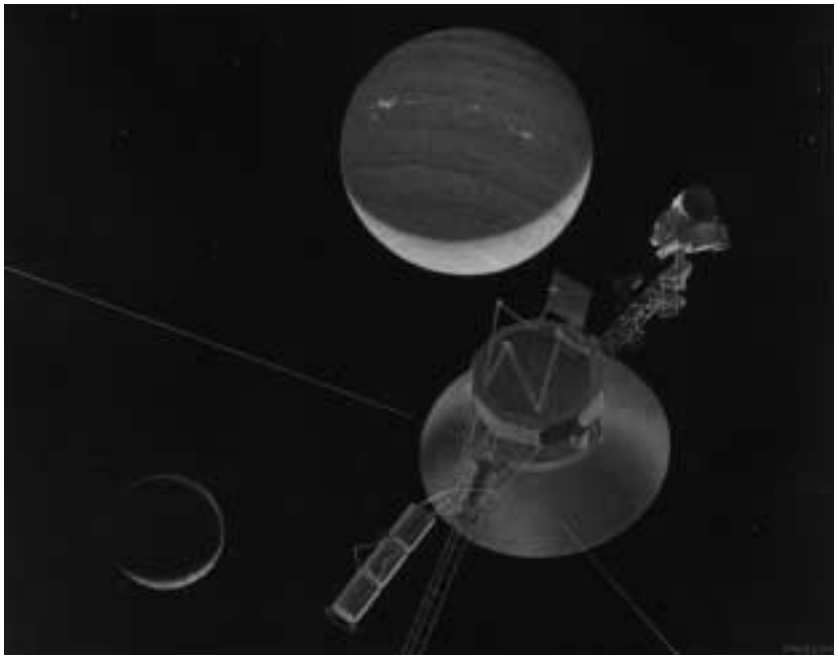

Figure 2. Voyager *at Neptune, with the Moon Triton in the Background.*

spectively. When system-level analysis is required to handle events that affect more than one spacecraft subsystem, a separate set of individuals coordinates the efforts of the relevant subsystem analysts. The total operations staff for the two *Voyager* spacecraft during peak activity periods (such as planetary encounters) consisted of over 100 individuals. This traditional approach was used successfully for the *Voyager* mission, resulting in enormous volumes of scientific data from brief flyby encounters with Saturn, Jupiter, Uranus, and Neptune. The success of *Voyager* has helped to enable new orbital missions to other planets, such as *Magellan* to Venus, *Galileo* to Jupiter, and *Cassini* to Saturn. Figures 2 and 3 show artist renditions of *Voyager* at Neptune and *Galileo* at Jupiter.

Despite the past successes, the increasing number and complexity of missions cause this operations approach to become less feasible for two reasons. First, the work force costs for supporting this style of operations for multiple simultaneous missions are too great to be sustained by current NASA budgets. Second, with the exception of *Voyager*, missions will be returning significantly higher volumes of engineering and science data on a more continuous basis than in the past.

MARVEL provides user interface functions, data access, data manipulation, data display, and data archiving within an X WINDOWS–MOTIF envi-

Figure 3. Galileo Orbiter and Probe at Jupiter, with the Great Red Spot at Upper Right.

ronment. The detailed expertise for anomaly analysis is supplied by embedded knowledge-based systems. In the event of anomalies, the appropriate knowledge bases provide an analysis and recommendations for corrective action. Conventional processing is implemented in C functions. The knowledge bases are embedded within the C program and are implemented in data-driven and goal-driven rules using a commercial expert system shell. The shell is written in C, which allows easy integration with the conventional code. MARVEL makes it possible for an analyst to effectively handle significantly more demanding real-time situations than in the past because it automatically performs numerous tasks that previously required human effort. As a result of MARVEL, it has become possible for individual analysts to be responsible for several spacecraft subsystems during periods of low and moderate spacecraft activity. Thus MARVEL reduces both the level of training and the cognitive load that are required to perform routine mission operations.

MARVEL hasdemonstrated that the use of automation enhances mission operations. Individual spacecraft analysts are no longer burdened with routine monitoring, information gathering, or preliminary analysis functions. They are able to view the results of the automation of these activities on displays associated with individual spacecraft subsystems at the click of a mouse button. This approach resulted in a reduced need for staffing, less work force dedicated to routine tasks, ear-

lier anomaly detection and diagnosis, leverage of scarce and valuable expertise, and reduced impact from personnel turnover. As a result, a MARVEL system for the *Galileo* mission (to Jupiter) is now under way, and MARVEL for the *Cassini* mission (to Saturn) is being considered.

Achieving Real-Time Performance

Fast systems are not necessarily real-time systems; however, in many applications, fast response time can be essential for meeting real-time constraints. *Real-time systems* have been defined as systems that have the "ability to guarantee a response after a (domain-defined) fixed time has elapsed" (Laffey, et al., 1988, p. 27) or that are "designed to operate with a well-defined measure of reactivity" (Georgeff and Ingrand 1989, p. 209). In other words, real-time systems must be able to reliably and predictably process data at rates as fast or faster than they are arriving. According to these definitions, knowledge-based systems have not yet been sufficiently demonstrated for complex real-time applications because in such applications, the amount of computation is nondeterministic, even in the presence of constant input data rates. This limitation is already being recognized as the primary limitation of AI systems, making it difficult to apply AI approaches where they might otherwise prove useful.

Although future approaches might make it possible for intelligent systems to adapt more flexibly and dynamically to real-time situations (Horvitz, Cooper, and Heckerman 1989; Hayes-Roth 1990; Schwuttke and Gasser 1992) without becoming overloaded, it is unlikely that any single new method will be able to handle all real-time situations. However, judicious use of existing AI methods can make it possible to obtain improved performance, both in current systems and in more dynamic systems of the future. The following paragraphs describe some of the methods used in MARVEL that enable knowledge-based techniques to enhance the capabilities of a real-time system without causing a negative impact on performance.

Knowledge-Based Methods Used Only Where Essential

For certain functions, such as diagnostics and anomaly correction, expert systems provide better implementational paradigms than more efficient conventional approaches. However, expert systems usually use interpreters to perform inferencing on the knowledge base rather than compile the knowledge base into native code. This approach tends to compromise performance and can pose difficulties in applications where the fastest possible response time is a critical factor in meeting real-time constraints (Barachini and Theuretzbacher 1988; Bahr 1990).

MARVEL achieves adequate response time by placing as much of the computing burden as possible into conventional algorithmic functions written in the C language. For example, C processes handle the initial tasks of allocating telemetry to a monitoring module and detecting anomalies. If a potential anomaly is found, the corresponding telemetry is passed to the appropriate expert system for verification. If the expert system concurs that the telemetry appears to be anomalous (without actually diagnosing the specific anomaly at this point), the subsystem monitor then performs an algorithmic check to determine if the anomalous telemetry is merely the result of data noise or corruption. After these preliminary tests are done, and a probability of anomaly occurrence is established, the subsystem monitor invokes knowledge-based processing for diagnosis of the anomaly and recommendation of corrective action. In MARVEL, knowledge-based processing is used only for knowledge-intensive tasks for which it is essential. All other tasks are implemented with C routines. This technique contributes to an overall response time that is sufficient for real-time monitoring.

Hybrid Reasoning for Improved Performance in Knowledge-Based Methods

MARVEL augments several types of reasoning with conventional software methods to improve performance. For example, MARVEL uses hybrid reasoning for detecting data that are uncertain or corrupted or of decaying validity. In the MARVEL system, there are two mechanisms for detecting data-integrity problems. The first mechanism is algorithmic: It uses algorithmic calculations to check the validity of incoming telemetry-based quantities, such as telemetry values and data modes, so that obviously noisy data can be eliminated from further processing. This technique is implemented at the level of the data management process and is used to monitor simple data types. The second mechanism is knowledge based in nature and is implemented in rules. This mechanism uses the method of expectation-based data validation (Chandrasekaran and Punch 1984). Data of questionable integrity are verified by cross-checking them with other data sources for correlation and corroboration. If an anomaly is indicated by a new incoming telemetry word, one can validate this hypothesis by examining known related data to see if they have values that one would expect if the hypothesis were true. If the related data corroborate the initial indication, then the knowledge-based system can conclude that the new data are valid, and the anomaly hypothesis is confirmed. Conversely, if the related data do not appear to be consistent with the new data, then the anomaly hypothesis is not proven. MARVEL's expert systems have been designed explicitly so that they do not disregard the new data, which

might provide the first evidence of a true anomaly that will eventually be confirmed by subsequent telemetry. Thus, whenever possible, the conclusions of the expert systems are based on patterns of consistent data rather than on a single piece of data in isolation.

Temporal Reasoning with Minimal Impact on Real-Time Processing

Real-time systems often need to reason about past events and the order in which they occurred. The MARVEL expert systems respond to events (symptoms) indicated in the spacecraft telemetry by attempting to identify and diagnose specific subsystem anomalies that caused an event. To make this response, the expert system might need to know about other spacecraft events that have occurred in the past and the sequence of their occurrence. This knowledge process involves temporal reasoning, which is implemented in MARVEL using dynamically updated structures, as shown in Figure 4.

The structures contain the name of the event, the name of an anomaly that might have caused the event, a Boolean flag indicating whether the event has occurred and is currently relevant, and an integer specifying the sequence in which the event occurred relative to other events. The anomaly identifier is necessary because a particular event can have bearing on the diagnosis of more than one anomaly (that might or might not have occurred). Thus, a single event can point to multiple structures that are each associated with a different anomaly. The Boolean flag in a structure is set when the event associated with the structure is detected from telemetry. When this flag is set, the relative time of the event is recorded in the structure. The validity of a Boolean flag expires after its corresponding anomaly is resolved, causing the flag to be reset so that it cannot contribute to the detection or diagnosis of the same anomaly unless the associated event occurs again.

These structures are intended to have minimal impact on performance. Once an event is detected, a structure is created for each anomaly whose diagnosis might depend on this event. Thus, the multiple pieces of evidence that confirm the occurrence of an event need only be evaluated once, regardless of how many anomalies might be related to this event. Also, event structures are not retained indefinitely. There is a time limit beyond which an event structure is considered no longer useful for identifying and diagnosing new anomalies. After this time limit expires, a structure's Boolean flag is reset to false, regardless of whether its associated anomaly has been diagnosed. This approach minimizes the number of event structures that are active or relevant at any one time, which, in turn, reduces the number of event structure comparisons that must be performed during a rule-evaluation cycle.

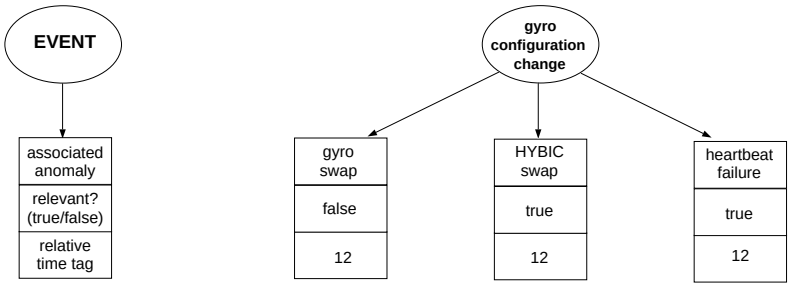

Figure 4. The Event Structure.
The figure on the left depicts the general form of the structure; the figure on the right shows a specific instance of an event associated with three different anomalies.

Multiple Knowledge Bases for Improved Focus of Attention

When significant events occur, real-time knowledge-based systems must focus their attention and resources on relevant parts of the search space to achieve adequate performance. Many expert system environments do not have an efficient method for focusing attention. One standard way to enable focus of attention is to apply different subsets of the domain rules within different contexts. MARVEL accomplishes this task with separate knowledge bases for each spacecraft subsystem and with rule contexts (mini-experts) within the individual knowledge bases.

A top-level data management process identifies incoming telemetry and determines which subsystem monitoring module to invoke for anomaly detection. When an anomaly is found, the subsystem monitor then invokes its corresponding expert system to perform the necessary analysis. This logical partitioning of input data among reasoning modules enables more rapid traversal of the search space and helps to ensure that conclusions and responses that are not relevant to the current analysis are not pursued. This approach has also contributed to the maintainability of the knowledge bases: Several smaller knowledge bases are easier to maintain than a single large one.

Knowledge-Based Reasoning and Transfer of Interprocess Control: An Example

To illustrate the mechanisms that are used in the process of anomaly detection and diagnosis in MARVEL, we give a specific example based on the partial rule network shown in figure 5. Initially, data representing

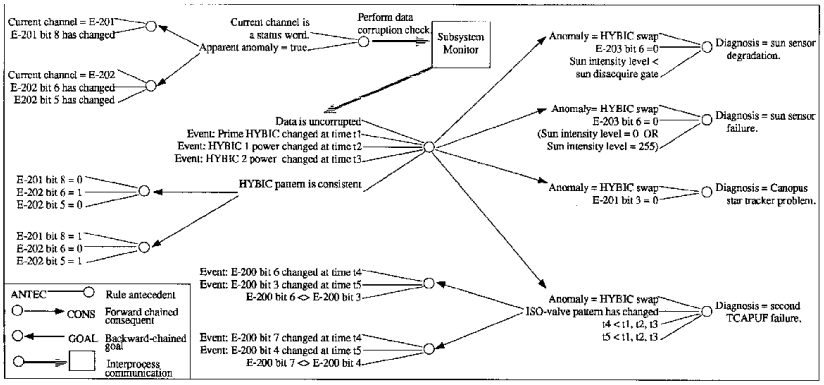

Figure 5. A Partial Rule Network for the Analysis of a Voyager *Articulation Control Subsystem Anomaly.*

the telemetry channel identified as status word E-201 is received by MARVEL's top-level data management process. This module recognizes that the channel is from the attitude and articulation control subsystem (AACS) and routes it to the AACS subsystem monitoring module. It is important to note here that this particular channel is received only if one or more of its bit values has changed from its last known value (in other words, redundant data are suppressed by an external telemetry processing system).

The subsystem monitor compares the given octal value of the status word to the current predicted value of the status word contained in a file. If the two values match, then the status word is ignored, and no further processing occurs (all appears to be well). If the two values are not equal, then a potential anomaly has arisen, and the subsystem monitor passes the status word to the AACS expert system using the C-to-expert-system interface functions.

The AACS expert system checks the bit values of the status word to see if, in fact, they would indicate an anomaly if they were valid (as opposed to corrupted during data transmission). The knowledge base contains domain knowledge that quickly enables it to recognize both anomalous and normal miscompares. If the status word appears to represent an anomaly, then the expert system sends a message back to the subsystem monitor through a call-back function, informing it of this fact. The subsystem monitor at this point initiates an algorithmic check to determine if the change in the status word was the result of data corruption or if it was caused by an actual anomaly.

Two facts are used by the subsystem monitor to examine whether

data corruption has occurred: Redundant status data are suppressed by the ground system, and status problems cannot be remedied without ground intervention. If the anomaly resulted from a mere corruption, then a new correct status word follows, but if the anomaly resulted from a problem on the spacecraft, then no new status word follows. If the monitor determines that the status word is valid (that is, not corrupted), then it notifies the expert system of this fact.

On receiving this final signal that knowledge-based reasoning is appropriate, the expert system initiates the diagnosis of the exact anomaly. From the example status word, E-201, the expert system deduces that the spacecraft has automatically swapped out a possibly faulty AACS component and switched to its backup unit. This event can be corroborated by examining two related values in status word E-202. Because the value of E-202 is not in the knowledge base, the expert system calls back to the subsystem monitor for the last known value of this status word. The expert system confirms that the values of E-202 do corroborate the indicated component swap. If the E-202 data had not borne out this conclusion, processing would have stopped, and control would have passed back to the subsystem monitor with no anomaly messages.

Next, the expert system attempts to deduce the cause(s) of the component swap. Such a swap usually occurs when *Voyager* detects an impairment in its ability to maintain its attitude. The cause of the impairment is normally indicated by the value of one or more other telemetry channels. The expert system retrieves the relevant channels from the subsystem monitor and searches for the cause. Based on these channels, the expert system concludes that the cause of the component swap was an abnormally low sun-sensor intensity value (which could mean that the sensor is bad or that the spacecraft is going off course). The expert system sends a message back to the subsystem monitor, indicating the nature of the anomaly and its cause. Control is then transferred back to the subsystem monitor, which relays this information to the MARVEL system operator by displaying a red window on the workstation screen and displaying the anomaly message (figure 6). At this point, the anomaly diagnosis cycle is completed, and control reverts to the subsystem monitor, which continues to monitor incoming data that are allocated to it by the data management process.

Implementation and Performance of the Distributed Architecture

MARVEL has previously monitored three of the most complex spacecraft subsystems (the computer command subsystem, the flight data subsys-

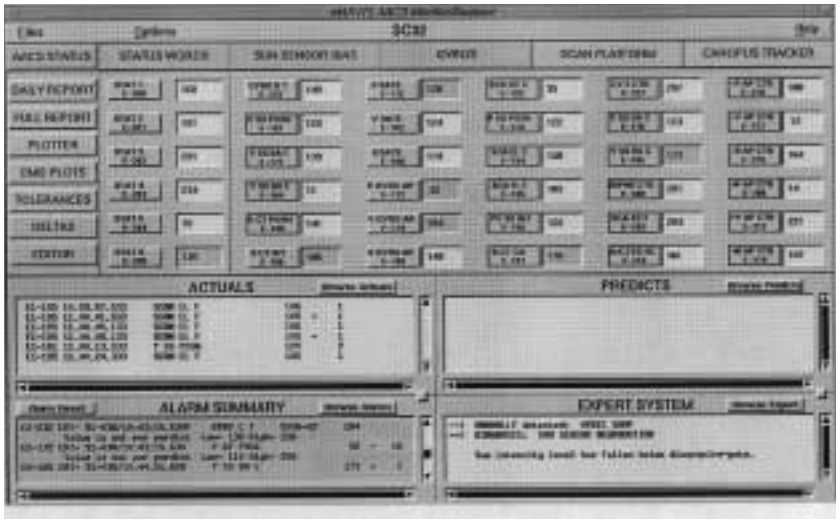

Figure 6. The Main Attitude and Articulation Control Subsystem Real-Time Display and Expert System Dialog.

tem, and AACS) for both *Voyager* spacecraft on a single workstation. However, there are a total of seven spacecraft subsystems that could ultimately be monitored. In addition, the incoming *Voyager* data are sparse compared to the data rates that will need to be handled by future applications of the MARVEL system. Therefore, we pursued a distributed implementation of MARVEL to improve performance for future application to more complex spacecraft.

There are many reasons for distributed problem solving (Bond and Gasser 1988). For example, distributed systems are often characterized by greater computational speed because of concurrent operation. Also, a distributed system can be significantly more cost effective because it can include a number of simple modules of low unit cost. Further, distributed systems can offer a more natural way to represent certain classes of problems that contain subtasks that can be naturally partitioned. Each of these reasons is considered important in the mission operations environment, and as a result, a distributed MARVEL environment was implemented.

Implementation of the Distributed Architecture

The distributed MARVEL architecture shown in figure 7 is based on a central message-routing scheme. The various software modules are allocated among a configuration of UNIX workstations. The data management module receives telemetry data from JPL's ground system. Each

of the three subsystem monitors provides functions such as validation of telemetry, detection of anomalies, diagnosis of causes, and recommendation of corrective actions. The latter two functions are provided through intelligent reasoning modules that are embedded within each of the individual subsystem monitors. The remaining modules include the display processes for each of the three subsystem monitors, the system-level reasoning module for diagnosing anomalies that manifest themselves in multiple subsystems (and therefore cannot be analyzed completely by any one subsystem alone), and the data-routing module for interprocess communication.

The interconnectivity of the distributed system is provided by a transmission control protocol–internet protocol (TCP-IP) central router program and a set of messaging routines that are linked to the subsystem processes. All MARVEL processes are connected to the central router by UNIX sockets. Each process registers with the router under one or more aliases (names) and a systemwide, unique group name. This approach allows several instantiations of MARVEL with different group names to operate simultaneously and independently. The router delivers each message according to the destination's alias(es) and group name only, allowing a message to be sent to one, some, or all processes within a group, depending on which processes share the destination alias. In MARVEL, each subsystem has a subsystem-specific alias of *<id>_subsystem* for receiving messages sent to one or more instances of the same subsystem. Each subsystem also has a generic alias of s*ubsystem* to receive global data sent to all subsystems. This characteristic allows an arbitrary number of processes to be included in MARVEL at run time. Furthermore, different copies of distributed MARVEL can be supported by the same router through the use of distinct group names. (These features are important when more than one analyst needs access to the same subsystem display at his/her own workstation.)

The implementation of the distributed MARVEL system involves the installation of a set of remotely executed functions that act as data and event handlers for the appropriate data types. For example, the data management module sends a message to a subsystem module that tells the subsystem to analyze newly arrived telemetry data. This analysis involves sending a message to the destination alias of *name_of_subsystem* with the event descriptor *data_analysis*. The telemetry data to be analyzed are buffered in the body of the message. After receiving this message, the network process registered as *name_of_subsystem* takes the data carried in the message and calls the function *data_analysis(data)*. This implementation scheme was shown to be compatible with (and integrated well with) both the SUN VIEW and X WINDOW systems.

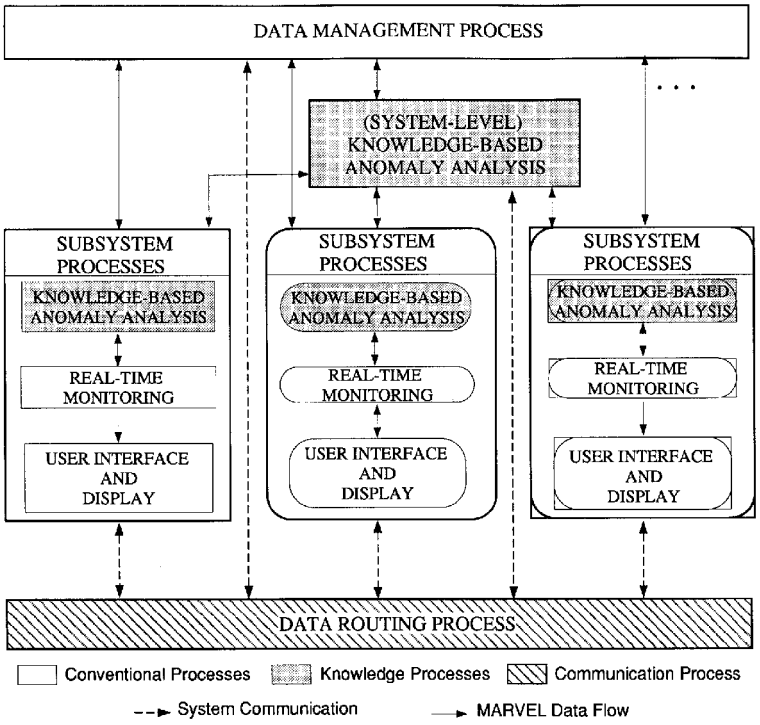

Figure 7. The Distributed MARVEL Architecture.

Performance of the Distributed Architecture

A general critical measure for evaluating computer systems is cost-per-unit performance versus fixed-total performance. This curve usually rises at a steeper than linear rate. Thus, in the uniprocessor domain, acquiring additional computing power is expensive. However, for some applications, the cost can be minimized through distributed processing. For distributed systems having tasks of identical work load and requiring negligible communication overhead, the critical measure curve is constant. For realistic systems with nonnegligible communication overhead, the critical measure curve is related to the speedup $S(N)$ (Fox et al. 1988), defined as

$$S(N) = T_{seq} / T_{conc}(N) .$$

In this equation, N denotes the number of processors, and T_{seq} and $T_{conc}(N)$ refer to the execution times of the sequential program and the distributed program on N processes, respectively. Distributed systems with speedup $S(N)= 0.8N$ are considered efficient (Fox et al. 1988); the minimum desired speedup for a distributed MARVEL system is $0.6N$.

The basic measurement of performance for the distributed MARVEL is the speedup $S(N)$. However, it was not possible to measure a unique value of speedup because of the heterogeneous nature of the MARVEL modules. This heterogeneity arises because the processing loads of the four basic components (the data management module and the three subsystem modules) are not identical. Thus, the logical alternative to this measurement was defined as the worst-case estimate of speedups for the individual subsystems. With a four-processor implementation, a speedup of 3.6, or $0.9N$, was observed. This result indicates that the MARVEL environment is a highly efficient distributed system. Two factors contribute to the success of these results. The first is the modularity inherent in the application (as is common in many other complex applications). The second factor is a distributed design that effectively minimizes the need for interprocess communication.

Distributed AI Research in Progress

Distributed AI involves coordinating the knowledge, goals, and plans of several independent, intelligent systems to solve problems that cannot easily be solved by any of the independent units acting alone. The MARVEL expert systems exchange information to cooperatively solve *system-level analysis problems*. These are problems that affect or are manifested in more than one spacecraft subsystem and, therefore, cannot be solved by any one of the subsystem monitors acting alone. To achieve this end, a higher-level expert system is being developed that coordinates the activity of the subsystem experts.

This system-level expert responds to input from the subsystem experts in a data-driven mode that begins with the arrival of telemetry. Anomalous telemetry is detected by the subsystem monitor and analyzed by the corresponding subsystem expert. The latter is invoked by the arrival of the anomalous data. The subsystem experts have knowledge of subsystem anomalies that could have possible system-level impact. When such an anomaly is recognized, the relevant information is communicated by the subsystem expert(s) to the system-level expert. The system-level expert then requests additional information as needed (from the subsystem experts and subsystem monitors) to perform the appropriate analysis.

Discussion

The development of MARVEL has shown the value of a rapid development approach that emphasizes top-down design and bottom-up implementation. The implementation was modular and incremental, with

frequent deliveries (every 5 to 10 months) of new or enhanced capabilities. The result is an automated tool that began as a simple software module for automating straightforward tasks and that evolved over a period of five years into a sophisticated system for automated monitoring and analysis of multiple spacecraft subsystems. The initial modular design enabled MARVEL to be developed incrementally, with each subsequent delivery providing greater breadth to the application. This approach was instrumental to the success of the effort because it was compatible with available budgets and encouraged user and sponsor confidence with frequent demonstration of results. In addition, the approach influenced the validation and use of MARVEL, as described in the next two subsections.

Validation of MARVEL

The validation of MARVEL was ad hoc largely because of a lack of formal procedures for testing AI systems. Two methods were used: carefully engineered test cases and online validation (involving parallel operations with human analysts). Most problems were detected with the use of test cases, but some were not detected until the software was used in an online mode. Newly delivered modules were subject to an online validation period, typically on the order of one month. The purpose of the validation period was to continually compare the results of manual approaches with those obtained by MARVEL, so that reasonable levels of confidence in the automated system could be obtained without risk to ongoing operations.

The primary advantage of this approach was its minimal impact on development costs. However, there were several disadvantages, which under ideal circumstances would cause us to avoid the ad hoc testing approach in the future. On isolated occasions, minor bugs in MARVEL went undetected until the end of the parallel-operations phase. Not detecting these bugs temporarily undermined user confidence, particularly with users who were not enthusiastic about automation. A second disadvantage is that without formal validation procedures, there were occasional questions about whether MARVEL should be accepted formally as official ground software for mission operations. The current lack of solid answers in this area would prevent the use of MARVEL's AI modules for certain tasks that are considered mission critical but has not prevented the use of these modules in an advisory mode.

Use and Benefit of MARVEL

MARVEL has been in active use since it was first deployed in 1989. In its current version, it performs real-time monitoring functions for the

three subsystems for which it was developed. These functions previously required the presence of human analysts for a minimum of 8 hours for each subsystem every day; during planetary encounters, human presence was required on a 24-hour basis. In addition, MARVEL automatically performs a variety of non–real-time functions that previously required analyst attention. These functions save anywhere from 30 minutes a week (for clock drift analysis) to 2 hours a day (for daily report generation). During the time that MARVEL has been online, it has not failed to detect any anomalies that occurred within its domain. During parallel operations, several of these anomalies were detected by MARVEL prior to their being detected by the human analyst. On two occasions, MARVEL detected anomalies that operations personnel believe might have been overlooked completely by human analysts because the quantities of data that were being transmitted at these times were larger than could reasonably be handled without automated assistance.

Initial emphasis on productivity enhancement resulted in an early version of MARVEL that (according to the responsible operations supervisor) would have made real-time CCS subsystem work force reductions of 60 percent (3 out of 5 analysts) possible during the Neptune encounter had MARVEL been approved for stand-alone, rather than parallel, operations. Subsequent to the Neptune encounter, significant work force reductions were implemented for all spacecraft subsystems, not primarily because of MARVEL but because of postencounter budget cuts. However, it should be noted that MARVEL played a substantial role in simplifying the transition to reduced work force for the subsystems for which it was available.

The initial emphasis on productivity enhancement temporarily curtailed the development of MARVEL's expert systems because it was perceived that the presence of expert systems did not improve efficiency of operations. This perception stemmed from the correct observation that anomaly analysis was only required in the presence of spacecraft anomalies, which did not occur with sufficient frequency to warrant an automated approach, particularly because human confirmation of the expert system analysis would still be required.

However, the postencounter work force reductions brought about renewed interest in expert system development. However, the goal of this development is no longer work force reduction but the preservation of mission expertise. The current analysts are new to the mission and, for the most part, do not have the experience of the previous staff members. In addition, the new personnel will have little opportunity to gain such experience: Although the *Voyager* interstellar mission is scheduled to continue until approximately 2018, spacecraft activity is at a relatively low level. Thus, there are far fewer opportunities for learn-

ing about the spacecraft and its operation than during planetary encounters. There is concern that analysts with the experience to handle future anomalies will be less readily available or that they will have retired. As a result, MARVEL's expert systems are being expanded to provide information to the current analysts that is based on the expertise of former analysts. The system-level anomaly analysis work that was described previously is part of this effort.

Summary

This chapter presented methods for combining conventional software with AI techniques for use in real-time problem-solving systems. The methods described were presented in the context of the MARVEL system for automated mission operations, which has provided a continuous and evolving demonstration of the success of the approach since *Voyager*'s Neptune encounter in August 1989. These techniques were implemented in a distributed environment that will accommodate the more rigorous real-time demands of NASA's more recently launched interplanetary missions.

Acknowledgments

The research described in this chapter was carried out by the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration. The authors want to acknowledge discussions with John Rohr; encouragement from Dave Eisenman and Jim Marr; mission expertise provided by Roy Otamura, Gene Hanover, Ralph Ellis, and Enrique Medina; and support from JPL's *Voyager* Project, Flight Project Support Office, and Director's Discretionary Fund.

References

Bahr, E., and Barachini, F. 1990. Parallel PAMELA on PRE. In P*arallel Processing of Engineering Applications*, ed. R. A. Adey, 209–219. New York: Springer-Verlag.

Barachini, F., and Theuretzbacher, N. 1988. The Challenge of Real-Time Process Control for Production Systems. In Proceedings of the Seventh National Conference on Artificial Intelligence, 705-709. Menlo Park, Calif.: American Association for Artificial Intelligence.

Bond, A. H., and Gasser, L. 1988. *Readings in Distributed Artificial Intelligence*. San Mateo, Calif.: Morgan Kaufmann.

Chandrasekaran, B., and Punch, W. 1987. Data Validation during Diag-

nosis: A Step beyond Traditional Sensor Validation. In Proceedings of the Sixth National Conference on Artificial Intelligence, 778–782. Menlo Park, Calif.: American Association for Artificial Intelligence.

Fox, G., et al. 1988. *Solving Problems on Concurrent Processors,* volume 1. Englewood Cliffs, N.J.: Prentice Hall.

Georgeff, M. P., and Ingrand, F. F. 1989. Monitoring and Control of Spacecraft Systems Using Procedural Reasoning. In Proceedings of the Space Operations-Automation and Robotics Workshop, 209–217.

Hayes-Roth, B. 1990. Architectural Foundations for Real-Time Performance. *Artificial Intelligence Journal* 26: 251–232.

Horvitz, E. J.; Cooper, G. F.; and Heckerman, D. E. 1989. Reflection and Action under Scarce Resources: Theoretical Principles and Empirical Study. In Proceedings of the Eleventh International Joint Conference on Artificial Intelligence, 1121–1127. Menlo Park, Calif.: International Joint Conferences on Artificial Intelligence.

Laffey, T.; Cox, P. A.; Schmidt, J. L.; Kao, S. A.; and Read, J. Y. 1988. Real-Time Knowledge-Based Systems. *AI Magazine,* 9(1) (Spring 1988): 27-43.

Schwuttke, U. M. 1991. Intelligent Real-Time Monitoring of Complex Systems. Ph.D. diss., Dept. of Electrical Engineering, Univ. of Southern California.

Schwuttke, U. M., and Gasser, L. 1992. Real-Time Metareasoning with Dynamic Trade-Off Evaluation. In Proceedings of the Eleventh National Conference on Artificial Intelligence. Menlo Park, Calif.: American Association for Artificial Intelligence. Forthcoming.