Lessons Learned: Automated Event Recognition in Distributed Data Environments

M.B. Hudson, D. Schreckenghost, and C. Thronesbery

NASA Johnson Space Center, 2101 Nasa Road One, Houston, TX 77058

Abstract

We investigated issues with recognizing events when monitoring large amounts of live data coming from distributed systems. We did this by reporting on a system that was deployed in NASA/JSC's Water Research Facility. The system included complex event recognition software that recognized significant events and anomalies when monitoring the Water Recovery System. We share our experiences and lessons learned after running the system for a year and a half. We discuss the issues that were brought about by operating in a distributed data environment. We believe that these issues will need to be addressed by any system that performs complex event recognition in such an environment. This is partly due to the fact that recognizing events is sequential by nature, and operating in a distributed data environment is parallel by nature. In this paper we discuss our solutions to these issues and point out areas that require further research for future growth of this technology.

Introduction to the Problem

We investigated the problem of automated event recognition in distributed environments by developing a system to monitor data coming from NASA's Water Research Facility (WRF). Using automated event recognition software, we developed a system that, upon recognition of an event, automatically notified appropriate personnel to take action. The notification was done via pager or email. This approach provided two benefits. First, we were able to capture the knowledge of domain experts when defining the events that needed to be recognized and addressed. Second, automated monitoring relieved the necessity of using people to monitor 24 hours/day, 7 days/week and allowed them to focus their attention on problem solving and other jobs rather than on data monitoring.

The use of event recognition in live distributed data environments is particularly interesting because it is a relatively new use of technology that addresses a very common problem in a broad range of industries. The problem of monitoring large amounts of distributed data exists in government and commercial arenas, including hospitals, chemical plants, offshore drilling rigs, nuclear plants, military environments, and air and naval stations. In short, any environment that requires vigilant non-invasive monitoring of people and processes in order to maintain a safe state for operations is a candidate application.

Data monitoring is often done manually. For instance, at NASA, telemetry data are streamed from the shuttle or the space station to mission control and mission control is manned around the clock to ensure that all systems are operational. At the NASA/JSC we are investigating ways to automatically monitor data to detect important events and notifying people when predefined events or anomalies occur.

We have developed an automated event recognition system for monitoring the water recovery system in the WRF at JSC. This application combines signals from four different subsystems and uses complex pattern recognition software to recognize events traditionally distinguishable only by human expertise. To do this, we employed complex event recognition software developed by a third party and modified by them to fit our environment. But even with the added benefit of customizing this software to conform to our environment, issues arose that we did not foresee. It was not until we put the system into operational use that many problems surfaced.

While we used the Complex Event Recognition Architecture (CERA) (Fitzgerald, Firby, and Hanneman., 2003) developed by Dr. James Firby and Dr. William Fitzgerald for event recognition, we believe that the lessons learned in deploying this software are of general interest to the pattern recognition community. This paper describes the issues arising from the characteristics of the distributed data environment that will need to be addressed by any automated monitoring system operating in such an environment. By making the research community aware of these problems and the solutions we found effective, we hope to promote further growth in the area of automated event monitoring.

Application – Detecting Autonomous Control Events

We used CERA to provide the event recognition capability in the Water Research Facility at NASA/JSC. The WRF is used to evaluate new hardware for purifying water (called water recovery). The Water Recovery System (WRS) is an example of such hardware tested in the WRF. The WRS consists of four systems: a biological water processor (PPBWP), a reverse osmosis system (RO), an air evaporation system (AES), and a post-processing system (PPS). The WRS is operated by autonomous control software (Bonasso, et al., 2003). Control engineers are responsible to keep an eye on the autonomous controller and intervene should problems arise. Thus, control engineers look for events indicating anomalous conditions in the WRS, such as a shutdown of one or more of the systems, an unsuccessful slough (which is supposed to unclog a filter), or a loss of communications with other systems. The Distributed Collaboration and Interaction (DCI) System was deployed in the WRF to assist control engineers in interacting with the automated WRS (Schreckenghost, et al., 2005). Complex event detection software (i.e., CERA) was deployed as part of the DCI system. Recognizers for events of interest to control engineers were defined using the CERA event definition language. CERA monitored data from the WRS and compared it to these definitions. When a predefined event of interest occurred, CERA exported the event to the DCI agent system, which notified the responsible control engineers.

We deployed the DCI System January 28, 2004. It supported testing in the WRF until August 30, 2005. During much of that time, CERA successfully recognized the predefined events. The appropriate control engineers were automatically notified by pager, email or fax when a predefined event occurred. The event was sent and supporting data were made available so that the user could quickly assess the situation and determine what was happening in the WRF. This allowed the engineers to spend most of their time working on other projects, knowing that if anything important happened they would be alerted immediately. The DCI Event Detection System based on CERA provided 24 hour/day, 7 day/week monitoring without requiring human presence in the WRF to do the monitoring.

CERA patterns are defined in event recognizers. A user describes an event to be captured by defining a recognizer for that event. A recognizer is a pattern of sub-events and a clause telling what actions to take when the event is recognized. The pattern consists of a list of sub-events connected by a relation. CERA supports the following relations:

- ALL – all signals must be seen for the event to be recognized
- ONE-OF – only one signal in the group must be seen for the event to be recognized
- IN-ORDER – a sequence of signals must be seen for the event to be recognized.
- WITHIN – a sequence of signals must be seen within a specified time period for the event to be recognized
- WITHOUT – a period of time must pass without a specified signal being seen for the event to be recognized
- James Allen's 13 Interval Relations (Allen, 1983) – an event can be described using any of James Allen's definitions given two signals.

Events are signaled when the pattern is observed. For example, a WRS-Shutdown is an event that can be described by the successful shutdown of its four sub-systems. The recognizer for a WRS Shutdown might look like this:

```
(define-recognizer  WRS-Shutdown
  (pattern '(ALL
              (PPS-Shutdown-complete)
              (PPBWP-Shutdown-complete)
              (RO-Shutdown-complete)
              (AES-Shutdown-complete)))
  (on-complete
     (Signal '(WRS-Shutdown complete)start end)))
```

The start time and end time of the signal is calculated from the start and end times of events in the pattern.

The patterns can be hierarchical, that is, there can be relations nested within the relation. Sub-events are modeled as separate recognizers that send a signal when they complete. These signals can be used by a higher level recognizer. For example, PPS-Shutdown might be defined like this:

```
(define-recognizer PPS-Shutdown
  (pattern '(In-Order
              (Air-and-water-flow halted)
              (UV-lamps off)
              (Decreasing-water-pressure)))
  (on-complete)
   (Signal '(PPS-Shutdown-complete) start end)))
```

When CERA receives the signals showing all three sub-events in the PPS-Shutdown pattern have completed, the recognizer completes and it issues a signal "PPS-Shutdown complete". This signal satisfies one of the steps in the WRS-Shutdown recognizer. This is how low level recognizers can be linked to higher level recognizers to form a hierarchy of events.

The System Architecture

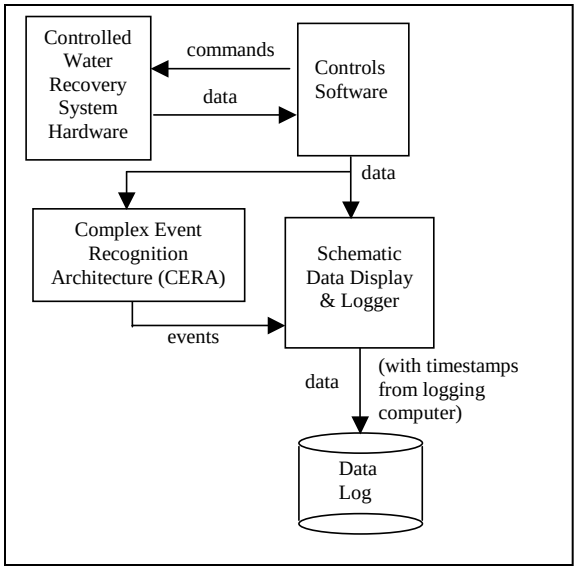

Figure 1. Architecture of Distributed Software System Supporting Event Detection for the WRS

The architecture of the distributed software system supporting event detection is shown in Figure 1. The controls software had the only direct link to the controlled water recovery system hardware. The low-level controls that provided data to both CERA and to the logging software did not have easy access to the current time of day, so timestamps were not affixed to the data until it reached CERA and the logging software. Since these two software processes were running on separate computers, there was an occasional mismatch of system time affixed to the data. This became a problem when a situation captured by CERA was displayed with CERA timestamps and then low level data plots were displayed from the logged data (with logger timestamps) to illustrate to the end user precisely what CERA had seen. When times were out of synch on the two computers, CERA could appear either very sluggish or prescient. We had an alternate, test-harness mode for running CERA directly from the data logs rather than from the live data directly from the controls software. In this test-harness mode, CERA recognition times were less perplexing.

In the architecture illustrated in Figure 1, the low-level controls software would bundle the data into batches and publish a complete batch of data every twenty seconds. Any change that took place during the last twenty seconds would be reflected in the new batch of data. On the receiving end of this batch of data, any changes must be considered to have been simultaneous. However, CERA was built to consider incoming data values sequentially. When CERA received a set data values in a single batch, it would process them sequentially rather than as a batch of simultaneous values. Within a single twenty second interval, it would be possible for a sensor value (pressure reading) to exceed a critical value and for a resulting control to respond (turn off a pump). The batching of data would cause these two events to appear as simultaneous events. Unfortunately, the serial processing of individual data values by CERA could occasionally cause the ordering of these events to be reversed. If the data representing the pump command appeared higher in the list of data values in the batch, then it would appear that the pump was commanded before the pressure reached critical value.

During the year and a half that the system was in use in the WRF we encountered a number of issues related to the handling of data and the modeling of events by the event recognition software. We describe these issues and how we solved them for the WRF application in the next sections.

Data Handling Issues

The first set of issues that need to be addressed arise from the fact that event recognition software must collect and process data from multiple, distributed data sources (such as the water recovery subsystems described above). Because of this, timing issues may arise - not only in the discrepancies in clock times among the multiple systems, but also in the lag time between the time the data was sampled and the time the data are processed. There is also the potential of a bottleneck when sampling data at too high a rate or losing information when sampling data at too low a rate.

Issue #1: Multiple Signals Same Timestamp

In distributed systems many data items are sampled at the same time. Therefore, event recognition software that processes data signals one at a time can introduce an artificial ordering on the data. We need to be able to process a group of multiple signals in the data queue that were sampled at the same time (which we call a timestamp), as if they were a single signal.

We encountered this problem using CERA, which assumes one signal per timestamp. This is due to the fact that the software evolved from the world of natural language processing where there is no notion of time stamped data. Instead, words are parsed in the sequence in which they occur. Although timing constraints were added to the event recognition software, they serve primarily to keep track of the start and finish time for each event. Timestamps for the start and finish of sub events are propagated to determine the start and finish of parent events.

Solution for the WRF Application: We had to be careful in how we created our recognizers to avoid this problem. Often we changed the relation of a recognizer from an "In-Order" (events must happen in a sequential order) to an "All" (all events must occur, but no order is specified) because we could not guarantee the order of the signals coming in. We considered reissuing signals, and trying to order the signals in the data queue to match the order found in recognizers, but none of these ideas really worked.

A better solution would have been to timestamp each data element and consider time as an integral part of the data. This requires the recognition software to reason over multiple simultaneous timestamps. In our case we didn't have this option since we didn't have this reasoning capability and it was not possible to change the way the data was being broadcast

Issue #2: Data Sampling Rates Affect Recognition Definitions

Typically data from hardware instrumentation is sampled and distributed at some fixed rate. If the sampling rate is too frequent, it can slow down the system – the system needs to process the data at least as fast as it is sampled. If the sampling rate is too low, it can cause the recognition software to miss the details of the event because they are not revealed in the data.

Solution for the WRF Application: To recognize events when the data rate was low and observations came in a set instead of sequentially, we had to make distinctions in the events based on our confidence in the recognition. We defined *observed events* to be those where we saw data for every event in the expected sequence of events. We defined *inferred or confirmed events* to be those where we saw a partial data set that verified the effects of the expected sequence of events (e.g., the system was observed in the desired state but we did not observe the transition to that state) and were thus confident the events had occurred. We defined *possible events* to be those where we saw data for some events in the expected sequence of events, but these events were not sufficient for us to be confident the event had occurred.

In the following example, we show event definitions for each of these event categories. In all these events, we are looking to recognize a slough event. A slough is a maintenance event on a bioreactor in a water recovery system. In this event, air is blown through a tube lined with bacteria to strip off the outer layer of bacteria that can clog the flow of water through the tube.

In the "observed" manual slough, we observed the tube clogged, we observed a slough, and then we observed the tube unclogged.

```
(define-recognizer (manual-slough-observed)
   (pattern '(IN-ORDER
      (tube ?n clogged)
      (tube ?n slough)
      (tube ?n unclogged)))
  (on-complete
    (signal '(Manual-Slough-Observed tube ?n)
                                          start end)))
```

In the "inferred" manual slough, we did not observe a slough, but we did see the tube was clogged and then unclogged. So we assumed a slough occurred to unclog it.

```
(define-recognizer (manual-slough-inferred)
   (pattern '(IN-ORDER
      (tube ?n clogged)
      (tube ?n unclogged)))
  (on-complete
    (signal '(Manual-Slough-Inferred tube ?n) start end)))
```

In the "possible" manual slough, we're not sure the tube was really clogged. The difference between a possible clog and a clog is that when there is a clog, the tube sustains a high pressure for an extended amount of time, whereas in a possible-clog there is high pressure briefly in the tube.

```
(define-recognizer (possible-manual-slough)
   (pattern '(IN-ORDER
      (tube ?n possible-clog)
      (tube ?n unclogged)))
  (on-complete
    (signal '(Possible-Manual-Slough tube ?n) start end)))
```

Similarly, we recognized the need for these event distinctions when looking for Post Processing System (PPS) safe state. The PPS Safing could take a few data samples to complete or it could complete in just one data sample. Observing the PPS safe state requires that the pump feeding water to the system be seen to change from on to off, the lamp relays be seen turning off, and either the air flow or water flow drop below operating levels. Confirming the PPS safe state does not require seeing the pump or relays transition from on to off, but just confirming that their current state is off.

Issue #3: Time Synchronization and Representation

Time synchronization is key for distributed systems such as the DCI application in the WRF. Unless time is synchronized, data processing can be done out of order and erroneous results concluded. To synchronize time, it is necessary to provide for a common time source and a consistent time representation. Since tools often make assumptions about the default time representation used internal to the tool, translators are needed between these

internal representations and the common time representation. The different time representations used for this application include absolute times (such as GMT, time since 1970, etc) and relative times (the amount of time between two events).

Solution for the WRF Application: To ensure time synchronization, we provide two capabilities: (1) a time server that was available for use by all processes in the distributed system, and (2) automatic updating of local computer time using an NTP time server serving atomic clock time. Additionally we provide translators between different time representations (such a GMT, time since 1970, etc.). Event recognition required three different versions of time. We had to be diligent in keeping track of which time formats were being used where.

Event Modeling Issues

This section describes some of the data modeling issues that came up as a result of trying to accurately depict events in the domain of automated control of space systems.

Issue #4: Event Persistence

Persistent events are events that are continuously evaluated to determine if they still hold true. Transient events are events that are observed true at some point in time, but do not necessarily continue to be true at later times. We found it necessary in the space domain to recognize both persistent events and transient events. For example, we need persistent events when recognizing complex state changes where later events are preconditioned on earlier events (only see water flow increase after you turn on the water pump). In other cases, we observed systems move through a sequence of transient states (such as changing to a standby state until the system warmed up). CERA supported the recognition of transient events but did not support the recognition of persistent events.

Solution for the WRF Application: In cases where we needed persistence, we used a technique involving name-value pairs and mutually exclusive states. For example, we could use name-value pairs to model the water pump in both the on and off positions. Then, when we needed the water pump to stay in the on position while waiting for another event, we would model it as below:

```
(pattern '(In-Order
            (Water-pump ON)
            (Waterflow-increased)))
```

In this example, if we see the water pump turn on, the recognizer waits until it sees the waterflow increase before completing. In the meantime, if the pump is turned off a (Water-pump OFF) signal is issued and CERA's contradiction mechanism causes this recognizer to fail. (The contradiction mechanism will cause a recognizer to fail if all elements except the last in a signal, match all elements except the last in the pattern. If the last elements do not match, the signal is said to "contradict" the pattern item.) When the recognizer fails, all the information about recognizing this event is deleted and it starts over as if nothing has been recognized yet.

If we did not need persistence, we would model the water pump event as a simple event (Water-pump-on), rather than modeling it as a name-value pair. This way any event, even a "Water-pump-off" event, would not affect it. (This is because CERA's contradiction mechanism only works when a state is modeled using two or more elements, e.g. a name-value pair. A single element such as "(Water-pump-on) "cannot be contradicted.)

Issue #5: State Transitions

We discovered when modeling events about system control that we need to recognize state transitions. We define a transition as observing the state of a device change from the expected initial state to the expected final state. This is similar to detecting an event sequence; that is, looking for one event state followed by another event state. But, in the CERA sequence operator (which we believe is typical of other models of sequence), each event is considered to have occurred the *first* time it was observed. For a transition, we are interested in the *last* time the initial state is observed and the first time the final state is observed. This temporal distinction made it difficult to use the standard sequence operator when detecting transitions.

Solution for the WRF Application: We added a new Transition operator that reflected the last time the initial event was observed and the first time the final event was observed. This operator was useful in detecting the initiation of mode changes in systems (such as startup, shutdown, or standby), since they often start with a state transition. Without this operator, it was impossible to detect the time at which a system began a mode change.

Issue #6: Failure to Observe an Expected Event

For system control, recognizing the failure to observe an expected event can be as important as recognizing an event. Specifically, we need to recognize when an observed triggering event is not followed by a response event within a specified period of time. For example, in the PPS we expect for the automated controller to safe the PPS within 15 minutes of a bed breakthrough (an event requiring PPS maintenance). If this safing does not occur within 15 minutes, we want to issue an alarm (e.g., PPS bed breakthrough with no PPS safe). Detecting the absence of an event during some time period is not commonly supported in event recognition software.

Solution for the WRF Application: Detecting the absence of an event required a new CERA operator. Jim Firby modified CERA's core capabilities to add the After-Without operator. The After-Without operator looks for the first event and then looks for a second event for a specified amount of time. If the second event is found, the operator fails. If the second event is not found, the operator succeeds. Thus, the operator succeeds only when it sees one event, followed by the absence of another.

Issue #7: Combinatoric Explosion due to Partial Recognition

One of the challenges we experienced in the domain of automated control was not completing an event as soon as expected. In these cases we would observe some but not all of the necessary sub-events, resulting in CERA monitoring for the final sub-events for a long period of time. In these cases where an event is partially recognized, CERA recognizes and bookkeeps all instances of these partial solutions, which incurs considerable overhead for extended periods of time. Bookkeeping and cleaning up these instances can bog down the system because it is necessary to store all possible combinations for satisfying the recognizer until the recognizer completes.

Solution for the WRF Application: During operations in the WRF, we observed CERA detect the partial solutions a number of times. In one case, the second event of the PPS startup (i.e., turning on the UV lamps) occurred 4 hours late. As a result, the PPS_Startup_Confirmed recognizer was partially recognized with each set of data (sampled every 15 seconds). This caused the PPS Startup Confirmed recognizer instance to increase in size as it stored every matching instance in this four hour period. By the time the UV lamps were turned on, state history had grown to 220,000 combinations. The resulting delay in processing was around 5 hours. To fix this problem in the short-run, we stopped using operators that stored partial solutions. We have not found an easy solution to the combinatoric problem in the long term. However, we have found no cases for this domain in which it was necessary to keep all possible combinations of events that occur during a lengthy transition period. Thus, for this application, it would be useful to provide operators that only store the first time a match occurs or the latest time a match occurs.

Issue #8: Reusable Event Definitions

In modeling an application domain, it can be useful to define reusable recognizers for recurring events (such as mode changes). For example, a PPS Safe is a common mode-change event that is a sub-event in many other events. Rather than duplicating software for these common events each time they occur in a recognizer, it is more efficient to code these sub-events as separate recognizers and compose larger recognizers from them. We had difficulty defining separable reusable events within CERA because it does not fully implement event abstraction and encapsulation. While separate recognizers can be encoded, there is no way of describing relations among multiple recognizers (i.e., no way to indicate one event contains another).

Solution for the WRF Application: To provide a capability to create reusable event definitions, we implemented a simple extension to the CERA representation for relations among multiple recognizers. We stored all information about a recognizer and its parent linkage in a hash table and used this to reconstruct the larger recognizers. While helpful, this approach incurs additional overhead and does not provide true hierarchy among the recognizers.

Lessons Learned

We learned that in a near-real-time distributed data environment, reasoning over time becomes a central issue. This is especially important in a distributed environment because multiple signals can be received having the same timestamp. As such, an event recognition system has to manage more than just the order in which events arrive - it needs to explicitly reason about the time the data was sampled. It needs to be able to treat multiple signals simultaneously as if they were a single signal, otherwise an artificial ordering can be introduced. Currently we do not know of any event recognition software that reasons about time.

Data from distributed sources can be received out of order due to transmission delay. The event recognition software should handle signals being slightly out of order due to transmission delay by processing based on the time the data were sampled instead of the time the data were received. This approach constrains the minimum response time of the system, since the software must wait until signals from all sources have been received before processing a data set.

Because selecting the data sampling rate is often not an option when fielding applications, knowledge of the data sample rate must be considered when developing and integrating the event recognition software If the system cannot keep up with a high data rate, it may be necessary to build bridge software between the data source and the event recognition software that reduces the data rate. Additionally, recognizers for the same event may be written differently for different data sample rates. For example, events that occur in sequence with a high data rate may all be recognized at the same time with a low data rate. As a result, if the sampling rate changes, the pattern to describe the event may need to be changed as well.

When operating in domains with distributed processing, time synchronization among processes is essential to avoid processing data in the wrong order. The time representation and synchronization techniques should be part of the initial system design and should be handled consistently throughout the system. Since internal time representations can vary among software, it is suggested that techniques for translating between different time formats be defined early in development. It is also important to model time as data available to the systems for reasoning. Object-oriented techniques are particularly suited to such modeling, since they can encapsulate conversion functions and provide a clean interface to different time formats

We found that usually the event descriptions (patterns to be recognized) evolved the more they were used. We added new recognizers as we realized we needed to detect slightly different versions of the same event. Many of these variations arose because of limited data availability due to low data sample rate. We added recognizers to distinguish "confirmed", "observed", "inferred" and even "possible" events in order to show the confidence we had that the events actually occurred.

Even when using expressive event definition languages such as CERA provides, complex applications like the WRF event detection software can require language extensions. For detecting control events, we added new relations (e.g., Transition, After-Without). We identified needed customization of existing relations (e.g., bookkeeping of instances in the All relation). And we developed techniques for adding new types of events, such as persistent events. Event recognition software should provide a regular means of extending the language.

We found that the ability to define reusable recognizers reduced implementation time and reduced the potential for inconsistencies among recognizers. Such a capability also is needed to aid users in specifying events, since it enables defining event libraries that can be composed by users into customized events. Reusable event definitions require the ability to describe relations among a set of recognizers. Sub-events must be able to signal the containing event when they are recognized and sub-events must persist until the containing event is recognized.

We became aware of a combinatorial problem when trying to keep track of all possible solutions that might solve a pattern. This arose when the event recognition software was in a state of partial recognition (i.e., recognized some but not all of the necessary sub-events in an event) for a lengthy time period. However, we have found no cases for the space system control domain in which it was necessary to keep all possible combinations of events that occur during a lengthy period with partial recognition. Thus, for this application, it would be useful to provide operators that only store the first time a match occurs or the latest time a match occurs. While these extensions would help for our domain, we understand that under certain circumstances it is necessary to track all possible solutions. We recognize this is a research issue for the community.

Conclusions

The domain of space system control where we deployed our event recognition application has characteristics, such that, even when using a tool as flexible as CERA, we still ran into unforeseen issues. The ability to easily capture expert knowledge and rapidly test the system in operational use is needed to identify and resolve issues quickly. We found that accurately defining complex patterns for event recognition relies heavily on evolution through use. New events were required when test objectives changed. Modified events were needed as test configurations changed, or our understanding of operations improved. As a result, defining event recognizers proved to be one of the more costly aspects of using this software. To make event detection software viable for operational use requires that end-users define recognizers instead of software experts, as done in this test. It also will require standardizing the process of event definition so that different users can define events consistently within a domain. But the richness of the event detection language makes it difficult for non-experts to use without assistance. We propose that ease of use of the event language and event consistency within a domain would be improved by providing the following capabilities as part of a development environment for end users:

- domain ontologies that define a common language for specifying events,
- event libraries that define reusable recognizers for common control concepts, such as latching or bang-bang control,
- input data filters that compute and signal statistics on parameters in the data stream, and
- a test harness that supports executing recognizers over imported data sets.

We were able to find workarounds for most of the issues described in this paper. Solving issues with time and combinatorial explosion, however, are the most challenging problems. In the end, we delivered what we had promised – a system that recognizes significant events in a distributed environment and automatically alerts the appropriate people to handle them. This was due in part to the flexible and robust event recognition software provided by INET and to the generous support of its creators.

Acknowledgements

We want to acknowledge Dr. Michael Shafto, manager of NASA's Software, Intelligent Systems, & Modeling for Exploration, who sponsored this work. We also wish to acknowledge the efforts of Pete Bonasso/Metrica, Tod Milam/SKT, and Cheryl Martin/Applied Research Laboratories on the design and implementation of DCI. Finally we want to thank Dr. Will Fitzgerald and Dr. Jim Firby for their help in using and extending the CERA software.

References

Allen, James. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26:832{843, 11 1983.

Bonasso, R. P., Kortenkamp, D., and Thronesbery, C. 2003. Intelligent Control of A Water Recovery System: Three years in the Trenches. *AI Magazine* 24 (1): 19-44.

Fitzgerald, Will; R. James Firby; & Michael Hanneman. (2003). *Complex Event Recognition Architecture: User and Programmer's Manual*, Revision 2.

Fitzgerald, Will; R. James Firby; & Michael Hanneman. (2003). Multimodal Event Parsing for Intelligent User Interfaces. *Intelligent User Interfaces*. Orlando: ACM.

Schreckenghost, D., P. Bonasso, M.B. Hudson, C. Martin, T. Milam, and C. Thronesbery. (2005). Teams of Engineers and Agents for Managing the Recovery of Water. *AAAI Spring Symposium 2005 on Persistent Agents.* Palo Alto, CA: AAAI.

Schreckenghost, D., C.Thronesbery, and M.B. Hudson (2005), Situation Awarement of Onboard System Autonomy. *The 8th International Symposium on Artificial Intelligence, Robotics and Automation in Space*, München, Germany.