Source of Acquisition
NASA Johnson Space Center

SITUATION AWARENESS OF ONBOARD SYSTEM AUTONOMY

Debra Schreckenghost[1], Carroll Thronesbery[2], Mary Beth Hudson[2]

[1]Metrica, 1012 Hercules, Houston, TX, USA 77058, ghost@ieee.org
[2]S&K Technologies, S&K Technologies, 201 Flint Ridge Plaza, Suite 102, Webster, TX 77598, c.thronesbery@jsc.nasa.gov, mhudson@sktcorp.com

ABSTRACT

We have developed intelligent agent software for onboard system autonomy. Our approach is to provide *control agents* that automate crew and vehicle systems, and *operations assistants* that aid humans in working with these autonomous systems. We use the 3 Tier control architecture to develop the control agent software that automates system reconfiguration and routine fault management. We use the Distributed Collaboration and Interaction (DCI) System to develop the operations assistants that provide human services, including situation summarization, event notification, activity management, and support for manual commanding of autonomous system. In this paper we describe how the operations assistants aid situation awareness of the autonomous control agents. We also describe our evaluation of the DCI System to support control engineers during a ground test at Johnson Space Center (JSC) of the Post Processing System (PPS) for regenerative water recovery.

1. INTRODUCTION

Establishing a human presence in deep space will require changing the way manned space operations are conducted. Due to longer communication delays and extended mission duration, NASA can no longer rely on large teams on Earth to support astronauts round-the-clock. As a result, astronauts must become more independent from ground support. Autonomous crew and vehicle systems enable such crew autonomy.

The use of system automation does not eliminate the need for crew to interact with these systems, however. The crew must maintain awareness of system events, the actions taken by autonomous control, and the health of automated systems. They must perform periodic maintenance on system hardware. In cases where control automation cannot resolve anomalies, the crew must fix or workaround the problem.

We have developed intelligent agent software for onboard system autonomy. Our approach is to provide *control agents* that automate crew and vehicle systems, and *operations assistants* that aid humans in working with these autonomous systems. We use the 3T control architecture to develop the control agent software that automates system reconfiguration and routine fault management. We use the Distributed Collaboration and Interaction (DCI) System to develop the operations assistants that provide human services, including situation summarization, event notification, activity management, and support for manual commanding of autonomous systems.

In this paper we describe how we use the DCI System to develop operations assistants that aid situation awareness of onboard autonomy by (1) detecting and notifying users of important events as they occur, (2) logging these events chronologically so the user can inspect and review them well after they occur, and (3) encapsulating complex event sequences as summary events that can be decomposed down to the constituent events, if needed. We also describe our evaluation of the DCI System to support control engineers during a ground test at JSC of the Post Processing System (PPS) for regenerative water recovery.

2. LIFE SUPPORT SYSTEM AUTONOMY

To achieve onboard autonomy, it is necessary to provide control automation for crew and vehicle systems that perform routine system reconfiguration and fault management. We have used the 3T control architecture [1] to develop such autonomous control agents. The 3T architecture consists of the following layers of parallel control processing:

- **Planner**. a hierarchical task net planner that coordinates automated control activities for the hardware systems to ensure that resource and time constraints are met. The Planner also re-plans these activities when a task fails to complete or events make activities no longer possible.
- **Sequencer**. a reactive planner that dynamically constructs operational procedures based on situational context. The Sequencer decomposes activities from the Planner into executable steps, based on the current state of the system. The Sequencer also monitors for anomalies and reacts to them. When activities have been completed or failed, the Sequencer informs the Planner.
- **Skill Manager.** traditional closed loop control modules for each effector and sensor. Activation of these modules is coordinated by the Skill Manager. The Skill Manager takes commands

Proc. 'ISAIRAS 2005 Conference', Munich, Germany, 5-8 September 2005 (ESA SP-603, September 2005)

from the Sequencer and passes them to the correct skills, as well as returning to the Sequencer the results from commands executed by these skills.

The 3T architecture has been used extensively to control life support systems during ground tests of hardware for regenerative water recovery [2] and air revitalization [3]. The layered architecture is used to handle the uncertainty inherent in complex domains like life support. The three layers execute in parallel, with bi-directional communication between the layers. Control commands flow down through the layers and feedback flows back up through the layers to close the control loop. Each layer is designed to take safe action should communication with other layers be lost. If a command fails at any level, a repair action can be initiated (e.g., re-planning at the deliberative level, selection of an alternative sequence at the reactive level). Each layer operates at a different time constant, allowing high speed controllers at the low level of the architecture to operate in parallel with the slower, deliberative algorithms at the high level. The time constants for control of life support are 1 second for the Skill Manager, 30 seconds for the Sequencer, and a few minutes for the Planner, Each layer also abstracts task and state information differently, with information becoming more abstract as you move up the layers. The abstractions for control of life support are activation of individual effectors for the Skill Manager, sequencing of groups of effectors (i.e., procedures) for the Sequencer, and scheduling of procedures to manage resources (i.e., potable water, oxygen) for the Planner.

3. AGENTS FOR SITUATION AWARENESS

3.1 DCI System

The DCI System was developed to aid distributed human teams in working with autonomous control systems. DCI provides each person in the team with a personal agent, called a Liaison Agent. Each Liaison Agent provides services to help its user in maintaining situation awareness of control autonomy, and in conducting manual activities and coordinating them with autonomous activities. To aid situation awareness, the DCI system provides the following capabilities:

- **Data Monitoring and Event Detection.** monitor heterogeneous data sources to discern when data values indicate a meaningful or significant change in system state or health has occurred,
- **Event Notification.** decide which team members should be notified about events, and notify them based on team protocols specifying how and when personnel in different roles should communicate,
- **Situation Summarization.** analyze and interpret sets of events to aid crew in understanding of what happened during a situation.

To aid activity coordination, the DCI system provides the following capabilities:

- **Automated Activity Planning.** build activity plans for the team that achieve team objectives and adjust these plans in response to situation changes or manual input,
- **Tracking Activity Execution.** use telemetry, user state, and information from the user to assess the completion status of planned activities,
- **Command Authorization.** coordinate activities not in the plan with planned activities to ensure concurrent activities do not interfere with each other or with actions taken by control autonomy,
- **Procedure Support** (in work). provide the team with electronic procedures and track the execution of steps in these procedures, with an emphasis on support for joint human-automation procedures.

The Liaison Agent maintains a model of its user's activities, location, and roles in the team that helps customize these services for the needs of each team member. Figure 1 illustrates the DCI architecture.

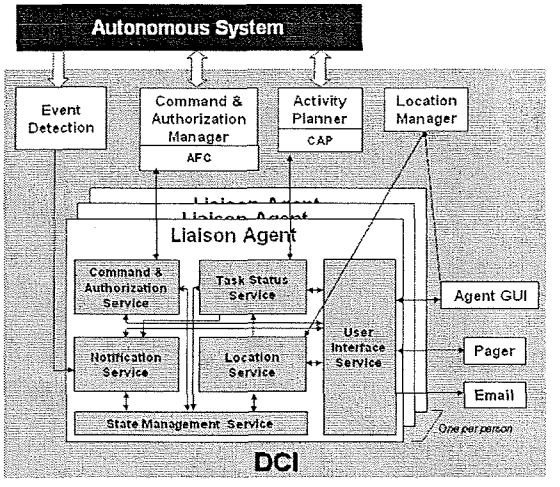

Figure 1. DCI Architecture

In this paper we describe DCI support for situation awareness. For additional information on the DCI System, see Schreckenghost, et al. [4].

3.2 Data Monitoring and Event Detection

The DCI System provides two Data Monitoring and Event Detection capabilities: the Event Detection Assistant (EDA) and the Complex Event Recognition Architecture (CERA) [5, 6]. EDA implements limit sensing and triggered events predicated on simple conditionals. CERA implements detection of situations consisting of sets of events organized temporally and hierarchically. EDA can provide input to CERA.

The Event Detection Assistant provides a library of pattern matching functions implemented in Java. It monitors telemetry data from hardware systems as well as more abstracted information computed by software systems such as EDA or the 3T control agent. For example, EDA receives all Caution and Warning events detected by the 3T control agent, identifies the events designated as critical, and exports this subset of events to the DCI users. When the event pattern is observed in the monitored data, EDA generates an XML message describing the observation. The event XML message consists of the following: (1) ID, (2) timestamp, (3) category, (4) subject, (5) urgency, (6) importance, (7) body, (8) sender, (9) receivers, and (10) response time. Most of these fields can be further decomposed. The content of the event is expressed in the body. The body always has a text block and may have the following additional information: (1) agent or domain, (2) notification category, (3) changed user state value, or (4) situation structure. The contents of the message are specified using the following ontologies that can be extended to new domains of application:

- Domain: physical system the event refers to
- EventType: the format of the event content
- Notification: Caution & Warning classification

These XML-formatted events are exported from EDA using CORBA event channels.

The Complex Event Recognition Architecture (CERA) automatically constructs summaries of events that develop over time or that have complex relationships among them (e.g., cascading events after a failure). To detect complex events, CERA compares incoming data such as telemetry or computed data to conditions in pre-built *Event Definitions* defined by system experts. The Event Definition consists of two parts: the *pattern* that must hold true in the data and the action that is performed *on completion* of the pattern. Patterns consist of conditions and the temporal relationships among them. Conditions are defined as either (1) logical relations on incoming data (atomic events), or (2) an encapsulation of collections of events as a more abstracted event. Temporal relations in patterns include conjunction (ALL operation), disjunction (OneOf operation), ordered sequence (InOrder Operation), and Allen's temporal interval relations [7]. The primary action executed on completion (i.e., upon recognition of a pattern) is signalling the consequences of event recognition. These consequences include (1) notification that the event occurred, and (2) notification that side effects occurred. The consequences of recognition can be determined during execution by including logical branching based on data bindings. For example, if a lamp relay is > 0, the recognizer would signal "(confirm_lamp_on true)" otherwise it would signal "(confirm_lamp_on false)".

An example of a CERA recognizer to detect the startup of the PPS is shown below. In this example, the pps-startup-event is recognized when three encapsulated events are signalled in an ordered sequence. When the event completes, it signals that the event has occurred (i.e., (pps-startup complete))

```
(define-recognizer (pps-startup-event)
  (pattern
    '(in-order
       (transition_air_and_water_flow started)
       (confirm_uv_lamp1_on true)
       (increasing_input_air_flow true)))
  (on-complete
    (start end bindings data pattern)
    (binding-translate () bindings
      (store-and-signal-event '(pps-startup complete) start end pattern
        "PPS Startup" "Startup, PPS system start."))))
```

Encapsulation into abstracted events results in hierarchy among events. For example, the event pps-startup-event encapsulates a pattern on 3 sub-events: (1) transition air and water flow started, (2) confirm uv lamp1 on, and (3) increasing input air flow true. Each of these events has a separately defined recognizer. Because complex events can unfold in more than one way, Event Definitions can encode more than one event pattern as a valid indicator that a situation has occurred. For example, a safing situation for the PPS requires observing either low input air flow or low water pressure, but not both. CERA will capture which of these events was observed in a given situation.

When constituent events are recognized, they are exported to the Notification Services of the Liaison Agents of all DCI users. Similar to the EDA, an XML-formatted message is exported over a CORBA event channel when an event is recognized. The content of these messages comply with the formats and the ontologies described previously.

The message formats and ontologies we have described constitute a standard interface definition and communications protocol for passing events to DCI users. This interface and protocol can be used to integrate new Data Monitoring and Event Detection capabilities with the DCI System. Adding events from new domains requires (1) encoding messages in an XML format recognized by DCI, (2) extending the domain ontologies used in the interface definition for the new events if needed, and (3) exporting encoded messages over a CORBA event channel. We have used this interface and protocol to integrate DCI with the Intelligent Briefing Response Assistant (IBRA), an XML rule-based system developed by SKT, Corp. [8].

3.3 Event Notification

Once events are detected by the Data Monitoring and Event Detection functions just described, they are passed to the Liaison Agents of the DCI system. Each

Liaison Agent provides a Notification Service to ensure its user receives the right information at the right time to perform his or her job. The Notification Service receives all events and compares them to notification rules (called *Notice Specifications*) that encode the operational protocols for human communication used in mission training [9]. Each Notice Specification consists of (1) a *Notice Condition* that defines patterns in the content of an event that must match for the event to be passed to the user, and (2) a *Notice Directive* that defines what presentation medium to use and how urgently and emphatically to inform its user of events that are passed. Notice Specifications are encoded as XML rules.

The Notice Condition is used to filter incoming notices. This condition is represented as a set of pattern-matching triplets, related by conjunctive and disjunctive logical operators. Each triplet consists of (1) a *property* of a notice, (2) a *matching function*, and (3) the *value* to match against. The property of a notice identifies a content field in the notice. The value to match against identifies the desired value for this content field in the notice. The matching function defines the type of comparison performed between the property and the matching value. To ensure Notice Conditions are extensible to the concepts and language of a new domain, a property corresponds to an ontology and a matching value correspond to a field within this ontology. These ontologies describe features of the incoming event (e.g., alarm or alert) as well as states of the user (e.g., roles, location). The matching functions defined for the Notification Service include string comparisons, ordinal comparisons, integer comparisons, and ontological comparisons that allow us to consider hierarchies of abstraction. An example of a Notice Condition to detect when the category of an incoming notice is "comms" is shown below:

```
<Condition>
  <AtomicMatch>
    <PropertyName>EventCategory</PropertyName>
    <MatchingFunction>OntologySubClassOrEqual</MatchingFunction>
    <InstanceToMatch>
      <OntologyEntry>
        <OntologyName>EventCategoryOntology</OntologyName>
        <OntologyValue> Event.DomainEvent.LifeSupportEvent.Comms
                                                    </OntologyValue>
      </OntologyEntry>
    </InstanceToMatch>
  </AtomicMatch>
</Condition>
```

The Notice Directive is used to annotate an event with information needed to present the event to the user. It assigns the *latency* tolerated in notifying the user and how emphatically the user's *focus of attention* should be shifted to the event. Using urgency information in the incoming event, the latency is assigned one of the following values (1) immediate – the notice should be presented to the user without delay, (2) deferred – the notice should be presented to the user when available, and (3) archive – the notice can be viewed at any time. Using urgency information in the incoming event, the focus of attention is assigned one of the following values: (1) ShiftToPrimary – the notice should be presented emphatically to the user to focus attention on it, (2) ShiftToSecondary – the notice should be presented such that the user is aware of the notice but not distracted by it, and (3) NoShift – the notice should be presented with no special emphasis (i.e., put in background). The Notice Directive also assigns one or more *presentation modalities* to the event from the set of possible modalities. The assignment of modality can be conditionalized on whether the user is online to the DCI System or not (what we call *user presence*). For the current DCI system, modalities include (1) display in the Notice Viewer if the user is logged into DCI, (2) queue for display in the Notice Viewer when the user logs into DCI at a later time, (3) email to the user, and (4) page the user. Events that match a notice specification are passed with assigned latency, focus of attention, and modalities to the user interface for presentation to the user. An example of a Notice Directive is shown in Figure 2.

```
<NotificationDirective>
  <Latency>immediate</Latency>
  <FocusOfAttention>primary</FocusOfAttention>
  <ModalityImplication>
    <Condition>
      <AtomicMatch>
        <PropertyName>UserPresence</PropertyName>
        <MatchingFunction>OntologySubClassOrEqual
                                              </MatchingFunction>
        <InstanceToMatch>
          <OntologyEntry>
            <OntologyName>CrewPresenceOntology
                                                  </OntologyName>
            <OntologyValue>Online</OntologyValue>
          </OntologyEntry>
        </InstanceToMatch>
      </AtomicMatch>
    </Condition>
    <IfTrue>
      <Modalities>
        <Modality>DISPLAY</Modality>
      </Modalities>
    </IfTrue>
    <IfFalse>
      <Modalities>
        <Modality>DISPLAYQUEUE</Modality>
        <Modality>PAGER</Modality>
      </Modalities>
    </IfFalse>
  </ModalityImplication>
</NotificationDirective>
```

Figure 2. Example of Notice Directive

This directive will shift the user's attention immediately to the incoming notice. If the user is online to DCI, it will use the DCI GUI for notification. If the user is offline, it will use a pager for notification and queue the message for viewing in the DCI GUI.

The Notification Service uses an XML pattern matcher to compare these rules to incoming data. When the Notice Condition matches, the Notice Directive is applied to annotate the notice for presentation. When more than one Notice Specification matches, the

Notification Service combines the directives of the matching specifications by (1) applying the most salient assignment for both latency and focus of attention, and (2) combining all modalities. The Notification Service is implemented using Java, with ontology-based pattern matching for notice routing based on VESPR [8], an XML rule-based system built using JES, the Java implementation of the CLIPS rule-based system developed at NASA.

This annotated notice from the Notification Service is passed to the User Interface Service (UIS) for presentation. The UIS uses latency, focus of attention, and modality to construct the user notification. For notices that are displayed using the DCI user interface, the UIS builds and maintains a persistent model of the user interface state, permitting users to view events long after they have been received. For notices that are emailed or paged, the UIS formats and sends these notices. Examples of how the user interface management software uses these saliency annotations include (1) determining the degree of emphasis when displaying a notice (interrupting or peripheral) or (2) assigning the urgency codes to pager messages.

When a user's roles change, the information needed to perform his or her job changes. The Liaison Agent uses its knowledge of user roles and location to determine which specifications are applicable at any time. It will load the applicable Notice Specifications whenever its user receives a changed role assignment.

Because Notice Specifications are based on standard operational protocols for communication, they should be defined and managed by the organization managing the human team (e.g., Crew Office for astronauts). It is possible for users to customize their notification by defining Individual Notice Preferences, but these personal rules are not permitted to compromise or override the rules defined by the organization.

Once notices are received by the User Interface Service, they are available to be viewed in the DCI user interface. DCI provides a toolbar that is displayed whenever the user logs into his or her agent. The user can access notices from the toolbar by clicking on the Notice icon to access to the Notice Viewer. When urgent or important notices arrive, the Notice icon will change appearance and an audible beep is annunciated. From the Notice Viewer the user can view all notices passed by the Liaison Agent for it user. Notices are grouped into the following categories: (1) domain – notices about the control software and the hardware it controls, (2) group – notices about the changes in location, activity, and roles of each person in the operations team, and (3) From ARIEL – notices to the user from his or her Liaison Agent. There is a tab for accessing notices in each category. Additionally, there is an ALL tab where every notice passed by the Liaison Agent can be viewed. Notices in a tab are shown as a list, sorted chronologically. Each notice in a list is annotated with the following information:

- time received: time the notice was received by the Liaison Agent; notices are sorted using this time
- subject: short description of the notice contents
- category: corresponds to notice format and sender
- location: user location when notice was received
- mode: other modalities used to notify the user; if the user is paged, a "P" is displayed in this field

This information can be viewed from the list of notices in a tab. Selecting a notice from this list shows the contents of the notice in the window at the bottom of the viewer. A notice contains the information below:

- time: time the event occurred
- event: short description of the event
- details: body of the notice message
- importance: criticality of information in the notice
- urgency: how quickly to inform the user

Additionally, some notices contain a button that can be used to launch a Situation Viewer showing the details of a related situation captured by CERA. Figure 3 shows an example of the Notice Viewer display.

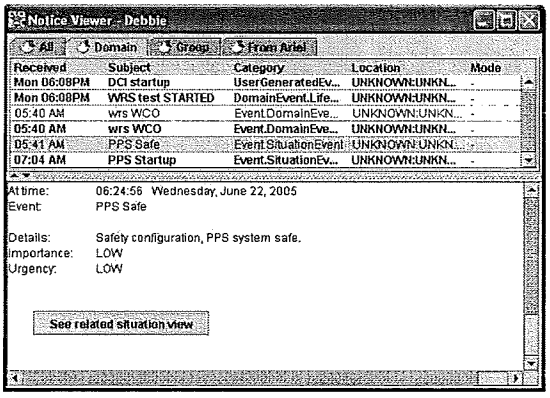

Figure 3. Notice Viewer Display

3.4 Situation Summarization

Situation summaries are needed to assist the crew and ground experts in developing an understanding of the events comprising a complex situation after these events have occurred and evidence of these events is no longer observable in system state. This can include nominal situations such as system mode reconfiguration requiring multiple state changes (e.g., startup) or complex configuration changes (e.g., restringing a power distribution node). It also can include off-nominal situations, such as system failures or crew repair sequences.

One way the DCI System supports situation summarization is to provide the ability to inspect the data analyses and interpretation performed by CERA and to relate these analyses to the underlying data that supports them. The DCI Situation Viewer displays the analyses as a hierarchical set of events with temporal relations among them. These events include both the actions taken by automation or humans and the associated system changes that these actions effect (e.g., command a flow valve open and observe a rise in flow rate through the valve). A summary of the situation can be attained by viewing the top level of the event hierarchy (see figure 4). In this example, we see that safing the PPS consists of first observing air and water flow are halted, next confirming the UltraViolet (UV) lamps are turned off, then observing a drop in either the input air flow or the input water pressure.

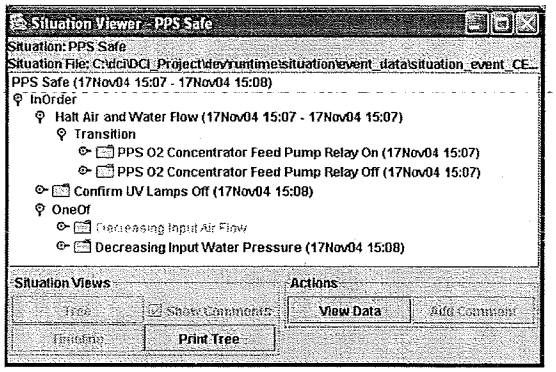

Figure 4. Top Level of Event Hierarchy

To clarify details about particular events or to get more information about how an event occurred, the user can drill down into the hierarchy (see figure 5). These details include an indication of which events in the hierarchy were observed in this case (shown in black text) and which events in the hierarchy were not observed in this case (shown in grey text). Showing the events that were observed in relation to all the events that might have been observed reminds the user of how the system can operate and aids the user in comparing similar situations occurring at different times. In this example, reviewing the event details reveals that all three UV lamps have been confirmed off. It also reveals that the O2 concentrator feed pump was observed turning off to indicate that air and water flow is halted. Finally, it reveals that a drop in input water pressure occurred first for this situation.

An atomic event at the bottom of the hierarchy is associated with the data set used to detect the event. The data values that triggered the primitive event are captured. Additionally, the set of parameter values observed over the period of time when the situation occurred are available for inspection. This data set can be viewed as a plot, as a table, or as a set of summary statistics (i.e., minimum, maximum, average value). Figure 6 shows a data plot supporting the conclusion that decreasing input water pressure was observed first. A closer look at this data reveals that input water pressure dropped below 1.4 at 15:00 while the input air flow dropped below 200 around 7 minutes later.

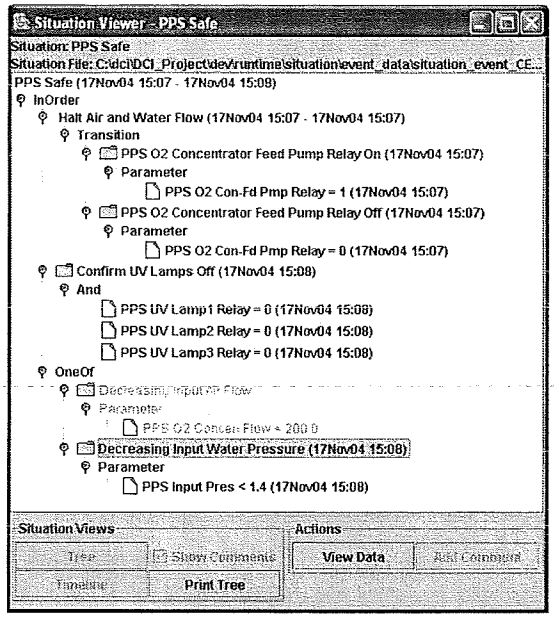

Figure 5. Expanded Event Hierarchy

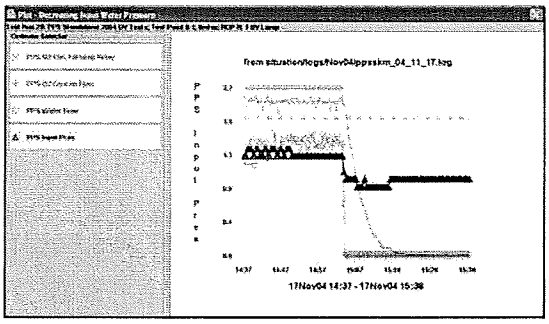

Figure 6. Data Indicating Input Pressure Drop

Another way the DCI System supports situation summarization is to provide a chronologically ordered list of events in the Notice Viewer. These events can come from a variety of sources, including the 3T control agent, CERA, EDA, and operations personnel. Since the notices that are passed to the user depend upon the user's assigned roles, this set of events represents a view of the system customized to the user's job. From the notices in Figure 3, the user can determine that DCI was restarted and the WRS test started near 6pm Monday. The system shutdown at

5:41 the next morning when the feed tank was empty (WRS WCO), then was restarted at 7:04 AM.

DCI provides the Event Logger for reviewing notices that have been cleared from the Notice Viewer but have been archived. The user loads archived notices from either database or file. The loaded notices are grouped into a tab, so you can compare notices from different archives. Notices are sorted chronologically and shown in a list much like the Notice Viewer. Selecting a notice from the notice list shows its content in the window below. To aid searching the archive for particular notices, the user can enter the following types of filters: (1) time regions, (2) strings in the subject field, and (3) notice categories. The Event Logger uses these filters either (1) to show all notices that match the filter, or (2) to show all notices that do not match the filter.

4. RELATED WORK

The DCI System has developed adaptive strategies for filtering and routing notices to a distributed user group based on the users' roles in that group. This differs from other notification research that addresses the needs of individuals without considering their group memberships [10, 11]. The DCI approach using notice specification most closely resembles Bradshaw's use of ontology-based notification policies in the KaOS system [12]. Commercial notification software such as Stirling Systems Group JobMon and Bear Mountain Software's Topper address remote notification via paging using static routing specifications. DCI provides for dynamic notice routing by using it's knowledge of user roles and location to adjust what notices are sent to a user and what notification mechanism is used (e.g., pager, email, etc.).

DCI supports situation awareness with high level summaries that describe important state changes and anomalies. On demand, user's can view the evidence for these conclusions about state and the specific actions of humans and automation that affect them. This approach is consistent with Endsley's definition of situation awareness as "the perception of the elements in the environment within a volume of time and space, the comprehension of their meaning and the projection of their status in the near future" [13].

5. EVALUATION OF THE DCI SYSTEM

The Crew and Thermal Systems Division (CTSD) performs ground-based testing of regenerative water recovery systems in the Water Research Facility (WRF) at JSC. It consists of a waste water collection facility, a water analysis laboratory, and the hardware systems used to regenerate water. One such system is the PPS. A ground test of the PPS was conducted at JSC from January 2004 to January 2005 to evaluate the effectiveness of different bed substrate configurations in removing trace inorganic salts and a set of UltraViolet (UV) lamps in removing trace organic carbon in recycled water. The PPS hardware is automated using the 3T control agent described previously. Similar to the way crew will interact with automated systems, control engineers are on-call for routine PPS maintenance and to resolve PPS problems the automation cannot fix. These engineers perform other duties than control engineering most of the time. Each week a different control engineer is assigned to fulfil the Prime role, which makes him responsible to handle all problems with the PPS. In addition to the Prime, there is a Backup engineer, who handles issues when Prime is not available, and a Coordinator, who is the resident expert on the control automation. Both Prime and Backup change weekly while Coordinator is a permanent role.

In previous ground tests without the DCI System, the Prime engineer manually checked the system every 3-4 hours to see if it was functioning properly. This meant there could be hours of delay before problems were handled. The engineer would first review schematics displaying current system state. If he perceived a problem, he would then bring up tables of logged data to try and identify the problem. Both of these tasks could be performed remote from the PPS hardware. If this provided insufficient information to identify the problem, or if a recovery action was required, the engineer would go to the site of the PPS hardware and control automation where additional information from the automated software was available and where he could adjust the PPS as needed.

The PPS ground test is the first use of DCI to assist control engineers in supervising automated crew life support systems [14]. Instead of monitoring the PPS periodically, control engineers rely on the DCI System to contact them if an interesting event or a problem occurs and to assign anomaly handling tasks when needed. Using secure internet access via Virtual Private Network (VPN), control engineers can access the DCI System from their offices at JSC, from offices located off-site, and from home. When an engineer becomes Prime, the notice specifications used by his Liaison Agent change to ensure that he is contacted first when important events occur or problems arise (i.e., page). For example, the Prime is notified when a tank of water has been processed, or when the PPS beds lose their effectiveness in polishing water. Control engineers use situation summaries created by DCI to better understand anomalies and how to fix them. They also use these summaries to verify that nominal reconfigurations of the PPS are timely and correct.

6. CONCLUSIONS

After a year of using the DCI system in the WRF, control engineers rely on it to maintain situation awareness about the test and to detect PPS problems. The DCI System operated near continuously in support of the PPS ground test from January 2004 through January 2005. Resetting the DCI System was done periodically to clear system buffers and improve system performance after extended operation. From June 2004 to January 2005, DCI was restarted every 11 days on the average, and ran for up to 30 days continuously in some cases. When DCI was restarted, it was typically down for less than 10 minutes.

DCI situation awareness software was used throughout this test. The notification capability operated reliably once the notice specifications were tuned to ensure notices were routed to the proper personnel. The data monitoring and event detection capability required minor adjustments throughout the course of the test. Changes include (1) modifying or writing new Event Definitions for CERA and EDA as users changed what they wanted to see or how the PPS test was conducted, and (2) debugging errors in Event Detection software, including bugs the CERA software from INet.

One of the more surprising results of this evaluation is that pagers are not reliable for infrequent but critical notification. We found that users who did not need a pager for anything else forgot to carry them, forgot to change the batteries in them, or forgot to turn them on. This delayed user awareness of important notices (i.e., anomaly notices) by as much as 10 hours in some cases. As a result, we plan to investigate alternative notification approaches, such as cell phones.

Based on the results from evaluating the DCI System in the WRF, we propose that intelligent operations assistants such as DCI enable more autonomous space operations. They permit safely shifting greater responsibility to the crew for managing autonomous crew and vehicle systems by supporting supervision and coordination of concurrent human and autonomous activities. Such capabilities provide for the high level of onboard autonomy needed for the Crew Exploration Vehicle and human lunar return missions.

7. ACKNOWLEDGEMENTS

We want to acknowledge Dr M. Shafto, manager of NASA's Software, Intelligent Systems, & Modelling 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.

8. REFERENCES

1. Bonasso P, Firby J, Gat, E., Kortenkamp D., Miller D, & Slack M. Experiences with an Architecture for Intelligent, Reactive Agents. *Journal of Experimental Theory of Artificial Intelligence*. 9: 237-256. 1997.

2. Bonasso P., Kortenkamp D., & Thronesbery C. "Intelligent Control of Water Recovery System: 3 years in the Trenches" AI Magazine 24 (1): 19-44 2003

3. Schreckenghost D., Ryan D., Thronesbery C., Bonasso P., & Poirot D. Intelligent control of life support systems for space habitats. *IAAI 1998*. Madison, WI. July 1998.

4. Schreckenghost D., Martin C., Bonasso P., Kortenkamp D., Milam T., & Thronesbery C. "Supporting group interaction among humans and autonomous agents" *Connection Science* 14(4) 361-369 2002.

5. Fitzgerald W., Firby J., Phillips A., & Kairys J. Complex event pattern recognition for long-term system monitoring. *AAAI Spring Symposium Workshop on Human Interaction with Autonomous Systems in Complex Environments*. Mar 2003 104-109.

6. Thronesbery C., & Schreckenghost D., Situation Views: Getting Started Handling Anomalies. *IEEE International Conference on Systems, Man, and Cybernetics*. Washington, D. C. Oct 5-8, 2003

7. Allen, J. Maintaining knowledge about temporal intervals *Communications of ACM* 26:832-843 11 1983

8. Malin J., Molin A., and Thronesbery C. Managing and instructing information assistants. *AAAI Spring Symposium Workshop on Persistent Assistants: Living and Working with AI*. March 2005.

9. Schreckenghost D., Martin C., and Thronesbery C. Specifying organizational policies and individual preferences for human-software interaction. *AAAI Fall Symposium on Etiquette for Human-Computer Work* 32-39. North Falmouth, MA: AAAI Press 2002.

10. Horvitz E., Jacobs A., & Hovel D. 1999. Attention-sensitive alerting. *Proceedings of UAI '99*, Stockholm, Sweden. Morgan Kaufmann, 305-313 Jul 1999.

11. Schmandt C., Marmasse N., Marti S., Sawhney N., and Wheeler, S. Everywhere messaging. *IBM Systems Journal* 39(3&4): 660-677 2000.

12. Bradshaw J. et al. Representation and reasoning for DAML-based policy and domain services in KAoS and Nomads. *AAMAS 2003*. Melbourne, Australia, ACM Press. 2003.

13. Endsley M. Design and evaluation for situation awareness enhancement. *Human Factors Society 32nd Annual Meeting*, Santa Monica, CA. 1988.

14. Schreckenghost D., Bonasso P, Hudson M., Martin C., Milam T., & Thronesbery C. Teams of engineers and agents for managing the recovery of water. *AAAI Spring Symposium Workshop on Persistent Assistants: Living and Working with AI*. March 2005.