Abstract

We have used adaptive interface techniques when developing and operating autonomous control software for manned space systems like life support or robotics. We adapt the interface using both a model of control tasks and a model of user roles. The purpose of this adaptation is to support changing user roles resulting from adjusting the level of autonomy of the control software. We describe progress to date on applying adaptive interface techniques for manned space operations.

Introduction

We are developing user interface software that supports the development and use of autonomous control software for manned space support systems (e.g., life support, robotics). An important aspect of these interfaces is assisting users in interacting with the autonomous control system. Our approach for providing such capability is to support different levels of control autonomy, ranging from fully autonomous operation to manual selection and execution of tasks.

We use adaptive interface technology in developing user interfaces for adjustable control autonomy (Kortenkamp, et al, 2000). These interfaces are adapted for the following:

- changes in task allocation corresponding to a change in control autonomy
- changes in control situation, including failures, that require altered level of human awareness or interaction
- changes in the user’s roles

We have used models of both the task and the user to adapt the interface. We describe our approach in this paper.

Task Models for Adaptation

The Three-Tier (3T) software architecture (Bonasso, et al. 1997) that we use for autonomous control includes task-level control. As such, control tasks are represented explicitly in the application software. When the application executes, control tasks and the results of these tasks are monitored at the user interface. Information about which tasks were selected, executed, completed successfully, or failed can be used to define control events of interest to the user. We use these control events to reconfigure the display and logging of information at the user interface. Control events are defined in pairs – an initiating and a terminating event. We define this event pair, and the associated specification of display changes as an event trigger. The occurrence of the initiating control event in the autonomous system changes the display configuration per the specification in the event trigger. This change persists until the terminating event occurs. Figure 1 shows an example defining an event trigger.

Specific ways in which the user interface can be adapted include the following:

- show or hide a new display window
- initiate or terminate logging of pre-specified data groups
- initiate or terminate verbose reporting of tasks executed
- execute a custom user interface function

We plan to add the capability to initiate or terminate highly salient alerts (audio, color changes) at the occurrence of a control event. This is particularly important for eyes-off, hands-occupied operations.

These techniques for adapting the user interface can be used in a variety of ways. They can be used to notify the user of interesting control situations. For example, we can pop up a window showing a history of control tasks recently executed when a fault management task is executed. A second use of this adaptive capability is to notify the user of the need for information from the user or to request a manual action. For example, we can use an audio signal to notify a human working away from the workstation of a pending manual task. A third use of adaptation is to collect new information for later inspection by a user. For example, if an operation of interest occurs while the user is working elsewhere, we can log data values not normally stored that give insight into the effect of this operation on the environment. Since event triggers can be defined or changed during software execution, the user can adjust them to the changing needs of a developing situation. For example, after executing a failure repair operation, the user may define an event trigger to notify him when the repaired system is next started up.
The 3T system is a layered control architecture. Each layer of control represents a different control abstraction (low-level, task-level, plan-level). Under normal circumstances, the user intervenes at the plan-level or the top of the task-level (initiate start-up sequence). Under exceptional circumstances (anomalies, novel situations), we adapt the user interface to provide the capability to issue low-level commands to the system controllers. This adaptation can be triggered based on events indicating that upper layers of control are not operating correctly (e.g., control process stops communicating, autonomous anomaly response fails to correct a problem). Both the user and the autonomous system can determine when these exceptional conditions have occurred.

The user interface to 3T is both adaptable and adaptive. It is adaptable because it can be reconfigured manually by the user to support different task objectives. The user can change the display configuration using the capabilities listed in Table 1. Examples of the adaptable interface include selecting a set of data values to monitor in a new window and displaying an interface to execute a manual procedure.

<table>
<thead>
<tr>
<th>Table 1. User Adaptation of 3T</th>
</tr>
</thead>
<tbody>
<tr>
<td>• View of task agenda</td>
</tr>
<tr>
<td>• Inspect changes to memory</td>
</tr>
<tr>
<td>• Review message log</td>
</tr>
<tr>
<td>• Execute a manual task (single step)</td>
</tr>
<tr>
<td>• Execute an operating procedure (multi-step)</td>
</tr>
</tbody>
</table>

The user interface to 3T also is adaptive because it can be reconfigured automatically based on changes in the control tasks. Using control events from the autonomous controller to trigger reconfiguration, the interface can adapt to support a variety user tasks, including the following:

• Human Supervision of Autonomous Control
• Human-assisted Anomaly Management
• Joint Human-Computer Task Performance

We give an example below of how the 3T user interface adapts to support each of these tasks.
Human Supervision of Autonomous Control

We can use the control event triggers described previously to adapt the user interface by initiating or terminating data logging when significant control events occur. For example, the initiation of a startup or shutdown sequence for life support hardware represents a period where the hardware configuration is very dynamic. To get an accurate history of control during this transient period, the human supervisor may need to monitor additional sensors as well as increase the rate at which data are sampled. An event trigger defines which control tasks should initiate more detailed logging and which control tasks should terminate such logging. The user interface will then monitor incoming data for these control events and will change data logging based on which events are received. Figure 1 shows how to define a special data log using triggers.

A related capability we are developing is displays to inform the user of new log files created when specific events occur (including failure events). A summary timeline of control tasks will be updated with a new archive icon whenever a special data log is created. A mechanism for viewing these files (plotter, message list) can be launched by clicking on the archive icon on the timeline. Figure 2 illustrates this design concept (Schreckenghost and Thronesbery, 1998).

![Figure 1: Defining a special data log using triggers.](image1.png)

- Archives: Battery failure, Daily Logs
- View log
- View plot

![Figure 2: Accessing Data Logs Created Automatically.](image2.png)

Human-assisted Anomaly Management

During autonomous operation, the human supervisor occasionally monitors high level information summaries. When an anomaly occurs in system operation, the supervisor needs more detailed (and often different) information. We also use control event triggers to implement adaptive reconfiguration of displays, in response to an anomaly.

System anomalies can be indicated by the execution of a failure handling "method". The 3T user interface software monitors for the execution of these methods, and selects which display to show based on which method is executed. For example, the user interface can pop up a detailed history of recent control events that is hidden during nominal operations or a manual anomaly recovery procedure when an anomaly occurs (see figure 3). This helps the supervisor understand what autonomous control actions have been taken in response to the anomaly as well as what manual actions are needed.

Because the autonomous control software that we are using detects when the control actions fail to have the intended effect, we can use triggers to associate custom display configurations with the failure of a software control action. Control action failures can result from a failure of the hardware controllers, network failures affecting distributed controllers, or novel control situations not handled by the control software.

![Figure 3: Manual Anomaly Recovery Procedure.](image3.png)
Joint Human-Computer Task Performance

We developed a checklist display (Schreckenghost, 1999) for use when performing joint human-computer tasks. This display keeps a history of all low level activities performed to accomplish the joint high-level task objective. The checklist display is adaptive, because it reconfigures based on information about the control tasks executed during joint task performance.

The autonomous control software dynamically assigns tasks to either human or software “agents”. These assignments can be altered during task execution either by the human or the autonomous system. The 3T checklist interface uses information about which agent is assigned to a task to reconfigure the checklist display. As shown in figure 4, manual tasks (figure 4a) have a different colored background (blue background) than autonomous tasks (white background; figure 4b). Manual tasks also are annunciated using sound. Thus, the same task information will be displayed differently depending upon which agent was assigned to perform the task.

The checklist display also adapts automatically to the changes in the control situation by selectively providing the user with access to custom control displays, based on which tasks have executed. These custom controls can be popped up by selecting the button next to a manual task in the checklist (see figure 4a). The custom controls are only available to the user when the situated control task context specifies that a specific manual task should be executed.

User Models for Adaptation

We have identified a variety of user roles for autonomous control software. These user roles break down into two categories: system development and system operation. During system development, the interface should support the following user roles:

- Control software engineering: provide manual commanding and data feedback at each level of control abstraction (planning, task, closed-loop), access to software internals, and hardware emulation for testing.
- User interface development: provide software to customize displays for the application (e.g., build schematics, define procedures or event triggers)
- System integration and testing: issue commands to each tier of architecture, permitting manual replacement of a control tier, and review environmental feedback on the effects and timing of control activities at each control level

During system operation, the interface should support the following user roles:

- Operations supervision: access standard operating procedures, review integrated views of system operation, software
- Fault management and opportunity response: access anomaly response procedures, review activity and data histories, and issue commands at all levels of control
- Joint task performance: support task handover between manual and autonomous operations, assist computer-based manual tasks, and provide insight into autonomous activities
• System repair and maintenance: provide procedures for repair and maintenance, including selective disabling of sensors and controllers.

The user’s ability to issue commands to the control software should vary with user role. System development roles require unrestricted access to system commands, including the ability to execute arbitrary functions in the control process (see figure 5 for an example of a developer interface for system commands). Commanding available for system operation roles is typically constrained to standard operating procedures and anomaly response procedures (see figure 6 for an example of an operations interface for system commands). User changes to commands are limited to parameter changes (such as a control setpoint). Only the more catastrophic operational situations will require unrestricted access to commands.

The user’s access to information about the system also should be adapted based on the role of the user. For example, we have developed software for building schematic views of control data. System developers using this display can change the components on the schematic while system operators cannot. Conversely, system operators have easy access to data displays (real-time plots, data tables) that are of little interest while constructing the schematic view of system components. User tasks related to checkout and maintenance of control hardware require views of low-level sensors not typically needed for normal operations. Adaptation does not necessarily restrict access to information or capability. In some cases it changes the emphasis that the display places on the information or capability. For example, when executing tasks jointly with a robot (what we call traded control) we provide displays for tracking manual and autonomous activities to help in coordinating these activities. These displays highlight information about manual tasks, using high saliency encoding like color and sound.

Although we have developed different interfaces to support changes in user role, we have not yet implemented adaptation of the control interface based on user role. We expect that role will be specified by either the user or the autonomous system based on allocation of tasks. Since it is possible that a user will fulfill multiple roles simultaneously, the user interface should support as many roles as needed and should adjust as user roles change.

References


