Software Testbed for Developing and Evaluating Integrated Autonomous Systems

James Ong, Emilio Remolina, Axel Prompt
Stottler Henke Associates, Inc.
1670 S. Amphlett Blvd., suite 310
San Mateo, CA 94402
650-931-2700
ong, remolina, aprompt @stottlerhenke.com

Peter Robinson, Adam Sweet, David Nishikawa
NASA Ames Research Center
Moffett Field
Mountain View, CA 94035
650-604-5000
peter.i.robinson, adam.sweet, david.nishikawa @nasa.gov

Abstract— **To implement fault tolerant autonomy in future space systems, it will be necessary to integrate planning, adaptive control, and state estimation subsystems. However, integrating these subsystems is difficult, time-consuming, and error-prone. This paper describes Intelliface/ADAPT, a software testbed that helps researchers develop and test alternative strategies for integrating planning, execution, and diagnosis subsystems more quickly and easily. The testbed's architecture, graphical data displays, and implementations of the integrated subsystems support easy plug and play of alternate components to support research and development in fault-tolerant control of autonomous vehicles and operations support systems.**

Intelliface/ADAPT controls NASA's Advanced Diagnostics and Prognostics Testbed (ADAPT), which comprises batteries, electrical loads (fans, pumps, and lights), relays, circuit breakers, invertors, and sensors. During plan execution, an experimentor can inject faults into the ADAPT testbed by tripping circuit breakers, changing fan speed settings, and closing valves to restrict fluid flow. The diagnostic subsystem, based on NASA's Hybrid Diagnosis Engine (HyDE), detects and isolates these faults to determine the new state of the plant, ADAPT. Intelliface/ADAPT then updates its model of the ADAPT system's resources and determines whether the current plan can be executed using the reduced resources. If not, the planning subsystem generates a new plan that reschedules tasks, reconfigures ADAPT, and reassigns the use of ADAPT resources as needed to work around the fault. The resource model, planning domain model, and planning goals are expressed using NASA's Action Notation Modeling Language (ANML). Parts of the ANML model are generated automatically, and other parts are constructed by hand using the Planning Model Integrated Development Environment, a visual Eclipse-based IDE that accelerates ANML model development. Because native ANML planners are currently under development and not yet sufficiently capable, the ANML model is translated into the New Domain Definition Language (NDDL) and sent to NASA's EUROPA planning system for plan generation. The adaptive controller executes the new plan, using augmented, hierarchical finite state machines to select and sequence actions based on the state of the ADAPT system. Real-time sensor data, commands, and plans are displayed in information-dense arrays of timelines and graphs that zoom and scroll in unison. A dynamic schematic display uses color to show the real-time fault state and utilization of the system components and resources. An execution manager coordinates the activities of the other subsystems. The subsystems are integrated using the Internet Communications Engine (ICE), an object-oriented toolkit for building distributed applications.

TABLE OF CONTENTS

1. INTRODUCTION 1
2. MODELING CHALLENGES.....2
3. INTEGRATION CHALLENGES 3
4. INTELLIFACE SOFTWARE TESTBED 3
5. RESULTS.....10
6. FUTURE WORK 10
7. SUMMARY 11
REFERENCES.....11
BIOGRAPHY 12
ACKNOWLEDGEMENTS.....12

1. INTRODUCTION

Fault tolerant autonomy requires the integration of planning, adaptive control, and state estimation subsystems. For example, diagnostic systems analyze sensor readings, commands, and other data to identify faulty components and their fault states. When a fault occurs, the planning subsystem must determine whether the available resources can execute current plans and, if they cannot, how the plans should be revised.

We developed Intelliface/ADAPT to support research and development of fault-tolerant autonomous systems, This software testbed enables rapid development and testing of alternative strategies for integrating intelligent planning, execution, and diagnostic subsystems. The testbed's architecture, graphical data displays, and implementations of the integrated subsystems support easy plug and play of alternate components to support research and development in robust control of autonomous systems.

978-1-4799-5380-6/15/$31.00 ©2015 IEEE

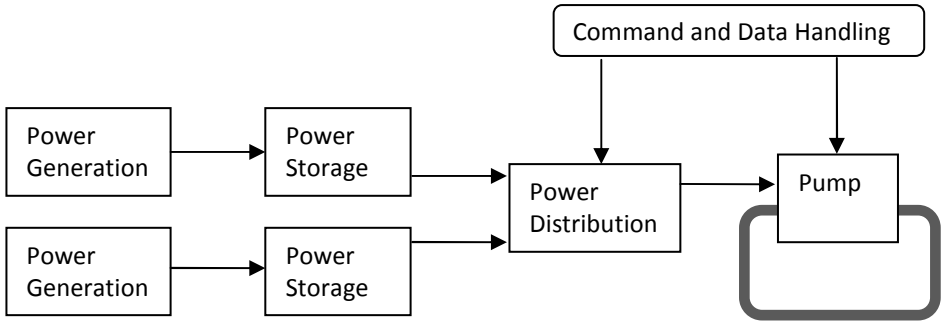

Figure 1 – Resources Depend Upon Other Resources in Order to Operate

Intelliface/ADAPT controls the Advanced Diagnostics and Prognostics Testbed (ADAPT) [1][2] at NASA Ames Research Center. ADAPT is an electrical system comprising batteries, electrical loads (lights, fans, and pumps), invertors, relays, and circuit breakers, wires, and sensors. ADAPT was originally developed to support diagnostics research, for this project (open loop system – no closed loop control). It supported a series of diagnostic competitions affiliated with the Diagnosis conference (DX), in 2009 [3], 2010, 2011, and 2013.

For this project, we hypothesized experiments and tasks to be planned and performed using ADAPT's lights, fans, and pumps, sometimes in parallel and sometimes in sequence. Intelliface/ADAPT detects and diagnoses faults manually injected into ADAPT, determines the impact of the fault on the ADAPT system's capabilities, determines whether the impaired capabilities affect the system's ability to execute the current plan, and, if necessary, generates a new plan that can be carried out by the damaged system.

An automated planning system uses heuristic search to select, configure, and schedule actions that perform tasks and achieve or maintain desired state conditions requested by users. It accepts as input a problem statement and a planning domain model, both expressed in a planning domain modeling language such as the New Domain Definition Language (NDDL) [4] or the Action Notation Modeling Language (ANML) [5]. The problem statement specifies the expected state of the world during the planning period, the tasks to be carried out, desired goal states to be achieved, and additional requirements such as deadlines for completing tasks or achieving desired states. The planning domain model specifies the world knowledge needed by the planner. Some planning domain languages, including ANML, encode this world knowledge as a library of actions, their effects on the world, and the conditions that must be true in order for the action to be executable. The planner outputs a plan that specifies actions to be performed, their timing, and the resources assigned to support each action.

An adaptive controller executes each action in the plan by selecting and sending a sequence of commands to the plant. The adaptive controller can use conditional logic to modify the selection or timing of commands to adjust how actions are performed in particular situations. Adaptive control can also monitor state variables during execution to confirm the successful achievement of target state conditions or the completion of planned actions. If execution problems occur, the adaptive controller can report these problems to a human operator and/or the automated execution manager. It shares knowledge of the system with both the planner and state estimator.

The state estimation system estimates the present state of the plant using the knowledge of the current and prior timepoints of the world based on its analysis of sensor data, commands, and other information. State estimation systems can provide diagnostic functions that analyze sensor readings, commands, and other data to identify faulty components and their fault states. A state estimation system could provide prognostics that estimate the type and timing of future failures. It could also estimate when damaged parts of the plant will be repaired and can resume operations.

2. MODELING CHALLENGES

Determining how best to model resources for planning is challenging because the model must include those resources that are directly used by activities as well as resources required only indirectly. For example, suppose that a cooling activity requires the operation of a device such as a pump. One might model the pump, in combination with the coolant loop, as a resource that provides a thermal control capability. However, in order to operate, the pump requires a power resource. To access this power source, an uninterrupted electrical path between the power storage and the pumps is needed. In addition, command and data handling are necessary to monitor and control the equipment and to configure the electrical path. Thus, this cooling activity requires a pump, coolant loop, power source, electrical distribution, and command and data handling resources. In Figure 1, each arrow indicates a dependency between a pair of resources. For example, the pump relies upon power distribution in order to operate, and power distribution relies on power storage in order to have power to deliver.

Changes in operating rules, system configuration, usage patterns, and other assumptions can require revisions to the resource model. For example, some resources may be so abundant that it is not necessary to reason about how to allocate its use. However, if the supply of a resource decreases or if demand increases, a resource that was once plentiful (and therefore possibly not even modeled) may become limited, relative to demand. In this case, it would be necessary to revise the planning domain model, so the planner can reason about the availability and ensure that plans allocate their use effectively.

For example, our planning domain model encodes an operating rule stating that a battery could be connected to at most one electrical load bank at a time. This rule exploits the fact that each battery can normally power all loads in any one load bank. However, this operating rule might become invalid if electrical loads were added to a load bank or if the battery's capacity degraded, so that all loads in the load bank, in combination, required more power than the battery could provide. Or, suppose that only one battery were operational. It might be necessary to power some of the loads in one load bank and additional loads in a second load bank. As long as the total power requirements can be met by the available battery, it might be desirable to relax the operating rule and configure the battery to drive some of the loads in each of the two load banks. However, this relaxation would require modeling the power consumption of each load and the power output of each battery to avoid overloading the battery. The modelling challenge is to define the union and intersection of knowledge categories between the planner, scheduler, adaptive controller and state estimator.

3. INTEGRATION CHALLENGES

Typically, diagnosis, planning, and execution subsystems use different models of the system, so integration usually requires translation between models. For example, the diagnosis function normally identifies physical (e.g., hardware) or logical (e.g., data, software) state of all components, both nominal and off-nominal. It also models the connectivity between components and the modes of operation. By contrast, planning and scheduling systems use resource models that are only as detailed as is necessary to support the correct selection, scheduling, and execution of planned actions and are often grounded in the hardware/logical components. In practice, planning resources are usually modeled more abstractly and coarsely than the components identified by the diagnosis function.

Developing and maintaining these interfaces by hand is time-consuming and error-prone. The various subsystems rely upon their respective knowledge bases and data models, which encode assumptions about the system's configuration, state, and operating procedures, sometimes in subtle ways. Because the diagnosis, planning, and execution systems use different models, interfaces among these subsystems must translate between these models, as shown in Figure 2. Changes in system state, system configuration, operating rules, and other assumptions can require revisions to the models and interfaces. Implementing these revisions can be labor-intensive and error-prone. Because assumptions that underlie the modeling decisions are often subtle and implicit, it is easy to change the models in ways that violate these unstated assumptions.

4. INTELLIFACE SOFTWARE TESTBED

Figure 3 shows the exchange of data among Intelliface/ADAPT software modules. The ICE Network protocol is used to integrate these modules. The following scenario demonstrates how Intelliface/ADAPT responds to a fault.

1. Initially, the *User* submits to the *Execution Manager* a file that specifies experiments to be run.

2. The *Execution Manager* sends a Plan Request to the Planner. The request comprises two parts: the planning goals and a description of the current and projected resource availabilities based on the current and projected state of the ADAPT system.

3. The *Planner* combines the planning request with the static portions of the planning domain model to create a planning problem, expressed in ANML. This planning problem is then sent to an automated planner. The Planner either returns a plan that satisfies the request, or it reports a planning failure because it was unable to generate a valid plan that satisfied the request.

4. The *Execution Manager* sends the plan to the

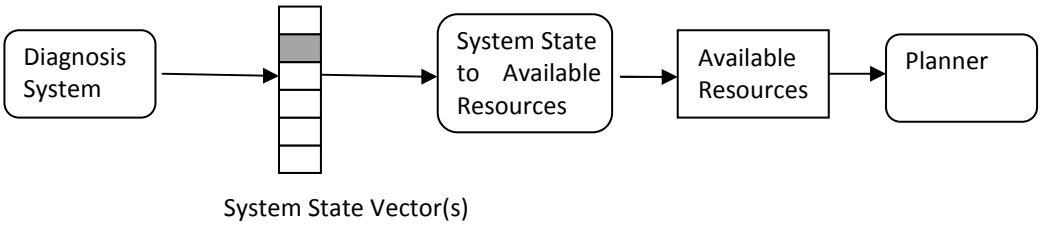

Figure 2 – Translation Between Diagnostic State Vector and Model of Available Resources

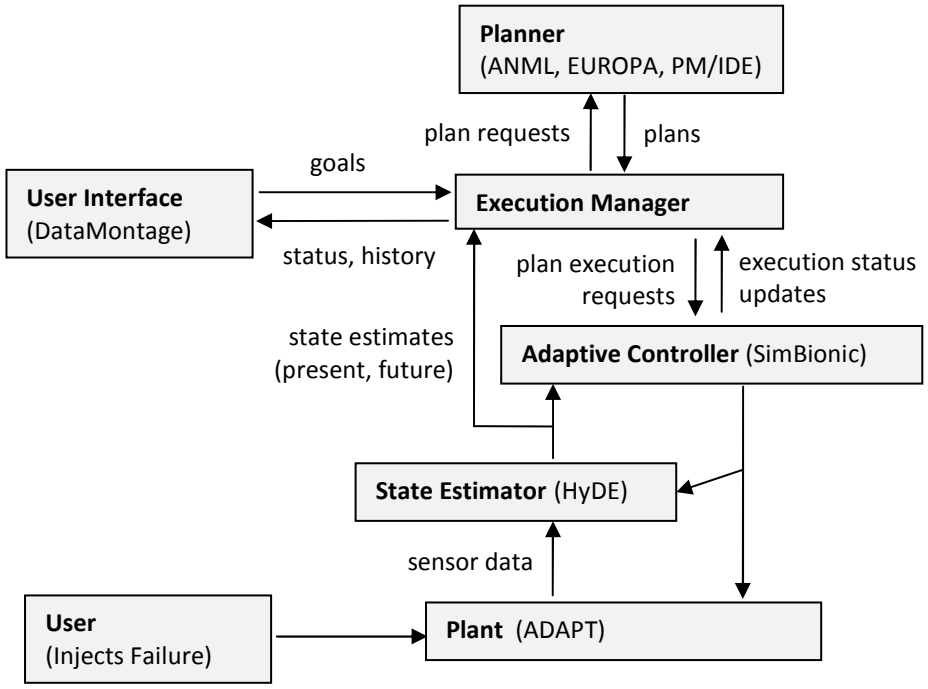

Figure 3 – Intelliface/ADAPT Architecture

Adaptive Controller for execution.

5. The *Adaptive Controller* executes each planned action by running a finite state machine (FSM) that sends commands to the Plant. When the start time of each planned action is reached, the Adaptive Controller starts the appropriate augmented finite state machine to execute the action. The Adaptive Controller can track states and events and perform actions in parallel by running multiple FSMs simultaneously,

6. During the execution of each action, the Adaptive Controller module monitors the estimated state and uses branching and looping logic to select or tailor alternate lower-level actions as appropriate. The Adaptive Controller module notifies the Execution Manager if it cannot execute a planned action or if it detected anomalous conditions during execution. For ADAPT, execution of an action that closes a relay might check the value of the relay position sensor to confirm that the relay indeed closed. Commands are also sent to the State Estimator. This information enables the State Estimator to compare sensed and commanded states, so it can detect and diagnose problems.

7. The *Plant* (ADAPT) responds to commands received by the Adaptive Controller by opening/closing relays to provide power to pumps, lights, and fans. The Plant continuously outputs data from current, voltage, flow rate, and other sensors. There are two configurations of Intelliface/ADAPT. In one configuration, the Plant is a simulation of the ADAPT system, implemented using MatLab® [6] and Simulink® [7]. In the second configuration, Intelliface/ADAPT controls the physical ADAPT testbed at NASA Ames Research Center.

8. The *User Interface* shows the current plan and sensor data in Gannt charts, timelines, and time-series graphs.

9. The *User* can inject one or more faults into the Simulink simulation of ADAPT or physically on the ADAPT hardware testbed.

10. The State Estimator estimates the state of the Plant. Specifically, the State Estimator embeds a diagnostic reasoning system that diagnoses faults in the physical ADAPT testbed based on sensor data and commands. This system was developed by NASA Ames using the Hybrid Diagnostic Engine (HyDE) [8][9].

11. The *Execution Manager* revises the resource model based on the diagnosed fault(s) and compiles a list of resources believed to have become unavailable. If there are planned actions that use or will use any of the impacted resources, the Execution Manager requests a new plan that avoids using any of the unavailable resources.

12. The *Planner* generates a new plan and sends it to the Execution Manager.

13. The *Execution Manager* sends the new plan to Adaptive Controller for execution.

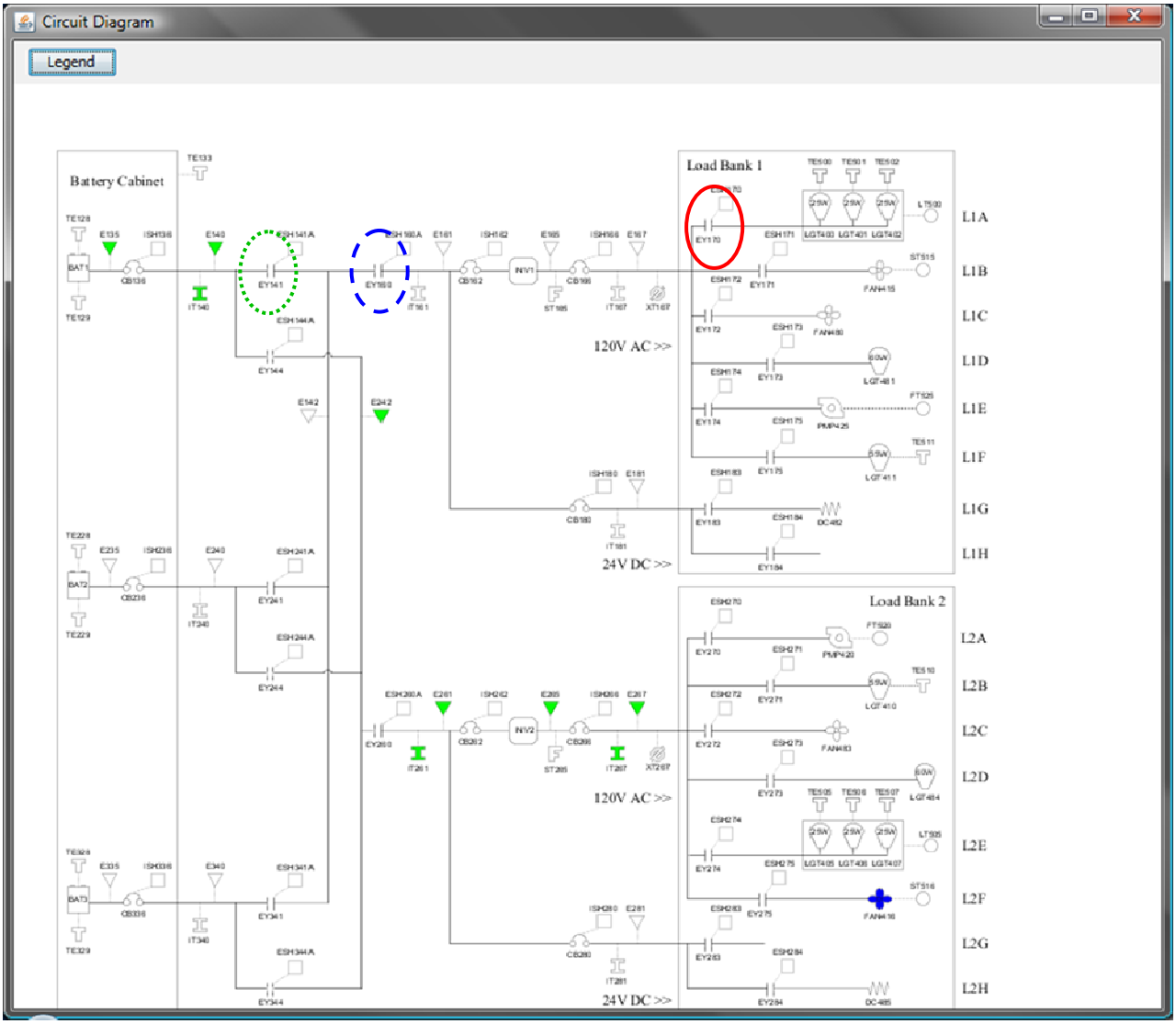

Figure 4 – Schematic of ADAPT System

14. The *User Interface* shows the new and old plans in a Gannt chart, so users can compare them.

15. The *ICE Data Distribution Layer* integrates the compoenents of Intelliface.

16. *ADAPT A/D layer* is controlled through LabVIEW® software [10].

Plant (ADAPT)

The Advanced Diagnostics and Prognostics Testbed (ADAPT) [1][2] is an experimental testbed at NASA Ames Research Center. ADAPT supports research in automated diagnosis and advanced user interfaces [11] for system diagnosis and recovery.

ADAPT is an electrical system comprising:

- 3 batteries (labeled BAT1, BAT2, BAT3) displayed on the left side of the schematic.
- Relays (IDs start with EY) that open or close to establish electrical paths between batteries and loads.
- Circuit breakers (IDs starts with CB) that are designed to trip open when excessive current flows, in order to protect the other components.
- Electrical wires that connect components. They are drawn in the schematic as lines and are assumed to never fail.
- Sensors that sense the state of the other components.
- Electrical loads such as fans, pumps, and lights, grouped into two load banks (labeled Load Bank 1

and Load Bank 2) on the right side of the schematic.

- 2 Invertors (labeled INV1 and INV2) that convert 24V DC power provided by the batteries to 120V AC power required by the loads.

Each battery is capable of providing power to all of the loads in a single load bank. Relays can be set to link either of the two load banks to any of the three batteries. Because there are three batteries, it is possible to power both load banks, even when one of the three batteries is inoperative. Thus, redundant batteries and electrical paths enable the electrical system to be configured so that loads can be powered even when faults occur.

Additional relays control whether power is supplied to individual loads within each load bank. For example, as shown in Figure 4, relay EY170, circled in red, controls whether power is supplied to lights LGT400, LGT401, and LGT402, assuming that power is supplied to Load Bank 1. Thus, to provide power from battery 1 to these lights, relays EY141 (oval, green dotted outline), EY160 (oval, blue dashed outline), and EY170 (oval, red solid outline) must all be closed and operational (not in a fault state).

NASA developed the ADAPT test bed to support research in automated diagnosis. From a diagnosis perspective, the devices were simply loads that needed to be powered by the electrical distribution system, and how the devices were used to support tasks was not important. However, in order to use ADAPT as a testbed for automated planning, we needed to invent hypothetical activities that use these devices, as well as constraints on those activities. Thus, we hypothesized experiments, each composed of tasks that used the various types of fans, lights, and pumps in ADAPT. We assumed that load bank 1 would power devices in room 1, and load bank 2 would power devices in room 2. We invented activities that use loads in each of the two load banks. For example, the activity Dim Lighting uses at least 50W of light, and Bright Lighting uses at least 100W. Minimum Cooling uses the small fan, Medium Cooling uses the large fan, and Maximum Cooling uses both fans. Some activities must be performed in room 1, some must be performed in room 2, and some can be performed in either room. Each Crew Task includes one or more activities.

For example, Crew Task A requires simultaneous execution of Minimum Cooling and Dim Lighting, and Crew Task B requires simultaneous execution of Maximum Cooling and Bright Lighting. Experiments are the largest unit of work and are each composed of an ordered sequence of crew tasks and activities. A typical planning goal might specify the performance of several experiments.

User Interface

The User Interface enables users to send planning goals and tasks to the Execution Manager. The User Interface also displays the current plan in a Gannt chart (Figure 5).

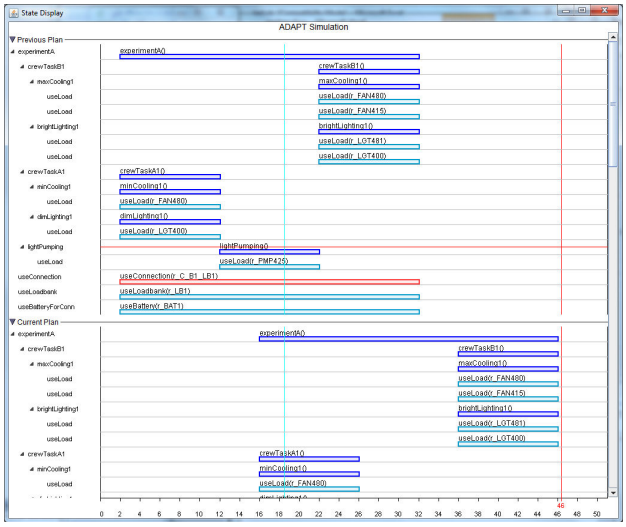

Figure 5 – Intelliface/ADAPT Gannt Chart

The Gannt Chart display enables users to compare the previous plan with the current plan after re-planning has been requested. Each horizontal bar shows the times when a particular type of action, as specified by an ANML action definition, is scheduled for execution. Violet bars represent high-level actions, which decompose into lower level actions, and light-blue bars represent lowest-level actions that have no subactions. The bar colored red represents an action that can no longer be executed because some of its required resources are no longer available. Its hierarchical timeline feature enables users to click on triangular buttons to open or close timelines. Opening a timeline displays additional lower-level timelines immediately below that show when lower-level subactions are scheduled.

In the Color-Coded Schematic Display (Figure 6), blue rectangles or polygons show resources in use, and red rectangles or polygons show resources that are inoperative, according to the State Estimator. Yellow regions show several resources identified by the State Estimator in an ambiguous diagnosis as possibly inoperative. The red circle highlights a component that has faulted. In this example, relay EY 141 has faulted. Because the connection resource between battery 1 (upper left) and load bank 1 (upper right) is inoperative, load bank 1 is powered by battery 2.

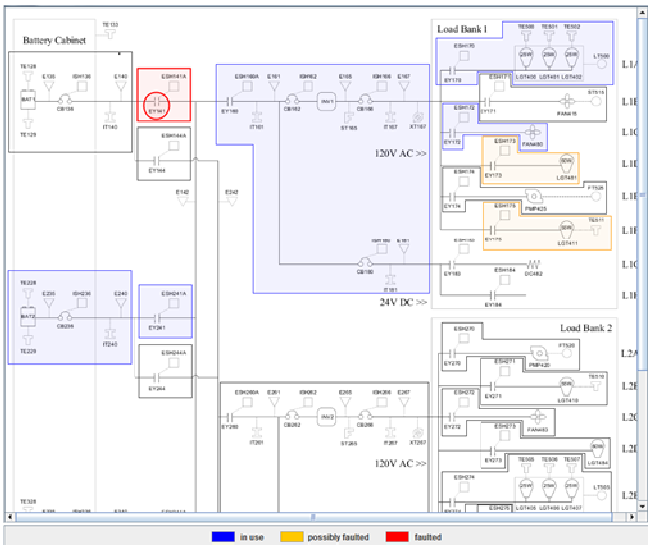

Figure 6 – Intelliface/ADAPT Color-Coded Schematic Highlights Resource Boundaries and Their Statuses

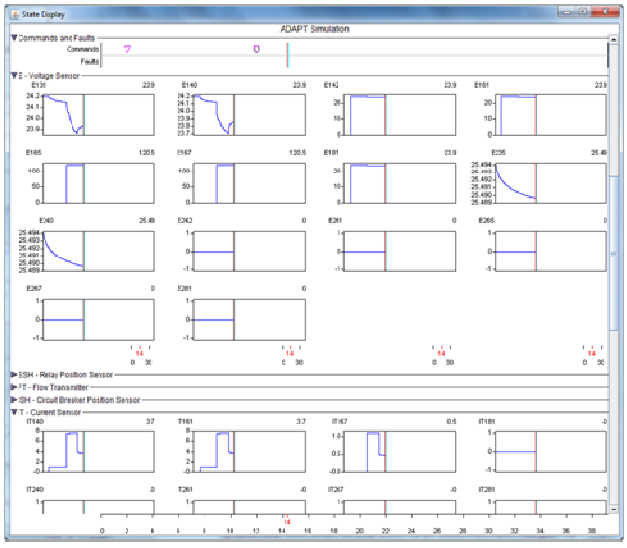

Figure 7 – Intelliface/ADAPT Graph and Timeline Display

The Graphical Data Display (Figure 7) shows sensor data and command in an interactive array of graphs and timelines. You can zoom and scroll within the graphs and timelines in unison. This display is implemented using Stottler Henke's DataMontage™ data visualization system [12].

Planner

The Action Notation Modeling Language (ANML) [5] is a planning domain modeling language developed at NASA Ames Research Center. ANML models can specify fluents that represent the state of the world in terms of discrete or continuous time-varying variables and functions. For example, the following declares a time-varying function that represents the location of each robot:

```
fluent Location atLocation(Robot r);
```

ANML actions assign values to fluents to specify effects on world, and they specify required Boolean conditions that refer to these fluents. For example, the following action specifies the condition that the battery level must

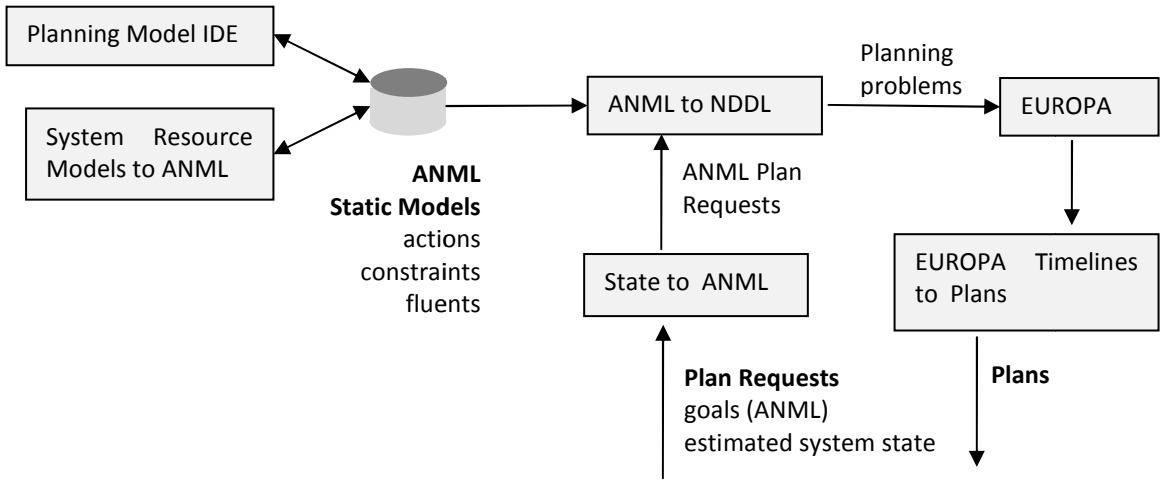

Figure 8 – Intelliface/ADAPT Planning Subsystem

be > 0 when the action begins to execute. At the end of the action's execution, an effect is that the battery level is set to 100.

```
action recharge(Robot robot) {
  [start] batteryLevel(robot) > 0;
  [end] batteryLevel(robot) := 100;
}
```

ANML actions have quantitative durations and can optionally decompose into subactions. For example, the following action decomposes into two subactions, calibrate and getImage, which are executed sequentially:

```
action takeImage(Robot robot, Location
location) {
  duration := 10;
  [all] contains
    ordered(
      calibrateCamera(robot),
      getImage(robot, location));
}
```

Because ANML is an expressive, high-level planning language, if was easier to generate ANML models, both manually and automatically, compared to lower-level languages such as the New Domain Definition Language (NDDL), used by NASA's EUROPA planner. However, no planner currently exists that accepts a sufficiently large subset of ANML as input, so we prototyped an ANML to NDDL translator, so NDDL models could be provided as inputs to the EUROPA planning system.

We developed an initial ANML model by hand that specified the static model, comprising actions, constraints, and fluent declarations, as well as the plan request, comprising planning goals, resource availabilities, and the estimated state. We then revised the ANML model so that some parts of the model could be generated automatically. Thus, the revised ANML model served as target output for two automated code generators.

The System Resource Model to ANML code generator accepts as input a resource model stored in a multi-sheet Microsoft Excel spreadsheet. This resource model provides a declarative description of the resource types, resource instances, dependencies among resources, and mutual exclusion constraints that prevent two resources from being used at the same time. The System Resource Model is assumed to change only occasionally, so it is used to create the static portion of the ANML model. The code generator translates the model into ANML statements.

The State to ANML code generator translates a list of failed components and their fault modes into ANML

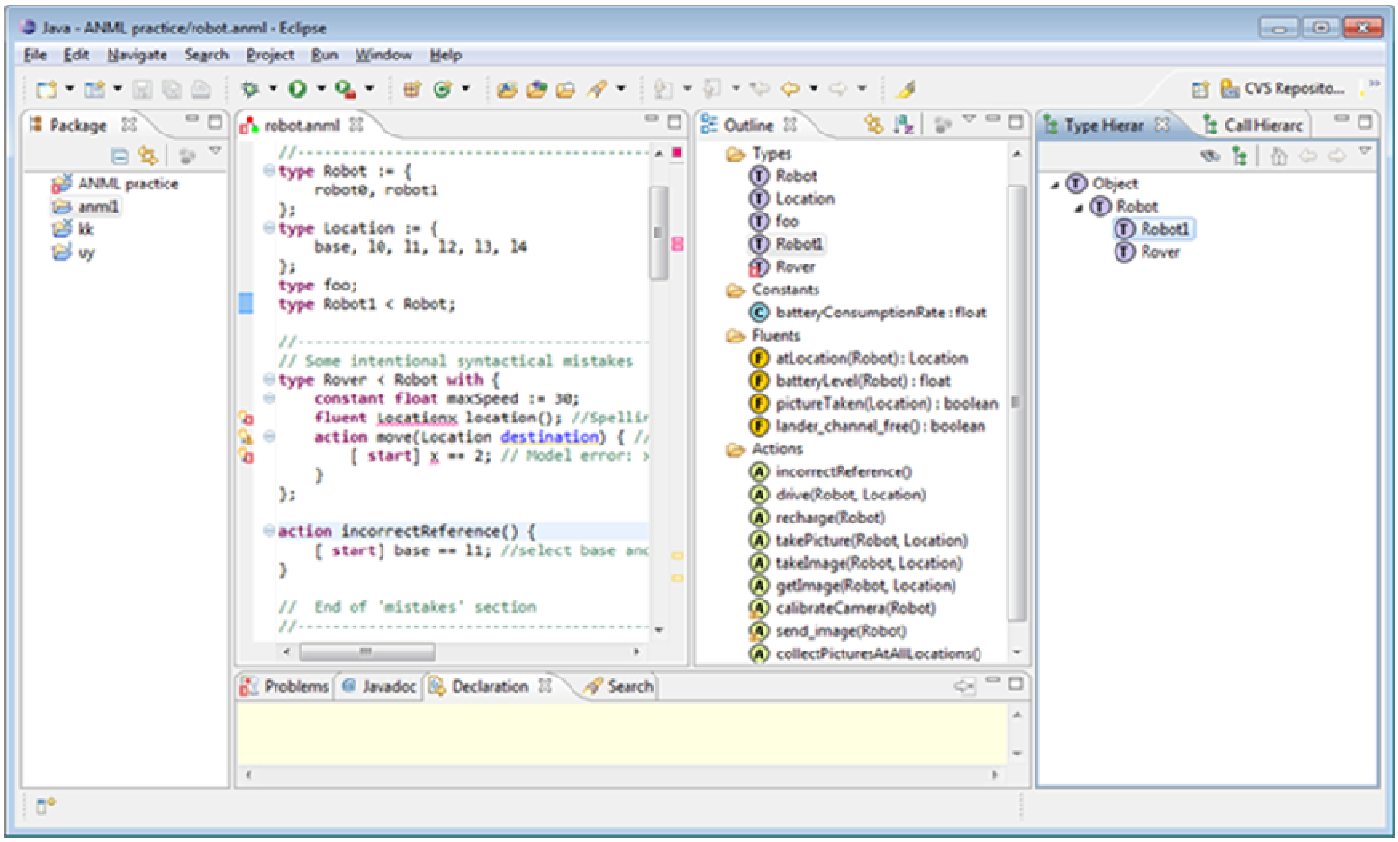

Figure 9 –Planning Model Integrated Development Environment Main Window

statements that specify the reduced availability of ADAPT resources. In the current version of Intelliface/ADAPT, every functional (non-sensor) component is associated with a resource, and a resource is assumed to be unavailable if any of its components have faulted.

The output of the EUROPA software is a collection of data structures, called timelines, that encode EUROPA's model of the planning problem and its solution. The EUROPA Timelines to Plans module translates the planning solution encoded in these timelines into a format that is understood by the Adaptive Controller.

Portions of the ANML model that cannot be generated automatically are currently created by hand. The Planning Domain Integrated Development Environment (PM/IDE) [13] is an Eclipse plug-in and collection of model visualization tools that accelerates the development of ANML models.

Adaptive Controller

An *adaptive controller* executes each action in the plan by selecting and sending a sequence of commands to the plant. The adaptive controller can use conditional logic to modify the selection or sequencing of commands in response to the state of the system or of the world to adjust how actions are performed in particular situations. Adaptive control can also monitor execution to confirm successful achievement of target state conditions or completion of planned actions. If problems occur during execution, the adaptive controller can report these problems to a human operator and/or automated execution manager.

The Adaptive Controller is based on Stottler Henke's SimBionic® intelligent agent toolkit [14]. SimBionic can run multiple hierarchical finite state machines in parallel to execute concurrent planning actions. The graphical SimBionic editor, shown in Figure 10, enables developers to quickly specify each finite state machine. Rectangle nodes can be configured to call Javascript functions or Java methods to perform actions or assign values to Simbionic variables. Ovals specify conditions that determine whether to transition from a rectangle node. If none of a node's conditions are true at a clock tick, no transition occurs, and the conditions are reevaluated at the next clock tick.

State Estimator

The *State Estimation* module estimates the present and future state of the plant and the world based on its analysis of sensor data, commands, and other information. State estimation systems can include *diagnostic systems* that analyze sensor readings, commands, and other data to identify faulty components and their fault states. Diagnostic output can be thought of as a vector in which

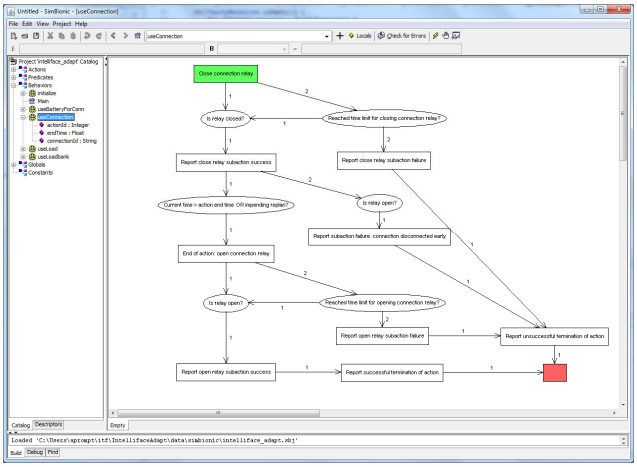

Figure 10 – SimBionic Editor

each element in the vector specifies the state (nominal or off-nominal) of each component in the system. Diagnostic systems identify faulty subsystems and components in enough detail to enable other processes (which map state to function) to determine the system's reduced functionality and performance, so risks to the crew, vehicle/habitat, or mission can be identified and plans can be revised as necessary. In addition, diagnoses must be sufficiently specific to support restoration of system functions, although diagnoses are often ambiguous due to under-instrumented hardware. For example, a specific diagnosis that identifies a replacable unit is more helpful for repair than a diagnosis that identifies the subsystem containing the unit.

The State Estimator's diagnostic function is provided by a NASA-developed diagnostic reasoning system that was implemented using NASA's Hybrid Diagnostic Engine (HyDE) [8][9]. HyDE is a model-based diagnosis engine that uses candidate generation and consistency checking to diagnose faults in stochastic hybrid systems. A HyDE model of a system consists of many connected component models.

An example of a HyDE component model used on ADAPT is shown in Figure 11. This is a screenshot of the graphical HyDE modeling tool, and this example is of a relay component, which shows many of the elements in a HyDE component model. The defined modes are represented as circles, and may include nominal and fault modes. Each mode will have constraints on the component model variables (in the figure, the input voltage, the output voltage, and the relay position, shown as orange squares with a V). The mode's constraints are only active in that mode and define the component behavior in that mode. All allowable transitions between the modes are defined and shown as arrows between the circles. The transitions may have guards enabled by commands to the component (in the figure, the green box with a squiggle in it), by values of the component's variables, or they may be unguarded. The unguarded

transitions are treated as failure transitions by HyDE. HyDE will perform a diagnosis when it detects that the constraints in the current mode of any component are no longer consistent, and it will search for failure transitions in all components which lead to failure modes in which the constraints are currently consistent. A longer description of HyDE and the HyDE diagnostic system used on ADAPT is given in [9].

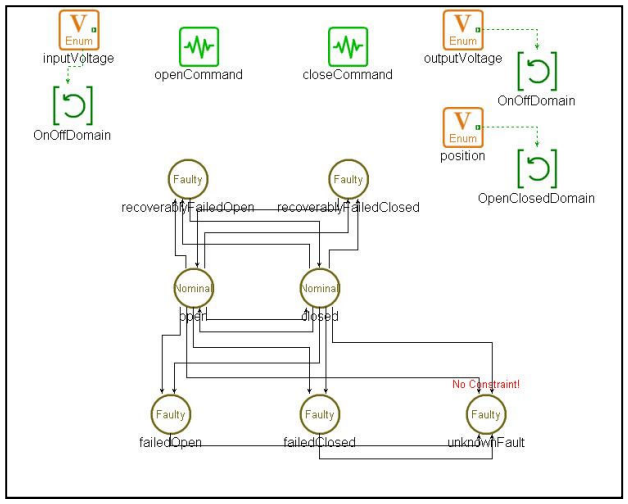

Figure 11 – Screen Capture of the HyDE Relay Component Model Used on ADAPT. Note the Use of Finite States for Both Nominal and Off-Nominal State Definition.

5. RESULTS

We implemented two configurations of Intelliface/ADAPT. The first configuration monitors and controls a Matlab/Simulink simulation of the ADAPT testbed, developed by NASA. Faults are injected using the Matlab/Simulink user interface. The second configuration monitors and controls the ADAPT hardware testbed in the ADAPT Lab at NASA Ames Research Center. Replacing the ADAPT simulation with the ADAPT hardware testbed was easy because both systems used the ICE toolkit and common interfaces. The testbed implements test scenarios in which the system diagnoses and replans in response to injected failures of fans, lights, and pumps as described in section 4.

6. FUTURE WORK

The following list describes ways in which Intelliface/ADAPT could be enhanced:

- *Preserve Work Already Performed* - When a fault occurs, part of an experiment may have already been completed. The current implementation of Intelliface/ADAPT re-executes each experiment from the beginning. An enhanced version of Intelliface/ADAPT could preserve work already performed. However, this type of enhancement would require Intelliface/ADAPT to know whether partially-completed experiments can be safely resumed after a delay.

- *Minimize Changes to the Plan* - When a fault occurs, Intelliface/ADAPT currently generates a new plan from scratch. When re-planning, it is sometimes preferable to minimize changes to the previous plan.

- *Handle Complex Mappings between Faults and Resource Changes* - Intelliface/ADAPT currently associates each electrical component with one resource. Each resource is assumed to be either fully operational or completely non-functional. If a component enters any of its fault modes, its associated resource is assumed to be malfunctioning and incapable of being used in any way. Also, each resource is assumed to be used exclusively for one task, or it is assumed to be sharable among any number of tasks. Although these were reasonable assumptions for ADAPT, for other plants, a component failure might reduce the quality or quantity of a type of resource rather than make the resource completely unavailable. Supporting this capability would require the autonomous system to reason using more complex mappings between component faults and changes in resource quantity and quality.

- *Handle Ambiguous Diagnoses* - If the State Estimator returns more than one possible, competing diagnosis, Intelliface/ADAPT currently makes the pessimistic, simplifying assumption that all of the components associated with any of the possible diagnoses have faulted. This assumption may be too conservative because there might exist a plan that can be executed if either of two resources is unavailable, whereas there might not be any plan that can be executed if both resources are unavailable. It also assumes that fault signatures are additive and not mutually exclusive.

- *Auto-Generate Diagnostic Models* – Intelliface currently generates portions of the planning domain model from an application-neutral model of the system's resources. An enhanced version of Inteliface/ADAPT might auto-generate parts of other models used by Intelliface/ADAPT, such as the State Estimator's diagnostic model, based on machine-readable descriptions of the plant. This will require a common domain core that can be shared among most of the tools. Achieving this will allow for a reduction in locations where all the domain knowledge is formalized. This will in turn reduce the time for Intelliface to reconfigure all of its components when a new hardware/software configuration is defined.

- *Plan and Execute Diagnostic Actions* - Intelliface/ADAPT uses HyDE to diagnose faults by analyzing sensor data and commands that are reported continuously by ADAPT. When this diagnostic method is unable to identify the faults unambiguously, it might be possible to plan and execute diagnostic actions that experimentally determine the fault. Since some of these actions might compete with mission tasks for resources or attention by the crew, it may be necessary to plan diagnostic actions in a way that minimizes interference with those tasks.

- *Plan Using Prognostics* - The Intelliface/ADAPT state estimation system could be enhanced in the future to provide *prognostics* that estimate the type and timing of future failures. A state estimation system could also estimate when repairs will complete, enabling a subsystem to resume operations. This information could be used by the Planner to plan future actions more effectively.

7. SUMMARY

Intelliface/ADAPT is integrated with NASA's Advanced Diagnostics and Prognostics Testbed (ADAPT). It detects and diagnoses faults injected into ADAPT, determines the impact of each fault on the ADAPT system resources, determines whether the reduced resources affect the system's ability to execute the current plan, and, if necessary, generates a new plan that can be carried out by the damaged system. Intelliface/ADAPT integrates several NASA-developed technologies including the Action Notation Modeling Language (ANML), EUROPA planning system, and a HyDE-based system that diagnoses problems in ADAPT.

The architecture, data displays, and baseline implementation provided by the Intelliface/ADAPT testbed enable development, demonstration, and evaluation of new strategies for integrating intelligent subsystems to create fault-tolerant autonomous systems.

REFERENCES

[1] Advanced Diagnostics and Prognostics Testbeds (ADAPT) Web site: http://ti.arc.nasa.gov/tech/dash/diagnostics-and-prognostics/adapt-diagnostics/

[2] S. Poll, A. Patterson-Hine, J. Camisa, D. Garcia, D. Hall, C. Lee, O. Mengshoel, C. Neukom, D. Nishikawa, J. Ossenfort, A. Sweet, S. Yentus, I. Roychoudhury, M. Daigle, G. Biswas, and X. Koutsoukos. Advanced Diagnostics and Prognostics Testbed. 18th International Workshop on Principles of Diagnosis, pp. 178-185, May 2007.

[3] Kurtoglu, T., Narasimhan, S., Poll, S., Garcia, D., Kuhn, L., de Kleer, J., Gemund, A., & Feldman, A. (2009). First international diagnosis competition-DXC'09. *Proc. DX'09*, 383-396.

[4] EUROPA Web site: http://code.google.com/p/europa-pso/wiki/WhatIsEuropa.

[5] Smith, D., J. Frank, W. Cushing, "The ANML Language". ICAPS-08 Poster Session.

[6] The MathWorks, Inc. (2014) MATLAB User's Guide (R2014b). 3 Apple Hill Drive, Natick, MA.

[7] The MathWorks, Inc. (2014) SIMULINK User's Guide. 3 Apple Hill Drive, Natick, MA.

[8] HyDE Web site: http://ti.arc.nasa.gov/tech/dash/diagnostics-and-prognostics/hyde-diagnostics/

[9] Sweet, A. (2008) Testing HyDE on ADAPT, NASA technical memo 2008-214570, Jan 2008.

[10] National Instruments. Labview [Computer software]. Austin, TX.

[11] Spirkovska, L., et al, (2011) Advanced Caution and Warning System, Final Report - 2011. NASA Technical Report NASA/TM-2013-216510.

[12] DataMontage Web site: http://www.stottlerhenke.com/datamontage/

[13] Ong, J., E. Remolina, D. E. Smith, M. S. Boddy (2013) A Visual Integrated Development Environment for Automated Planning Domain Models. AIAA Space 2013 Conference. San Diego, CA, Sept. 10-23, 2013.

[14] Fu, D., R. Houlette, J. Ludwig (2007) An AI Modeling Tool for Designers and Developers. 2007 IEEE Aerospace Conference Proceedings. Big Sky, Montana, March 4-9, 2007.

BIOGRAPHY

James Ong *received an MBA from Boston University, an MS in computer science (artificial intelligence) from Yale University, an MS in electrical engineering and computer science from UC Berkeley, and a BS in electrical engineering from the Massachusetts Institute of Technology. At Stottler Henke, he leads the development of tools that support automated planning, autonomous systems, and robotics research at NASA. He also leads the development of mission planning, decision/task support, and training applications and tools for the Department of Defense.*

Emilio Remolina *received an M.A. in Mathematics from Universidad de Los Andes, Colombia and a Ph.D. in Computer Science (artificial intelligence) from the University of Texas at Austin. For NASA, he led the development of integrated development environments for NASA's PLEXIL and ANML planning languages, and he contributed to the development of the Intelliface software testbed. He leads the development of the Visual Planning Execution and Review (ViPER) tool for military planning, and he has led the development of simulation-based intelligent tutoring systems for the Department of Defense.*

.

Axel Prompt *is a Software Engineer at Stottler Henke. He received a B.S. in Electrical Engineering and Computer Science from UC Berkeley in 2012. He contributed to the development of the Intelliface software testbed and an integrated development environment for NASA's ANML planning language.*

Peter Robinson *received a B.A. in Computer Science from U.C. Santa Cruz in 1987 He wrote the NASA SBIR subtopic integrating diagnosis and planning. He also was the Contract Officer's Representative (COR) for the Intelliface Phase I and Phase II projects.*

Adam Sweet *is a research engineer in the Diagnostics and Prognostics group at NASA Ames Research Center. He graduated with an MS in Mechanical Engineering from UC Berkeley in 1999, and has worked at Ames ever since. His project experience encompasses robotics, hybrid system simulation, model-based diagnosis, and flight software development for nanosatellites.*

David Nishikawa is a Computer Engineer in the Intelligent Systems Division (Code TI) at NASA Ames Research Center, and was the primary software designer for the ADAPT system. He is a graduate of CSUH (now East Bay) and has been at NASA for over 30 years. He began his career in flight research support, worked in the National Full Scale Aerodynamics Complex (NFAC), and now supports the Diagnostic and Prognostics group.

ACKNOWLEDGEMENTS

Intelliface was developed with funding from NASA under contract NNX12CA06C. Development of PM/IDE and the ANML to NDDL code generator was funded by NASA under contract NNX11CC18C. Development of DataMontage was funded in part by the Department of Defense under contract DAMD17-02-C-0030 and by NASA under contracts NNX09CA07C and NNX13CA04C.