Applying Model-based Diagnosis to a Rapid Propellant Loading System

Charlie Goodrich*. Sriram Narasimhan, Matthew Daigle**, Walter Hatfield*, Robert Johnson*, Barbara Brown***

*NASA Kennedy Space Center USA (e-mail:
charles.h.goodrich@nasa.gov,walter.h.hatfield@nasa.gov,robert.g.johnson@nasa.gov),
**University of California Santa Cruz USA (e-mail:sriram.narasimhan-1@nasa.gov, matthew.j.daigle@nasa.gov),
***NASA Ames Research Center USA (email:barbara.l.brown@nasa.gov)

Abstract: The overall objective of the US Air Force Research Laboratory (AFRL) Rapid Propellant Loading (RPL) Program is to develop a launch vehicle, payload and ground support equipment that can support a rapid propellant load and launch within one hour. NASA Kennedy Space Center (KSC) has been funded by AFRL to develop hardware and software to demonstrate this capability. The key features of the software would be the ability to recognize and adapt to failures in the physical hardware components, advise operators of equipment faults and workarounds, and put the system in a safe configuration if unable to fly. In December 2008 NASA KSC and NASA Ames Research Center (ARC) demonstrated model-based simulation and diagnosis capabilities for a scaled-down configuration of the RPL hardware. In this paper we present a description of the model-based technologies that were included as part of this demonstration and the results that were achieved. In continuation of this work we are currently testing the technologies on a simulation of the complete RPL system. Later in the year, when the RPL hardware is ready, we will be integrating these technologies with the real-time operation of the system to provide live state estimates. In future years we will be developing the capability to recover from faulty conditions via redundancy and reconfiguration.

1. INTRODUCTION

Rapid propellant loading deals with transferring large amounts (~50,000 gallons) of cryogenic propellant from a storage tank to a vehicle tank. The vehicle tank, though large, is relatively fragile. The vehicle designers have prescribed strict pressure limits for the tank and associated valves. When the cold cryogen liquid first contacts the warm tank and connecting piping, it boils and vaporizes, generating large volumes of gas. This translates to high flow rates and pressures. For this reason, cryogen loading is generally done very slowly using a very structured sequence of events: chilling the cryogen pumps and associated transfer pipes, chilling the vehicle with cold gas, followed by small quantities of cryogen liquid, slow filling of the vehicle with liquid to a predefined level, fast filling the tank to near operating level, slowly topping off the tank to the maximum quantity and incremental replenishment of cryogen liquid that constantly boils off from the vehicle tank prior to launch.

Normally, the process takes about 2-3 hours and a crew of 4 to 8 engineers watches all the equipment on the ground and in the vehicle during this sequence. The engineers are familiar with the physics of cryogenic liquid flow, the pressures and temperatures involved, typical faults that occur with temperature and flow measurements, stuck valves, faulty position indicators, high or low pressure measurements and occasional blown fuses and open circuit breakers. They stand ready to diagnose problems and devise workarounds for faults to complete the loading process and launch the vehicle.

The US Air Force would like to build some of the engineering expertise of the cryogen loading team into the computer system that controls the loading operation. They would like to launch the rocket with a crew of three sergeant-level personnel who may have no engineering experience and limited training in cryogen operations. To this end, the US Air Force Research Laboratory (AFRL) has funded a prototype project at NASA Kennedy Space Center (KSC) to explore rapid cryogenic propellant loading (RPL). The task has two primary goals:

1. Reduce the entire loading operation time to 45 minutes (as opposed to the normal 2-3 hours).

2. Develop software that automates the loading of the vehicle tank with cryogens with minimum operator supervision.

The project is expected to proceed in three phases. In Phase I a small-scale hardware test bed will be built. Supporting software that performs passive diagnosis and some implicit fault accommodation and recovery will also be developed. Phase II will deal with building a larger scale hardware which would include fuel and oxidizer and perform automated umbilical mating and leak detection. Phase III would be a flight demonstration of the developed hardware and software.

The software task of the Phase I project had the following objectives:

1. Development of a preliminary cryogenic test configuration
2. Development of use cases for the preliminary configuration
3. Development of a simulation model of the preliminary cryogenic test configuration (a "mini-model") to conduct tests using the preliminary configuration and use cases
4. Evaluation of the fault detection isolation and recovery (FDIR) application development tools using the mini-model and use cases
5. Development of the RPL Control Architecture
6. Update of the mini-model to the demonstration test bed simulation
7. Testing of the selected Control Architecture with the demonstration test bed simulation.
8. Integration of the Control Architecture with the actual cryogenic test bed hardware
9. Testing of the Control Architecture with the cryogenic hardware

Requirements 1-4 were completed and demonstrated to AFRL in December 2008. In this paper we describe the work that was done in completing the requirements, the setup for the demonstration and some representative scenarios and results.

Our approach to solving this problem was to use model-based technologies where the models themselves would be component-based. The primary reason for this approach was that we would be testing diagnosis algorithms on different systems from the same domain (preliminary configuration, complete RPL configuration, actual hardware). Moreover the models were built while the system configuration was being finalized. Hence component-based models would allow us to compose different system configurations by instantiating appropriate components and connecting them according to the system structure. This approach also allows us to tie anomalous behavior to component faults in a straightforward way.

We selected MATLAB/Simulink® as the language of choice for the simulation model. To support requirement 4, it was decided to evaluate two model-based diagnosis technologies, namely, Hybrid Diagnosis Engine (HyDE) (Narasimhan and Brownston 2007) and Knowledge-based Autonomous Test Engineer (KATE) (Jamieson et al. 1985, Scarl et al. 1987, Goodrich 1995). These two technologies were chosen because of the availability of developers of the two systems. Other technologies may be considered in future years.

The rest of the paper is organized as follows. Section 2 describes the mini-model configuration and faults selected for testing based on use-cases analysis (Requirements 1 and 2). Section 3 describes the MATLAB/Simulink models with fault injection capabilities that were used to generate data for testing (Requirement 3). Section 4 describes HyDE and how models of the mini-model were built in HyDE. Section 5 describes KATE and the efforts to create the mini-model

configuration in KATE. Section 6 presents the experimental setup (which was exactly what was demonstrated) that was used to test HyDE and KATE on nominal and faulty data from the simulation and the results from a few representative scenarios.

2. MINI-MODEL CONFIGURATION

2.1 Preliminary Test Configuration

Fig. 1 presents the preliminary cryogenic test configuration (referred to as the mini-model) used for evaluating the model-based diagnosis technologies. The storage tank and vehicle tank both have vent valves (C3 and C4) to regulate the ullage pressure. Three transfer lines connect the tanks to regulate flow between. The main transfer line (top most) and the auxiliary transfer line (second from top) contain pumps that can be controlled to regulate the flows at various rates depending on pump RPM. Both lines contain manually operated valves (G1, G2), remotely operated valves (C1, C2) and flow control valves (A1, A2) to control the flow. The manually and remotely operated valves can be full open or full closed only, whereas the flow control valves can be controlled to be open to any desired percentage. A re-circulation line (bottom most) controlled only by a flow control valve is used to send back excess liquid to the storage tank. The path to the vehicle tank is also controlled by a flow control valve to regulate the ratio of flow going to the vehicle tank and the flow re-circulating to the storage tank.

2.2 Definition of Nominal Operation and Fault Scenarios

Experienced propellant loading engineers stipulated the components and their capacity and likely scenarios for nominal loading operation satisfying the requirements put forth by AFRL. For the sake of the preliminary testing, we decided to ignore the chill down phase, assuming instead that all components were already ready for transferring cryogens. The stages of nominal operations were identified as (i) initialization (ii) slow fill (iii) fast fill (iv) topping (v) replenish and (iv) drain-back in the event of a launch scrub. It was assumed the system would follow this pre-determined sequence except when recovery actions required changes to the sequence.

Engineers also specified likely sensor and component failures based on general knowledge about the components involved and also on prior experience with liquid oxygen loading for the Space Shuttle. The typical faults were identified as sensor (pressure and flow) failures, pump failures and valve (stuck open or stuck closed failures). Some other common failures involving power and data acquisition modules were not included for this initial configuration.

Since we were using a simulation to generate data we had the flexibility of defining which values of the system could be sensed. Typical quantities like flows, pressures, valve positions and tank levels were included corresponding to typical measurement points in the actual hardware. Selected observation points are marked with white circles in Fig. 1.
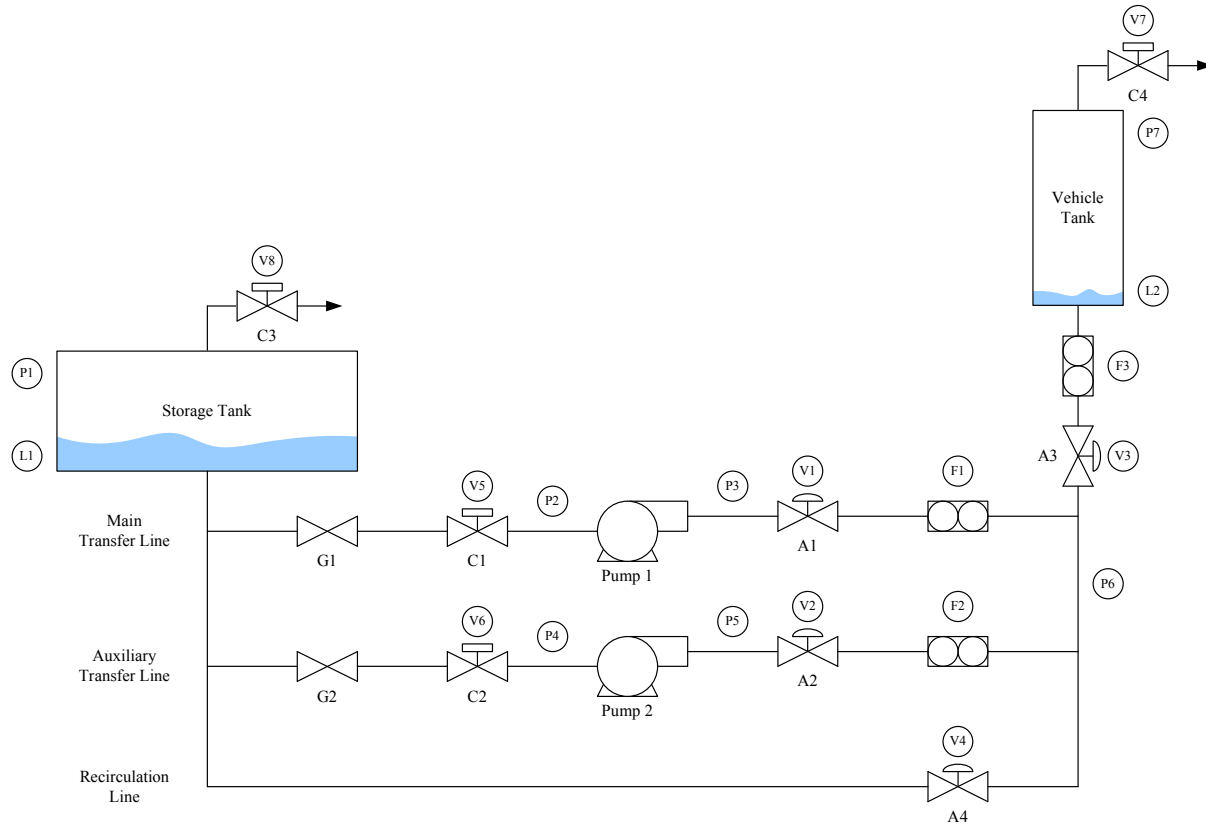
Fig. 1. Mini-model configuration

We were able to take advantage of the flexibility to test the sensitivity of model-based diagnosis tools when faced with different sets of sensor observations. In the future, we hope to use this as a valuable tool to determine sensor placement and diagnosability.

3. MATLAB SIMULATION

In order to evaluate our model-based diagnosis tools, we developed a simulation of the system. The simulation serves as a virtual test bed where we can easily study a large number of fault scenarios to develop our diagnosis models and test our algorithms. Clearly, if an accurate and realistic simulation is developed, then less work is required in migrating diagnosis algorithms to the actual system.

To this end, we developed a physics-based simulation of the system in MATLAB/Simulink. We adopted a component-based modeling paradigm, where parameterized simulation models of generic components including tanks, valves, pipes, pumps, and sensors were developed within a component library. The overall system model is constructed by instantiating the different components from the component library, specifying their parameters, and connecting the components to each other in the appropriate fashion. The top level of the Simulink model which corresponds to the mini-model schematic is illustrated in Fig. 2.

Each component model includes their associated fault modes. For example, a control valve may become stuck at a particular position, or its orifice may become partially blocked. The fault mode, time of fault injection, and fault magnitude (where applicable) can all be specified. In general, each fault mode is mapped to a change in component mode and a fault-dependent magnitude parameter. Because each fault mode is parameterized within the Simulink model, a fault can be injected programmatically (i.e., the fault mode, injection time, and magnitude are specified) either at the beginning of the simulation, or while the simulation is running. The component model of a valve with fault injection capabilities is shown in Fig. 3.

4. HYDE

4.1 Description

Hybrid Diagnosis Engine (HyDE) is a model-based reasoning engine for hybrid (discrete + continuous) diagnosis. HyDE is able to diagnose multiple discrete faults using consistency checking between prediction from hybrid models and sensor observations. HyDE models are component-based and are similar to simulation models. Component models are expressed in terms of behavior in different modes of operation (called locations) including faulty modes with transitions representing the conditions under which the system changes locations. The system model is composed by connecting shared variables between the various components.
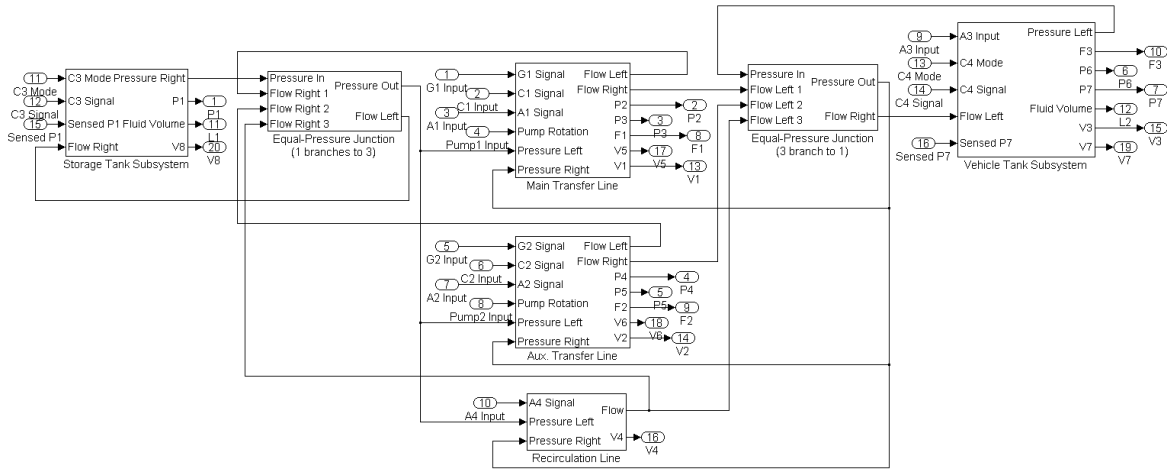
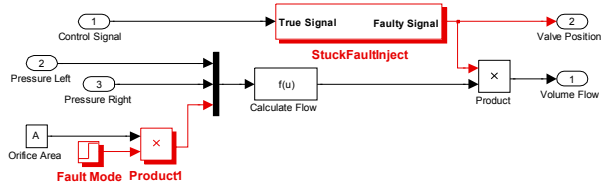Fig. 2. Simulink model of mini-model configuration

Fig. 3. Valve component model with fault injection
capabilities

The HyDE reasoning engine uses the model both for simulation and candidate generation. A set of consistent candidates, each of which may include multiple hypothesized faults, represents the diagnosis. At each time step the candidates are tested for consistency against sensor observations (using simulation) and if found inconsistent new candidates are generated (using candidate generation). For details please refer to (Narasimhan and Brownston 2007).

4.2 HyDE Models of Mini-model Configuration

The HyDE model for the mini-model configuration (Fig. 4) was constructed to look like the Simulink model described earlier. Each component and subsystem in the Simulink model is replicated in the HyDE model. Variables inside each component are also exactly the same as in the Simulink model. This also allowed the connections between components via shared variables to be duplicated in the HyDE model. The behavior within each component had to be modified so as to be represented as hybrid automata (finite state automata with equations in each state). This was done by identifying the modes of operation of the components including the fault modes based on fault injection capabilities. The equations within each mode can be determined by substituting appropriate parameter values corresponding to the different modes as specified in the MATLAB model. The HyDE model of a valve is illustrated in Fig. 5.

The reasoning parameters for HyDE had to be fine-tuned to provide the best performance. The fixed-step Runge-Kutta ODE solver was implemented to simulate behavior across time-steps. There were still some numerical problems since numerical precision used in MATLAB was far superior. As a result the tolerance for fault detection had to be increased (the default was 2%) for some observed variables. For fault isolation the maximum fault size was set to 3 (indicating a maximum of 3 faults in the system). The diagnostic results from HyDE are presented in Section 6.
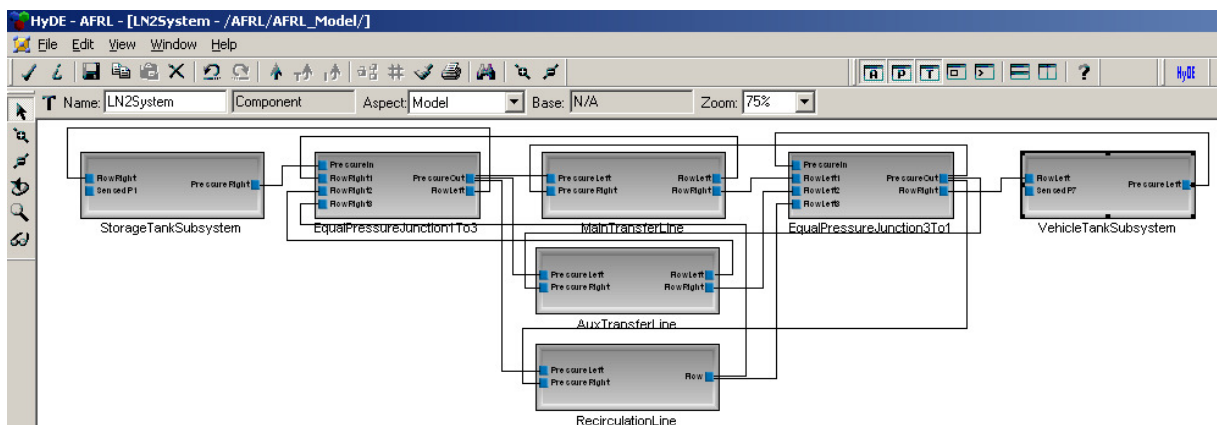

Fig. 4. HyDE model of mini-model configuration

5. KATE

5.1 Description

KATE is a C++ model-based diagnosis system developed by NASA circa 1993 at Kennedy Space Center (Jamieson et al. 1985, Scarl et al. 1987, Goodrich 1995). It was developed to employ Fault Detection, Isolation and Recovery (FDIR) technology to deal with component faults during the Space Shuttle Countdown.

The Shuttle's Launch Processing System (LPS) was programmed to sequence through normal operations and "hold" the countdown if unanticipated sensor readings are encountered. KATE was developed to help Shuttle engineers decide whether such an exception is significant, diagnose the problem, and decide whether or not to continue the countdown.

KATE is a generic software shell for performing model-based FDIR. Each KATE application requires a knowledge-base or model of the system. The same model-based reasoning algorithms are used for each application. The main reasoning systems are:

- Simulation: using a component-based model of a system, the simulation subsystem generates a prediction of behavior of the application system.

- Monitoring: by comparing the predicted sensor values generated by the simulation subsystem with the actual sensor readings of the application system, the monitoring
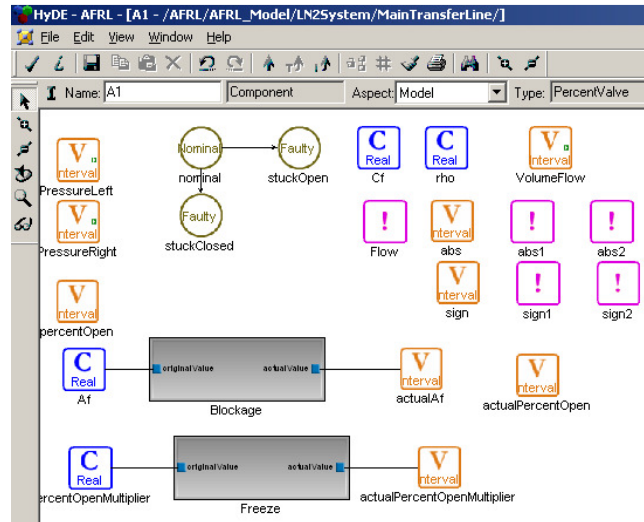

Fig. 5. HyDE model of Valve component

subsystem is able to provide system health status.

- Diagnosis: When a discrepancy between the predicted and actual sensor values is detected, the diagnosis subsystem exploits the structural and functional relationships of the components in the model to determine the failed component(s) that would explain the discrepancy.
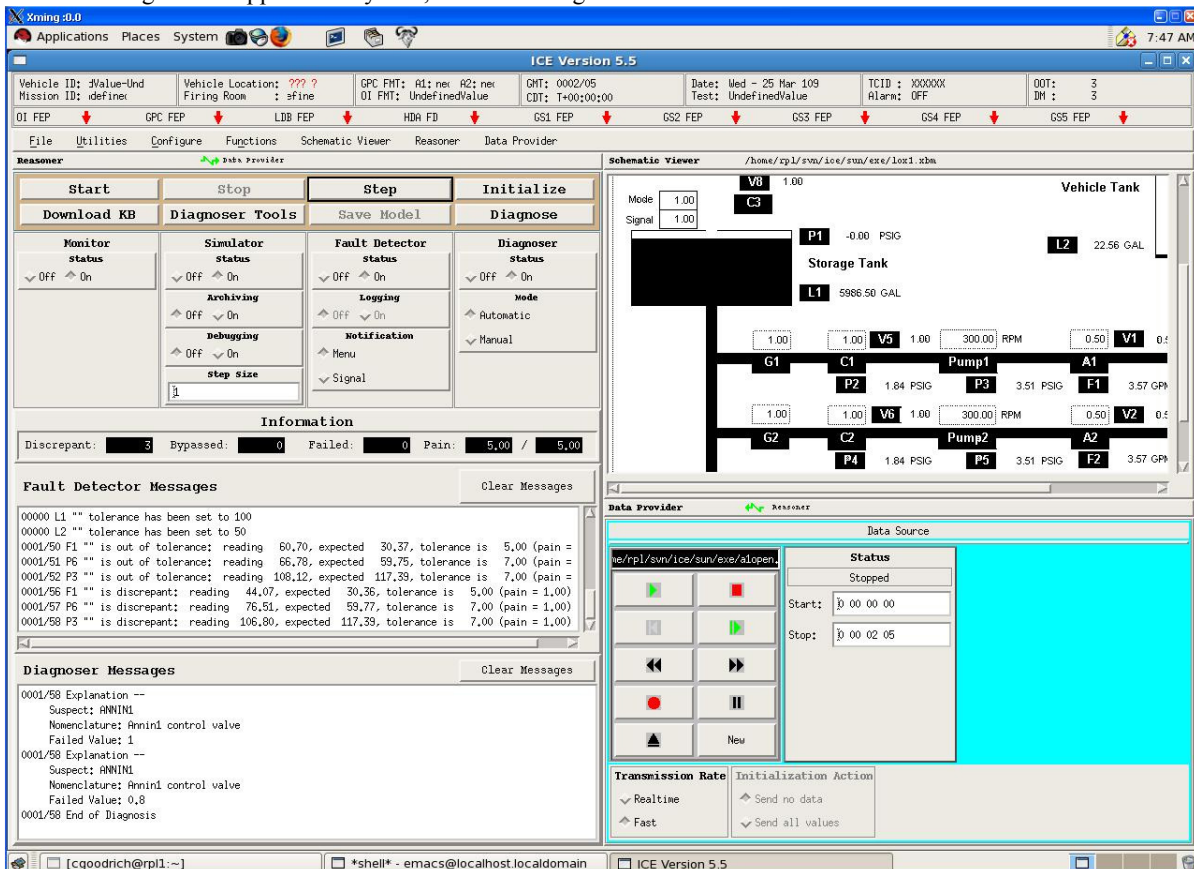

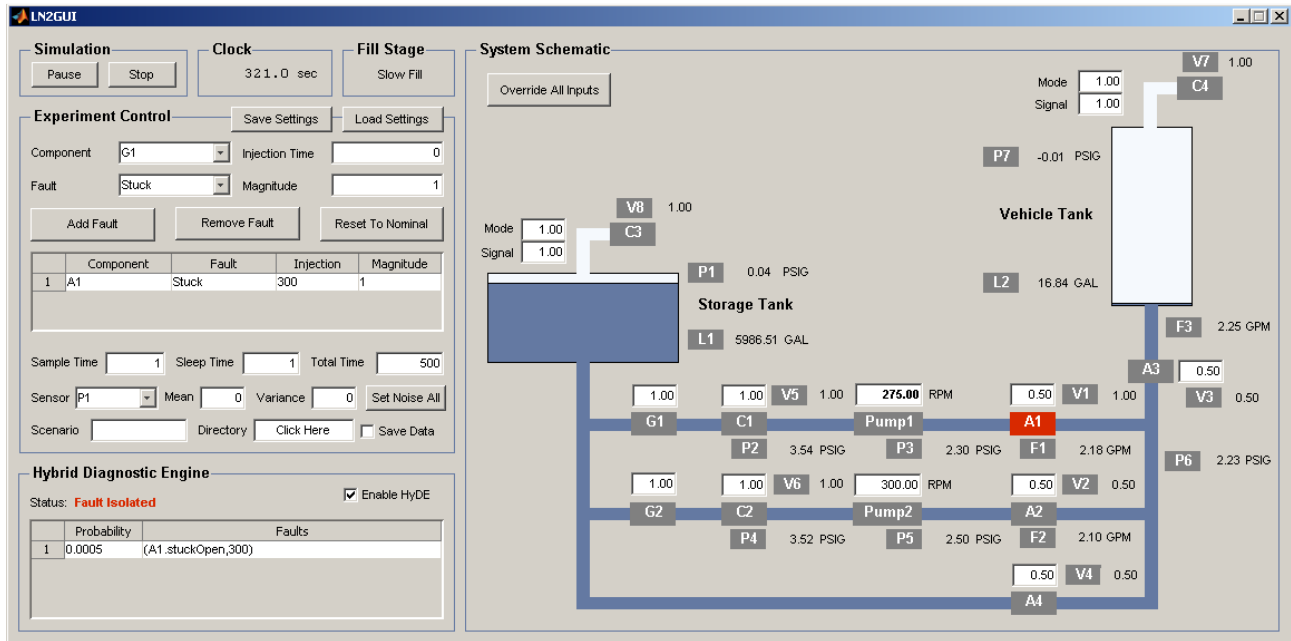Fig. 6. KATE in action for mini-model configuration

Fig. 7. GUI used in demonstration

5.2 KATE models of mini-model configuration

The KATE model for the mini-model configuration was built by identifying all the components in the mini-model and considering their likely failure modes. The components include propellant tanks, pumps, valves and sensors. The mini-model uses component behavior defined for the Space Shuttle' KATE LOX application developed in 1986 and refined in 1994. Additional component models were used from a USAF Advanced Launch Operations demonstration developed in 1993. KATE's mini-model was constructed by defining connections between components selected from these pre-existing libraries. Some modifications of the C++ code in the libraries were made to account for the parallel pumping defined in the mini-model.

The resulting "flat file" model simply specified mini-model components from the libraries and connections between the components. KATE was able to diagnose faults by reading data generated by the Matlab simulation. The diagnostic results from KATE are presented in the next section.

6. EXPERIMENTAL SETUP, DEMO AND SCENARIOS

In order to demonstrate the simulation and diagnosis capabilities we created command line and visual interfaces. Using the command line interface, the user can specify an experiment, parameterized by a set of faults to inject, the amount of noise for each sensor, and the sample time of the sensors. The simulation is then automatically executed with these parameters, and the resulting input and output data is written to files for input to the diagnostic tools. Currently we support the creation of data files recognized by HyDE and KATE tools.

The visual interface, a MATLAB GUI linked to the Simulink model, provides an overview schematic of the mini-model

configuration and allows the user to setup the experiment. It sends the user specified parameters to Simulink and runs the simulation for the specified time. All input and output values are updated on the GUI as the simulation progresses. The user can also change the inputs online to recover from failures, and click on any variable in the GUI to see a time series plot. Fig. 7 shows the GUI in action.

In addition, we have also implemented an interface from the GUI/Simulink to HyDE. This was achieved by creating a MEX function in C++ that acts as the interface. The MEX function, which is linked to HyDE static library, creates an instance of HyDE, loads the appropriate model and configuration parameters into HyDE and then executes HyDE API functions in step with the simulation. Command and sensor data from the simulation is sent to HyDE at each time step. The diagnosis candidates from HyDE are listed in the GUI (lower left side). When the user clicks on each candidate, faulty components are colored red in the schematic to provide a visual indication.

Due to lack of time, an interface between the GUI and KATE was not created. We chose instead to exercise KATE using the data files generated (either using the command line interface or the GUI).

We present and discuss results from 4 scenarios. These were the same 4 scenarios used in the demonstration. The nominal operational sequence was selected to mimic the stages of a realistic cryogenic propellant loading. The first scenario was the complete nominal scenario with the goal of validating the realism of the simulated data. Three fault scenarios were selected based on use case analysis that determined some common failures (discussed in Section 2). Note that HyDE did not use the valve position sensors in any of the scenarios.

The first fault scenario was a flow control valve (A1) stuck open during the slow fill stage (at 300 s). The flow control

valves can be controlled on a continuous scale from 0 to 1 to regulate the flow. There is a flow control valve in the main and auxiliary lines and in the common discharge line leading to the vehicle tank (Fig. 1). When one of these valves is stuck open it is necessary to regulate the flow in that line using other means. HyDE and KATE are able to diagnose this situation right after observations for time 300 s are available. We determined that the recovery action for this fault was to regulate the flow by decreasing the Pump 1 speed to 275 RPM during slow fill to achieve the required net flow to the vehicle tank. Fig. 8 shows the profiles of some key measurements for this scenario.

The second fault scenario was a vent valve failure (stuck closed). The operation of this valve is modeled as an autonomous mode change (is not externally commanded). Its nominal behavior is to regulate the ullage pressure of the vehicle tank between 2 and 7 psig. Because the state of the valve is not directly commanded, this adds a level of difficulty to the reasoning process. Further, this was a critical fault scenario because if nothing was done, then the ullage pressure in the tank would keep increasing, resulting in catastrophic failure. The other interesting feature of this failure is that it can only be detected when the vent valve is predicted to be in open state since the behavior when the valve is in the closed state and the stuck closed state is identical. The scenario chosen failed the valve when it is in the closed state (1100 s). At 1254 s, the ullage pressure (P7) exceeds 7 psig, and the valve should have opened. At this point, HyDE and KATE are able to diagnose that the valve is stuck closed, although they cannot determine the time at which the failure occurred since it may have occurred at any time during which the valve was in the closed state. In this situation, the system is unsafe and loading operations must be aborted. Therefore, the recovery action is to drain the propellant back from the vehicle tank to the storage tank by shutting down the pumps and opening up all the valves. Fig. 9 illustrates the profiles of some key measurements in this scenario.

The third fault scenario was a double fault case where a pump (Pump 1) and a down stream flow sensor (F1) failed. Only HyDE was run on this scenario since KATE currently does not support multiple fault diagnosis. Additionally, we also decided to use only flow sensors to demonstrate the sensor fusion capabilities of HyDE when limited sensing is available. When the pump in the main transfer line fails, it is expected that the flow sensor in the main transfer line would indicate this failure. However, if the flow sensor has failed, it is necessary to use other sensors (namely flow sensors in the auxiliary transfer line and the vehicle tank subsystem) to diagnose the pump failure. HyDE is able to do exactly that albeit taking a couple of time-steps since the effects on the other flow sensors are not immediately seen. The recovery action for this situation was determined to be increasing the pump RPM in the auxiliary transfer line to 375 RPM and opening up the flow control valve in that line completely, in order to regain the nominal flow into the vehicle tank. Fig. 10 shows the profiles of some key measurements for this scenario.

Although we presented only three fault scenarios, we tested HyDE and KATE extensively by varying the number of faults injected (only single faults for KATE), the time at which the faults are injected, the sensor noise, and which sensor observations were available to the diagnosis engines. For KATE we tested only the scenarios described earlier in this section. However for HyDE we performed more systematic tests. There were 25 components that could be faulted with approximately 2 fault modes each. This in combination with 6 operating regions (Section 2.2) gave us 300 interesting single fault cases. We also tested ~100 double fault cases, all of which were a combination of a component and a sensor fault. All the single faults were diagnosed but there were 2 double fault cases that could not be diagnosed by HyDE. In one case a combination of a vent valve failure and corresponding pressure sensor failure resulted in fault effects not being seen until a few time steps later. In another scenario, HyDE was not able to distinguish between manual operated valve failure and remotely operated valve failure (position sensors were not used) and picked the wrong one because it had higher prior probability of failure.

7. CONCLUSIONS AND FUTURE WORK

In this paper we presented an application of model-based simulation and diagnosis work to a scaled-down version of a rapid propellant loading system. We demonstrated the detection and isolation of some common failure scenarios. In the process we also developed a component-based simulation in MATLAB and showed its usefulness in testing and evaluating FDIR applications.

The next steps in the project involve the completion of requirements 5-9 detailed in the introduction. We are in the process of developing a software architecture based on the Internet Communication Engine (ICE) (Henning 2004) to support requirement 5. ICE uses a "publish-subscribe" protocol for communication. In the architecture which was adopted from the Advance Diagnostics and Prognostics Test bed (ADAPT) (Poll et al. 2007), the MATLAB simulation will subscribe to the user inputs (including commands) and publish sensor data. The operation of the simulation is similar to the Virtual ADAPT simulation. HyDE and KATE will subscribe to the user inputs and sensor data and publish diagnosis results. The MATLAB model is being updated to reflect the hardware test bed being constructed (requirement 6). HyDE and KATE models will be built for the new configuration and a demonstration of the results is scheduled for late April 2009 (Requirement 7). When the test bed hardware has been built, the control architecture can be easily modified by replacing the MATLAB simulation with the LABVIEW module performing the data acquisition on the hardware (requirement 8). HyDE and KATE models can then be tested against data generated from the actual hardware (requirement 9).

Phase II of the project would deal with the implementation of automated synthesis of recovery actions based on the current state of the system. This may have to be interleaved with the diagnosis process for 2 reasons: (i) the diagnosis engine frequently reports ambiguities and recovery actions may have
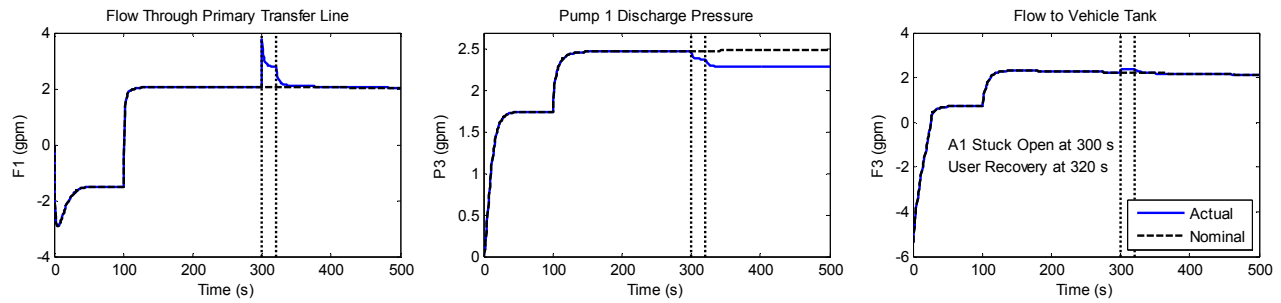
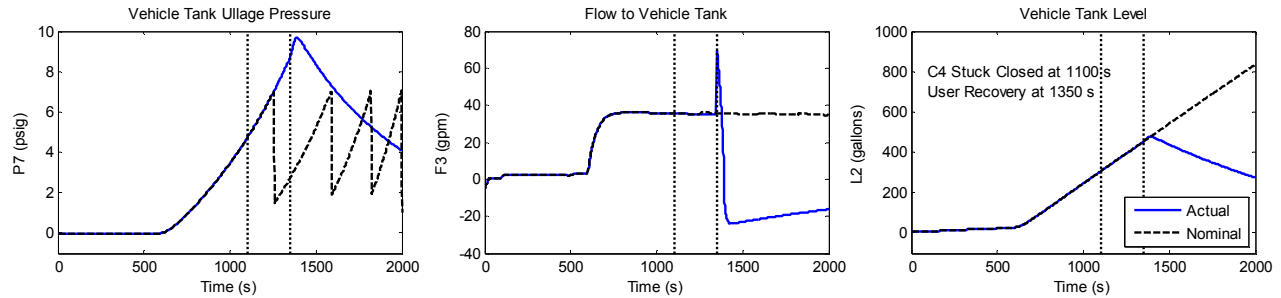Fig. 8. Fault Scenario 1 with stuck control valve
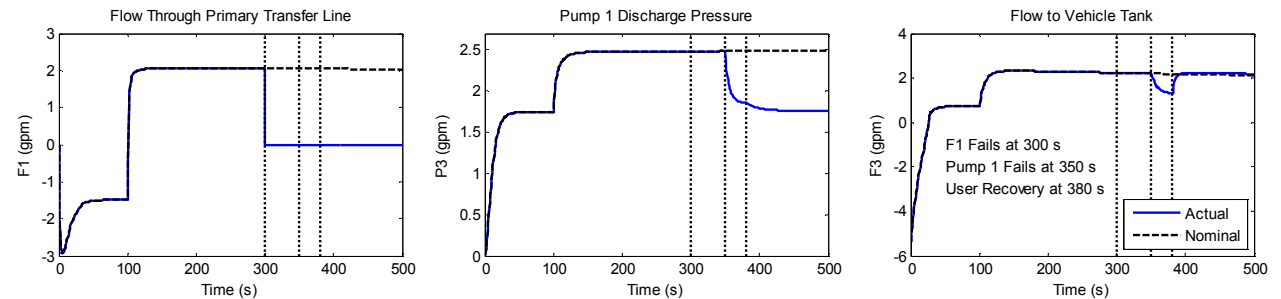

Fig. 9. Fault Scenario 2 with stuck vent valve

Fig. 10. Fault Scenario 3 with double fault

to be taken to avoid catastrophic effects of any faults reported, if time is taken to wait for further disambiguation (ii) in some cases recovery actions may be used to provide information helpful in disambiguating faults.

REFERENCES

S. Narasimhan and L. Brownston. HyDE – A General Framework for Stochastic and Hybrid Model-based Diagnosis. In *Proc. 18th International Workshop on Principles of Diagnosis (DX '07),* pages 162-169, Nashville, USA, May 2007.

J. Jamieson, E. Scarl, and C.I. Delaune. A Knowledge Based Expert System for Propellant System Monitoring at the Kennedy Space Center. In *Proceedings of the 22nd Space Congress.* Cocoa Beach, FL, Apr 1985.

E.A. Scarl, J.R. Jamieson, and C.I. Delaune. Diagnosis and Sensor Validation through Knowledge of Structure and Function, *IEEE Transactions on Systems, Man, and Cybernetics,* 17(3):360-368, May/June 1987.

C. Goodrich. A Method for Diagnosing Time Dependent Faults using Model-Based Reasoning Systems, *FLAIRS (Florida AI Research Symposium).*

M. Henning. A New Approach to Object-Oriented Middleware. *IEEE Internet Computing*, 8(1): 66-75, 2004.

S. Poll S. A. Patterson-Hine, I. Roychoudhury, M. Daigle, G. Biswas, et al. Evaluation, Selection, and Application of Model-based Diagnosis Tools and Approaches. *AIAA-2007-2941. AIAA Infotech@Aerospace*, Rohnert Park, CA, May 2007.