riacs Research Institute for Advanced Computer Science

The Livingstone Model of a Main Propulsion System

Anupa Bajwa

Adam Sweet

RIACS Technical Report 03.04

Dec. 2002

The Livingstone Model of a Main Propulsion System

Anupa Bajwa, RIACS/NASA Ames

Adam Sweet, QSS/NASA Ames

RIACS Technical Report 03.04

Dec., 2002

We present a component-based model of a Main Propulsion System (MPS) and say how it is used with Livingstone in order to implement a diagnostic system for integrated vehicle health management IIVHM) for the Propulsion IVHM tTechnology Experiment (PITEX). Given this model and the observed external commands and observations from the system, Livingstone tracks the state of the MPS over discrete time steps by choosing trajectories that are consistent with observations. We discuss how the compiled model fits into the overall PITEX architecture.

This work was supported in part by the National Aeronautics and Space Administration under Cooperative Agreement NCC 2-1006 with the Universities Space Research Association (USRA).

The Livingstone Model of a Main Propulsion System

Anupa Bajwa
USRA-RIACS at NASA Ames
MS 269-4, Moffett Field, CA 94035
650-604-1851
abajwa@mail.arc.nasa.gov

Adam Sweet
QSS Group Inc., at NASA Ames
MS 269-1, Moffett Field, CA 94035
650-604-2979
asweet@mail.arc.nasa.gov

Abstract—Livingstone is a discrete, propositional logic-based inference engine that has been used for diagnosis of physical systems. We present a component-based model of a Main Propulsion System (MPS) and say how it is used with Livingstone (L2) in order to implement a diagnostic system for integrated vehicle health management (IVHM) for the Propulsion IVHM Technology Experiment (PITEX).

We start by discussing the process of conceptualizing such a model. We describe graphical tools that facilitated the generation of the model. The model is composed of components (which may map onto physical components), connections between components and constraints. A component is specified by variables, with a set of discrete, qualitative values for each variable in its local nominal and failure modes. For each mode, the model specifies the component's behavior and transitions. We describe the MPS components' nominal and fault modes and associated Livingstone variables and data structures.

Given this model, and observed external commands and observations from the system, Livingstone tracks the state of the MPS over discrete time-steps by choosing trajectories that are consistent with observations. We briefly discuss how the compiled model fits into the overall PITEX architecture. Finally we summarize our modeling experience, discuss advantages and disadvantages of our approach, and suggest enhancements to the modeling process.

TABLE OF CONTENTS

1. INTRODUCTION

Model-based diagnosis (MBD) is the task of identifying faults in a physical system given a model of and observations from that system. The model consists of components and connections. The model is used to test assumptions about the state of a component. The assumption is "tried out" in the model – that is, the observations from the system are compared with values expected (predicted) by the model. When they match, the assumptions are accepted as accurate representations of the system's state.

Model-based tools, in general, are more flexible than rule-based tools. In fact, the same model may be used in one of three ways: simulation, diagnosis or recovery. In simulation, the commands and state of a system are given, and the model is used to predict the observations (sensor readings) from the system. In diagnosis, the commands and the observations are given, and the model is used to find the possible system states. Finally, in recovery, the observations and the current state are known (assumed), and the model is used to find a series of commands that will place the system into a desired state.

In MBD, there is often a distinction between the actual model and the diagnostic engine–the program that carries out the reasoning or inferencing. Livingstone is a qualitative, discrete, consistency-based reasoner that has been developed at NASA's Ames Research Center (ARC). As a model-based tool it can also be used for simulation and recovery.

The original version of Livingstone [1] was Lisp-based. The current, C++, version of Livingstone, called L2, can track multiple trajectories of the system over time [2]. Over the years, Livingstone has been a part of several diagnosis efforts:

- Deep Space 1 – it was used for mode identification, and mode recovery, during an onboard experiment, called the Remote Agent Experiment, in May 1999 [3]
- In-Situ Resource Utilization (ISRU) – Kennedy Space Center explored using Livingstone to confirm that

[1] 0-7803-7651-X/03/$17.00 © 2003 IEEE
[2] IEEEAC paper #1444, Updated December10, 2002

commands are executed and to diagnose faults in the ISRU.

- Advanced Life Support System – JSC has worked on integrating Livingstone with the control system of the Life Support project.
- X-37 – L2 was slated to fly as an onboard experiment [4] to monitor electro-mechanical actuators for control surfaces and electrical power system (though the X-37 program is currently under review).
- International Space Station -- ARC is researching issues in modeling hardware and software elements for some ISS subsystems, such as Command and Data Handling, and in using the model to assist ground operators with fault analysis.

This paper describes a component-based, discrete system model of a propellant storage, conditioning and feed system on a spacecraft such as the X-34 Reusable Launch Vehicle (RLV) [5]. This model is one part of an integrated architecture for diagnosis [7], developed by the PITEX team, comprising researchers at NASA's Glenn Research Center (GRC), Ames Research Center (ARC) and Kennedy Space Center (KSC). From a model consideration, two other key parts are the monitors and the Real Time Interface (RTI). The monitors take observations and map them into discrete values used in the model. The RTI sends these values to the Livingstone inference engine and decides when to request a diagnosis from Livingstone. It is not required that all observations be known, since Livingstone uses consistency-based reasoning. Secondly, by using a discrete representation, instead of a continuous one, Livingstone reduces the state space that must be stored and searched.

2. MODELING APPROACH

Before building a model there are several steps that the modeler has to consider. These are discussed in this section.

Domain understanding

Gaining an understanding of the physical system, or device, is the first step in the model generation process. For our modeling task it is sufficient to capture just the minimum amount of information about the system that will be needed for the specific purpose of diagnosis. Often knowledge of the physical system is available in varied formats such as schematics, nominal operations timeline and telemetry spreadsheets. A detailed discussion with the system designers, if they are available, is a key step in gaining a better understanding of the functionality of the system.

Basic information for a modeler

How do faults change the behavior of the device? In heuristic diagnosis there is a direct relation between the failure that occurs and its manifestation at observation points. In MBD, one models how failures change the *local* behavior of a part of the system (which is represented as a component in the model), and models the influences between parts. More details on such modeling issues are available in [6].

An essential part of diagnosis is making inferences based on observations. So some key considerations are what fault types are to be modeled and what observations are available. Observed signals usually have some noise associated with them, which affects accuracy. Observations also have varied ranges. Knowledge about these ranges can help determine the optimal level of abstraction required for the nominal and fault behaviors of the modeled components. In addition, the types of sensors, and the number and placement of sensors (whether redundant or not) determine what observations are available for system diagnosis.

What makes an adequate model

Diagnosability of a model is the issue of whether a particular diagnosis can be sufficiently distinguished from others, given the observables of the system. If a model satisfies this requirement for a particular, predetermined, set of fault scenarios it is an adequate model for that set of faults [6]. These are the faults that were taken into account during model generation. This does not imply that these are the only faults that can be diagnosed. For instance, there often is an "unknown" failure mode for each component that acts as a catchall mode for other faults.

Though only component faults are modeled, the key idea is that the diagnostic engine will be able to use these to detect many system-level faults, including multiple-component faults. The component faults have to be known *a priori* and often are available from Failure Modes and Effects Analysis (FMEA). Once these components are connected, the engine provides the ability to detect system-wide faults.

Basic properties of a model

Sometimes diagnosis is part of a larger task such as troubleshooting or preventive maintenance. When that is the case, an adequate model is one that assists a maintenance engineer in advising on the lowest-cost repair. That is, effective repairs are advised, no ineffective repairs are advised and the advice is provided in a timely manner.

In turn, this expects that the diagnostic engine provides correct diagnoses, provides no incorrect diagnoses (that is, there are no "false alerts") and does this in a timely manner. For the model this implies that the components (and connections) capture the nominal and faulty behavior of the system. A good model has states that are uniquely identifiable, has an optimal number (usually the smallest number) of components and connections and the cost of probing the system (number of sensors) is not too high. In practice, because most diagnosis modeling is done after a system has been designed, the modeler is simply given a system with a certain number of sensors.

Structure of a model

A system model consists of components and connections.

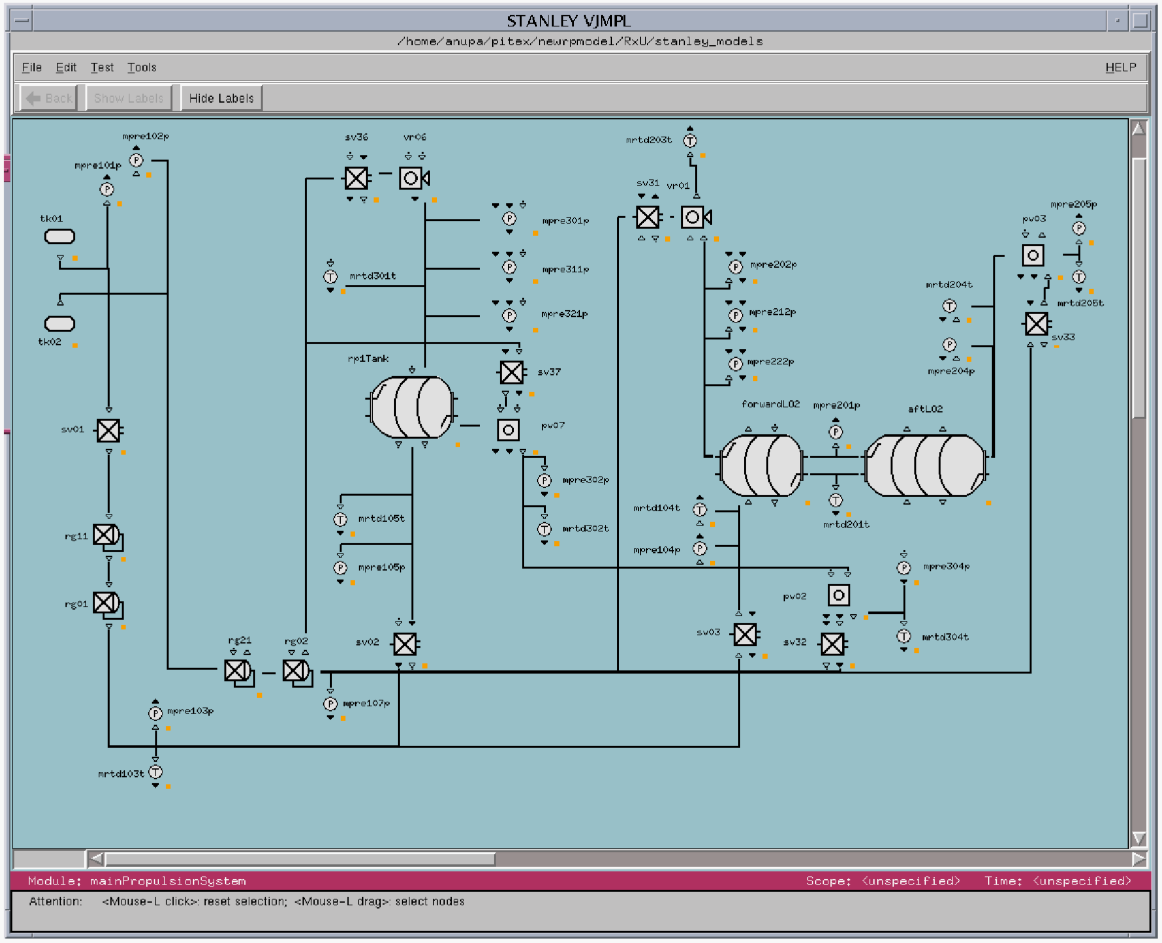
Components have terminals (inputs and outputs), modes (nominal mode, and one or more faulty modes including “unknown”) and specification of their observable behaviors (in the nominal and fault modes). Components represent parts of the system that contribute to the relevant behavior of the system -- for instance, parts that can fail and how they can fail and probability of that failure occurring.

Connections represent influence of components on other components. The modeler’s thought process includes assessing how nominal and faulty behavior of two or more connected components can manifest at the visible (and modeled) observation points. When creating a connection between two components the modeler defines the constraints between them. In Livingstone, a constraint is a clause that represents permissible assignments to variables.

3. MPS Model

For Livingstone a model is a data structure that represents the real-world device, the faults of which Livingstone diagnoses. To build a model of the MPS we can use a (Tcl-based) graphical user interface called Stanley to create a schematic-like model on the Stanley “canvas”. Stanley then uses this to generate the model files that will be read by Livingstone. Livingstone models can be stored in several different model file formats. A detailed description of these, and of other Livingstone terminology, is available at: http://ic.arc.nasa.gov/projects/mba/projects/L2/doc/index.html

The Stanley model of the X-34 MPS is shown in Figure 1. It shows liquid oxygen (LOX) and rocket propellant (RP-1) subsystems, pressurization and pneumatic subsystems, and vent and feed lines. Detailed descriptions of the subsystems are in [5]. The designed flight profile of the X-34 had two phases during its captive carry stage: LOX conditioning and bleed phase. This stage was followed by the burn stage, where the rocket engine fired, and thereafter the flight ended with an autonomous landing. The PITEX model was scoped to diagnose the MPS during captive carry.



Module: mainPropulsionSystem Scope: <unspecified> Time: <unspecified>

Attention: <Mouse-L click>: reset selection; <Mouse-L drag>: select nodes

Figure 1 – Model of the Main Propulsion System

4. Model Datatypes

Livingstone variables can take on one of a discrete set of values. For example, a variable named "sign", can take the values "negative", "zero", and "positive". Often, a variable is defined to divide the (continuous) real-number line into the discrete values relevant to the diagnosis of a system. Variables can also be grouped into structures for modeling convenience. However, Livingstone reasons about each variable individually regardless of the grouping.

One variable type, called a threshold, has been defined for the MPS model, and has been used extensively throughout it. A threshold is a defined variable that can take on two values, "aboveThreshold" and "belowThreshold". It divides the real-number line into two bins at a specified point. These thresholds can be on values for a sensor reading (e.g., 170°R) or can be on the derivative of a sensor reading (e.g., 0.04 psi/s).

The threshold data type is used to create other data types for the MPS model. There is a structure called a range, which contains two threshold data types, an upperBound and a lowerBound. There are constraints defined between the thresholds that assert the upperBound is greater than the lowerBound. If, for a particular range, a monitor reports a reading that is above the lowerBound threshold and is below the upperBound threshold, then the value is within that range. Most of the model constraints are implemented on this level.

Furthermore, the model contains structures built from multiple ranges. For example, the feedLineTemperature structure is composed of two ranges, loxTemp and ambientTemp. The order of the ranges is specified for each particular structure; however, note that they can overlap. This hierarchy of datatypes is illustrated in Figure 2.

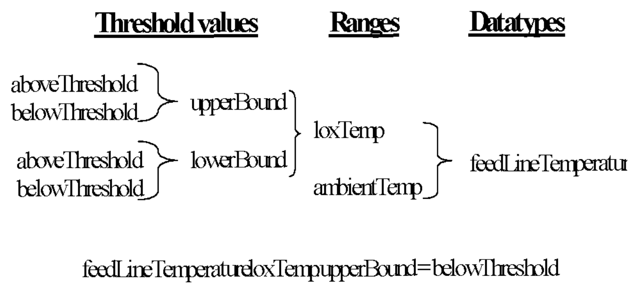


Figure 2 Model datatypes

5. Components

A Livingstone component contains a set of variables (which also includes a set of modes). The variables can be of any Livingstone datatype, as described above. The set of modes is the list of possible states for the component. Each mode contains constraints that will be active when the system is in that mode. Thus constraints are logical statements involving the variables defined within the mode. The types of allowable constraints include those commonly used in programming languages (e.g. equals, and, or, if). In general, a component relates its inputs to its outputs based on its current mode. For example, when a valve's mode is open its output pressure will be equal to its input pressure.

There are two types of modes, nominal modes and fault modes. In the current version of Livingstone, the modeled system can transfer between nominal modes only when given a command. The same system command can cause multiple components to change mode.

Fault modes differ from nominal modes in two ways. First, a component can transition into any fault mode from any other mode, as long as the constraints in the fault mode are consistent with the current observations. Secondly, a fault mode has a probability associated with it. This probability indicates the likelihood of that particular fault mode, and is constant: the probability of a fault cannot depend on the current system mode. Livingstone translates the probability number into an order of magnitude comparison called a rank. A rank is defined to be the negative of the logarithm of the probability, rounded to the nearest integer. Hence, the ranks are positive integers starting from zero, and higher numbers indicate less likely faults. The ranks also are additive. If a candidate in a diagnosis contains two components in a failed state, one failure of rank 1 and one failure of rank 2, the overall rank of the candidate is 3.

The nominal and fault modes of all modeled components are presented in Table 1. There are four subsystems in the MPS:

- pressurization subsystem provides pressurized helium to the propellant tanks during the bleed and burn phases
- pneumatics subsystem provides pressurized helium to open and close the pneumatic valves
- Liquid Oxygen (LOX) subsystem has two tanks for storing liquid oxygen: a forward tank and an aft tank. All of the logic is implemented in the forward tank and the aft tank is simply a pass-through component.
- Rocket Propellant (RP-1) subsystem has one RP-1 tank

The model propagates simple constraints on observations to track the system state. However, in the LOX or the RP-1 tank, the amount of storage (tank level) by itself is not uniquely indicative of the component's state. Moreover, there are not an adequate number of level sensors in the tank. Therefore, the time-derivative of the pressure sensor reading is used to indicate the state.

Thus the state of the LOX tank is determined by the states ("open" or "closed") of the three valves leading into and out of the tank, and is defined, in terms of pressure-rates, as

Table 1. Nominal and Failure Modes of Components of the Main Propulsion System

Subsystem	Component	Nominal Mode	Failure Mode	Rank
Pressurization	Helium tank (tk01)	Output pressure > secondary regulator setpoint		
	Enabling valve (sv01)	When valvePosition=open, input=output When valvePosition=closed, output=ambient		
	Regulators (rg01, rg11)	When Pin>=setpoint, output=setpoint When Pin<setpoint, output=input	Regulating high Regulating low Unknown	2 2 4
	Pressurization valves (sv02, sv03)	When open and pressure >= primary setpoint, flow>0 When closed, flow=0	Stuck open Stuck closed Unknown	3 2 5
	Microswitch (mmsw102x, 103x)	PositionReading = sensedPosition	Faulty	3
	Pressure sensors (mpre101p, 103p)	PressureReading = sensedPressure	Faulty	2
	Temperature sensor (mrtd103t)	TemperatureReading = sensedTemperature	Faulty	2
Pneumatics	Helium tank (tk02)	Output pressure > secondary regulator setpoint		
	Regulators (rg02, rg21)	When Pin>=setpoint, output=setpoint When Pin<setpoint, output=input	Regulating high Regulating low Unknown	2 2 4
	Solenoid valves (sv31, sv32, sv33, sv36, sv37)	When open and pressure>=primary setpoint, flow>0 When closed, flow=0	Stuck open Stuck closed Unknown	3 2 5
	Microswitch (mmsw205x, 303x)	PositionReading = sensedPosition	Faulty	3
	Pressure sensor (mpre102p, 107p)	PressureReading = sensedPressure	Faulty	2
LOX	LOX tanks (forwardLO2, aftLO2)	(see table)	Unknown	6
	Vent relief valve (vr01)	When open, flow>0 and temperature in tankMixture(LOX) range When closed, flow=0 and temperature in ambient range	Stuck open Stuck closed Unknown	2 1 4
	Engine inlet valve (pv03)	When open, pressure>=pressurizationIndicator When closed, flowIn = flowOut = 0	Stuck open Stuck closed	2 1
	Microswitch (mmsw203x, 213x)	PositionReading = sensedPosition	Faulty	3
	P. sensors (mpre104p, 201p, 204p, 205p)	PressureReading = sensedPressure	Faulty	2
	Controller pressure sensors (mpre202p, 212p, 222p)	PressureReading = sensedPressure and sensor is not biased	Biased Faulty	1 2
	T. sensors (mrtd104t, 201t, 203t, 204t, 205t)	TemperatureReading = sensedTemperature	Faulty	2
RP-1	RP-1 tank (rp1Tank)	(see table)	Unknown	6
	Vent relief valve (vr06)	When open, flow>0 and temperature in tankMixture (RP-1) range When closed, flow=0 and temperature in ambient range	Stuck open Stuck closed Unknown	2 1 4
	Engine inlet valves (pv07, pv02)	When open, pressure>=pressurizationIndicator When closed, flowIn = flowOut = 0	Stuck open Stuck closed	2 1
	Microswitch (mmsw304x,314x, 301x, 311x)	PositionReading = sensedPosition	Faulty	3
	Pressure sensors (mpre105p, 302p, 304p)	PressureReading = sensedPressure	Faulty	2
	Controller pressure sensors (mpre301p, 311p, 321p)	PressureReading = sensedPressure and sensor is not biased	Biased Faulty	1 2
	T. sensors (mrtd105t, 301t, 302t, 304t)	TemperatureReading = sensedTemperature	Faulty	2

shown in Table 2. Similarly, the RP-1 tank model is shown in Table 3.

Table 2. PITEX LOX Tank Model

Sv03 Flow	Vr01 Flow	Pv03 Flow	Predicted Pressure Value
zero	zero	zero	HeatingRate
positive	zero	zero	PressurizationRate
zero	positive	zero	VentingRate
zero	zero	positive	BleedRate
positive	zero	positive	PressurizationRate

Table 3. PITEX RP-1 Tank Model

Sv02 Flow	Vr06 Flow	Pv07 Flow	Predicted Pressure Value
Zero	zero	zero	Below pressurizationRate, above bleedRate
Positive	zero	zero	PressurizationRate, Above ambientPressureLevel
Zero	positive	zero	VentingRate, Below ambientPressureLevel
Zero	zero	positive	BleedRate
Positive	zero	positive	PressurizationRate
Positive	positive	zero	Below ambientPressureLevel
Positive	positive	positive	Below ambientPressureLevel

6. TESTING THE MODEL

For PITEX, the model has been tested in two stages: in isolation and as part of the integrated diagnostic system. (A third test of the model, using formal verification tools, has been done on a current version of the model but that effort is not addressed in this paper.)

To test in the context of the model, the model is loaded into Livingstone in a standalone mode. The modeler then inputs commands and observations interactively, or through a scenario, which correspond to a particular sequence of commands and observations. This method of testing is most useful in finding the set of component interactions that Livingstone predicts for the system. Many unexpected interactions can be caught at this stage. However, there are some limitations. Often, the modeler simply inputs the observations that he feels is relevant to the situation being tested, whereas the full set of observations would have changed the possibilities that Livingstone reports in a diagnosis. In addition, the actual monitors may input observations that the modeler didn't expect. This indicates a modeling error, and this class of errors may not be

Table 4. Diagnosis of PITEX scenarios.

PITEX Scenario	DESCRIPTION	TIME OF FAULT	DIAGNOSIS	Rank
1	Open microswitch, MMSW205X, fails on the LOX vent/relief solenoid valve, SV31.	3301.70 and 7260.8	Microswitch MMSW205X failed SV31 Failed Closed and LOX vent/relief valve VR01 Failed Open	3 4
2	Close microswitch, MMSW213X, fails on the LOX feed valve, PV03.	9410.00	Closed microswitch MMSW213X Failed	3
3	RP-1 feed valve, PV02, fails closed after the RP-1 bleed has been initiated.	9359.00	PV02 Stuck Closed SV32 Stuck Closed	2 2
4	RP-1 vent/relief pneumatic valve, VR06, fails open.	9379.00	VR06 Stuck Open MPRE103P faulty	2 2
5	Primary RP-1 tank pressurization valve, SV02, sticks closed.	9383.86	SV02 Stuck Closed SV02 Stuck Closed and SV02.openMicroswitch faulty SV02.openMicroswitch faulty and MPRE103P faulty SV02 unknown fault	2 5 5 5
6	Primary RP-1 tank pressurization valve, SV02, sticks open.	9384.71	SV02 Stuck Open SV02.openMicroswitch faulty SV02 unknown fault	3 3 5
7	Open microswitch, MMSW205X, fails on the LOX vent/relief solenoid valve, SV31. After that SV31 fails closed.	3301.70	SV31 Stuck Closed and SV31.openMicroswitch faulty VR01 Stuck Closed and SV31.openMicroswitch faulty	5 5
8	GHe pressurization system pressure regulators, RG11 and RG01, both regulate high.	9000.00	MPRE103P faulty RG11 regulates high and RG01 regulates high	2 4
9	Two of the LOX vent line pressure sensors, MPRE202P and MPRE212P, fail high.	0.00	MPRE202P faulty and MPRE212P faulty MPRE202P biased and MPRE212P biased MPRE202P faulty and MPRE212P biased MPRE202P biased and MPRE212P faulty	4 4 4 4
10	Two of the LOX vent line pressure sensors, MPRE202P and MPRE212P, fail low.	0.00	MPRE202P faulty and MPRE212P faulty MPRE202P biased and MPRE212P biased MPRE202P faulty and MPRE212P biased MPRE202P biased and MPRE212P faulty	4 4 4 4

identified without overall system testing.
Testing the model as a part of the integrated diagnostic system can begin once the model is stable and the integrated diagnostic system is implemented. On PITEX, this involves generating simulated data, processing these continuous data through the monitors, and then through the RTI and on to Livingstone. The testing process entails running a fault scenario, inspecting the Livingstone diagnosis, investigating the reasons for any spurious results, correcting them, and re-running the scenario. There are a variety of reasons for spurious results: model inconsistency, noise issues, RTI issues, and simulated dataset not matching expected values. Getting the end-to-end system working is an iterative process. For instance, the model-monitor teams have to work together to establish the thresholds of interest and may have to iterate on changes to the model and the monitors in order to accommodate all scenarios. Changes include adding a new fault mode to a component (e.g. microswitch failure), or adding a new variable to track the state of the system.

Table 4 shows summary results for all of the PITEX fault scenarios. "Time of fault" is the time at which the fault is first observable in the physical system. The failure candidates diagnosed by Livingstone are shown, along with the rank of failure. All candidates are correct and, in almost all cases, the lowest ranked candidate corresponds to the injected fault.

7. Accomplishments and Limitations

The process of developing and testing the model has driven development of the core tools, such as L2, used with the model. One of the fault scenarios studied identified a bug in Livingstone, which was corrected. Some problems were also identified, and fixed, in the Stanley graphical development environment.

Livingstone has brought several advantages to the problem of diagnosing the X-34 Main Propulsion System. First, it is fast. Because it uses a discrete model, the state space of the system is smaller. Also, it provides multiple candidates, each of which could contain multiple component faults. As shown in the diagnosis table in the previous section, many of the fault scenarios considered could have several possible explanations.

There are many possible topics for future work in this area. First, the scope of the model could be increased. Many possible faults are not explicitly modeled, such as tank leaks. In addition, the scope could be extended beyond the captive carry phase of the flight. Modeling the burn and landing phase would highlight many issues in diagnosing different operating modes of the vehicle. Finally, scalability effects have been investigated only briefly. It would be interesting to see how the time required for a diagnosis increases with the number of components, or with changes in Livingstone parameter settings during runtime.

A current limitation is that the components are all created individually, and that they fill specific roles in the model. Even similar components, such as solenoid valves, have different models depending on their placement in the system. It could be useful to make the components more generic, so they can be reused in the various parts of this model and even in other future models. The creation and rigorous testing of generic components would reduce the development and testing times for future models. An effort investigating the issues with modeling generic components is currently in progress.

Finally, more through testing could be done by generating large numbers of simulations that vary the type and time of the injected faults. Effort is underway to make the simulations more adaptable to generate more scenarios; thereafter, the model can be tested on large numbers of possible fault scenarios. Better measures for false alerts and missed alerts of the system can be obtained, which can help guide the refinement of the diagnostic system.

8. Conclusions

PITEX has demonstrated that qualitative, model-based reasoning can be used for diagnosis of a real-world system, such as an MPS, with continuous sensor values. Several new ideas have been developed to accomplish this, including derivative monitors and the current RTI policy. Since multiple NASA centers worked on different parts of the integrated architecture, PITEX provided an excellent opportunity for teamwork and technology exchange.

9. Acknowledgements

The current Stanley model of the MPS has its roots in the original model developed for NITEX by William Millar and others, in concert with monitor designers. The PITEX effort also leverages continued maturation of the Livingstone inference engine and supporting model development tools.

References

[1] Williams, B. C., and Nayak, P. P., "A model-based approach to reactive self-configuring systems." *Proceedings, AAAI-1996.*

[2] J. Kurien and P. Nayak, "Back to the future for consistency-based trajectory tracking." *7th National Conference on Artificial Intelligence (AAAI 2000).*

[3] D. Bernard, G. Dorais et al, "Spacecraft Autonomy Flight Experience: The DS1 Remote Agent Experiment" AIAA 99-4512, 1999.

[4] M. Schwabacher, J. Samuels, L. Brownston, " NASA Integrated Vehicle Health Management Technology

Experiment for X-37." *SPIE AeroSense 2002.*

[5] P. Sgarlata and B. Winters, "X-34 Propulsion System Design." *33rd AIAA/ASME/SAE/ASEE Joint Propulsion Conference and Exhibit, July 6-9, 1997.*

[6] Dirk Carel van Soest, "Modeling for model-based diagnosis." *Thesis*, Universiteit Twente, 1993.

[7] C. Meyer, H. Cannon et al, "Propulsion IVHM Technology Experiment Overview." *IEEE Aerospace Conference, March 8-15, 2003.*

Anupa Bajwa *is a scientist at NASA Ames' Research Institute for Advanced Computer Science (RIACS). She has worked on developing a Livingstone model of a Main Propulsion System for the Propulsion IVHM Technology Experiment (PITEX). Previously she has worked on assessment of conflict prediction software for Air Traffic Control and Surface Movement Advisor at airports, and has also worked on software for Condition-Based Maintenance of aircraft engines. She has specialized in Aerospace Engineering, with a B.Tech. from The Indian Institute of Technology, Bombay, an M.S. from The Ohio State University and a Ph.D. from The Pennsylvania State University.*

Adam Sweet is a researcher with QSS Group Inc. at NASA Ames Research Center. He has worked on creating models and simulations of physical systems in order to test diagnostic and autonomy research software.
He has a B.S. and an M.S. in Mechanical Engineering from the University of California at Berkeley.