Advanced Diagnostics and Prognostics Testbed

Scott Poll, Ann Patterson-Hine, Joe Camisa, David Garcia[1], David Hall[1], Charles Lee[2], Ole J. Mengshoel[3], Christian Neukom[1], David Nishikawa, John Ossenfort[2], Adam Sweet[1], Serge Yentus[1]

[1]QSS Group, Inc., a subsidiary of Perot Systems Government Services; [2]SAIC; [3]RIACS

NASA Ames Research Center

M/S 269-1, Moffett Field, CA 94035

{Scott.Poll, Ann.Patterson-Hine}@nasa.gov

Indranil Roychoudhury, Matthew Daigle, Gautam Biswas, Xenofon Koutsoukos

Institute for Software Integrated Systems, Department of Electrical Engineering and Computer Science,

Vanderbilt University, Nashville, TN 37235

gautam.biswas@vanderbilt.edu

Abstract

Researchers in the diagnosis community have developed a number of promising techniques for system health management. However, realistic empirical evaluation and comparison of these approaches is often hampered by a lack of standard data sets and suitable testbeds. In this paper we describe the Advanced Diagnostics and Prognostics Testbed (ADAPT) at NASA Ames Research Center. The purpose of the testbed is to measure, evaluate, and mature diagnostic and prognostic health management technologies. This paper describes the testbed's hardware, software architecture, and concept of operations. A simulation testbed that accompanies ADAPT, and some of the diagnostic and decision support approaches being investigated are also discussed.

Introduction

Automated methods for diagnosing problems with system behavior are commonplace in automobiles, copiers, and many other consumer products. Applying advanced diagnostic techniques to aerospace systems, especially aerospace vehicles with human crews, is much more challenging. The low probability of component and subsystem failure, the cost of verification and validation, the difficulty of selecting the most appropriate diagnostic technology for a given problem, and the lack of large-scale diagnostic technology demonstrations increase the complexity of these applications. To meet these challenges, NASA Ames Research Center has developed the Advanced Diagnostic and Prognostic Testbed with the following goals in mind:

(i) Provide a technology-neutral basis for testing and evaluating diagnostic systems, both software and hardware,

(ii) Provide the capability to perform accelerated testing of diagnostic algorithms by manually or algorithmically inserting faults,

(iii) Provide a real-world physical system such that issues that might be disregarded in smaller-scale experiments and simulations are exposed – "the devil is in the details,"

(iv) Provide a stepping stone between pure research and deployment in aerospace systems, thus create a concrete path to maturing diagnostic technologies, and

(v) Develop analytical methods and software architectures in support of the above goals.

NASA missions have always provided challenging problem domains for model-based diagnosis researchers. Early demonstrations (Williams *et al.*, 1998) were typically limited to particular subsystems or components, and were run in "shadow-mode," in parallel with the conventional control and fault protection software. However, the increasing complexity of aerospace systems has made advanced diagnostic capabilities a necessity to ensure the reliability and safety of missions. Although they are well-suited to address the needs of aerospace missions, many of the techniques that fall under the "model-based diagnosis" umbrella face opposition from project managers, who may be reluctant to consider techniques with limited flight experience.

We have developed the Advanced Diagnostics and Prognostics Testbed at NASA Ames Research Center to enable maturation of advanced fault management techniques. ADAPT provides a representative physical domain that serves as a benchmark for performance assessment and comparison of algorithms and concepts that enable automated diagnosis of complex systems exhibiting hybrid, i.e., both continuous and discrete, behavior. The testbed documentation includes engineering schematics, component descriptions, fault descriptions, and system engineering analyses, such as functional models, Failure Modes and Effects Analysis, and Testability Analysis. Additional information includes a simulation and an Application Programming Interface (API) to facilitate integrating diagnostic applications with the testbed. We will provide the documentation, simulation model, API, as well as nominal and faulty testbed data to researchers in the diagnosis community.

The process of developing advanced diagnostic applications is just as important to the verification and validation of these systems as the specific algorithms that are applied. ADAPT establishes a problem domain with known fault signatures and the capability to inject failures during operation of the testbed. By implementing and testing different

diagnostic approaches on ADAPT, we aim to improve performance assessment methods and the comparison of diverse health management strategies.

This paper presents a description of ADAPT that includes its hardware and software architectures, the concept of operations, and a simulation testbed that emulates the real system. We also discuss some of the diagnostic approaches that are currently being developed on the testbed.

Testbed Description

The ADAPT lab is shown in Figure 1. The equipment racks in the background can generate, store, distribute, and monitor electrical power. The initial testbed configuration functionally represents an exploration vehicle's Electrical Power System (EPS). The EPS can deliver AC (Alternating Current) and DC (Direct Current) power to loads, which in an aerospace vehicle would include subsystems such as the avionics, propulsion, life support, and thermal management systems. A data acquisition and control system sends commands to and receives data from the EPS. The testbed operator stations are integrated into a software architecture that allows for nominal and faulty operations of the EPS, and includes a system for logging all relevant data to assess the performance of the health management applications. The following sections describe the testbed hardware, diagnostic challenges, concept of operations, and software.

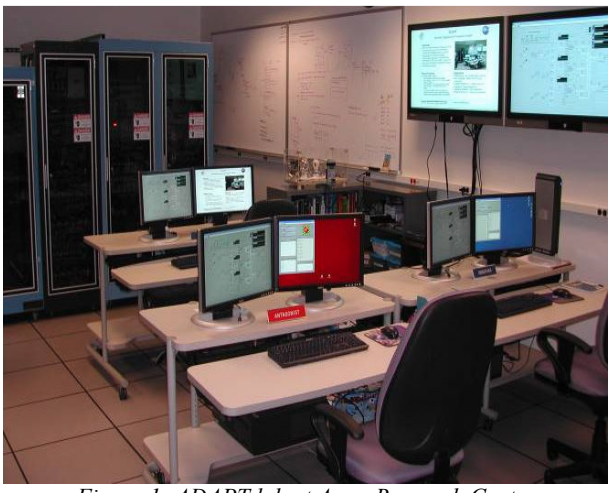

Figure 1: ADAPT lab at Ames Research Center.

Hardware Subsystems

Figure 2 depicts ADAPT's major system components and their interconnections. Three power generation sources are connected to three sets of batteries, which in turn supply two load banks. Each load bank has provisions for 6 AC loads and 2 DC loads.

Power Generation. The three sources of power generation include two battery chargers and a photovoltaic module. The battery chargers are connected to appropriate wall outlets through relays. Two metal halide lamps supply the light energy for the photovoltaic module. The three power generation sources can be interchangeably connected to the three batteries. Hardware relay logic prevents connecting one charge source to more than one battery at the same time, and from connecting one charging circuit to another charging circuit.

Power Storage. Three sets of batteries are used to store energy for operation of the loads. Each "battery" consists of two 12-volt sealed lead acid batteries connected in series to produce a 24-volt output. Two battery sets are rated at 100 amp-hrs and the third set is rated at 50 amp-hrs. The batteries and the main circuit breakers are placed in a ventilated cabinet that is physically separated from the equipment racks; however, the switches for connecting the batteries to the upstream chargers or downstream loads are located in the equipment racks.

Power Distribution. Electromechanical relays are used to route the power from the sources to the batteries and from the batteries to the AC and DC loads. All relays are the normally-open type. An inverter converts the 24-volt DC

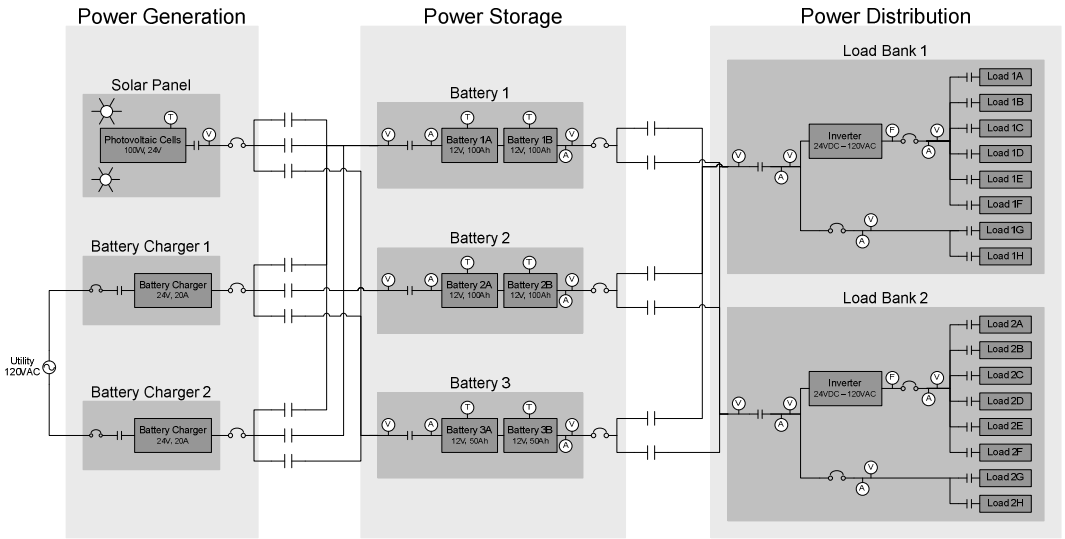

Figure 2: Testbed components and interconnections.

battery input to a 120-volt rms AC output. Circuit breakers are located at various points in the distribution network to prevent overcurrents from causing unintended damage to the system components.

Control and Monitoring. Testbed data acquisition and control use National Instrument's LabVIEW software and CompactFieldPoint (cFP) hardware. Table 1 lists the modules that are inserted into the two identical backplanes. The instrumentation allows for monitoring of voltages, currents, temperatures, switch positions, light intensities, and AC frequencies, as listed in Table 2.

Table 1: Testbed backplane modules.

Module	Description	Channels
cFP-2000	Real-time Ethernet Module	NA
cFP-DI-301	Digital Input Module	16 (x2)
cFP-DO-401	Digital Output Module	16 (x2)
cFP-AI-100	Analog Input Module	8
cFP-AI-102	Analog Input Module	8 (x2)
cFP-RTD-122	RTD Input Module	8

Table 2: Testbed instrumentation.

Sensed Variable	Number of Sensors
Voltage	22
Current	12
Temperature	15
Relay Position	41
Circuit Breaker Position	17
Light Intensity	3
AC Frequency	2

Diagnosis Challenges

The ADAPT testbed offers a number of challenges to health management applications. The electrical power system shown in Figure 2 is a hybrid system with multiple system configurations made possible by switching among the generation, storage, and distribution units. Timing considerations and transient behavior must be taken into account when designing diagnosis algorithms. When power is input to the inverter there is a delay of a few seconds before power is available at the output. For some loads, there is a large current transient when the device is turned on. As shown in Figure 3, system voltages and currents depend on the loads attached, and noise in the sensor data becomes more pronounced as more loads are added. Due to the low probabilities of failure, seeding/inserting faults is needed. Through an antagonist function described in the next section, it is possible to inject multiple faults into the testbed.

Concept of Operations

Unlike many other testbeds, the primary articles under test in ADAPT are the health management systems, not the physical devices of the testbed. To operate the testbed in a way that facilitates the study of health management technologies, the following operator roles are defined:

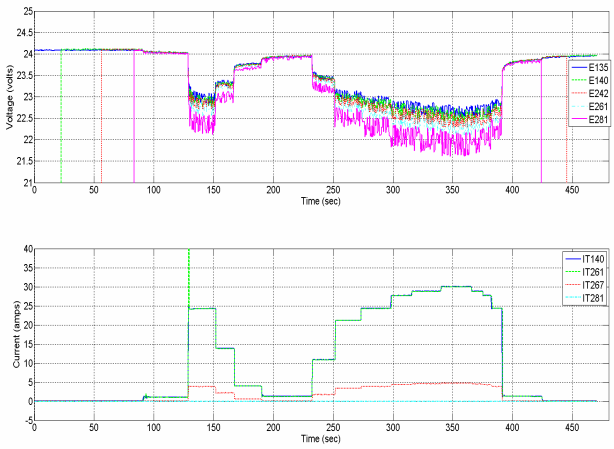

Figure 3: Sample testbed voltages (top) and currents (bottom) for a battery discharging to loads.

- *User* – who simulates the role of a crew member or pilot operating and maintaining the testbed subsystems.
- *Antagonist* – who injects faults into the subsystem either manually, remotely through the *Antagonist* console, or automatically through software scripts.
- *Observer* – who records the experimental data and notes how the *User* responds to the faults injected by the *Antagonist*. The *Observer* also serves as the safety officer during all tests and can initiate an emergency stop.

During an experiment, the *User* is responsible for controlling and monitoring the EPS and any attached loads that are required to accomplish a mission. The *Antagonist* disrupts system operations by injecting one or more faults unbeknownst to the *User*. The *User* may use output from a health management application (test article) to determine the state of the system and choose an appropriate recovery action. The *Observer* records the interactions and measures the effectiveness of the test article.

The testbed has two primary goals: (i) performance analysis of, and comparisons among, different test articles, and (ii) running of system studies. With the hardware and the supporting software infrastructure described in a subsequent section, experiments may be conducted using a variety of test articles to evaluate different health management technologies. The test articles may be connected to different interface concepts for presenting health management information to the human *User*, to study how the person performs in managing system operations.

Software

The testbed software model supports the concept of operations that includes the previously-mentioned operational roles of the *User* (USR), *Antagonist* (ANT), *Observer* (OBS), and *Test Article* (TA), along with the *Logger* (LOG) and the *Data Acquisition* (DAQ) roles. The *Logger* collects and saves all communication between the various components. The DAQ computer interfaces with the National Instruments data acquisition and control system. It sends command data to the testbed via the appropriate

backplane modules and receives testbed sensor data from other modules.

The underlying data communication is implemented using a publish/subscribe model in which data from publishers are routed to all subscribers registering an interest in a data topic. To enforce testing protocols and ensure the integrity of the data, filters based on role and location limit the topics for which data can be produced or consumed. Table 3 lists the message topics and the topic publishers and subscribers.

Table 3: Publish/subscribe topics.

Topic	Publisher	Subscriber
Sensor Data	DAQ	ANT,OBS,LOG
Antagonist Data	ANT	TA,USR,LOG
User Command	USR,TA	TA,ANT,OBS,LOG
Antagonist Command	ANT	DAQ,OBS,LOG
User Message	USR	OBS,LOG,ANT
Note	OBS	LOG,ANT
Fault Data	TA	USR,OBS,LOG
Diagnostics	TA	USR,OBS,LOG
Experiment Control	OBS	LOG

The following constraints are enforced on the various system components when they are operating on the ADAPT computer network:

- The *DAQ* can only read command data sent by the Antagonist and only sends sensor data which it gets directly from the instrumentation I/O subsystem. When no faults are injected, the *Antagonist* commands are the same as the *User* commands. The *DAQ* software can run only on the *DAQ* computer. It is the only software that connects directly to the instrumentation I/O subsystem.
- The *Antagonist* can only read sensor data sent by the *DAQ* and command data sent by *Test Articles* and *Users*. It forwards sensor data, which it may have modified by fault injection. It also sends command data, which are read by the *DAQ* and the *Logger*.
- The *Test Article* and the *User* cannot see *DAQ* sensor data. They have access only to *Antagonist*-generated sensor data, which are identical to *DAQ* sensor data when no faults are injected. The *Test Article* and *User* cannot read text data (a *Note*) sent by the *Observer*. The *Test Article* can read *User* commands and *Antagonist* data. It can send diagnostics data and *User* commands.
- The *Observer* sends an experiment control record to initiate a testbed experiment. It also sends text data to describe observations of the ongoing experiment. The *Observer* cannot send any data other than control and text data.
- The *Logger* reads all data sent over the ADAPT network. The *Logger* assigns a unique experiment ID for each new experiment.

A test article may be integrated with the testbed by installing the application on one of the ADAPT computers or by connecting a computer with the test article application to a preconfigured auxiliary system that acts as a gateway to the ADAPT network. A test article uses an API to subscribe to commands and data, and publish diagnosis results to the ADAPT message server. Java and C++ interfaces are supported.

VIRTUAL ADAPT: Simulation Testbed

We are also developing a high-fidelity simulation testbed that emulates the ADAPT hardware for running offline health management experiments. This environment, called VIRTUAL ADAPT, provides identical interfaces to the application system modules through wrappers to the ADAPT network. The physical components of the testbed, i.e., the chargers, the batteries, relays, and the loads, are replaced by simulation modules that generate the same dynamic behaviors as the hardware test bed. Also, like the actual hardware, VIRTUAL ADAPT subscribes to antagonist commands and publishes corresponding sensor data. As a result, application systems developed on VIRTUAL ADAPT can be run directly on ADAPT, and vice versa. Therefore, applications can be developed and tested using VIRTUAL ADAPT, and then run as test articles on the actual system. The simulation environment also provides for precise repetition of different operational scenarios, and this allows for more rigorous testing and evaluation of different diagnostic and prognostic algorithms.

In order to mirror the real testbed, we have addressed several issues that include (i) one-to-one component modeling, (ii) replicating the dynamic behavior of the hardware components, (iii) matching the possible configurations, and (iv) facilitating the running of diagnosis and prognosis experiments. The following sections provide a more detailed description of our approach to addressing these issues.

Component-Oriented Modeling

Our approach to component-oriented compositional modeling is based on a top-down process, where we first capture the structural description of the system in terms of the components and their connectivity. Component models replicate the component's dynamic behaviors for different configurations. The connectivity relations, represented by energy and signal links, capture the interaction pathways between the components. Complex systems, such as a power distribution system, may operate in multiple configurations. The ability to change from one configuration to another is modeled by switching elements. Sets of components can also be grouped together to define subsystems. The modeling paradigm is implemented as part of the Fault Adaptive Control Technology (FACT) tool suite developed at Vanderbilt University (Manders *et al.* 2006). FACT employs the Generic Modeling Environment (GME) to present modelers with a component library organized as hierarchical collection of *components* and *subsystems* and a graphical interface for creating component-oriented system models (Ledeczi *et al.* 2001). The VIRTUAL ADAPT models created using FACT capture a number of possible ADAPT testbed configurations.

Each component includes an internal behavior model and an interface through which the component interacts with other components and the environment. The interfaces include two kinds of ports: (i) *energy ports* for energy exchange between the component and other components, and (ii) *signal ports* for input and output of signal values from the component. The current VIRTUAL ADAPT testbed component library includes models for the chargers, batteries, inverters, loads, relays, circuit breakers, and sensors that exist on the current ADAPT hardware testbed. For experimental purposes and "what if" analyses, new components can be added to the library by modifying existing component models or by creating new ones. System models are built by creating configurations of component models.

Many ADAPT testbed components are physical processes that exhibit hybrid behaviors, i.e., mixed continuous and discrete behaviors. These components are modeled as Hybrid Bond Graph (HBG) fragments (Mosterman and Biswas 1998). HBGs extend the bond graph modeling language (Karnopp, Margolis, and Rosenberg 2000) by introducing junctions that can switch *on* and *off*. Bond graphs are a domain-independent, topological modeling language based on the conservation of energy and continuity of power.

Building complete HBG models for a system requires detailed knowledge of the system configuration and component behaviors, as well as component parameters. This knowledge is typically obtained by consulting system designers and experts, extracting information from device manuals and research papers, and using experimental data collected during system operations. When experimental data is used, unknown parameters and functional relations associated with the models are estimated using system identification techniques. Often, this is a difficult task that requires significant analysis.

Model validation is performed by comparing simulated behaviors with data collected from experimental runs on ADAPT. A number of parameter estimation iterations may be necessary to obtain an accurate model of the system, keeping in mind the tradeoff between model accuracy and model complexity.

Generating Efficient Simulation Models

VIRTUAL ADAPT models created using the FACT tool suite can be translated into MATLAB Simulink® models using a systematic model transformation process (Roychoudhury *et al.* 2007) that is implemented using GME interpreters. The two-step process first transforms the HBG models into an intermediate block diagram representation, and then converts the block diagram representation into an executable Simulink model. The use of the intermediate block diagram provides the flexibility of generating executable models for other simulation environments with minimal effort.

Using naïve methods to generate executable hybrid system models requires pre-computation of the model for all possible system configurations or modes of operation. This is space-inefficient. The alternative is incremental generation of model structure at runtime when mode changes occur, which is time-inefficient (Daigle *et al.* 2006). We use a structurally static system model where the correct operating configuration of each bond graph element is selected online when mode changes occur, thus reducing excessive space and time costs at runtime. These algorithms exploit causality in bond graphs and, in addition to incremental generation, can also minimize the use of high-cost fixed point or algebraic loop solvers. Algebraic loop structures may arise in the Simulink structures to accommodate the switching functions in the HBG models.

In addition to generating nominal behaviors, the simulation system provides an interface through which sensor, actuator, and process faults with different "fault profiles" (e.g., abrupt versus incipient) can be injected into the system at specific time points. The Simulink model then generates faulty system behavior, which can form the basis for running diagnosis, prognosis, and fault-adaptive control experiments. In general, many different user roles and test articles, such as controllers, fault detectors, diagnosers, and interfaces for observing behaviors, can be tested.

Example: Battery Component Modeling

We demonstrate our modeling approach by developing a model of the batteries on the ADAPT testbed. The battery component model is developed from an electrical equivalent circuit model (Ceraolo 2000), shown in Figure 4 (left). The model computes the output battery voltage and the current flowing from the battery to a connected load when the battery is discharging; or from the charger to the battery, when it is charging. In both situations, some of this current goes into charging or discharging the batteries, and the rest is lost to parasitic reactions (e.g., gas production)

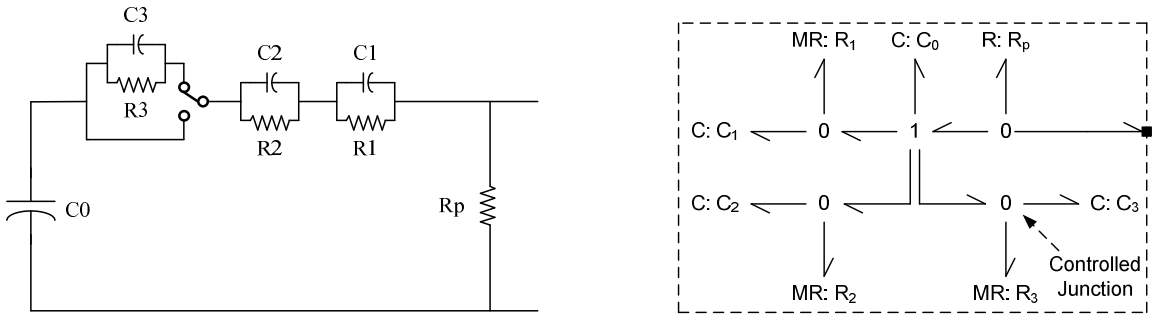

Figure 4: The equivalent circuit of the battery (left) and its corresponding hybrid bond graph (right).

modeled by a resistive element, R_p. The capacitor C_0, which has a large capacitance value, models the steady-state voltage of the battery. The steady-state voltage of the battery is a linear function of this capacitance value and the current amount of charge in the battery. The remaining resistor-capacitor pairs model the internal resistance and parasitic capacitance of the battery. All of the parameter values are nonlinear functions of system variables, such as state of charge and temperature. The nonlinear charging and discharging of the battery is captured as distinct modes of operation. Moreover, the internal battery model components differ for the different modes of operation. For example, the R_3-C_3 pair is only active during the charge mode. The two configurations are modeled by a switch in Figure 4 (left). Other configuration changes include switching between a load and a charger in the discharge versus charge modes.

Figure 4 (right) shows the HBG model of the equivalent circuit of the battery. The capacitors and resistors are in one-to-one correspondence with the electrical circuit elements. The hybrid nature of the battery is modeled by a controlled 0-junction, which is switched on and off depending on the direction of current through the battery. Most of the parameters in the battery HBG are nonlinear, and these nonlinearities are captured by making the bond graph element parameters nonlinear functions of system variables, such as the battery state of charge (SOC) and depth of charge (DOC). These two variables are computed in another portion of the model, not shown in Figure 4. As an example, the resistance of R_2 is proportional to the natural logarithm of the DOC. The variable parameters are indicated by prefixing their type-name with the letter "M", e.g., MR.

We have performed extensive system identification to estimate the parameters of the battery model, and have obtained good matches to actual observed behavior. Figure 5 shows a comparison of actual and simulated battery voltage in the battery discharge mode. The battery begins at a steady state, and as soon as a load is attached, it begins to discharge. As the battery nears its terminal voltage, the load is taken offline, and the battery begins to approach its new steady-state value. A battery charger is then connected, and the charging process generates an increase in the observed battery voltage.

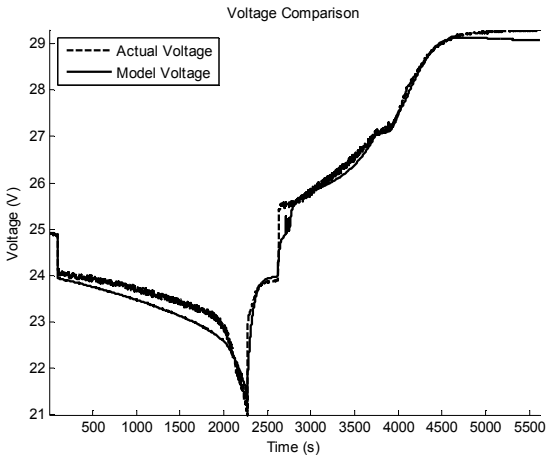

Figure 5: Comparison of actual and simulated battery voltages through discharge, no load, and charge modes.

Fault Injection

ADAPT supports the repeatable injection of faults into the system. Most of the fault injection is currently implemented via software using the *Antagonist* role described previously. Software fault injection includes one or more of the following: 1) sending commands to the testbed that were not initiated by the *User*; for this case the *Test Article* will not see the spurious command to the testbed, since it was not sent by the *User*; 2) blocking commands sent to the testbed by the *User*; 3) altering the testbed sensor data; for this case the *Test Article* and the *User* will see the altered sensor data since these reflect the faulted system. The sensor data can be altered in a number of ways, as illustrated in Figure 6. For a static fault, the data are frozen at previous values and remain fixed. An abrupt fault applies a constant offset to the true data value. An incipient fault applies an offset that starts at zero and grows linearly with time. Excess sensor noise is introduced by adding Gaussian or uniform noise to the measured value. Future work will add intermittent data faults, data spikes, and the ability to introduce more than one fault type for a given sensor at the same time. By using these three approaches to software fault injection, fault scenarios may be constructed that represent diverse component faults.

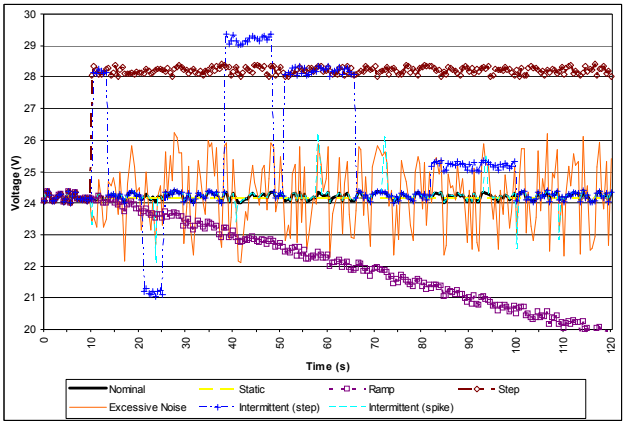

Figure 6: Example fault types.

In addition to the faults that are injected via software, faults may be physically injected at the testbed hardware. A simple example is tripping a circuit breaker using the manual throw bars. Another is using the power toggle switch to turn off the inverter. Relays may be failed by short-circuiting the appropriate relay terminals. Wires leading to or from sensors may be short-circuited or disconnected. Additional faults include blocking portions of the photovoltaic panel and loosening the wire connections in power-bus common blocks. Faults may also be introduced in the loads attached to the EPS. For example, one load is a pump that circulates fluid through a closed loop with a flow meter and valve. The valve can be closed slightly to

vary the back pressure on the pump and reduce the flow rate. Table 4 lists the faults that are injected into the testbed.

Since some fault scenarios may be costly, dangerous, or impossible to introduce in the actual hardware, VIRTUAL ADAPT also provides fault injection capabilities. For example, degradation in the batteries can be simulated as an incipient change in a battery capacitance parameter. Other parametric faults can also be injected and simulated. In addition, VIRTUAL ADAPT permits experimentation with fault scenarios that cannot be realized in the hardware, such as an inverter malfunction. Currently, mostly discrete failures (e.g., relay failures) and sensor errors are introduced into ADAPT, so the simulation provides added functionality by enabling injection of other types of fault scenarios.

Table 4: Faults injected into testbed.

Fault Description	Fault Type
Circuit breaker tripped	discrete
Relay failed open	discrete
Relay failed closed	discrete
Sensor shorted	discrete
Sensor open circuit	discrete
Sensor stuck	discrete
Sensor drift	continuous
Excessive sensor noise	continuous
AC inverter failed	discrete
PV panel blocked	continuous
Loose bus connections	continuous
Battery faults	continuous
Load faults	discrete, cont.

Test Articles

The test articles to be evaluated in ADAPT are health management applications from industry, academia, and government. The techniques employed may be data-driven, rule-based, model-based, or a combination of different approaches. Health management concepts to be investigated include fault detection, diagnosis, recovery, failure prediction and mitigation.

The technologies currently integrated with the testbed include model-based reasoning tools HyDE – Hybrid Diagnostic Engine (Narasimhan, Dearden, and Benazera 2004); FACT – Fault Adaptive Control Technology (Manders *et al.* 2006); and TEAMS-RT – Testability Engineering and Maintenance System Real Time (Deb *et al.* 1998). These tools are from government, academia, and industry, respectively. Each test article uses the ADAPT API to connect to the ADAPT message server, subscribing to the appropriate commands and data, and publishing the diagnosis results.

Each tool employs different abstraction, modeling, and reasoning methodologies. For example, TEAMS-RT typically discretizes continuous-valued sensor data into pass/fail test results. Cause-effect dependencies in a failure space, multi-signal model that link causes (components) to effects (test results) are used to isolate the fault. In contrast, FACT includes a hybrid observer that tracks continuous system behavior and mode changes. Statistical tests are used to detect a fault. Qualitative and quantitative fault signatures are used for fault isolation.

We aim to explore the advantages and disadvantages of the different approaches to health management to better understand their applicability to different fault types and operational contexts.

Performance Assessment

The purpose and scope of assessments and experiments as they relate to diagnostic and prognostic techniques and systems can vary significantly. We now identify a few different classes of assessments.

I. *Platform assessment:* Processors (CPUs) and operating systems may react differently to different diagnostic workloads. Therefore, it can be of interest to test an implementation on different computational platforms.

II. *Implementation assessment:* The programming language and the data structures can have a substantial impact on performance; hence it may be of interest to compare different implementations of the same algorithm.

III. *Algorithm assessment:* Different algorithms may solve the same computational problem. For instance, the problem of computing marginals in Bayesian networks can be solved using clique tree clustering or other approaches. Typically, this type of assessment involves the use of problem instances (Mengshoel, Wilkins, and Roth 2006).

IV. *Technique assessment:* The problem of electric power system diagnosis can be addressed using, for example, model-based reasoners, Bayesian networks or artificial neural networks.

V. *System assessment:* Overall system performance may also depend on the human(s) in the loop, and how they use and interact with different automated diagnostics systems.

Initial efforts will focus on classes III, IV, and V. The assessment of different diagnostic algorithms will consist of metrics determined from the compilation of several test runs, which include nominal and faulty behavior. For a single test run, it will be classified as a false alarm if a fault was not injected during the run but the test article reported one or if the test article reported a fault before it was injected. If the test article does not report a fault when one was injected, it will be classified as a missed alarm. The correctness and precision of fault isolation will also be determined for each run. By combining the results over several runs, false alarm rates, missed alarm rates, and isolation rates for the test article will be measured. Additional metrics include fault detection and isolation times. Not all of the metrics will apply to each test article. For example, a particular technique may only perform fault detection without isolating the cause of the fault. Additional studies will include variations in the amount and variability of data available to the test articles, i.e., only data from voltage sensors is available or some data is available intermittently.

Conclusions

This paper describes a testbed at NASA Ames Research Center for evaluating and maturing diagnostic and prognostic concepts and technologies. The electrical power system hardware, together with the software architecture and unique concept of operations, offers many challenges to diagnostic applications such as a multitude of system modes, transient behavior after switching actions, multiple faults, and load-dependent noise. A simulation model is available to facilitate the development and testing of health management applications. Future work will include the integration of more test articles, loads, faults, operational scenarios, and evaluation techniques. In meeting the five goals of the testbed listed in the Introduction, much greater knowledge of the trade-offs among diagnostic technologies will be available to system designers for future programs. Many current technology assessments have relied upon trade literature or technical publications of an algorithm's definition and performance. ADAPT provides a technology-neutral basis for the comparison of various techniques and a highly configurable operational environment for the evaluation of an algorithm's performance under specific operational requirements and fault conditions.

Acknowledgments

The authors thank Somnath Deb, Sudipto Ghoshal, Charles Domagala and Venkat Malepati from Qualtech Systems, Inc. for creating the TEAMS model of ADAPT under NASA contract number NNA06AA51Z and the TEAMS-RT application for the testbed under NASA contract number NNA06AA64C, and for their many discussions on the design of the diagnostic experiments. Contributions by RIACS were based upon work supported by NASA under award NCC2-1426. Contributions by Perot Systems were supported by NASA under contract NNA04AA18B and contributions by SAIC were supported by NASA under contract NAS2-02091. Contributions by Vanderbilt University were supported by NASA USRA grant 08020-013, NASA NRA grant NNX07AD12A, and NSF CNS-0615214.

References

Williams, B. C.; Nayak, P.; and Muscettola, N. 1998. Remote Agent: To Boldly Go Where No AI System Has Gone Before. *Artificial Intelligence* 103(1–2):5–48.

Mosterman, P. J.; and Biswas, G. 1998. A Theory of Discontinuities in Physical System Models. In *J Franklin Institute* 335 B(3):401–439.

Karnopp, D. C.; Margolis, D. L.; and Rosenberg, R. C. 2000. *Systems Dynamics: Modeling and Simulation of Mechatronic Systems.* New York: John Wiley & Sons, Inc., 3rd edition.

Manders, E. J.; Biswas, G.; Mahadevan, N.; and Karsai, G. 2006. Component-Oriented Modeling of Hybrid Dynamic Systems Using the Generic Modeling Environment. In *Proceedings of the 4th Workshop on Model-Based Development of Computer Based Systems*.

Narasimhan, S., and Biswas, G. 2007. Model-Based Diagnosis of Hybrid Systems. *IEEE Transactions on Systems, Man, and Cybernetics*, Part A, 37(3):348–361.

Roychoudhury, I.; Daigle, M.; Biswas, G.; Koutsoukos, X.; and Mosterman, P. J. 2007. A Method for Efficient Simulation of Hybrid Bond Graphs. In *Proceedings of the International Conference on Bond Graph Modeling (ICBGM 2007)*, 177–184.

Daigle, M.; Roychoudhury, I.; Biswas, G.; and Koutsoukos, X. 2006. Efficient Simulation of Component-Based Hybrid Models Represented as Hybrid Bond Graphs. Technical Report ISIS-06-712, Institute for Software Integrated Systems, Vanderbilt University, Nashville, TN, USA.

Ceraolo, M. 2000. New Dynamical Models of Lead-Acid Batteries. *IEEE Transactions on Power Systems* 15(4):1184–1190.

Mosterman, P. J., and Biswas, G. 1999. Diagnosis of Continuous Valued Systems in Transient Operating Regions. *IEEE Transactions on Systems, Man and Cybernetics, Part A* 29(6):554–565.

Mengshoel, O. J.; Wilkins, D. C.; and Roth, D. 2006. Controlled Generation of Hard and Easy Bayesian Networks: Impact on Maximal Clique Tree in Tree Clustering. *Artificial Intelligence*, 170(16–17):1137–1174.

Ledeczi, A.; Maroti, M.; Bakay, A.; Nordstrom, G.; Garrett, J.; Thomason IV, C.; Sprinkle J.; and Volgyesi P. 2001. GME 2000 Users Manual (v2.0).

Narasimhan, S.; Dearden, R.; and Benazera, E. 2004. Combining Particle Filters and Consistency-Based Approaches for Monitoring and Diagnosis of Stochastic Hybrid Systems. *15th International Workshop on Principles of Diagnosis (DX04)*, Carcassonne, France.

Deb, S.; Mathur, A; Willitt, P; and Pattipati, K. 1998. Decentralized Real-Time Monitoring and Diagnosis. *IEEE Transactions on Systems, Man, and Cybernetics.*