Fuel Cell Diagnostic Assistant

Dana Kulic
Department of Mechanical Engineering
University of British Columbia
2324 Main Mall
Vancouver, BC V6T 1Z4 Canada
+1 604 822 3147
dana@mech.ubc.ca

ABSTRACT

This paper describes a diagnostic assistant software project which has been developed for use with a portable fuel cell system. The diagnostic assistant is intended for use in a manufacturing environment to diagnose fuel cell systems which fail factory acceptance testing. Because diagnostic procedures for the fuel cell system are still being developed, the assistant is designed to allow procedures to be modified without software changes. The test configuration interface allows users to create and modify diagnostic procedures and test hardware configurations in a simple graphical interface, without having to modify any of the underlying program. The runtime interface executes test procedures created in the test configuration interface, and is designed to allow the user to quickly and easily diagnose the system, and provide good overview of the system status and test progress. The first prototype of the fuel cell diagnostic assistant has been implemented and tested by potential users. This paper describes the design and implementation of the assistant system and the user testing.

Keywords

Diagnostic software systems, fuel cell diagnostics, interactive user interface.

1 INTRODUCTION

The fuel cell industry has traditionally been focused on research and development and proving concepts via hand made prototypes. As the technology matures, focus has been shifting from research to manufacturing. If fuel cells are to be mass-produced, improvements are needed in many processes which are currently performed with little automation and frequently require highly trained workers.

One such process is fuel cell diagnostics. This paper describes a prototype system for fuel cell diagnostics which has been developed at Ballard Power Systems. The system assists users in diagnosing system failures of a portable fuel cell system during factory acceptance testing.

Overview of the Fuel Cell System

The Ballard Nexa is a 1.2 kW fuel cell system designed for integration into portable power generation devices by Original Equipment Manufacturers (OEMs). The system consists of a 1.2 kW Ballard fuel cell and ancillary hardware that is used to control the fuel cell power generation process. The fuel cell is composed of multiple individual cells which are stacked together to form a fuel cell stack. The fuel cell, when connected to an external load, produces electrical power by chemically reacting oxygen from air and hydrogen. Water is the only byproduct of the reaction. In addition to the fuel cell stack itself, the major subsystems are cooling, air supply and fuel supply. Each subsystem consists of several sensors and actuators, which are controlled by custom developed controller software and hardware. In addition to sensing and actuating the system components, the controller periodically transmits information about its status and all current sensor readings and actuator settings over a serial bus interface. The controller can also be commanded via the serial interface to enter “Diagnostics” mode. In this mode, calibration settings can be modified and actuators can be manually driven to specified control points.

Overview of the Current Diagnostic Process

The initial factory acceptance test (FAT) on each Nexa module is executed on a fully automated test station. This test station executes a test sequence confirming the operation of each sensor and actuator in the “Diagnostics” mode. Sensors are confirmed by comparing the measurement reported by the Nexa with measurements

from independent sensors mounted at the test station. If all sensors and actuators are operating properly, the system is started up and tested under multiple load points.

If any of the sensor/actuator tests fails, or if the system fails at any point during startup or operation, the failed unit is transferred to a Rework station for diagnosis. The user knows at which point in the test sequence the unit failed, but has very little information as to what component in the system actually caused the failure. The rework station is equipped with all the functionality of the automated station, and the user can display all of the information being transmitted from the Nexa controller. A programmable loadbank is also mounted on the test station and can be used to simulate various types of loading. However, the large amount of information being displayed simultaneously causes the user difficulties in isolating the cause of the problem.

Another problem is the lack of diagnostics experience with the Nexa system in manufacturing. Since this fuel cell technology has only recently been transferred to manufacturing, systematic diagnostic procedures have not yet been developed for the system. As a result, the trial and error approach is used, where components are replaced until something is found which fixes the problem.

The Diagnostic Assistant

The goal of the diagnostic assistant project was to develop a new system which would help the user to visualize and analyze the large amounts of data collected by the system. The new system, implemented on the existing Rework test station, would assist the user in identifying and isolating causes of failure in the fuel cell system. One of the primary goals of the new system was to incorporate the existing knowledge about the relationship between symptoms and causes into a more useable form. The knowledge had to be organized and represented in a way that is easily accessible and easily understood by the user. In addition, the assistant program can be used to ensure that each user follows a consistent and re-traceable procedure to identifying failures.

The focus of the user interface of the system was to present the data in a way that draws attention to anomalies and inconsistencies in the data, without overloading the user with information. The system should provide visibility into how it is "thinking", so that users can understand why the system is prescribing certain actions, and gain a better understanding of both the Nexa system and the diagnostic assistant.

One of the key features required was that the system be easily configurable and scaleable, so that as new knowledge about isolating and identifying failures is gained, it can be easily incorporated into the program without requiring software rewrites. A set of procedures to be followed for each failure type already exists in the form of flow charts written in text form. However, since both the system and the tests are fairly new, these procedures tend to change as more becomes known about failure symptoms within the system, and as more testing functionality is added. These test procedures should be configurable without having to re-program the system.

The diagnostic assistant was designed as two separate interfaces, one for configuring and specifying diagnostic test procedures, and one for executing these procedures with a malfunctioning fuel cell system. A first prototype of the diagnostic assistant has been developed and tested by potential users of the system. Section 2 of this paper describes related work in interactive diagnostic interfaces, focusing on industrial applications. Section 3 describes the design of the diagnostic interfaces. Section 4 presents the user testing performed on the prototype and discusses user responses. Finally, Section 5 concludes the paper and discusses future work on the project.

2 RELATION TO PREVIOUS WORK

A large body of work exists in the field of interactive user interface research and design. Commercially, many interactive assistants are available today, ranging from Clippie the Microsoft Office Assistant to on-line travel booking tools, system configuration Wizards, etc.

Norman [10] provides general guidelines for user interfaces. Mackinlay [9] describes a design approach based on the theory of graphical presentation.

There are many works detailing different implementations of diagnostic assistants for various industrial applications. Chiu [5] describes an electronic recovery assistant for use in interactive assistance during printer maintenance. The assistant is implemented as a hypertext web application, and also contains video footage of corrective procedures.

Koch, Isle and Butler [8] present a diagnostic assistant system used for power plant maintenance and troubleshooting. The system prompts the user for the present conditions and symptoms of the unit and provides step-by-step instructions for the maintenance technician. The technician can also access wiring schematics, component descriptions and pictures and video help files. However, no allowances are made for user modification of the knowledge base.

Kant [7] describes the implementation of an interactive problem solving system to identify hydrocarbons and their flow paths from oil-well data. The task of analyzing this data is divided into many subtasks; algorithms and programs already exist for some of these subtasks, while others require new designs. The system contains a graphical editor for constructing tasks from existing code modules. The system is clearly aimed at an expert user with programming experience, as the task configuration

system also requires users to configure many system parameters and program new modules.

Vale et al. [11] describe a user interface developed for assisting users during fault analysis and service restoration in the Portuguese power transmission network. The user interface is constructed of 3 levels of detail at which the operators can view all or detailed parts of the network. Users can modify the level of detail being displayed in the detailed views. In case of an incident, a separate window is displayed showing the location of the incident. Flashing of the incident zone is used to alert the user. In [12], the same system is extended to provide a more structured interface for novice users, and a more flexible interface for advanced users. An expert system explanation server is also added to provide explanations into the systems actions to the user.

Davidson et al. [6] develop a set of functional requirements for a troubleshooting advisory system for use with mechanical equipment. A jet engine troubleshooting implemented is developed based on the requirements. The troubleshooting assistant guides the user through a structured diagnostic network.

In addition to specific industrial applications, there are numerous articles focused on research and prototype systems. Cebulka [4] describes a prototype system which uses an iconic script editor to allow the user to graphically modify the control flow of a program by manipulating a directed graph of icons. One shortfall of the described system is that the user only has access to a predefined set of primitive icons, and cannot create additional primitive nodes.

Bartram and Ovans [2], and Bartram et al. [1] describe an intelligent user interface designed for use with supervisory control systems, such as those used in process control. Their work is focused on displaying context sensitive information to the user and presenting and facilitating multiple simultaneous events requiring user attention and response. The display is configured dynamically to present detailed views of zones requiring attention, while at the same time maintaining global context.

Blythe et al. [3] describe a system that allows users to add new knowledge to a system without having to re-program the system. The user adds knowledge by selecting from a predefined set of English paraphrases, which map to a formal representation in the system internal language. The system also interactively guides the user through the knowledge addition process. One drawback of this text-based system is that it can be difficult to visualize the flow of control and information in a text based system.

3 SYSTEM DESIGN

Requirements

In addition to hardware and communications requirements imposed by the existing test station and the Nexa controller, the following primary goals were identified for the new system, based on consultation with users of the existing system and an analysis of the diagnostics process:

- Help users to better visualize the fuel cell system and the diagnostic test procedures
- Provide a structured framework that allows each user to follow a consistent and re-traceable procedure when diagnosing failed systems
- Provide an easy-to-use interface for modifying test procedures
- Provide the capability to re-configure test station hardware without software changes

The first two requirements concern the interface the user interacts with while performing a diagnostic procedure. The last two requirements apply to how the user interfaces with the system when she is trying to modify the test procedure or the system configuration. Based on these two separate sets of requirements, two separate interfaces were developed: the test configuration interface and the runtime interface. The test configuration interface is used to create and modify diagnostic procedures and to create and modify different hardware configurations. The runtime interface is used on the test station to diagnose a failed system. The runtime interface executes the diagnostic procedures created in the test configuration interface and displays the relevant test data to the user.

Test Configuration Interface Design

User Interface Design

The test configuration interface is used to configure the hardware and communication interfaces of the test station, and to create and modify test execution sequences. The user of the test configuration interface is assumed to be the “expert” in diagnostic procedures. The user is also assumed to be familiar with the test station hardware interfaces. The configuration user does not need to be familiar with the underlying implementation of the diagnostic program, and should be able to modify procedures and hardware configurations without programming or software knowledge.

The goal of the test configuration interface design was to allow the user to specify the diagnostic sequence in an easy and familiar format, while allowing the user as much flexibility as possible to create various test scenarios. For these reasons, a graphical test configuration interface was chosen, which represents the test sequence as a flow chart (or decision tree). Each node in the tree represents a simple test, which is termed a test node. The leaves of the decision tree represent an identification of the failure cause. A test node is represented as a sequential series of actions: Actuate, Wait, Measure and Compare. The flow chart of a test node is shown in Figure 1. An identification node is

used to represent a termination of the diagnostic procedure. Identification nodes do not contain repair procedures, because repairs are usually not performed by the system diagnostics personnel, they are usually returned to the component manufacturing team.

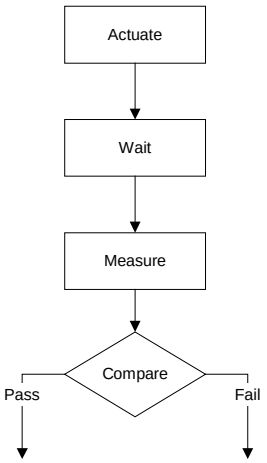

Figure 1 - Test Node Flow Chart

The test configuration interface is divided into several panels. When the user first starts the program, the main test panel appears. A picture of this panel is shown in Figure 2. This panel is used to graphically specify the test execution sequence using a decision tree diagram.

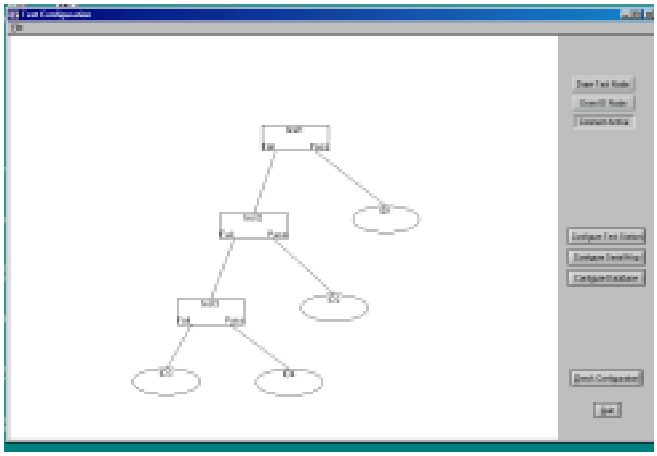

Figure 2 - Test Configuration Interface Main Panel

The test decision tree consists of test nodes and identification nodes. Nodes are placed and connected on the drawing area using the buttons in the top right corner and using familiar clicking and dragging operations. The user configures each node by double clicking on the node. When a test node is double-clicked, the test node edit window pops up. The test node edit window is shown in Figure 3. Each test is divided into four subtasks: Actuate, Wait, Measure and Compare. In the Actuate subtask, the user specifies which actuator will be activated. The Wait subtask is used to specify how long the test should wait after operating an actuator. The Measure subtask is used to specify which sensors are to be measured for this test. Since the system measures all configured sensors by default, the Measure subtask need only be used to specify manual measurements. The Compare subtask is used to specify the test pass/fail criteria. In the first prototype version of the software, only a simple compare operation with two operators is enabled, this can later be modified to include more complex Boolean operations. The user specifies two measurements to be compared, and the operator to be used to compare them.

Figure 3 - Test Node Edit Window

When an identification node is double-clicked, the identification edit window pops up. This window is used to specify what to display to the test operator when a problem has been identified by the diagnostic procedure. The description should be used to instruct the test operator which component has failed, and what the appropriate corrective action is. The identification inputs are quite simple, and no facility is provided for interactive guidance during the corrective action. This approach was chosen because in most cases, the technicians performing the diagnostic tests do not repair a failed component, they replace the component with a functioning one and report the failure.

From the main panel, the user can also call up lower level panels to configure the test station hardware, the communications format of the Nexa controller, or the test data database layout, via the configuration buttons on the right side of the screen.

Software Architecture

The software design is divided in three layers, each representing an increasing level of abstraction. The bottom, interface layer contains software modules that interface with the file system. Two separate file structures are used, one for storing the hardware configuration, and

one for storing the test sequence configuration. This allows a hardware configuration file to be re-used for multiple test sequences. The middle, translation layer contains modules that configure components of the test file. These modules include the test edit module and the identification edit module. Finally, a single top layer module handles the top-level user interface and the graphical test description.

The test configuration interface is implemented in the LabWindowsCVI environment from National Instruments. This environment was chosen because it provides an easy way to create user interface panels by automating a lot of the user interface code. At the same time, the application programming can be done in the familiar C environment.

Runtime Interface

User Interface Design

The runtime interface is used to execute the diagnostic test sequences created using the test configuration interface. The goals of the runtime interface are to display to the user parameters of interest while the diagnostic sequence is running, to prompt the user to perform manual actuation or measurement operations, to receive user feedback from these operations and to indicate to the user the results of the diagnostic procedure. This user interface needs to handle a variety of user types, ranging from the novice to the expert user. Some novice users will have only rudimentary knowledge of the Nexa system or the test station layout. Therefore, this interface needs to provide a greater level of structure to ensure that all users follow the correct diagnostic procedure.

The design of the runtime interface was guided by the need to present a large amount of data to the user without overwhelming her. For this reason, sensors and actuators were grouped by functionality into several subsystems, which are displayed on different screens. Using several screens also allowed larger displays such as gauges and thermometers to be used, which help the user to get an estimate of the sensor value by quick visual inspection, without having to read the numeric display. A strip chart is also shown on all the subsystem screens to enable the user to view time varying behavior of parameters. This design also takes advantage of the fact that the test sequence designer knows which subsystem is currently being tested, and can specify the subsystem the user should be directed to. The runtime interface will then display the specified subsystem by default, but the user can select to view the other subsystems.

The runtime interface is divided into six user interface panels. The base panel is always present in the bottom section of the screen, while the other panels alternate based on the current test being executed.

The base panel displays information on variables that are important for every test node and is displayed at the bottom of the screen throughout test execution. This panel displays the current status of the test station hardware, the loadbank and the Nexa system. The panel also displays the test and test node currently being executed, and contains links to any help files associated with the test node. From this panel, the user also controls the operation of the test; she can choose to stop the test, restart the test, or step back to the previous node.

The main panel displays a schematic diagram of the Nexa fuel cell system, showing all the subsystems and their components and the locations of all the system sensors and actuators. Besides each system sensor or actuator, an indicator is displayed showing the current value of that sensor/actuator. In the lower right corner of the screen, any warnings being issued by the controller are displayed.

The main panel provides an overview of the system. For a more detailed view, the sub-system panels are used. The fuel cell is divided into 5 major subsystems: Fuel Cell Parameters, Cooling System, Fuel Supply System and Air Supply System. Each subsystem panel contains two sets of indicators, one displaying parameters reported by the Nexa controller, and one displaying the same parameters as measured by the test station. Large meter and thermometer type displays are used in addition to the numeric displays to allow parameter changes to be easily noticeable. The software monitors each sensor to ensure that it stays within its allowable range, if a sensor exceeds its allowable range the sensor numeric indicator turns red.

Each panel also contains a strip chart that can be used to view the time-varying behavior of the parameters. The user can select which parameters are viewed by clicking on the desired parameter. Figure 4 shows a sample sub-system panel, in this case the fuel cell parameters panel. At the bottom of the screen is the base panel.

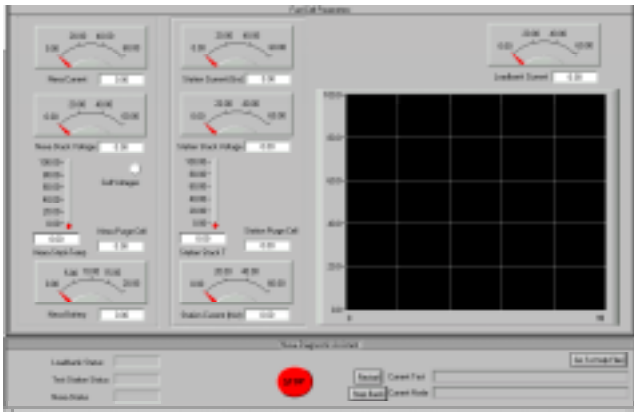

Figure 4 - Fuel Cell Parameters Panel

Software Architecture

The runtime interface follows the same design approach as

the test configuration interface, using the three layers of modules. The bottom, interface layer contains software modules which interface with the test station hardware, the Nexa and loadbank communications interfaces and access the filing system. The middle, translation layer contains a single test execution module. The test execution module executes the diagnostic test sequence based on the specification from the test configuration file. The top layer also contains a single module responsible for handling all the user interface panels.

Implementation of the runtime interface was initially started using the National Instruments Labview environment. This platform was initially chosen because the existing test station software was written in this environment, which meant that many of the low-level software modules could be re-used. However, the Labview environment does not allow for programmatic changes to the hardware configuration, or changes to the contents of the serial messages. For this reason, the runtime interface was also implemented using the LabWindowsCVI platform.

4 TESTING

Test Design

The first prototype system was tested by six employees of Ballard Power Systems. All six subjects tested the run-time interface; five subjects tested the test configuration interface. Two of the subjects were very knowledgeable of both the Nexa system and Nexa diagnostics procedures. Three of the subjects had good knowledge of the Nexa system, but were not familiar with Nexa testing procedures. One of the subjects was not familiar with either the Nexa system or the diagnostic procedures.

All the test subjects received a 30 minute demonstration of the system, during which both the test configuration and the run time interface were demonstrated. As part of the demonstration, a sample test procedure was created using the test configuration interface. This test procedure would be used to diagnose a system which had failed to startup during the automated FAT test. The test case starts the Nexa controller, and waits for a successful startup. If the startup is not successful, the test case retrieves data from the Nexa controller to determine what the cause of the startup failure is, and does further diagnostics to identify the faulty component.

Following the demonstration, each subject was asked to run the startup test case created during the demonstration using the runtime interface. To ensure that tests were consistent across subjects, and also due to lack of availability of the test station hardware, simulated data was used for the runtime interface test. The data used was from a previously tested system for which test data had been recorded. In the data used, the startup does not complete successfully, because the maximum allowable startup time is exceeded. There are several possible failures which would cause the startup time to expire before the startup sequence is completed successfully. In this case, the problem was caused by a bad connection between a sensor and the system control board.

After running the startup test case, each subject was asked to fill out a questionnaire about the effectiveness of the runtime interface. When testing the runtime interface, it would have been beneficial to compare the new interface with the existing diagnostic process. However, because of the lack of a structured procedure in the existing process, there is no good way to consistently compare the two interfaces. These issues, as well as the user responses are discussed in the section below.

Following the completion of the runtime test, each subject was asked to design a test case to diagnose a system which had failed a test of one of the system sensors during the FAT. In order to test the hardware configuration capability, part of the test case included configuring an additional sensor at the test station. Each subject was given a written description of the procedure to be programmed, which was adapted from existing manual procedures. The subjects were then asked to attempt to create this test case in the test configuration interface. Following the trial, the subjects were asked to complete a questionnaire assessing the test configuration interface, and comparing the new process with other test specification procedures.

Test Results

Runtime Interface

All of the users rated the new interface positively, but had many suggestions for improvement. During the test specification phase, the test designer specifies which sub-system will be affected by the test, so that during the runtime phase, this subsystem is displayed. In this way, the user was directed to look at data which was important to the test. As a result of this feature, most users found the data important to the test easy to see and understand. The directed views were preferred over the old system, where all the data was displayed on a single screen. However, the one novice user commented that despite this direction, he was still not sure what to focus on, and felt that the test executed too fast. Users also appreciated the large meter and gauge displays, but suggested that the numeric displays also be made larger. The idea of having chart displays was seen as beneficial, but users wanted to have more control over the chart display, including control over the time axis, the scale, and better color selection for the chart traces and their labels. Users also indicated that red should not be used as a trace color on the chart, as red was used to indicate an out of range sensor. The current system of indicating an out-of-range sensor was criticized by the majority of users; however, there was disagreement as to what would constitute a better design. Some users felt that

more visual cues were needed, such as a flashing indicator, while others considered an out-of-range sensor a safety concern, and wanted the system to take active measures to return the sensor to its allowable range, such as stopping the test and disabling any actuators.

Users also felt that the system did not provide enough information about where it was in the test sequence. One suggestion was to have an additional subsystem panel showing the test decision tree, with the current test highlighted. Another suggestion was to highlight sensors/actuators which are currently being considered by the test sequence. Users also felt that more information should be placed on the base panel, including basic information about the fuel cell stack performance. Users also wanted to be allowed to start multiple tests without having to reload the software.

One big issue raised by some of the subjects was safety. Users did not feel that it was adequate to allow the test designer to ensure that the system was shut down safely following each test procedure. The system should incorporate a shutdown procedure which is run at the end of each test, which ensures that all the system actuators are shut off. Another suggestion for ensuring safety was to only allow out-of-range sensor values for a limited time before performing the safety shutdown procedure.

Subjects assessed the new interface as better than the existing interface, but it was difficult to objectively measure this claim. The existing system does not have any structured procedures, therefore, the time taken to diagnose a system is heavily dependent on the experience of the operator. For example, during the running of the test case, one experienced user immediately noticed the problem, before the test procedure had finished running. On the other hand, the novice user found that test case had already diagnosed the problem and finished execution before he had determined what he was supposed to be looking at. The advantage of the new system is that it ensures that each operator follows the same procedure during diagnostics. However, the experiences of the novice user suggest that the system might also benefit from having an additional training mode, where the diagnostic procedure is executed slower.

Test Configuration Interface

Users reacted very positively to the test configuration interface. Most comments on the test configuration interface indicated that the interface does not provide enough features to adequately define all tests. Users wanted to see multiple ways to define the equality comparison operator, such as adding an accuracy definition, or a range of acceptable values. Users also wanted more flexibility when having to query the user. In addition to the numeric input provided, a Yes/No input dialog box was suggested.

The hardware configuration feature was assessed positively. Users wanted to have the ability to modify the range checking parameters in the hardware configuration screen.

Users also wanted to see the capability of the test configuration interface increased to allow multiple tests to be joined together. This feature was planned by the designer, but was not included in the initial prototype version tested. Users offered many suggestions on how to implement this added capability, especially on making the user's position in the hierarchy of tests easy to see.

Users also suggested making changes to the graphical drawing interface to make it more user friendly and more similar to the familiar drag and drop paradigm. The concept of representing the test sequence graphically was rated very positively, and users rated this method as preferable to other methods for specifying test configurations.

Again, it was difficult to compare the test configuration interface to the existing process, since diagnostics procedures are currently executed on an ad-hoc basis. Some users have written their own flow charts in text or tabular form, but hand written procedures do not ensure that the diagnostic sequence is followed consistently.

5 CONCLUSIONS

User response to the new system has been very encouraging, but has also identified several areas for improvement. The system is currently being modified to incorporate the results of the user testing. A shutdown safety module will be added to the runtime interface, as well as safety monitoring initiating a shutdown if critical sensors exceed their allowable ranges. The test configuration interface is also being extended to include some of the features identified during the user tests. Once these modifications are completed, the system will be installed on the test station for commissioning on the manufacturing floor. As a result of the user testing, several other potential applications are being considered, including use of the interface for diagnostics by service personnel in the field.

Once this initial version is commissioned on the test station, the system will be extended to automatically interface with the manufacturing databases. This will allow the diagnostic assistant to automatically determine which FAT test was failed by the unit, and start the corresponding diagnostic procedures automatically. Once a diagnostic identification is made, the system will also interface to the tracking database, to allow automated data collection on system failures.

ACKNOWLEDGEMENTS

This project was carried out with support from Product Development and Manufacturing teams at Ballard Power

Systems. Their support and many insightful suggestions and comments are gratefully acknowledged.

REFERENCES

[1] Bartram, L., Henigman, F., and Dill, J. (1995) The Intelligent Zoom as metaphor and navigation tool in a multiscreen interface for network control systems. In *Systems, Man and Cybernetics, 1995. Intelligent Systems for the 21st Century, IEEE International Conference on*, Volume: 4, 3122 –3127.

[2] Bartram, L. and Ovans, R. (1995) A Dialogue-Based Approach to the Design of User Interfaces for Supervisory Control Systems. In *Systems, Man and Cybernetics, 1995. Intelligent Systems for the 21st Century, IEEE International Conference on*, Volume: 4, 3144 –3149.

[3] Blythe, J., Kim, J., Ramachandran, S. and Gil, Y. (2001) An Integrated Environment for Knowledge Acquisition. In Lester, J.C. (ed.) 2001 International Conference on Intelligent User Interfaces, January 14 – 17, Santa Fe, New Mexico, 13 – 20.

[4] Cebulka, K. D. (1990) WISE: An Intelligent Interface for User Modification of Applications. In *Systems, Man and Cybernetics, 1990. Conference Proceedings, IEEE International Conference on*, 637 – 639.

[5] Chiu, S. (2000) Error Recovery Assistant for Operators of Industrial Automation. In *Proceedings of Human Interface Technologies*, 2000, 13 – 20.

[6] Davidson, P.L, Halasz, M, Phan, S. and Hakima, S. A. (1990) Intelligent troubleshooting of complex machinery. In *Proceedings of the third international conference on Industrial and engineering applications of artificial intelligence and expert systems*, Charleston, South Carolina, 16 – 22.

[7] Kant, E. (1988) Interactive Problem Solving Using Task Configuration and Control. In *IEEE Expert*, Volume: 3 Issue: 4, Winter 98, 36 - 49.

[8] Koch, C. G., Isle, B. A. and Butler, A. W. (1988) Intelligent User Interface for Expert Systems Applied to Power Plant Maintenance and Troubleshooting. In *IEEE Transactions on Energy Conversion*, Vol. 3, No. 1, March 1988.

[9] Mackinlay, J. (1998). Applying a Theory of Graphical Presentation to the Graphic Design of User Interfaces. In *Proceedings of the ACM SIGGRAPH Symposium on User Interface Software*, 1988, 179 – 189.

[10] Norman, D. A. (1988) *The Design of Everyday Things*, New York: Basic Books.

[11] Vale, Z. A., Faria, L., Ramos, C., Fernandes, M. F. and Marques, A. (1996) Towards More Intelligent and Adaptive User Interfaces for Control Center *Application In Intelligent Systems Applications to Power Systems, 1996. Proceedings, ISAP '96*, 2- 6.

[12] Vale, Z.A.; Ramos, C.; Faria, L.; Malheiro, N.; Silva, A.; Marques, A. (1999) in *International Conference on Human Interfaces in Control Rooms, Cockpits and Command Centres, 1999*, 446 - 451.