Technical Requirements for Application of Operator Support Technology across Multiple Plant Systems

Operator Support Technologies for Fault Tolerance and Resilience

Nuclear Engineering Division

About Argonne National Laboratory

Argonne is a U.S. Department of Energy laboratory managed by UChicago Argonne, LLC under contract DE-AC02-06CH11357. The Laboratory's main facility is outside Chicago, at 9700 South Cass Avenue, Argonne, Illinois 60439. For information about Argonne and its pioneering science and technology programs, see www.anl.gov.

DOCUMENT AVAILABILITY

Online Access: U.S. Department of Energy (DOE) reports produced after 1991 and a growing number of pre-1991 documents are available free via DOE's SciTech Connect (<http://www.osti.gov/scitech/>)

Reports not in digital format may be purchased by the public from the National Technical Information Service (NTIS):

U.S. Department of Commerce
National Technical Information Service
5301 Shawnee Rd
Alexandria, VA 22312
www.ntis.gov
Phone: (800) 553-NTIS (6847) or (703)
605-6000 Fax: (703) 605-6900
Email: **orders@ntis.gov**

Reports not in digital format are available to DOE and DOE contractors from the Office of Scientific and Technical Information (OSTI):

U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831-0062
www.osti.gov
Phone: (865) 576-8401
Fax: (865) 576-5728
Email: **reports@osti.gov**

Disclaimer

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor UChicago Argonne, LLC, nor any of their employees or officers, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of document authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof, Argonne National Laboratory, or UChicago Argonne, LLC.

Technical Requirements for Application of Operator Support Technology across Multiple Plant Systems

Operator Support Technologies for Fault Tolerance and Resilience

prepared by
Yu Tang and Richard Vilim
Nuclear Engineering Division, Argonne National Laboratory

September 30, 2017

(This page intentionally left blank)

ABSTRACT

Argonne National Laboratory (ANL) and Idaho National Laboratory (INL) are jointly developing and evaluating operator aids to: (i) facilitate more timely response to plant faults and grid disturbances; (ii) achieve better management of plant upsets and improved operator performance; and (iii) ultimately improve nuclear plant safety, production, and cost management. The overall objective of the collaborative effort in this third and final year of the project was to conduct human performance tests using nuclear plant operators in the Human Systems Simulation Laboratory (HSSL) control room at INL to assess the value of a computerized operator support system (COSS) as an aid to the operator. This report describes the work performed by ANL. A companion report describes the work performed by INL and the results of the human performance tests.

Much of the effort in the third year at ANL was directed at preparations for performing tests on the full-scale simulator to demonstrate the utility of the COSS as an aid to the operator for performing fault diagnosis. A first task involved software preparations, primarily the adaptation of the general-purpose modular data-bridge capability developed in year-two and its application to the plant simulator software to be used in year-three of the project.

A second task involved selecting and working with a set of faults that were not only supported by the HSSL plant simulator, but that would additionally challenge an operator in a typical plant control room. The latter criterion was met by setting the fault severities below alarm threshold levels so that the operator was less likely to detect the deviation in the plant response. Accordingly, assessment tests could be directed at measuring the benefits to the operators of receiving an early warning of a developing fault condition from the COSS. The INL report should be consulted for the results of these tests.

A final task was to develop technical requirements for application of the operator support technology across multiple plant systems.

(This page intentionally left blank)

ACKNOWLEDGMENTS

The authors thank Suibel Schuppner, Program Manager at the U.S. Department of Energy, and Bruce Hallbert, Program Director at Idaho National Laboratory, for their support through the Nuclear Energy Enabling Technology Program. We also thank Ron Boring and Ken Thomas of Idaho National Laboratory and Roger Lew and Tom Ulrich of the University of Idaho for their collaborative efforts that made possible the operator performance tests conducted at the Human System Simulation Laboratory.

(This page intentionally left blank)

TABLE OF CONTENTS

ABSTRACT	2
ACKNOWLEDGMENTS	4
TABLE OF CONTENTS	6
LIST OF FIGURES	8
LIST OF TABLES	9
1. INTRODUCTION	10
1.1 Background	10
1.2 Approach	11
1.3 Organization	11
2. BRIDGE PROGRAM DEVELOPMENT FOR NPP COSS	12
2.1 Data transfer from COSS to Bridge	12
2.2 Graphic User Interface (GUI) 1	14
2.3 Configuration Setting	16
2.4 PID File	19
2.5 Variable File	22
2.6 HTTP Response Page	23
3. TESTING BRIDGE PROGRAM THROUGH FAULTS IN NPP COSS	24
3.1 CVCS Fault Cases	24
3.2 Test Results	27
3.3 Plot Sensor Data Time History	34
4. TECHNICAL REQUIREMENTS ACROSS MULTIPLE SYSTEMS	36
4.1 View from Operator Perspective	36
4.2 Ensuring a Well-Posed Problem	36
4.3 Maintaining Currency after Initial Fault Diagnosis	37
4.4 Computing Requirements	37
5. FUTURE DEVELOPMENT	37
5.1 Multi-threading	37
5.2 Drools6	38
5.3 Utility Program for Generating the PID JASON File	38
6. CONCLUSION	38
7. REFERENCES	39
APPENDIX A: SETUP BRIDGE PROGRAM DEVELOPMENT ENVIRONMENT	40

APPENDIX B: SCHEMA OF PID JSON FILE IN JAVA	41
APPENDIX C: INPUT FILE FOR GENERATING THE PID JSON FILE	42
APPENDIX D: VARIABLE FILE FOR NPP COSS SIMULATOR.....	43

LIST OF FIGURES

Figure 1: Bridge Program GUI in the initial state (State 1).....	15
Figure 2: Bridge Program GUI in State 3	15
Figure 3: The Configuration Dialog for changing default Settings	16
Figure 4: Configuration Setting options for PRO-AID.....	17
Figure 5: Configuration Setting Options for COSS Connection	17
Figure 6: Configuration Setting Options for HTTP Response.....	18
Figure 7: NPP CVCS Overview	20
Figure 8: Screenshot of CVCS PID in NPP Simulator	21
Figure 9: Piping included in the PRO-AID model (marked in yellow).....	21
Figure 10: Simplified Model of NPP CVCS Diagnosis.....	22
Figure 11: JSON file Displayed in Web Browser. Old Way (top) and New Way (bottom)	23
Figure 12: Screenshot of Bridge Program running on Simulator Server.....	24
Figure 13: mfCV01 fault location.....	25
Figure 14: mfCV03A and mfCV04 fault locations.....	26
Figure 15: mfCV06A fault location.....	27
Figure 16: mfCV11A fault location.....	27
Figure 17: GUI Display before Fault Injection.....	28
Figure 18: GUI Display after Fault Injection.....	28
Figure 19: Diagnosis Result for Fault Case mfCV01	31
Figure 20: Diagnosis Result for Fault Case mfCV03A	31
Figure 21: Diagnosis Result for Fault Case mfCV04	32
Figure 22: Diagnosis Result for Fault Case mfCV06A	32
Figure 23: Diagnosis Result for Fault Case mfCV11A	33
Figure 24: User Interface for Parsing and Retrieving Data from a JSON file.....	35
Figure 25: Time Histories of Pressure Sensor at SEAL 1A Outlet for Different Severity	35

LIST OF TABLES

Table 1. List of APIs exposed through sim_pipe.dll [1]	13
Table 2. List of APIs Original Name vs. Mingled Names.....	13
Table 3: The Relationship of Button State and Bridge Program State	14
Table 4: State Transition Table.....	14
Table 5: Default Configuration Settings for NPP Simulator	18
Table 6: List of Available Faults in CVCS of NPP Simulator	25
Table 7: Diagnosis Results for Faults Injected to NPP Simulator	29
Table 8: Severity Threshold Study for Triggering Alarm	33

1. Introduction

1.1 Background

The Advanced Sensors and Instrumentation (ASI) technology area under the Nuclear Energy Enabling Technologies (NEET) Program was created by the U.S. Department of Energy (DOE) Office of Nuclear Energy (NE) to coordinate instrumentation and controls (I&C) research across DOE-NE programs and to identify and address common needs.

Under the NEET ASI program, Argonne National Laboratory (ANL) and Idaho National Laboratory (INL) are jointly developing and evaluating operator aids to: (i) facilitate more timely response to plant faults and grid disturbances; (ii) achieve better management of plant upsets and improved operator performance; and (iii) ultimately improve nuclear plant safety, production, and cost management. Within this framework, ANL is developing methods for equipment fault diagnosis. INL is developing the human factor aspects of assisting operators in monitoring overall plant performance and in making timely, informed decisions on appropriate control actions for the plant condition. Integration of these two activities has taken place under the concept of an operator advisory system. This is a software entity whose purpose is to manage and distill the enormous amount of information an operator must process to understand the plant state, particularly in off-normal situations, and how the state trajectory will unfold in time.

In year-one of this project, ANL developed and tested the PRO-AID fault-diagnosis software. It is a rule-based system implemented in the modern Java programming language and Drools rule engine. The rules are based on the confluence equations derived from physics principles. PRO-AID was tested using computer simulations of twenty different faults introduced into the chemical and volume control system (CVCS) of a pressurized water reactor (PWR). The methodology and software were verified and validated through batch computer simulations.

In year-two of the project, ANL developed the Bridge Program which links the PRO-AID fault diagnosis program to the full-scale simulator at the Human System Simulation Laboratory (HSSL) at INL, creating a capability for real-time fault diagnosis in a control room environment. Specifically, Bridge Program retrieves sensor data in real time from the simulator, transfers data to PRO-AID for the diagnosis, and returns the diagnosis result back to computerized operator support system (COSS) for display to the operator.

The overall objective of the collaborative effort in the third year was to conduct human performance tests using nuclear plant operators in the HSSL control room at INL to assess the value of the COSS as an aid to the operator. This report describes the work performed by ANL in the final year of this project. A companion report [6] describes the work performed by INL.

1.2 Approach

Much of the third year was directed at preparations for performing tests on the full-scale simulator to demonstrate the utility of the COSS as an aid to the operator for performing fault diagnosis

A first task involved software preparations, primarily the adaptation of the general-purpose modular data-bridge capability developed in year-two for application to the specific plant and simulator software to be used in year-three for operator performance tests. This task entailed customizing the data transfer process and the graphical user interface for reporting fault diagnosis information, and preparing a plant specific P&ID file along with an associated file listing the variables to be read from the simulator. The bridge communication capability then had to be tested.

A second task involved selecting and working with a set of faults that were not only supported by the HSSL plant simulator, but that would additionally challenge an operator in a typical plant control room. The latter criterion was met by setting the fault severities below alarm threshold levels so that the operator was less likely to detect the deviation in the plant response. Accordingly, assessment tests could be directed at measuring the benefits to the operators of receiving an early warning of a developing fault condition from the COSS. The INL report [6] should be consulted for the results of these tests.

A final task was to develop technical requirements for application of the operator support technology across multiple plant systems. The engineering and human factors findings were to be generalized into technical requirements for applying COSS technology across multiple plant systems. This entails a multi-system capability that can detect and integrate fault symptoms from more than one plant system, and integrate them into a single fault diagnosis. At the plant level, many more options for responding to the fault may be available compared to options within a single plant system. A requirements document can serve as a technical reference for defining and testing a multi-system COSS, the development of which is beyond the scope of this project but would be appropriate for a follow-up project.

1.3 Organization

This report is divided into three major sections: the development of the Bridge Program, the testing of the Bridge Program, and a description of technical requirements for application across multiple systems. Section 2 describes the modifications that were made to the Bridge Program to work with a new simulator in place of the original GPWR simulator. In Section 3, the software tests performed on the new simulator to validate the data flow from the simulator to Bridge Program and to assess the PRO-AID fault diagnosis capability are described. Section 4 describes how the engineering and human factors findings can be generalized into technical requirements for applying COSS technology across multiple plant systems.

2. Bridge Program Development for NPP COSS

The Bridge Program was first created during year-two of the project for the GPWR simulator installed at that time in the HSSL at INL. The GPWR simulator was subsequently replaced with what we refer to as the *NPP simulator*, where NPP is taken to reference a particular nuclear power plant that cannot be named here for proprietary reasons. Therefore, the Bridge Program needs to be modified for the new simulator because the interface for data exchange is different from that of GPWR. This report documents the major changes made to the Bridge Program and tests performed to validate the implementations. How to setup the Bridge Program development environment is presented in Appendix A.

2.1 Data transfer from COSS to Bridge

The NPP simulator is a 3KEYMASTER software environment which is a Windows-based system developed by Western Services Corporation (WSC). It provides several interfaces for external programs to communicate with the simulator. These interfaces include the Dynamic Data Exchange (DDE), shared memory, named pipes, and TCP/IP. WSC provides Application Programming Interfaces (APIs) for the data exchange. Bridge Program chooses to use named pipes for the communication with the simulator. The interface APIs for named pipes are written in C++ under Win32 environment, and they are contained in a dynamic link library, `sim_pipe.dll`. Bridge Program is a Java application. For Bridge Program to call C++ functions, a Java package named Java Native Access (JNA) needs to be used to write wrapping functions to convert the Java function calls to the corresponding native API calls. Names of these APIs are given in [1], and they are reproduced and presented in Table 1. However, it was found during the code development that these wrapping functions did not work if names listed in Table 1 were used in the function calls. The reason for this failure was that these API names are mingled after compilation; therefore, API names need to be mapped to the corresponding mingled names before the function calls. A utility program, `dumpbin.exe`, provided in Microsoft Visual Studio package, was used to retrieve these mingled names. Listed in Table 2 are the original API names and the corresponding mingled names. Note that only those API functions used in Bridge Program are included in the table.

Table 1. List of APIs exposed through sim_pipe.dll [1]

```

Void pascal SetPipe(BOOL);
void pascal SetInstance(LPSTR aSimInstance);

BOOL pascal ConnectToServer(LPSTR);
BOOL pascal ConnectToServerNamed(LPSTR servername, LPSTR username);
void pascal DisconnectFromServer();

void pascal SendCommand(LPSTR);
BSTR pascal ExecCommand(LPSTR request);

BSTR pascal GetVar(LPSTR);
BSTR pascal GetCsvVar(LPSTR);

BOOL pascal OpenShMem();
void pascal CloseShMem();
int pascal GetVarPtr(LPSTR varname, int *TypeIndex, int *ArrDim);
BSTR pascal GetVarValue(int pVarData, int TypeIndex);
void pascal SetVarValue(LPSTR Value, int pVarData, int TypeIndex);
int pascal GetExeStatus();
void pascal SetByteValue(char pData, int pVarData);
void pascal SetShortValue(short pData, int pVarData);
void pascal SetIntValue(int pData, int pVarData);
void pascal SetFloatValue(float pData, int pVarData);
void pascal SetDoubleValue(double pData, int pVarData);
void pascal SetStringValue(LPSTR pData, int pVarData);
char pascal GetByteValue(int pVarData);
short pascal GetShortValue(int pVarData);
int pascal GetIntValue(int pVarData);
float pascal GetFloatValue(int pVarData);
double pascal GetDoubleValue(int pVarData);
BSTR pascal GetStringValue(int pVarData);

```

Table 2. List of APIs Original Name vs. Mingled Names

Original API Name	Mingled Name
SetPipe	_SetPipe@4
ConnectToServer	_ConnectToServer@4
ConnectToServerNamed	_ConnectToServerNamed@8
DisconnectFromServer	_DisconnectFromServer@0
SendCommand	_SendCommand@4
ExecCommand	_ExecCommand@4
GetVar	_GetVar@4
GetCsvVar	_GetCsvVar@4
SetVarValue	_SetVarValue@12
GetExeStatus	_GetExeStatus@0

2.2 Graphic User Interface (GUI) 1

The Bridge Program GUI is shown in Figure 1. It is the window the user sees when Bridge Program is running. It consists of three menu items at the top of the window, four buttons in the middle of the window, and a text area at the bottom of the window. Three menu items are File, Project, and Help. The File menu has two submenus – “Configuration” and “Exit”. The Project menu has four submenus – “Poll PVs”, “Get Status”, “Clear Text Area”, and “Save Data as JSON”. The Help menu has two submenus – “System Information”, and “About”. Description of File menu is given in the next Section. Descriptions for Project menu and Help menu can be found in [2]. The text area is the space used to display the diagnosis results. The four buttons are two Start buttons and two Shutdown buttons. The upper pair of Start and Shutdown buttons are associated with “Link with COSS” and the lower pair are associated with “Invoke PRO-AID”. In the following description, the upper two buttons are referred to as Start-1 and Shutdown-1, and the lower two buttons are referred to as Start-2 and Shutdown-2. These four buttons are either active or inactive during the program execution depending on the state of the Bridge Program. Therefore, the Bridge Program and the four buttons constitute a finite state machine. When running, the state of the Bridge Program can only be in one of the three states. These three possible states are:

State 1: COSS is not linked and PRO-AID is not invoked

State 2: COSS is linked and PRO-AID is not invoked

State 3: COSS is linked and PRO-AID is invoked

The initial state is State 1. Each button has two states: active and inactive. The button-click event causes the state transition. Table 3 defines the button state related to the Bridge Program state. Table 4 defines the state transition associated with the button-click event.

Table 3: The Relationship of Button State and Bridge Program State

Program State	Start-1 Button	Start-2 button	Shutdown-1 button	Shutdown-2 button
1	Active	Inactive	Inactive	Inactive
2	Inactive	Active	Active	Inactive
3	Inactive	Inactive	Active	Active

Table 4: State Transition Table

Current State Event	State 1	State 2	State 3
Start-1 click	State 2	-	-
Start-2 click	-	State 3	-
Shutdown-1 click	-	State 1	State 1
Shutdown-2 click	-	-	State 2

For example, the Start-2 click event will bring Bridge Program from State 2 to State 3. The GUI displays these four buttons as either active or inactive according to the state of Bridge Program. Figure 1 shows the four buttons in State 1 condition, and Figure 2 shows the four buttons in State 3 condition.

Figure 1: Bridge Program GUI in the initial state (State 1)

Figure 2: Bridge Program GUI in State 3

2.3 Configuration Setting

The first submenu of the File menu is Configuration. Click this submenu. A configuration dialog appears and is shown in Figure 3. At the upper left corner of the dialog, there are three setting categories. They are PRO-AID Settings, COSS Connection Settings, and HTTP Response Settings. Each of these categories can be expanded to reveal the setting options. Shown in Figures 4, 5, and 6, are setting options for these three categories, respectively. The default settings and brief descriptions of options for NPP simulator are given in Table 5.

Figure 3: The Configuration Dialog for changing default Settings

Figure 4: Configuration Setting options for PRO-AID

Figure 5: Configuration Setting Options for COSS Connection

Figure 6: Configuration Setting Options for HTTP Response

Table 5: Default Configuration Settings for NPP Simulator

Configuration Item	Default Value	Description
PRO-AID Settings		
PID File	pid_Palo_verde-2017.json	The PID file contains the connections between plant components.
Diagnostic Path	C:\PRO-AID\2017\DIAGNOSIS	The Diagnostic path is where JSON files containing the diagnosis results are dumped for each time step.
Variable File	variableNames_PaloVerde-2017.txt	The Variable File contains a mapping from component and sensor names in the COSS database to component and sensor locations in PRO-AID.
Single Fault	true	Whether to stop the PRO-AID diagnosis once a fault is detected?
N(Occurrence)	1	Frequency that the same fault being detected
Debug Logging	false	Whether to have debug logging?
COSS Connection Settings		
Host Server	.	The NPP Server location (a dot means the same computer).
Host Service	Not used	Reserved for GPWR
Host UDS	Not used	Reserved for GPWR

Executive	Not used	Reserved for GPWR
Executive PID	0	Reserved for GPWR
HTTP Response Settings		
Server Port	7890	The HTTP server port.
Response Page	currentmessage	The HTTP page where diagnosis result is posted.

Note that the file menu is disabled once the COSS link is initialized and connected. This is to prevent the user from changing any configuration settings after the connection. The reason is that after COSS is connected, Bridge Program creates a simulator object, which binds an http server with the port number specified in the configuration. This simulator object is not deleted even after the diagnosis process is ended because the user may still want to access the HTTP page. Therefore, it is a design decision to disable the file menu so that the user is not able to make any configuration changes and tries to create another simulator object.

2.4 PID File

Bridge Program requires two input files, PID JSON file and Variable File, to initialize a PRO-AID diagnostic session. The PID file for NPP simulator is described in this section, and Variable File is described in the following section.

PID JSON file provides the components and connection information of a piping and instrumentation diagram (PID) that PRO-AID is monitoring. It is in the JSON file format and must include four sections: sensors, loops, components, and adjacencies. Section “sensor” is an array of sensor objects. Each object has the following attributes: “componentName”, “loopName”, “primary”, “secondary”, “measureType”, and “measureLoc”. Section “loops” is an array of loop objects having the following attributes: “name”, “fluidType”, “type”, “isOpen”, and an array of components that it contains. Section “components” is an array of component object with attributes of “name”, “relativePressure”, “fluidType”, “type”, “isSeparated”, and an array that lists the loops to which it belongs. Section “adjacencies” is an array of length-2 arrays; each length-2 array specifies two sides of a heat exchanger. The schema of the PID file described above may be confusing and difficult to grasp. A better way to present this schema is to cast it into a Java object, PIDFileJSONObj, that contains three objects, LoopObj, SensorObj, CompObj, and a two-dimensional array, “adjacencies”. This representation is believed to be easier to understand. Definitions of these objects are included in Appendix B.

The NPP chemical and volume control system (CVCS) is shown in Figure 7. This diagram, however, does not provide information regarding the flow direction. This information can be found in a screenshot of PID diagram of the NPP simulator in Figure 8. In the PRO-AID diagnosis model, only a portion of the CVCS piping is included. The included piping is marked in yellow and is shown in Figure 9. It is further simplified to eliminate some unimportant components. The final piping is rearranged and shown in Figure 10. Note that in Figure 10, there

are red circles sticking out from the piping with a letter inside. These red circles represent sensors, and the letter inside the circle represents the type of sensor. P stands for pressure sensor, T for temperature sensor, F for flow sensor, and ΔP for differential pressure sensor. The simplified piping diagram shown in Figure 10 is used to prepare the PID JSON file for the PRO-AID diagnosis. The PID JSON file is not prepared manually because it is a tedious and error-prone process. Instead, one prepares another input file for a utility program written in Python that, in turn, generates the PID JSON file. The user manual for the utility program is available [Reference 3]. The input file for the utility program prepared for the NPP CVCS based on the PID shown in Figure 10 is included in Appendix C.

CVCS Overview

Figure 7: NPP CVCS Overview

Figure 8: Screenshot of CVCS PID in NPP Simulator

CVCS Overview

Figure 9: Piping included in the PRO-AID model (marked in yellow)

Figure 10: Simplified Model of NPP CVCS Diagnosis

2.5 Variable File

This is the second input file required for Bridge Program. It provides information that links component names in the PID JSON file to sensor and component names in the COSS simulator database. The file structure and format are described in [2]. Every line of the Variable File contains three pieces of information. The first is the name of an item in the COSS simulator database. The second is the corresponding item name in the PID JSON file. The third is a brief description of the item. Note that an item can be either a component or a sensor. However, there are no sensor names specified in the PID JSON file, but every sensor has a name in COSS simulator database. It is necessary to find a method to link the two. It was noticed that sensors are associated with components. Therefore, an artificial naming convention is created for sensors used in PRO-AID to link to the sensors in the COSS database. Per this naming convention, a sensor in PRO-AID is given a composite name that consists three information: component, type, and location, separated by a colon (":") as component:type:location. For example, a temperature sensor located at the outlet of Volume Control Tank (VCT) is named as vct:temp:out. The third piece of information in every line of the input file is a short description of the item. This information is not used in PRO-AID diagnosis, but it is a useful information for debugging. During the execution of PRO-AID diagnosis, Bridge Program retrieves data from COSS simulator and hands over to PRO-AID according to the mapping specified in the Variable File; therefore, the accuracy of this file is essential for the diagnosis. One should check and recheck

this file carefully. The Variable File for the NPP COSS simulator is included in Appendix D.

2.6 HTTP Response Page

During the execution of PRO-AID diagnosis, Bridge Program receives sensor data at discrete time-steps and the resulting diagnosis is produced. This diagnosis result needs to be communicated back to the COSS system for examination to determine if any action needs to be taken. To facilitate this data transfer, Bridge Program creates an HTTP server and post the diagnosis result in JSON file format for COSS system to dynamically request and parse. The diagnosis result also posts on an HTTP page that can be displayed by a web browser. The server port and the webpage can be configured before the execution of PRO-AID diagnosis. Currently, the default port is 7890 and the default webpage is “currentmessage”. Therefore, the link for a web browser on the simulator server would be <http://localhost:7890/currentmessage>.

The HTTP page is constantly overwritten with the new diagnosis result. The user needs to refresh the webpage to see the updated result. This is not convenient. A better approach is to use a Java Servlet to refresh the page automatically. This change may be implemented in the future. The old Bridge Program sends the result in JSON format to the HTTP server without any modification; as a result, the data displayed in the web browser is a one-line string. An example is shown in the top image of Figure 11. Clearly, it is difficult to read. The newly modified Bridge Program breaks the string into multiple lines and adds the diagnosis result for easy reading as shown in the bottom image of Figure 11.

Figure 11: JSON file Displayed in Web Browser. Old Way (top) and New Way (bottom)

3. Testing Bridge Program through Faults in NPP COSS

After finishing modifications to Bridge Program for the NPP simulator, tests were performed to validate the data flow from the simulator to Bridge Program and to assess the PRO-AID diagnosis capability. This requires uploading Bridge Program from ANL to the simulator at INL and executing the program from ANL remotely. Before uploading, Bridge Program is exported to a runnable JAR file to maintain the proprietary nature of PRO-AID. Descriptions on how to export Bridge Program to a JAR file and how to upload and execute the program remotely are given in [4]. Shown in Figure 12 is Bridge Program running on the simulator server in the foreground and the NPP simulator running at the background.

Figure 12: Screenshot of Bridge Program running on Simulator Server

3.1 CVCS Fault Cases

There are fifteen available faults in CVCS of the NPP simulator that can be injected into the simulator for testing. These faults are listed in Table 6.

Table 6: List of Available Faults in CVCS of NPP Simulator

Malfunctions - CV - CHEMICAL AND VOLUME CONTROL							
Name	Description	Type	Low	High	Units	Current	Final
mfCV01	CHG LN LK BTWN CH-V429 & CH-HV-524	V	0	100	%	0	AsIs
mfCV02	LD TEMP CNTRLR CHN-TIC-223 FAILURE	V	0	100	%	0	AsIs
mfCV03A	CHN-LV-110P LD CONTROL VLV FAILURE	V	0	100	%	36.7761	AsIs
mfCV03B	CHN-LV-110Q LD CONTROL VLV FAILURE	V	0	100	%	0	AsIs
mfCV04	LOCA OUTSIDE CNTMT DWNSTRM OF RHX	V	0	100	%	0	AsIs
mfCV05	LOCA OUTSIDE CNTMT UPSTRM OF CH-V418	V	0	100	GPM	0	AsIs
mfCV06A	CHA-P01 CHARGING PUMP TRIP	B	0	1			
mfCV06B	CHB-P01 CHARGING PUMP TRIP	B	0	1			
mfCV06C	CHE-P01 CHARGING PUMP TRIP	B	0	1			
mfCV07A	CHA-P01 CHG PUMP BECOMES AIRBOUND	B	0	1			
mfCV07B	CHB-P01 CHG PUMP BECOMES AIRBOUND	B	0	1			
mfCV07C	CHE-P01 CHG PUMP BECOMES AIRBOUND	B	0	1			
mfCV08	RWT RUPTURE	V	0	100	%	0	AsIs
mfCV09	BORIC ACID FILTER BLOCKAGE	V	0	100	%	0	AsIs
mfCV10	REACTOR MAKEUP WTR FILTER BLOCKAGE	V	0	100	%	0	AsIs
mfCV11A	RCP-1A RCP #1 SEAL FAILURE	V	0	100	%	0	AsIs
mfCV11B	RCP-1B RCP #1 SEAL FAILURE	V	0	100	%	0	AsIs
mfCV11C	RCP-2A RCP #1 SEAL FAILURE	V	0	100	%	0	AsIs
mfCV11D	RCP-2B RCP #1 SEAL FAILURE	V	0	100	%	0	AsIs
mfCV12A	RCP-1A RCP #2 SEAL FAILURE	V	0	100	%	0	AsIs
mfCV12B	RCP-1B RCP #2 SEAL FAILURE	V	0	100	%	0	AsIs
mfCV12C	RCP-2A RCP #2 SEAL FAILURE	V	0	100	%	0	AsIs
mfCV12D	RCP-2B RCP #2 SEAL FAILURE	V	0	100	%	0	AsIs
mfCV13A	RCP-1A RCP #3 SEAL FAILURE	V	0	100	%	0	AsIs
mfCV13B	RCP-1B RCP #3 SEAL FAILURE	V	0	100	%	0	AsIs
mfCV13C	RCP-2A RCP #3 SEAL FAILURE	V	0	100	%	0	AsIs
mfCV13D	RCP-2B RCP #3 SEAL FAILURE	V	0	100	%	0	AsIs
mfCV14A	RCA-LT-10X PRESSURIZER LEVEL TRANSMITTER FAILS	V	0	100	%	52.5991	AsIs
mfCV14B	RCB-LT-10Y PRESSURIZER LEVEL TRANSMITTER FAILS	V	0	100	%	52.5991	AsIs
mfCV14C	RCN-LT-10Z PRESSURIZER LEVEL TRANSMITTER FAILS	V	0	100	%	36.09	AsIs
mfCV15	PLCS MASTER CONTROLLER SETPOINT FAILS	V	0	100	%	52.6	AsIs

Since the PID file created for testing contains only a partial portion of the piping system in the CVCS, not all faults in Table 6 can be meaningfully included in the test. The faults included in the tests are mfCV01, mfCV03A, mfCV04, mfCV06A, and mfCV11A.

mfCV01

The location of mfCV01 fault is shown in Figure 13. This fault is caused by a leak at the pipe location right before Valve V524 (see Figure 10 for the location of V524).

Figure 13: mfCV01 fault location

The location of mfCV03A fault is shown in Figure 14. This fault is due to Valve V110P positioner failure causing letdown flow to increase.

mfCV04

The location of mfCV04 fault is also shown in Figure 14. This fault is due to weld failure at Valve V523 causing a leak and increase of letdown flow through Regen HX. This malfunction is non-recoverable.

Figure 14: mfCV03A and mfCV04 fault locations

mfCV06A

The location of mfCV06A fault is shown in Figure 15. This fault is due to the faulty relay causing breaker trip at charging pump CHA-P01. The breaker cannot be reclosed. The pump trip is annunciated. The decrease in charging line flow causes increased Regen HX outlet temperature.

Figure 15: mfCV06A fault location

mfCV11A

The location of mfCV11A fault is shown in Figure 16. This fault is due to seal failure at SEAL 1A causing pressure drop. At 100%, severity, pressure drop across the seal is almost zero PSID.

Figure 16: mfCV11A fault location

3.2 Test Results

During the execution of Bridge Program, the GUI displays the diagnosis result at the text area as the time advances. Before a fault is injected, the simulator is in a steady state and the display should show “No fault detected”. After the fault is inserted, the display shows the diagnosis result almost immediately. The GUI display for an example of PRO-AID diagnosis session is shown in Figures 17 and 18 for before and after the fault injection, respectively.

Figure 17: GUI Display before Fault Injection

Figure 18: GUI Display after Fault Injection

The accuracy of PRO-AID diagnosis, obviously, depends on the number of sensors in the piping system. This issue has been studied extensively in [5]. In that study, three categories are proposed to categorize a diagnosis result. They are complete, semi-complete, and incomplete diagnosis. Complete diagnosis means that the outcome of the fault case identifies exactly the specific faulty component or location and contains no other diagnoses. Semi-complete diagnosis means that the outcome of the fault case contains the correct information but may also contain other information not fully localized to the fault or the reasoning for the fault is not complete.

This outcome is still useful and helpful to operators. Finally, incomplete diagnosis means that the diagnosis result does not contain information related to the fault under consideration. Note that these categorizations are not useful in a live detection application because the category of a fault is not known a priori.

The test results for the five cases are summarized in Table 7.

Table 7: Diagnosis Results for Faults Injected to NPP Simulator

Fault Label	Diagnosis Result
mfCV01	<p>2 possible faulty components Component CHA-HV-524 due to Negative Logic Component junction_11 due to Negative Logic</p> <p>4 Pipe Blockages or Breaks junction_11 to CHE-UV-231P break unk_1 to junction_11 break junction_11 to CHA-HV-524 break CHA-HV-524 to s_reg_htx_CHE-E01 break</p>
mfCV03A	<p>4 possible faulty components Component CHB-UV-523 due to Negative Logic Component CHB-UV-515 due to Negative Logic Component CHE-LV-110P due to Negative Logic Component CHA-UV-516 due to Negative Logic</p> <p>2 Pipe Blockages or Breaks CHA-UV-516 to t_reg_htx_CHE-E01 break t_reg_htx_CHE-E01 to CHB-UV-523 break</p>
mfCV04	<p>5 possible faulty components Component CHB-UV-523 due to Negative Logic Component CHB-UV-515 due to Negative Logic Component CHE-LV-110P due to Negative Logic Component filter_rc due to Negative Logic Component CHA-UV-516 due to Negative Logic</p> <p>15 Pipe Blockages or Breaks normal_letdown to CHB-UV-515 break CHB-UV-515 to CHA-UV-516 break CHA-UV-516 to t_reg_htx_CHE-E01 break t_reg_htx_CHE-E01 to CHB-UV-523 break CHB-UV-523 to CHE-LV-110P break CHE-LV-110P to Letdown_HX break Letdown_HX to CHE-PV-201P break CHE-PV-201P to filter_rc break</p>

	CHB-UV-515 to CHA-UV-516 cloggage CHA-UV-516 to t_reg_htx_CHE-E01 cloggage t_reg_htx_CHE-E01 to CHB-UV-523 cloggage CHB-UV-523 to CHE-LV-110P cloggage CHE-LV-110P to Letdown_HX cloggage Letdown_HX to CHE-PV-201P cloggage CHE-PV-201P to filter_rc cloggage
mfCV06A	4 possible faulty components Component V339 due to Negative Logic Component CHA-P01 due to Negative Logic Component V316 due to Negative Logic Component UV-501 due to Negative Logic 8 Pipe Blockages or Breaks VCT to UV-501 break UV-501 to V316 break V316 to CHA-P01 break CHA-P01 to V339 break V339 to unk_1 break UV-501 to V316 cloggage V316 to CHA-P01 cloggage CHA-P01 to V339 cloggage
mfCV11A	3 possible faulty components Component SEAL_1A due to Negative Logic Component CHE-FV-241 due to Negative Logic Component junction_sw_3 due to Negative Logic 0 Pipe Blockages or Breaks

The diagnosis results listed in Table 7 are also presented graphically. For each fault case, the diagnosis result is highlighted in yellow on the PID for leaks or blockage piping and in orange for faulty components. They are presented in Figures 19 through 23.

Figure 19: Diagnosis Result for Fault Case mfCV01

Figure 20: Diagnosis Result for Fault Case mfCV03A

Figure 21: Diagnosis Result for Fault Case mfCV04

Figure 22: Diagnosis Result for Fault Case mfCV06A

Figure 23: Diagnosis Result for Fault Case mfCV11A

When injecting a fault into the system, one can specify the fault severity. If the severity exceeds a threshold, the alarm will be triggered. To show that PRO-AID is a worthwhile detection tool, we need to show that PRO-AID can detect low severity faults that do not trigger alarms. A series of tests were performed to determine the severity that would not trigger the alarm. Table 8 summarizes the findings.

Table 8: Severity Threshold Study for Triggering Alarm

Fault Label	Findings
mfCV01	<ul style="list-style-type: none"> At 10%, PRO-AID did not detect the fault. Alarm was not triggered Between 20% and 40%, PRO-AID was able to detect the fault. Alarm did not go off. Above 50%, PRO-AID detected the fault and alarm went off.
mfCV03A	Not tested.
mfCV04	At 1%, PRO-AID detected the fault 18 seconds after the fault became active. Alarm went off after 20 minutes.
mfCV06A	Alarm always triggered
mfCV11A	<ul style="list-style-type: none"> At 10%, PRO-AID did not detect the fault. Alarm was not triggered Between 20% and 40%, PRO-AID was able to detect the fault. Alarm did not go off. Above 50%, PRO-AID detected the fault and alarm went off.

Having successfully tested the five fault cases, the results were forwarded to project team members at INL. They selected the leak outside containment and seal failure for use in operator tests. The description of the tests and results are given in [6].

3.3 Plot Sensor Data Time History

Bridge Program retrieves sensor data at every time step and stores them in memory. When the user click “Save Data As JSON” under the Project menu, Bridge Program dumps data to files in JSON format. Two files are created; one contains the sensor data only, and the other contains all data for variable names specified in the Variable File. These files are created in the same folder as that of the Jar file. The names of these two files are “data Histroynnnn.json” and “sensor Histroynnnn.json” where nnnn is the timestamp for the time when the file is created. These two files need to be parsed to retrieve numerical data for plotting. Two Excel spreadsheet utility programs are created, DataHistory.xls and SensorHistory.xls, for processing data history JSON file and sensor history JSON file, respectively. They are located at

[\\nyx\Groups\PRO-AID\SoftwareNeeded\PlotSensorDataHistory\DataHistory](#) folder; and [\\nyx\Groups\PRO-AID\SoftwareNeeded\PlotSensorDataHistory\SensorHistory](#) folder.

These two files have the same user interface as shown in Figure 24, which has two buttons, “Parse File” and “Generate Data 4 Plot”. One clicks the “Parse File” button to parse the “JSON.txt” file and clicks the “Generate Data 4 Plot” to retrieve time history data. The parsed data are stored in “JSON” worksheet and the data for plotting are stored in “Data” worksheet. Before running the utility program, one needs to rename the JSON file generated from Bridge Program to JSON.txt. The utility program will parse JSON.txt by default.

Presented in Figure 25 are the time history plots for a pressure sensor at the SEAL 1A outlet. The percentages indicated in the figure are the fault severities specified for the fault before the injection. The variable name for this pressure sensor is cvxrcnpt152.

Figure 24: User Interface for Parsing and Retrieving Data from a JSON file

Figure 25: Time Histories of Pressure Sensor at SEAL 1A Outlet for Different Severity

4. Technical Requirements across Multiple Systems

The technical requirements for applying COSS technology across multiple plant systems are described here. [7] The COSS needs to be able to detect and integrate fault symptoms from more than one plant system, and integrate them into a single fault diagnosis. At the plant level, many more options for responding to the fault may be available compared to options within a single plant system.

The requirements serve as a technical reference for defining and testing a multi-system COSS, the development of which is beyond the scope of this project but would be appropriate for a follow-up project.

The ultimate implementation of this technology would be a plant level COSS that has all production and safety critical systems modeled in an integrated fashion, such that faults of all types, resulting in symptoms that are manifested in multiple plants systems, can be readily detected and diagnosed. Mitigation actions could then be applied across multiple plant systems to arrest the degraded plant conditions and return the plant to its normal operating conditions.

For a fully-functional COSS, an estimated 25-30 plant systems must be monitored to detect upsets that can challenge plant safety, initiate a plant transient, or cause some other situation that requires prompt operator actions. Technical requirements for achieving this are described below.

4.1 View from Operator Perspective

The current implementation is limited mainly to diagnosing faults in a single plant system without consideration for what is happening in other plant systems. So while, for example, a leak in the regenerative heat exchanger in the Chemical and Volume Control System might affect the conditions in the Pressurizer Pressure Control System, in the current implementation, the COSS would not associate a related drop in pressurizer level with the heat exchanger leak. However, for the operator the possible linkage of these two events is something that would be entertained by him as part of his fault diagnosis process. Therefore, to make the COSS of greater utility to the operator, there is a requirement that it be capable of discerning related interactions between multiple systems do to a fault in one of these systems.

4.2 Ensuring a Well-Posed Problem

Since PRO-AID is based on conservation balances it is a requirement that all forcing functions driving a system are known, i.e. measured. This includes being able to infer from sensors the mass, energy and momentum that enter from an adjacent system and the influence on these quantities of any actuators in the system. Invoking the concept of mass and enthalpy transport, it may not be necessary to measure some quantities directly but to instead use upstream measurements when available.

System-level models will see effects of a fault but all energy balances across unfaulted systems should be maintained. The faulted system will of course exhibit an energy imbalance. A plant overview model might be needed at the top of the hierarchy to monitor system boundary conditions for faults occurring at those locations.

Individual system models will need a set of variants that cover all normal operating configurations. For example, many systems have alternate flow paths for operational purposes. These could be alternate charging pumps in service, demineralizer loops in or out of service, etc. The model configuration variants can be generally recognized by valve, damper, pump, and breaker status.

4.3 Maintaining Currency after Initial Fault Diagnosis

COSS must recognize when system models have changed to another variant due to automatic or operator actions. Some of these actions will be expected in response to fault conditions. These actions may or may not clear the fault condition. For example, closing a valve might isolate a leak. The new model variant will determine whether a fault condition is still present.

Control actions in unfaulted systems will possibly alter conditions in both faulted and unfaulted systems. For instance, flow rates into these systems might change with certain pumps being started or shut down. Therefore, the models will need to compute balances on a new set of conditions. In some extreme cases, secondary faults can be created by these conditions. For example, a check valve (first fault) could fail leading to an over-pressurization of a piping segment and thereby causing a relief valve to lift or a flanged pipe joint to fail (second fault).

4.4 Computing Requirements

Due to processor limitations to maintain acceptable time response, multiple processors may have to be employed in a hierarchical architectural. This is so each of the various systems models can be processed every few seconds or so – similar to what an operator can do for a single system.

5. Future Development

5.1 Multi-threading

During tests, occasionally, the Bridge Program receives an exception in JavaFX Application thread unexpectedly and the GUI becomes nonresponsive. This behavior is not reproducible and, hence, unpredictable. After extensive Google searching, it is found that this behavior is caused by accessing a JavaFX object from a thread other than the JavaFX Application thread. Apparently, the graphical user interface of a JavaFX application is not thread-safe, which means that JavaFX objects of GUI can only be accessed and modified from the JavaFX Application

thread. Currently, Bridge Program creates a runnable object containing an infinite loop and spawns off a new thread to perform work specified in the loop. In every loop, the work includes retrieving sensor data from the simulator and performing the PRO-AID diagnosis; then, at the end of loop, it posts the diagnosis result to the TextArea of the GUI, which is a JavaFX object. If at this point of time, the user clicks any JavaFX objects on the GUI, this action may lead to a race condition that may cause memory inconsistency errors, and an exception is thrown. This error, however, occurs rarely. It will be addressed in the next development cycle. For now, the guidance for avoiding this error from happening is “do not click to activate any GUI object while PRO-AID is running”.

Currently, the frequency of PRO-AID diagnosis is once per second. If the number of sensors increases, the time for diagnosis process may exceed one second. Under this situation, if it is important to keep the frequency at one second per diagnosis, Bridge Program may need to spawn two threads, one retrieves data and the other one performs the diagnosis, in other words, to implement a multithreading reader-writer problem. This implementation is planned for the future if there is a need.

5.2 Drools6

The rule engine that is compatible with the current implementation of PRO-AID is Drools 5.6.0.Final, which is an old version and is not supported. The current version of Drools is Drools 7.1.0.Final. From Drools 5 to Drools 6, the algorithm used in the rule engine is changed; APIs and jar files are renamed. The old algorithm is the Rete algorithm. It is now the Phreak algorithm. Therefore, Drools version 6+ is not backwards compatible with Drools 5. Migration of PRO-AID from Drools 5 to Drools 6+ is not a simple task. It requires rewriting the PRO-AID-Rule Engine interface code and performing the regression tests. This task is planned for the next development cycle.

5.3 Utility Program for Generating the PID JASON File

The utility program used to generate the PID JSON File is written in Python 2.7. Python 2.7 is the last 2.x release. The End Of Life (EOL) date for Python 2.7 is 2020. Before 2020, this utility program needs to be migrated to Python 3. Note that Python 3 is not compatible with Python 2. Alternatively, In Bridge Program, a Java module can be written to provide the same functionality as that of the Python utility program.

6. Conclusion

Bridge Program has been successfully modified for the NPP simulator. Verification and validation of the program were demonstrated by performing real time fault detection tests on the INL HSSL. Good and accurate results are observed from the tests.

7. References

1. Western Services Corporation, “Interfacing with 3KEYMASTER v.3.5-1, User Guide”, August 2010.
2. A. Grelle, S. Passerini, and R. Vilim, Argonne National Laboratory, unpublished information, 2016.
3. A. Grelle, Y. Park, and R. Vilim, Argonne National Laboratory, unpublished information, 2014.
4. A. Grelle, “Transition Materials for PRO-AID & Bridge Program”, Argonne National Laboratory, unpublished information, 2016.
5. A. Grelle, Y. Park, G. Jeon, and R. Vilim, Argonne National Laboratory, unpublished information, 2015.
6. T. A. Ulrich, R. Lew, R. L. Boring, K. D. T., Brandon C. Rice, and C. M. Poresky, “Operator-in-the-Loop Study for a Computerized Operator Support System (COSS) – Cross-System and System-Independent Evaluations,” INL/EXT-17-43390, September 2017.
7. Ken Thomas, personal communication, Idaho National Laboratory, September 2017.

Appendix A: Setup Bridge Program Development Environment

[Data withheld – Please contact the authors.]

Appendix B: Schema of PID JSON File in Java

[Data withheld – Please contact the authors.]

Appendix C: Input File for Generating the PID JSON File

[Data withheld – Please contact the authors.]

Appendix D: Variable File for NPP COSS Simulator

[Data withheld – Please contact the authors.]

Nuclear Engineering Division

Argonne National Laboratory 9700
South Cass Avenue,
Argonne, IL 60439

www.anl.gov

Argonne National Laboratory is a U.S. Department of Energy
laboratory managed by UChicago Argonne, LLC