New PRODIAG Algorithm and Acceptance Test

Young Soo PARK[1], and Richard VILIM

1. Nuclear Engineering Division, Argonne National Laboratory, Lemont, IL, USA (ypark@anl.gov)

Abstract: This paper presents the improvement of an automated-reasoning computer program for nuclear power plant diagnosis, namely PRODIAG. PRODIAG, first developed at Argonne National Laboratory, is a physics-based fault diagnosis system which is plant configuration independent. To further enhance the code extensibility and maintenance, code upgrade has been made incorporating a modern object-oriented programming language, and an open-source automated reasoning engine. Verification test was also performed on a series of simulated faulty power plant operation data.

Keyword: Plant fault diagnosis, Expert system, Artificial intelligence, Object oriented program

1 Introduction

PRODIAG a computer-based diagnostic system, first developed at Argonne National Laboratory, for detection and identification of component faults in in the T-H processes in nuclear power plant system [1, 2]. PRODIAG adopts confluence-based reasoning to perform fault diagnosis in engineering systems that can be described by the conservation laws for mass, energy, momentum, and chemical species [3, 4]. Therefore, unlike other plant configuration-dependent rule-based diagnosis algorithms, the code is generally applicable to a wide variety of plant configurations by simply inputting the plant instrumentation diagram (PID) database.

However, the original PRODIAG code implementation in Prolog reflects the state of expert systems development in the mid 1990's, and the need for improvement was raised when it failed to provide the requisite degree of extensibility and maintainability for new applications. Therefore, a developmental work has been conducted to re-design and re-write the PRODIAG code consistent with current-day best-practices for software implementation of automated-reasoning algorithms. In particular, the reasoning process is inextricably wedded to the execution of rules as opposed to the recent trend where the rules are defined separately and then presented for evaluation by a reasoning engine. These shortcomings have been addressed through a new code architecture that uses an objected-oriented logic-programming language in place of the original Prolog language and uses an open-source automated reasoning engine in place of the original hardwired rule-evaluation sequence. The latter was realized by separating the rule base from the rule-evaluation sequence. The new code was developed using an object oriented language, Java, and utilized an open-source production engine, JBoss Drool.

This paper presents the architecture to implementation details of the code transition. Verification and validation studies were performed for the new PRODIAG.

2 Overview of the Original PRODIAG

The methods of PRODIAG are described in [1, 2]. Unlike the traditional event-oriented approach for diagnostics, PRODIAG used function-oriented approach that does not require pre-specification of the possible process component faults. The diagnostic strategy is carried out by separating the relevant information into three distinct but interacting knowledgebase: physical rules database (PRD), component classification dictionary (CCD), piping and instrumentation database (PIDB). Process symptoms are related to component faults through the three-step mapping illustrated in Fig. 1. Through this three-step mapping, the diagnosis is transformed into a function-oriented approach and confines the process-dependent information solely to the PIDB.

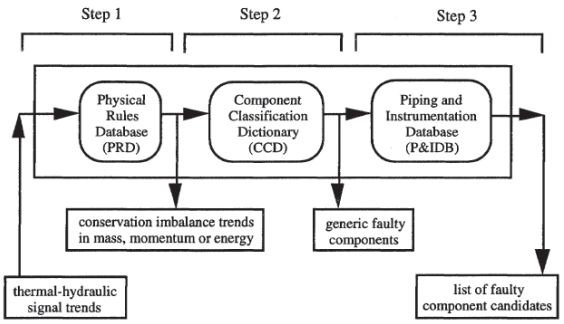

Fig.1. PRODIAG Diagnostic Process

Physical Rule Database (PRD)

PRODIAG' s physical rules are derived based on qualitative physics [3, 4] where a small number of qualitative trends, such as increasing, decreasing, and unchanging trends, are used to represent the values of continuous real-valued process variables applicable to process-independent basic elements, i.e. loops, control volumes, and components. First principle rules, mapping trends in T-H signals into imbalances of mass, momentum, and energy, were solely derived through the equations of state and the definitions of momentum, and applied to the relevant control volumes.

The qualitative physics rules of the PRD are of two classes: Q rules and CV rules. A Q rule indicates the type and trend of the imbalance in a control volume inferred from the trends in the T-H signals. Corresponding to the three balance equations of mass, momentum, and energy, there are there types of Q rules: Q_{mass}, Q_{mom}, Q_{eng}. The Q status can have one of three trends: increasing (↑), decreasing (↓), and unchanging (−). A CV rule infers the trend status of unmeasured T-H signals in a process component from the other T-H signals of the component and the Q status of the component.

As an illustration, consider the physics rule,

$$Q_{mass} = W_{out} - W_{in} \qquad (1)$$

where W_{in} and W_{out} are the control volume inlet and outlet mass flow rates, respectively, and Q_{mass} is the mass source/sink term in the mass balance. Transforming (1) into qualitative differential expressions, using De Kleer and Brown's methodology and notation [4], gives the following confluence:

$$[dW_{in}] - [dW_{out}] = -[dQ_{mass}] \qquad (2)$$

where the square brackets $[\cdot]$ represent the qualitative value or trend (↑, ↓, −) of the argument basic quantity. Equation (2) represents the general confluence, from which Q rules characterizing imbalances in Q_{mass} can be derived by applying the different trend combinations of W_{in} and W_{out} and using the operations of qualitative algebra. Thus,

$$\text{Rule (A)} \quad \text{If } W_{in}^{\uparrow} \text{ and } W_{out}^{\downarrow}, \text{ then } Q_{mass}^{\downarrow} \qquad (3)$$

$$\text{Rule (B)} \quad \text{If } W_{in}^{\downarrow} \text{ and } W_{out}^{\uparrow}, \text{ then } Q_{mass}^{\uparrow} \qquad (4)$$

Likewise, other Q rules are derived for defining momentum and energy with related process variables. PRODIAG includes a comprehensive set of rule bases modules, which include various physical rules.

Component Classification Dictionary (PIDB)

The component classification dictionary (CCD) provides generic classification of various process components. As such, the component faults detected as Q imbalances by the PRD can be linked with the responsible components. A small number of generic component types included in T-H processes are classified into predefined classes. For example, based on the three conservation equations, each generic component type, e.g. closed valve, open valve, and electrical heater, is functionally classified as a source or sink of mass, momentum, or energy.

Piping and Instrumentation Database (CCD)

To facilitate systematic application of physics rules for component level diagnostics in process independent manner, the T-H process is decomposed into basic elements or building blocks to form generic geometrical configurations to which the rules of the PRD can be applied. A top-down tree like decomposition definition is established, accordingly a T-H system is decomposed into loops defined between two end boundaries, and components (including junctions) which belong to each loops. The plant specific system configuration is described in the P&ID database, which includes the loop-specific information, component-specific, and inter-loop information.

Limitations of the Original PRODIAG

Despite the principal benefits, limitation was experienced in adapting the original PRODIAG to new diagnosis applications different from the original application. Despite the main benefits, the actual implementation realized in Prolog presented limitations in extensibility and maintainability when adapting to new applications. Mainly, this was a consequence of 1) failure to program using objected oriented techniques (the Prolog language does not support object-oriented programming, and 2) the decision to imbed the confluence-derived rules in the diagnostics search sequence, as opposed to keeping the two separate.

3 NEW PRODIAG ARCHITECTURE

To address the above mentioned shortcomings and support the re-write of the PRODIAG, various software development tools were evaluated. Various aspects were considered in support of 1) an inference capability, 2) an object-oriented capability, 3) a debugger, 4) a semi-natural English-type language, and 5) an integrated development environment (IDE). The tools examined were the Max-SAT solver [5], the VisiRule package [6], PyKE [7], and Drools (JBoss) [8]. The final choice was Drools JBoss, an open source tool with a Java-based rule engine. It supports forward chaining rules via the Rete algorithm and also partially supports backward chaining rules. Since the PRODIAG rules within the code can be formulated in forward-chaining reasoning form, the Drools JBoss support for production rule (if-then) was found to be adequate. Another significant merit is the capability for object-oriented code and the existence of a debugger in an integrated development environment. The language is Java-based and consequently benefits from a large Java-user community.

The overall software architecture of the program is illustrated in Fig.2, which consists of four parts: 1) PRODIAG Expert system, 2) PID Construction Program, 3) Read Data Interface Program, and 4) Graphic User Interface.

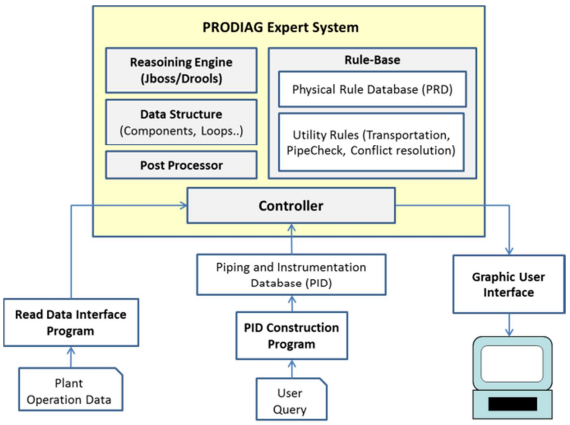

Figure 2. Software Architecture of New PRODIAG

Data Structure

The new PRODIAG uses the following data structures to keep track of static information (e.g. PID) and process variables.

- *Component*: A Component object keeps track of the component properties (e.g. component type, volume type, relative pressure, fluid type, location). The following types are supported: bearing, closed_valve, demineralizer, filter, heat_exchanger_cold, heat_exchanger_hot, heater, junction, open_valve, pump, seal, seal_barrier, sump, system, tank, and turbine.
- *Loop*: A Loop object keeps track of the properties of the loop and the components in the loop. It also keeps track of intermediate diagnosis results, such as faulty component candidates, for later use in the final diagnosis in the post processing.
- *THProcess*: This is the one instance for the whole life-cycle of the expert system. It keeps track of all the Loop, Component and Q instances. It also holds a handle to the rule-base engine.
- *Sensor*: A Sensor object stores the initial value of a process variable, and keeps track of its raw trends during the following updates.
- *Q*: Q objects are imbalance indicators that keep track of the inference output of various Q-rules in the rule base. There are two subclass, Q_{comp} and Q_{loop}, depends on the Q is associated with Component or Loop.

Rule-base

A major objective of the PRODIAG redesign activity was to modularize the program so that it could be easily modified or extended. Currently there are 21 different modules organized as agenda groups. These modules have no direct interactions with each other, and the collaborations are achieved mostly through updating the inserted Java objects: Component, Loop and Q.

Transportation: The transportation module replicates the enthalpy transportation and constant flow transportation procedure to fill the unfilled process variables, esp. temperature and constant flow. The transportation happens along any given loop that meets the conditions. Transportation, as in original PRODIAG system, is executed right before the primary Q-rules.

Module 00.00: Module 00.00 is designed to detect certain instrument errors. The design of the rules is straightforward: identifying a component with measurements that meet the rule condition, then report the error and stop the diagnosis.

Modules 10.0, 20.00, 30.00, 40.00, 50.00, 90.00, 100.00: defines the set of Q-rules of various categories.

Module 60.00, 70.00, 80.00, 100.00, 120.00: are the set of CV-rules designed to infer unmeasured variable trends for based on other measurements and Q trends.

Pipe Check: Pipe check is yet another undocumented module in the original PRODIAG program. This module is used to check the failure of a single pipe based on the immediate surrounding flow or pressure trends and their environment pressure attributes (relative pressure).

Conflict Resolution: This module includes all conflict resolution routines for conflicting Q assignments, so that multiple rules with action items on Q will not end up with any potential conflicting assignments.

Q Propagation: This auto-focus group has only one rule, which will propagate the component Q trends into the loop Q. Every time a component Q trend is set with conflicts resolved, this group will be pushed on top of the stack and propagate the component Q trend to loop Q.

Cleanup and Generic: This module contain some general rules for the purpose of housekeeping, to make sure rules in other modules can be executed as expected. This group is also an auto-focus group, so that can be executed as needed.

Automated Reasoning Process

The core of new PRODIAG is a business rule management system Drools, which provides a powerful reasoning engine for performing diagnosis. Java objects can be inserted into the rule base as facts, and therefore become accessible by the reasoning engine. A registered rule in the rule base consists of a set of conditions and a set of associated actions. Once the conditions of the rule are met, the actions will be executed. The rule base consists hundreds of rules in the form of standard Drool rules base. Each rule work independently, and the collaboration between rules are achieved through delicately conceived condition and action items. The effects of the rules are reflected in the modifications they made to the Java objects.

The diagnostic procedure works as follows: a new rule engine session is created, and data structures, such as *Components*, *Loops*, *Qs*, are initialized and registered as facts in the engine. A set of rules is also registered with a rule flow specification. During each iteration, a new batch of sensor reading is updated through TimeWindow selector, and rules are fired to perform diagnosis. The iteration continues until a conclusion is reached or the input data is exhausted. After each iteration, a post processing procedure is invoked to collect the inference results and present the final results. In general, there is no direct way to control the order of rule execution once we fire up the rule engine. However, through the *ruleflow* facility of Drools, we can group rules into group and control the overall flow. Using the "*ruleflow*" keyword, as in the example above, we categorized the rules based on the module they serve in, and use a flowchart tool to specify the orders of the group.

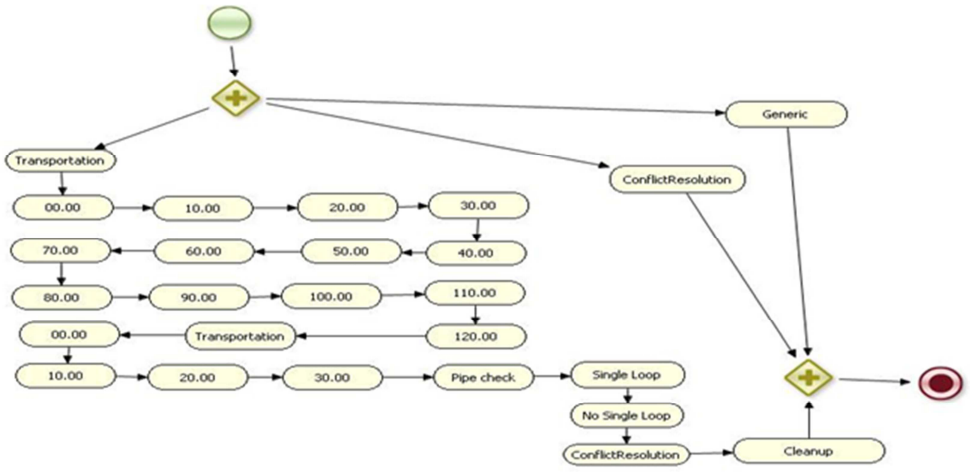

Fig.3. Schematic of Diagnosis Procedure

Fig. 3. shows the rule flow for PRODIAG. It has three branches, the left branch replicates the general order of original PRODIAG work flow: Transportation -> Q-Rules -> CV-Rules -> Transportation -> Q-Rules -> Diagnosis; the two other branches will be executed in parallel with the main branch and taking care of some routine updates and conflict resolution. The detail of the *ruleflow* groups can be found in [8]. Notice that the *ruleflow* is different from program flowchart. It only governs which set of rules should be activated after the rule groups before it has ceased fire, but it has no control over when an activated rule should be fired.

Post-Processing

After the rules cease firing, the reported faulty candidates are gathered and post processed in "ResultPool" (see ResultPool class). Because PRODIAG is a loop-based inference system, most faulty candidate information is stored in the class Loop. In general, there are nine categories of possible faults.

4 Acceptance Test

For verification and validation purpose, the new PRODIAG was tested with a test data generated from a power plant simulator. The diagnostic results were compared with those of old PRODIAG. The Chemical and Volume Control System (CVCS) of ComEd's Braidwood pressurized water reactor was selected as the test bed. Figure 4 shows the PID of the Braidwood CVCS system under test. The test data was generated to represent transient conditions for a range of postulated failure conditions.

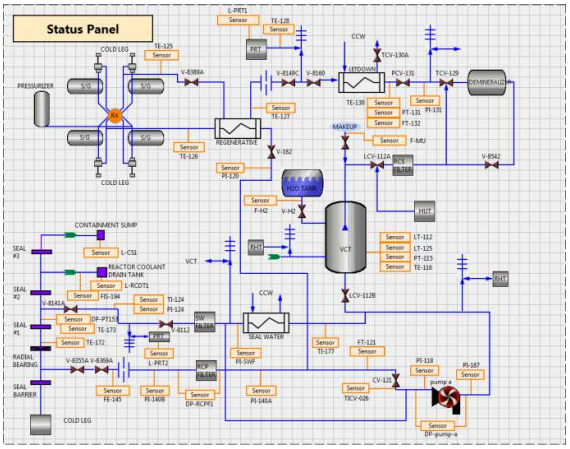

Fig.4. Plant Status Panel of the Braidwood NPS CVCS

In order to give users a realistic interface for plant status monitoring and control, a NPP operation training simulator is implemented which includes graphical user interface, plant control and protection system, and plant simulator. The control system is developed with EPICS, with graphic user interface implemented with Control System Studio [9]. As illustrated in Fig. 5, a Bridge program was developed to provide integration of the simulator and PRODIAG diagnosis system.

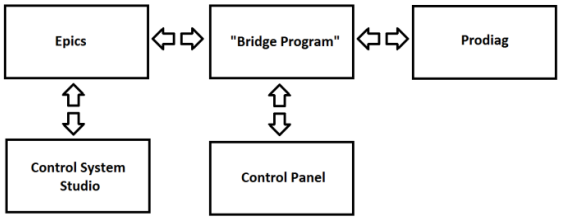

Fig.5. Overview of the control panel interfaces with PRODIAG and plant simulator

The test results are summarized in comparison with those of old PRODIAG diagnostic results on the same test cases. Total 20 failure cases were tested and the results are shown in Table 1. The diagnostic results are classified into two categories: component failures and piping failures (clogging or break). In the old PRODIAG results, the piping failure results are displayed and one sentence including the most outer boundaries. On the other hand, in the new diagnostic results, each piping segment failure is indicated. When combined which makes up the old diagnostic results. The diagnostic results matched most cases for the component failures, except CV09-150 case which was sensor failure case. The piping failure diagnostic results matched in most cases, but with different levels of accuracies in terms of pin-pointing the faulty segments. Table 1 shows a summary of comparison of diagnostics results.

4 Discussion

PRODIAG is a function-based approach for nuclear power plant diagnostics. Compared to the traditional process-based diagnostic systems, it has advantage in terms of portability and capability to diagnose unanticipated faults. However, due to the dated software structure, it had difficulty in software expansion for new application. To this end, a software update was performed to transform PRDIAG to modern day software practice. The objective is to adopt object oriented software structure, and separation of rules from the inference mechanism. The feasibility of the software is validated with simulation plant operation data. Despite the initial success, further validation of the code for robustness, and improvement toward model-based diagnostics is anticipated.

Acknowledgment

This work was supported by the U.S. Department of Energy, Office of Nuclear Energy, under Contract No. DE-AC02-06CH11357.

Table 1 Summary of Diagnostic Results Comparison

Diagnosis	Match	Predict Better	Predict Worse
Component Failure	CV04-100, CV05-310, CV06-100, CV07-100, CV08-480[a], CV10-100, CV12-65, CV14-65, CV16-95, CV18-70, CV21-50, CV24-65, CV27-150	CV01-100	CV09-150[b]
Piping Failure	CV04-100, CV24-65, CV25-45	CV06-100, CV07-100, CV12-65	CV01-100[c], CV05-310[c], CV13-45[c], CV22-65[c], CV23-65[c], CV26-20[c]

[a] This fault was a sensor failure case. The diagnosis is not correct in this case.

[b] This is a sensor failure case. The diagnosis is indeterminate.

[c] Piping failure pipe-run localization correct, but piping run is longer than diagnosed by original PRODIAG.

References

[1] J. REIFMAN and T.Y.C. WEI, "PRODIAG" A Process-Independent Transient Diagnostic System – I: Theoretical Concepts," Nuclear Science and Engineering, 131, 329-347, 1999.

[2] J. REIFMAN and T.Y.C. WEI, "PRODIAG" A Process-Independent Transient Diagnostic System – II: Validation Test," Nuclear Science and Engineering, 131, 348-369, 1999.

[3] J. DE KLEER and J. KURIEN, "Fundamentals of Model-Based Diagnosis," Proceedings of SafeProcess03, 2003.

[4] J. DE KLEER and J. BROWN, "A Qualitative Physics Based on Confluences," Artificial Intelligence, 24, Issue 1-3, Dec, 1984.

[5] M. CODISH, "Logic Programming with Satisfiability," Theory and Practice of Logic Programming, Vol. 1, Issue 8, 121-128, January 2008.

[6] R. KOWALSKI, VisiRule, Logic Programming Associates, London, England, 2012

[7] B. FREDERIKSEN, "Applying Expert System Technology to Code Reuse with Pyke, PyCon 2008, Chicago, IL, March 2008.

[8] P. BROWNE, JBoss Drools Business Rules, PACKT Publishing, March 2009. Experimental Physics and Industrial Control System [Web log] Retrieved Dec 29, 2014, from http://www.aps.anl.gov/epics/

[9] Control System Studio, [Web log], Retrieved Dec. 29, 2104, from http://ics-web.sns.ornl.gov/css/