A GENERIC FAULT PROPAGATION MODELING APPROACH TO ON-LINE DIAGNOSIS AND EVENT CORRELATION

Gregory M. Stanley* and Ramesh Vaidhyanathan
Gensym Corporation, 1776 Yorktown, Suite 700, Houston, TX, 77056 (USA)

Abstract: CDG (Causal Directed Graph) provides a methodology and framework for real-time fault management in large-scale systems, addressing the full life cycle of problem identification based on symptoms, diagnostic testing, and fault isolation, through recovery, as well as protecting the operator from "alarm flooding". It is based on generic fault propagation models, tied to an object-oriented domain representation and scalable algorithms. CDG combines the generality of FMEA models with on-line, asynchronous event correlation and diagnosis. The architecture of CDG is described and the modeling approach is discussed with examples. Event correlation and interactive diagnosis using CDG is illustrated through a nitric acid cooling system example.

Keywords: diagnosis, fault propagation model, digraph, knowledge engineering, automation, real-time systems, failure and recovery, FMEA, test planning

1. Introduction and Motivation

Operators of large systems are constantly bombarded by events in the form of messages and alarms, often resulting from a small number of root cause problems. Operators cannot adequately analyze all these events in real-time. Once events are detected, or arrive from an external system, operators need a system to:

(1) Filter out redundant events so attention can be focused on truly new information.
(2) Correlate events - recognize and summarize or group related messages for presentation to the operator, even if the root cause is not determined.
(3) Diagnose the root causes of problems, based on incoming symptoms and test results, automatically choosing tests to be run proactively in cost order.
(4) Run automated or guided manual tests as soon as there is evidence of a problem.
(5) Execute automated or guided manual corrective actions to address symptoms or fix problems.
(6) Communicate results to users and to other computer applications.

It is evident that fault management goals for the operations environment include more than just detection and isolation. These goals are partly addressed by alarm management, event correlation, or abnormal situation management applications. However, diagnostic testing and taking corrective actions in those applications are usually ad hoc.

In addition, application developers need to provide these capabilities quickly and reliably, and build re-usable applications. For practical deployment, installation must require minimal on-site manual customization. Automatic reconfiguration is also needed, to account for changes in equipment, topology, or operating mode.

CDG (Causal Directed Graph) provides a methodology and framework for real-time fault management in large-scale systems. CDG addresses these goals with a model-based framework to standardize development using generic (class-based) fault propagation models. CDG automatically looks at secondary data or runs more elaborate tests only when problems are suspected based on symptoms. CDG manages operations in domains as diverse as process plants, communications networks, and enterprise-wide software applications.

2. Basic Modeling Elements

A fault propagation model in CDG describes the propagation of failures via potential failure paths in the system by modeling how fault events will cause other symptom events and test-result events. It is a directed graph (digraph) model, with the nodes representing fault, symptom and test events, and the arcs connecting the nodes representing the "cause and effect" relationships or dependencies among these events. An arc from an event node "A" to event node "B" means, "if event A occurs then event B will occur (after the time delay specified in the arc)" or "event A causes event B (after the time delay specified in the arc)". The events represented in the model can take the values, "true", "false" or "unknown".

A "Fault" is an underlying independent root cause problem. Faults have associated corrective or mitigation actions specified by procedures that can be started against a specific domain object. Actions can be defined to execute when a fault is suspected as a cause of the observed symptoms and when a fault is confirmed. For a fault event, a true value indicates belief that the fault has occurred and a false value indicates belief that the fault did not occur. A fault may be "suspect" if it is a possible cause of observed symptoms.

A “Symptom” is an effect of underlying faults in the monitored system. A measured symptom arrives at CDG, unsolicited asynchronously from external systems. An unmeasured symptom status is used for failed sensors, and convenience in modeling, to represent those effects that are not measurable, but are important for fault propagation. For a symptom event, a true value means that the symptom has occurred and a false value means that the symptom did not occur within the time delays specified in the arcs.

A “Test” is an observable effect of faults in the monitored system, like a measured symptom. But, unlike symptoms, the observation can be requested at any time. Test results arrive asynchronously (and possibly unsolicited), just like a symptom. Tests have an associated set of actions applied to the monitored system, specified as an external procedure name that can be started against a specific domain object. Upon request, after some indeterminate amount of time, the result of the test is returned to CDG as a truth-value. Test procedures may be fully automated, may simply be a request to an operator, or a combination of the two. The true or false value for a test indicates whether the test passes or fails, respectively. The notion of "test" is powerful and general, (Simpson and Sheppard, 1994) and can imbed arbitrarily complex analysis and actions, as long as it returns a single truth-value.

3. Limitations of Other Diagnostic Techniques

Passive Symptom Monitoring

While desirable, passive symptom monitoring is not always good enough for fault isolation. Human troubleshooters know better, performing lab tests, visual checks, or device built-in-tests. They change controller tuning, put control loops in manual mode, or run a step test. The distinction between tests and symptoms can be used to improve scalability. In large systems, it is impractical to monitor every variable regularly and often. Instead key variables are monitored often, generating symptoms. Once an initial symptom indicates a problem, additional variables can be examined as tests to complete diagnosis. Unless tests are represented in a model linking them to faults, a diagnostic system can’t schedule tests, so tests are ad-hoc. In addition, tests need to be ranked based on cost factors such as disruption of the process, resource cost, or automated vs. manual testing.

Static Pattern Matching and Compiled Models

Static pattern matching (classification) includes neural net classifiers, Case Based Reasoning (CBR), and simple rule-based systems. During application development, each failure is hypothesized, and all expected symptom values are determined - a failure "signature". At run time, the pattern matcher determines the fault(s) with the signature closest to the observed symptom vector. This captures model knowledge, but is not robust enough to account for major dynamic changes in the monitored system, such as switch positions, controller modes, etc.

Unfortunately, large systems change often and developing separate pattern matchers for each possible combination of operating modes or topology wouldn't scale up. A model-based approach is required. However, if the models are compiled for use by pattern matchers, recompilation will still be needed with each change. This introduces delays, and interrupts ongoing diagnosis.

Systems requiring training from live data (e.g. neural nets or CBR) have additional problems in handling large systems, including generalization of results from failures involving symptoms in multiple objects, learning rarely-occurring faults, and deciding when old patterns are no longer relevant.

Time Window Problem

Static pattern matchers require periodic processing of a "snapshot" of data or events captured within a time window. Calculations are started fresh with each time window - results of previous analyses are not considered. Time delays between fault occurrence and symptom arrival cause problems. With only partial symptoms present, diagnostic conclusions can be wrong. Tuning the time window size to balance misdiagnosis vs. timely results is difficult. CDG processes individual events immediately as they arrive asynchronously, and event values are remembered until overwritten or timed out, without the need for a time window.

Reasoning with Missing Symptoms

While using pattern-matching techniques, the absence of a symptom event may be taken as evidence that the underlying symptom really isn't present (value of false), hence associated possible faults aren't present. That can lead to incorrect diagnosis if there are time delays between the fault and the symptom, or communications problems. Evidence based on missing symptoms should not be used until after the worst-case time delay. Additionally, when large delays are present, it is desirable to offer preliminary diagnosis before all symptoms have arrived. In that case, the only values that should be used are those for symptoms that have already arrived.

Single Failure Assumption

Many techniques assume there is a single failure. This is unrealistic in any large system where multiple faults have already occurred and previous faults continue to cause symptoms that may overlap with new problems. Static pattern matchers will fail to handle these situations, since a combination of faults leads to a pattern of symptoms far from any single fault’s failure signature. Encoding all 2- or 3-failure combinations does not scale up. Instead, users must partition the system into many small diagnostic subsystems, so that only one failure is likely in a given subsystem.

Other Modeling Approaches

Process models are often based on algebraic or differential equations, or qualitative versions such as Signed Directed Graph (SDG) that represent the propagation of process variable deviations. These models do not represent test procedures, relay or PLC logic, frequency domain information, discrete operational modes (controller modes, switch positions), or messages (events) from complex software in alarm systems, shutdown systems, optimizers, or intelligent instruments. "Smart sensor error code 30", "backup power starting", or "no response" messages don't fit the paradigm. CDG shares with SDG an orientation towards asynchronous event processing and propagation of information, instead of pattern matching. Like CDG, many SDG implementations have used generic, class-level models for automating diagnosis (Finch, et al., 1990), and HAZOP analysis (Vaidhyanathan and Venkatasubramanian, 1995).

Bayesian Network (BN) is a powerful modeling technique that can represent generic fault propagation knowledge. With the additional probability information, BN might perform better in cases when there are few symptoms or tests, and where tests are expensive. But BN introduces unscalable computing complexity and is not needed in data-rich environments with inexpensive tests.

Design models for "normal" operation are often used in diagnostic systems. However, faults invalidate design model assumptions. Pattern matches on observed deviations from "normal" can be used for diagnosis, but suffer the problems noted earlier for pattern matching and compiled models. CDG can accommodate such quantitative "normal" models in the definition of the tests and symptoms.

The fault propagation models used in CDG are similar to the information flow models used for model-based design for testability and integrated diagnosis (Simpson and Sheppard, 1994). CDG extends these models for on-line event correlation, interactive diagnosis and fault mitigation. Major extensions were a strong event orientation to handle processing of asynchronous events, handling symptoms as distinct from tests, run time selection of tests dependent on previous results (instead of offline generation of one fixed fault tree for all time), specification of generic models with specific model instantiation at run time, graphical input of models and eliminating the poorly-scaling matrix calculations.

The extended real-time Failure Environment Analysis Tool (FEAT) (Malin, et al., 1992) used "failure state information flow" models that are similar to the fault propagation models used in CDG. But FEAT was limited due to passive monitoring (no tests), lack of generic modeling, the use of compiled models and a complex matrix analysis to identify single and double faults.

Scalability

Scalability (ability to scale a system up to a large number of objects) is a central issue in fault management in large-scale systems. CDG addresses this issue through linear algorithmic complexity, "management by exception" to only instantiate a specific, localized model at run time when initial symptoms indicate a possible problem, models at a high level of abstraction, and an architecture supporting distribution over multiple computers.

4. Architecture

The architecture of CDG is presented in figure 1. The input specifications required for CDG are the Generic Fault Propagation Models and the Specific Domain Model. Using these specifications, CDG correlates incoming symptom and test result events, identifies suspect faults and performs diagnosis by selecting appropriate test and mitigation actions to resolve the suspect faults. The outputs from CDG are the diagnostic conclusions and test and action requests. The various components of CDG are described in detail in the following sections:

Generic Fault Propagation Models (GFPM)

GFPMs consist of the generic cause-effect relations among the fault, symptom and test events in the domain objects. The GFPM of a domain-object class defines the propagation of failures within its instances and to other classes of domain objects via relationships. This is a generic (class-level) "library" of models, independent of any specific topology of domain objects and relationships present in any particular system. The GFPMs can be developed from first-principles models, experience-based knowledge, or FMEA results. GFPMs are specified via the developer GUI graphically.

The event nodes and the arcs in the GFPM can be made conditionally dependent on the states of the domain objects by defining appropriate conditional methods on the nodes and the arcs. These conditional methods are evaluated during event correlation and only the nodes and arcs that are valid or available for the current states of the domain objects are used for correlation. In addition, state dependent methods are defined for the event nodes to calculate the cost of running a test, the cost of fixing a fault and the cost of not fixing a fault. The costs evaluated by these methods are used to rank the candidate tests for diagnosis and the fault mitigation actions.

Specific Domain Model (SDM)

The SDM is an object-oriented model of the specific system being monitored. It might model physical equipment, as well as more abstract entities affecting system performance, such as controllers or software applications. The SDM includes objects representing the monitored domain entities, their connectivity, containment and other relationships. The SDM can be imported from external databases or files or can be specified via the developer GUI.

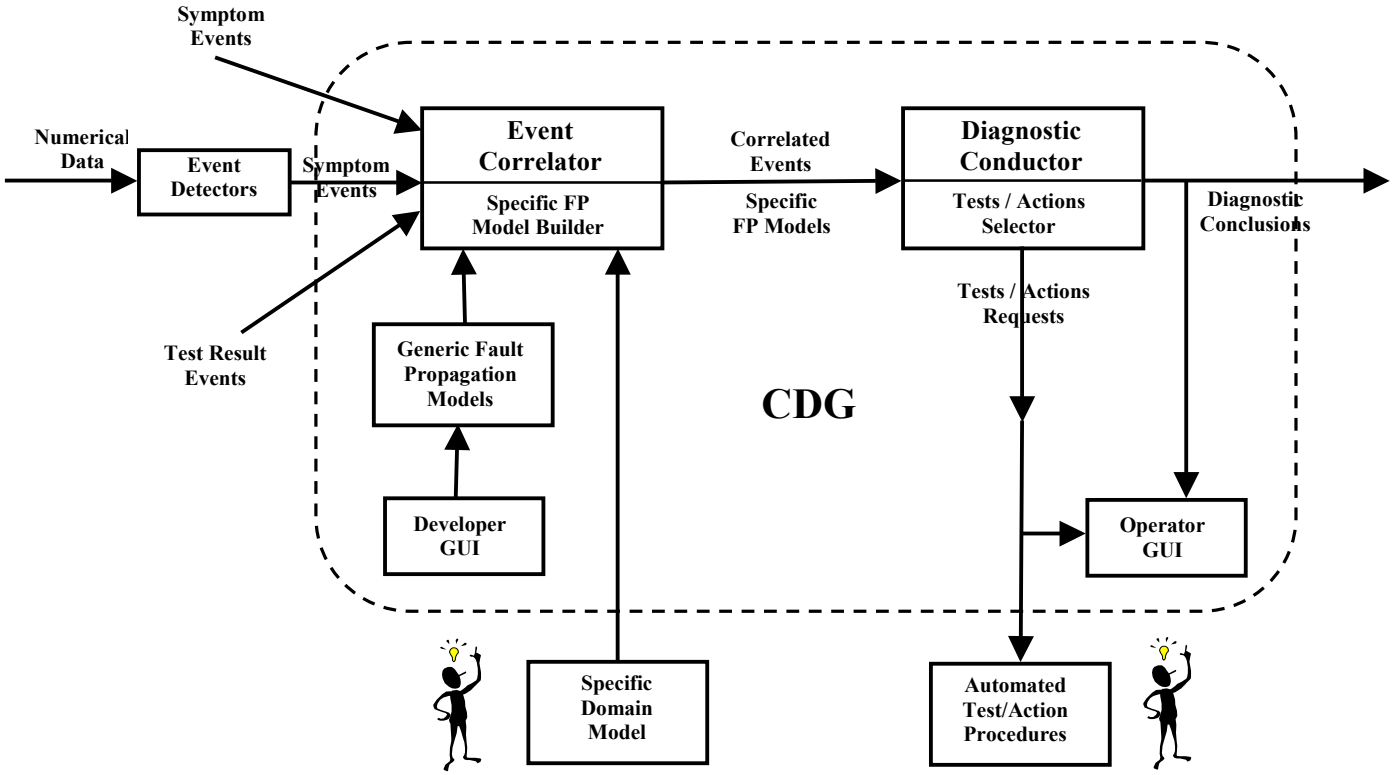

Figure 1. The Architecture of CDG

Event Correlator

The Event Correlator correlates asynchronous symptom and test result events input to CDG using Specific Fault Propagation Models (SFPMs). SFPM is a fault propagation model that describes the propagation of fault, symptom and test events within and across specific domain objects. The SFPM Builder constructs SFPMs at run time starting from the incoming events by appropriately combining the GFPMs and the SDM, just building enough event nodes to account for possible causes and effects of observed symptoms. "Event Detectors" are external applications that monitor and analyze the numerical data trend in the system, detect and generate appropriate symptom events to be input to CDG.

The event correlator recognizes that a group of events are related to each other based on their connectivity criteria in the SFPM such as the existence of a directed path or the fact that the events could be caused by common faults. Then the value of the incoming event is propagated in the SFPM to infer and predict the values of other events and to identify suspect faults. OR logic is used by default for the propagation of event values during correlation. Thus, "the value of a node is true if the value of one (or more) of its inputs is true". Conversely, "if the value of a node is false, then the value of all of its inputs ought to be false". Also using the OR logic, "if the value of an event is true, then all the upstream fault events become suspects". Hence, the event correlator identifies the suspect faults by searching upstream from the incoming symptom and test result events with a true value in the SFPM. This information is input to the Diagnostic Conductor.

During event correlation, if the value of a symptom event is predicted true by propagation and if the actual symptom with a true value does not show up within the time delays specified in the arcs, then the value of the symptom will be reset to false and re-propagated. Thus, the interim event correlation and diagnosis will be consistent with observed events at any time and the final correlation accounts for symptoms that did not occur. The default time delay is "infinite", so by default missing symptoms are not used as evidence. In addition to the OR logic, AND and NOT logic propagation relationships among the events could also be specified in the SFPM.

Diagnostic Conductor

The Diagnostic Conductor resolves the suspected faults by identifying appropriate candidate tests that when executed would provide additional information regarding those faults. These candidate tests are those which are the effects of the suspect faults. The Test Selector identifies these tests by searching downstream from the suspect faults in the SFPM. In the case of an automated test, a request is sent from CDG to execute the automated test procedure. Otherwise, the test is displayed on the operator GUI for approval before execution. The candidate tests are ranked based on cost criteria such as resource use, disruptiveness,

or the information value of a test. The test results are asynchronously input back to the event correlator for further correlation to reduce the number of suspect faults.

Based on the correlation of the test results, the suspected faults are either ruled out or concluded to have occurred. Diagnostic conclusions are output from CDG to the operator and to other external systems. Whenever CDG concludes a fault as a suspect or occurred, the Actions Selector will execute appropriate mitigation actions specified for the fault. Similar to the test procedures, these mitigation actions can also be automated procedures or may require operator intervention. These mitigation actions can also be ranked based on criteria such as the failure-rate, the cost of fixing, or the cost of not fixing the faults. The test and mitigation procedures can be defined using OPAC (OPerations expert ACtions) graphical procedure development tool or G2, external to CDG.

5. GFPM of the Controller and Sensor

A part of the GFPM of the controller and sensor are shown in figures 2 and 3, respectively. Each event in the GFPMs has a category attribute that specifies the name of the event and a class attribute that specifies the class of the domain object for which the event is defined. These attribute values are displayed at the side of the event nodes. The square icon nodes represent faults and the circular icon nodes represent symptoms and tests. The smaller circular icon nodes represent symptom-views and test-views, which are the views of symptoms and tests defined for another class of domain object. A propagation-relation that specifies the relation between the domain objects of the event-view node and the event node is defined on the arc connecting these nodes.

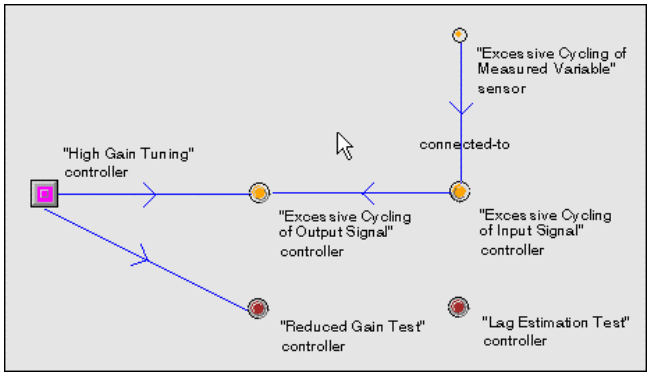

Figure 2. GFPM of the Controller

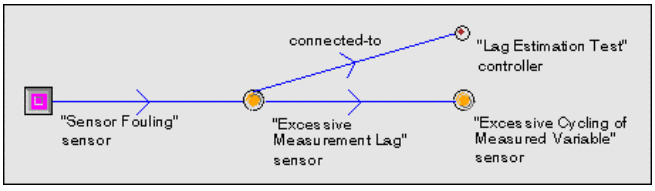

Figure 3. GFPM of the Sensor

In the GFPM of the controller, the fault "High Gain Tuning" represents the controller gain being set too high. This fault could cause the measured symptom "Excessive Cycling of Output Signal" from the controller. In addition, the measured symptom, "Excessive Cycling of Measured Variable" in a sensor connected to the controller could cause the symptom "Excessive Cycling of Input Signal" to the controller, which in turn could cause the measured symptom "Excessive Cycling of Output Signal" from the controller, due to control action.

The "Reduced Gain Test" in the controller is a candidate test for isolating the fault "High Gain Tuning" in the controller. This test involves a series of actions wherein the gain of the controller is reduced and the measured variable is monitored to determine if the cycling of the measured variable dissipates. A true or false result for this test will respectively confirm or rule out the "High Gain Tuning" in the controller as the fault that might have occurred. Mitigation actions such as "lowering the gain of the controller setting" or "switching the controller to manual mode" are defined for this fault.

In the GFPM of the sensor, the fault "Sensor Fouling" represents the accumulation of material around the thermowell of the sensor. This fault could cause the unmeasured symptom "Excessive Measurement Lag" in the sensor, which in turn could cause the measured symptom "Excessive Cycling of Measured Variable" in the sensor. The "Lag Estimation Test" in a controller connected to the sensor is a candidate test for isolating this fault. This test involves a series of actions wherein the controller is switched to manual mode from automatic mode, and then a step response test is performed by changing the manipulated variable and estimating the time-constant and the lag time by monitoring the effect on the measured variable. If the time-constant and the lag time of the sensor measurement are significantly higher than their normal values, the result of this test is true, else false. A true or false result for this test will respectively confirm or rule out the "Sensor Fouling" in the sensor as the fault that might have occurred. Mitigation actions such as "cleaning the thermowell of the sensor" or "replacing the sensor" are defined for this fault.

6. Nitric Acid Cooling System Example

In this section, event correlation and diagnosis using CDG is illustrated through a nitric acid cooling system example SDM shown in figure 4. In this system, a feedback temperature controller controls the temperature of nitric acid to a reactor by manipulating the coolant flow via a flow control valve. In addition to the GFPM of the controller and sensor described in the previous section, generic models for the flow control valve and heat exchanger are defined in CDG. The initial measured symptom, "Excessive Cycling of Measured Temperature" true, in the sensor will be detected by an event detector and this event will be input to CDG. Starting from this

symptom, CDG will build the SFPM by appropriately combining the SDM and the GFPMs in the model library.

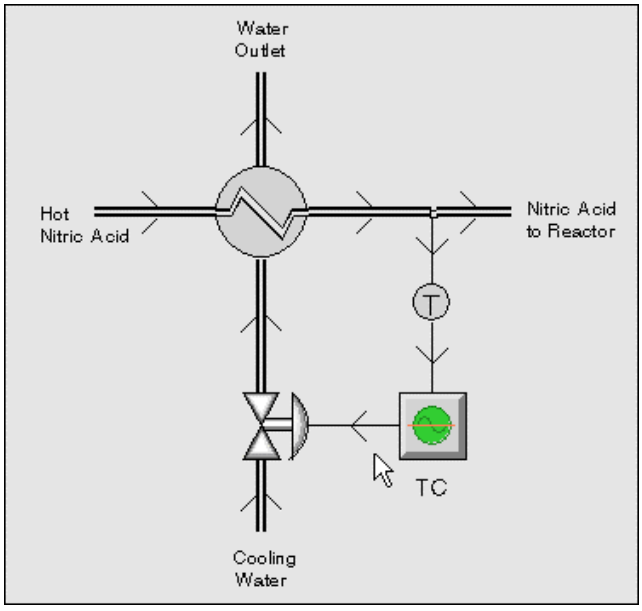

Figure 4. Nitric Acid Cooling System

By propagating the true value of the "Excessive Cycling of Measured Temperature" symptom in the SFPM, the CDG event correlator will identify the faults "High Gain Tuning" in the temperature controller and "Sensor Fouling" around the temperature sensor thermowell as the suspected root causes (assuming that the hot nitric acid inlet temperature is a measured variable and the corresponding possible cause "Excessive Cycling of Hot Nitric Acid Inlet Temperature" was already eliminated). This information will be fed to the CDG diagnostic conductor, which will then identify "Reduced Gain Test" and "Lag Estimation Test" in the controller as the respective candidate tests for resolving the suspect faults in cost order. The operator or automated procedures can then execute these test actions, and the test results are asynchronously input back to CDG. The CDG event correlator will propagate those test results in the SFPM to confirm or rule out the suspect faults.

Depending on the "Reduced Gain Test" result, the fault "High Gain Tuning" in the controller is confirmed or ruled out. Similarly, depending on the "Lag Estimation Test" result, the fault "Sensor Fouling" in the temperature sensor is confirmed or ruled out. Since CDG does not have a single fault assumption, either or both of these faults could be concluded to have occurred depending on the test results. The diagnostic conductor will inform the operator and external systems about the identification of these faults, and select appropriate mitigation actions defined for these faults for execution.

7. Conclusions

A generic fault propagation modeling approach has been developed for automating on-line event correlation and interactive diagnosis. The proposed modeling technique consists of novel concepts such as defining test and mitigation actions as part of the model and in-built state conditional dependencies. Based on this approach, a software product named CDG is currently being developed for real-time fault management in large-scale systems. CDG is part of Gensym's overall Operations Expert (OPEX) product line for managing operations environments.

CDG appropriately combines generic models with specific domain representation and builds focused specific models to investigate observed asynchronous events. Using the specific models, CDG recognizes that a group of events are correlated to each other, identifies suspected faults that could have caused the symptoms, and selects and executes candidate tests and mitigation actions to resolve the problems. CDG also provides graphical user interfaces for the development of the models, and for operator interaction. The test and mitigation actions can be defined using OPAC graphical procedure development tool or G2.

References

Finch, F. E., Oyeleye, O. O., and Kramer, M. A., "A Robust Event-Oriented Methodology for Diagnosis of Dynamic Process Systems", *Comp. and Chem. Engng.*, **14**(12), 1379 (1990).

Simpson, W.R., and Sheppard, J.W., *System Test and Diagnosis,* Kluwer Academic Publishers, Boston (1994).

Vaidhyanathan, R., and Venkatasubramanian, V., "Digraph-Based Models for Automated HAZOP Analysis", *Reliability Engineering and System Safety*, **50**, 33(1995).

Malin, J. T., Schreckenghost, D. L., and Rhoads, R. W., "Extended Real-Time FEAT", *Making Intelligent Systems Team Players: Additional Case Studies*, Section 4., NASA Technical Memorandum 104786, Johnson Space Center, Houston, TX (1992).

* Contact Greg Stanley at:
http://gregstanleyandassociates.com/contactinfo/contactinfo.htm