PATROL ROOT CAUSE ANALYSIS
A New Standard of Event Automation in an Enterprise Computing Environment

June 30, 1999

Michael R. Warpenburg
BMC Software, 10415 Morado Circle, Austin, TX, 78759 (USA)

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

Abstract: IT executives have identified Root Cause Analysis (RCA) of failures as a key factor of event automation within distributed enterprise computing environments. Integrity Reasoner, which is based on Gensym's SymCure technology, 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". By exploiting the existing PATROL infrastructure, an effective RCA solution for any application domain based on SymCure technology can be easily implemented and deployed.

Keywords: root cause analysis (RCA), PATROL, SymCure, diagnosis, fault propagation model, FEAT, digraph, knowledge engineering, automation, real-time systems, failure and recovery, test planning

(1) Introduction and Motivation

Modern event automation and monitoring technologies such as PATROL provide operators of enterprise computing environments with incredible amounts of observation information in the form of *events* and *alarms*. While this high level of system observability provides an incredibly powerful tool for monitoring a large systems, this power often reveals a greater need that, until now, has gone unanswered in the industry. These operators can be bombarded by hundreds of messages and alarms, usually resulting from a small number of *root cause* problems. Operators cannot adequately analyze all these events in real-time; they need the system to assist in pinpointing the root cause or a very small number of candidate root causes. This very important analysis function is now available for PATROL users. This paper introduces this new technology, explores how it works and how it allows the IT administrator to drastically improve the level of service within their organization.

(2) Application Availability Lifecycle

An overriding goal of the IT organization is, of course, to provide a high level service to their clients. While there are a number of aspects to this goal one very important and measurable aspect is that of *application availability*. Upon first consideration the notion of application availability is somewhat ubiquitous; it is more tangibly grasped in terms of its various components, each of which comprises the *application availability lifecycle*.

The application availability lifecycle allows us to discuss the various aspects of application availability in concrete terms. Stated simply, the application availability lifecycle is comprised of 5 distinct points:

(1) Point of Failure (PoF): the point in time where one or more application services are no longer available

(2) Point of Notification (PoN): the point in time where it is *detected* that the services are no longer available

(3) Point of Diagnosis (PoD): the point in time of the underlying root cause of the problem is identified

(4) Point of Recovery (PoR): the point in time where the failed services are restored

(5) Point of Postmortem (PoP): the point of time where information from the downtime experience is fed back into the management system (to allow proactive future management)

According to marketing studies, 80% of system downtime (from PoF to PoR) is spent diagnosing the problem to identify the root cause. The total wasted resource impact may be even greater in some situations where domain experts across multiple disciplines (and departments) must be employed. Moreover, organizations run the risk of becoming unresponsive if the number of "false alarms" becomes large (i.e. a modern day "boy who cried wolf" syndrome).

As the application availability lifecycle suggests, the greatest improvements can be achieved by reducing the diagnostic component expended in resolving problems. If the diagnosis and complexity time (from PoF to PoD) can be reduced, then the overall level of service within an organization will be drastically improved. This is precisely the component of the application availability lifecycle that is addressed by the PATROL Root Cause Analysis (RCA) system.

(3) Causal Directed Graphs (SymCure)

Gensym's SymCure technology is a general development and deployment environment for building intelligent systems to automate real-time fault management of complex systems. SymCure performs on-line event correlation and interactive diagnosis to address the full life cycle of problem identification: symptoms, root-cause analysis, diagnostic testing, fault isolation, and recovery. While SymCure is applicable to a wide variety of applications, it represents a breakthrough technology in the area of RCA. As we will see in later sections, this technology offers a number of significant advantages over competing technologies. When incorporated within BMC Software's powerful suite of PATROL products, SymCure becomes a cornerstone of a new standard in the area of event automation—the PATROL Root Cause Analysis system. The next several sections introduce the important aspects of the SymCure technology and discuss its role within the PATROL RCA system.

When 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.

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

(4) Basic Modeling Elements

A fault propagation model in SymCure 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 SymCure, 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 SymCure 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.

(5) 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. SymCure 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 which 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) which 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. SymCure shares with SDG an orientation towards asynchronous event processing and propagation of information, instead of pattern matching. Like SymCure, 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. SymCure can accommodate such quantitative “normal” models in the definition of the tests and symptoms.

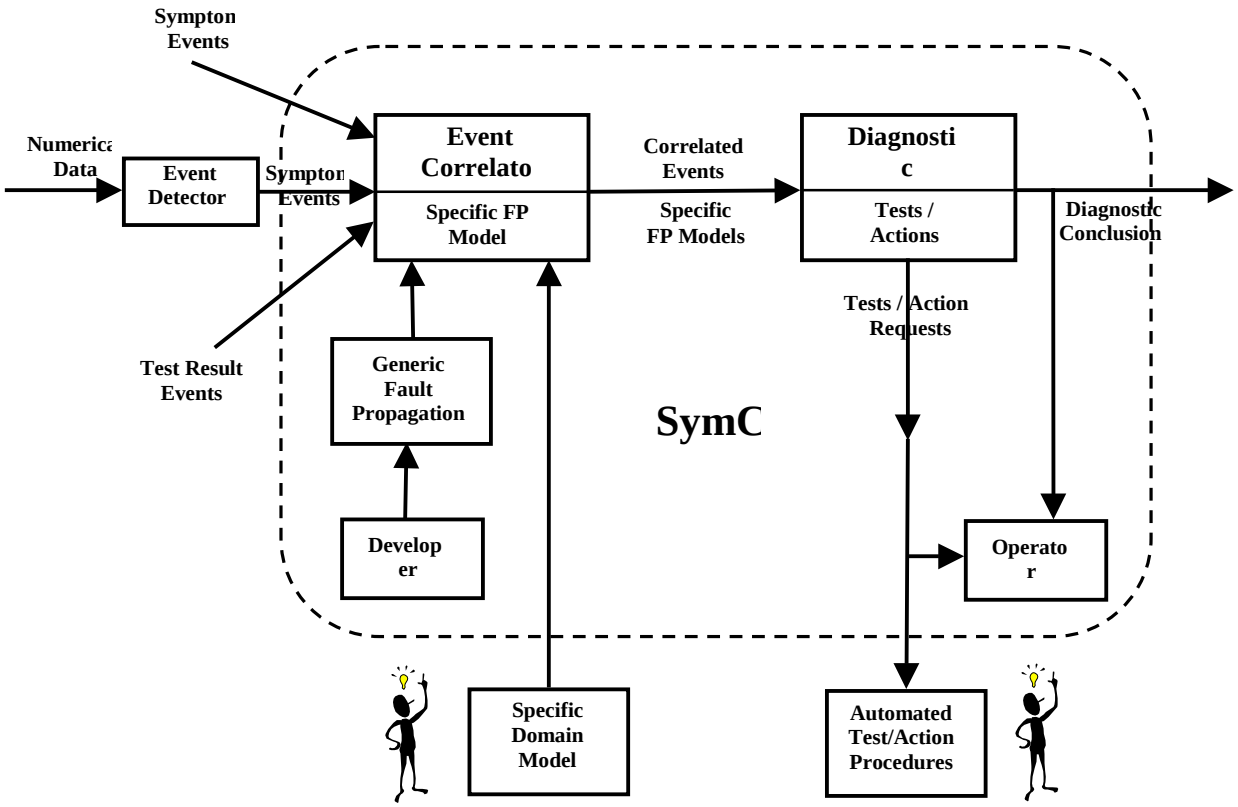
The fault propagation models used in SymCure are similar to the information flow models used for model-based design for testability and integrated diagnosis (Simpson and Sheppard, 1994). SymCure extends these models for on-line event correlation, interactive diagnosis and fault mitigation. Major extensions 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 which are similar to the fault propagation models used in SymCure. 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. SymCure 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.

Figure 1. The Architecture of SymCure



(6) Architecture

The architecture of SymCure is presented in figure 1. The input specifications required for SymCure are the Generic Fault Propagation Models and the Specific Domain Model. Using these specifications, SymCure 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 SymCure are the diagnostic conclusions and test and action requests. The various components of SymCure 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.

Event Correlator

The Event Correlator correlates asynchronous symptom and test result events input to SymCure 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 SymCure.
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 SymCure 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 SymCure to the operator and to other external systems. Whenever SymCure 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 SymCure.

(7) Exploiting the PATROL Standard

The PATROL RCA system derives its strength from combining two complementary technologies into a single solution. As described in the previous sections, SymCure provides a powerful technique for inferring root cause of system failures from an asynchronous stream of system failure events and alarms. However, effective deployment of this technology requires a solid infrastructure to reliably deliver relevant system events to the SymCure engine; additionally, the conclusions must be made available to the end user (in this case, the system operator) in a clear and useful fashion. The industry-standard PATROL suite of products provides such an infrastructure; augmenting this existing product suite with RCA capability is a natural evolution of PATROL systems that are deployed today.

The existing PATROL standard is exploited in a number of ways in yielding an effective RCA solution.

(1) *Existing KMs.* Existing KMs provide a wealth of system wellness information across a very large number of application domains. In addition to using this data for "normal" PATROL monitoring activities, RCA exploits this existing information by forwarding it to the SymCure engine for higher analysis. By applying this information to the relevant SymCure domain models, root causes of system failures can be determined.

(2) *Scalability.* PATROL spans multiple platforms over large distributed systems. As such, it provides a robust and proven delivery system for event and alarm data required by the SymCure engine.

(3) *Adaptability.* PATROL supports powerful *discovery* mechanism that allows it to be deployed out-of-the-box without the necessity of site-by-site configuration. In other words, the PATROL system discovers the existing "computing terrain" (relevant information varies from application domain to application domain but is handled automatically) when it is first turned-on and then maintains this information over time as it is used. The RCA solutions inherit this powerful mechanism and exploit it by virtue of the intrinsic separation of the *domain map* from the *fault propagation models* within the SymCure system.

(4) *Reusability.* RCA models can, in turn, be used by (incorporated by) other RCA models forming a

natural hierarchy of knowledge. Thus, RCA solutions inherit improvements and intelligence from other solutions as they evolve.

(8) Conclusions

PATROL RCA represents a breakthrough in event automation technology by incorporating a generic fault propagation modeling approach called SymCure. This approach employs novel modeling techniques such as defining test and mitigation actions as part of the model and using built-in state conditional dependencies.

PATROL RCA drastically reduces the time and resource intensive task of identifying root cause of failure allowing the IT administrator to drastically improve the level of application availability within their organization

SymCure appropriately combines generic models with specific domain representation and builds focused specific models to investigate observed asynchronous events. Using the specific models, SymCure 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.

PATROL RCA exploits the existing PATROL product infrastructure to derive additional benefit from the existing PATROL KM investment while at the same time inheriting PATROL scalability and adaptability. RCA solutions are reusable in that RCA solutions can, in turn, be used by other RCA solutions, inheriting improvements and intelligence from other solutions as they evolve.

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).