ADVANCED MAINTENANCE USING CAUSAL NETWORKS

Charles J. Sitter, Rockwell Collins, Cedar Rapids, Iowa

Gregory M. Provan, Rockwell Science Center, Thousand Oaks, California

Abstract

We present a flexible, powerful, easily maintainable diagnostics system that integrates diagnostic development functions into product development tools and processes.

Challenges of Developing Diagnostic Systems for Complex Applications

There exists a wide variety of modeling and inference methods for diagnostics, including expert systems, neural networks, and model-based approaches. All of these approaches can represent and analyze the same artifact but from different viewpoints.

Complex systems, such as commercial aircraft, represent a challenge both in modeling and inference. Next generation improvements in the performance of diagnostic systems will require, at the very least, higher fidelity models, guarantees on model consistency, and reasoning algorithms that are less susceptible to state-space explosion problems.

Moreover, analyses of the diagnostic models for complex systems must be correct and exhaustive. Incomplete analyses using probabilistic arguments become progressively more tenuous for complex systems and unacceptable for safety-critical applications.

Powerful and generic modeling paradigms and algorithms that seamlessly serve many analytic needs are key ingredients for developing the next generation diagnostic systems. Other critical aspects of models are capturing aspects of temporal and dynamic systems, and the ability to scale to complex and non-deterministic scenarios.

To address these challenges in the design of diagnostic systems for complex applications, we propose a model-based solution that integrates two state-of-the-art technologies: (1) causal network diagnostic modeling and inference algorithms and (2) distributed computing.

We argue that these two technologies can synergistically provide:

- The ability to model a fault using current design data and thus troubleshoot and repair a failure either (a) not previously identified or (b) with no existing maintenance procedures.
- The ability to provide diagnostics feedback to design engineering during the design and integration process.
- A significantly enhanced ability to scale to large complex applications while maintaining accuracy.
- The ability to guarantee computational requirements, *a priori*, based on model fidelity.

We organize this paper as follows. The following section summarizes the diagnostic representation, causal networks [1], and introduces a simple example of causal network modeling. The next section summarizes Rockwell Collin's avionics diagnostic system, briefly describing each major component of the system. The final sections provide conclusions and references.

Diagnostic Representation

This section introduces an example that we use to describe our approach, and the representation that we adopt for modeling, causal networks [1].

A causal network is a graph-based representation that is used for diagnosing failures of a system. Causal networks provide probabilistic, order of magnitude, and symbolic representations. They also have predictable scaling properties for the eventual embedded runtime diagnostic system.

A causal network encodes the causal relations of a system; e.g., in Figure 1, the transmitter in Tx causes Receivers Rx1 through Rx4 to receive data. A causal network is thus a good way of representing the flow of data in such systems, and hence reasoning about the root causes of data flow; e.g., Tx is the root cause of data being received by Rx1 through Rx4. Figure 2 shows the causal network structure for this simple example. Conversely, if data is not flowing as intended, the causal network can help track down the reason for the "breakdowns" of correct data flow; e.g., Tx could be the root cause of data being absent at Rx1 through Rx4. We call this diagnosing the faults in the system.

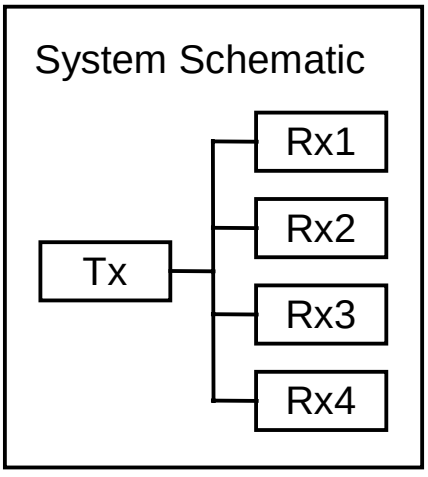

Figure 1: Simple Avionics Example

More formally, a causal network specifies the causal relationships among a set *V* of variables by encoding each variable in *V* with a node, and encoding the causal influence of V_1 on V_2 by a directed arc from V_1 to V_2. Hence a causal network is a directed graph (*N*,*A*) of nodes *N* and directed arcs *A*.

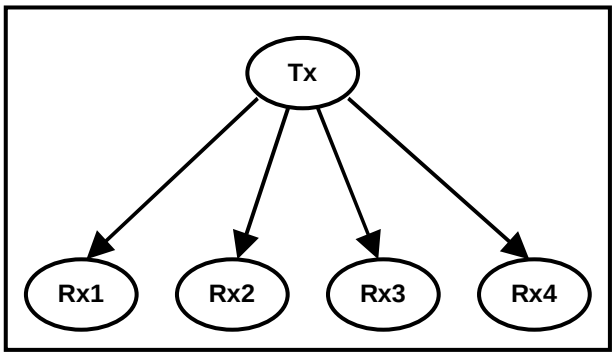

Figure 2: Causal Network Structure for Simple Avionics Example

The example we have just described covers only hardware, but note that the causal network modeling language allows functional specifications of both software and hardware.

Causal Network Specification Language

To specify a causal network model we need to define:

1. the variables in the model, which represent the components, e.g., Tx, Rx1, Rx2, Rx3, Rx4 from Figure 1;
2. the values for the variables, which represent the data that flows through various parts of the system;
3. the assumables, which are the variables that describe the operating characteristics of the components, such as OK or broken;
4. the quantifications, which relate the variables and assumables;
5. the evidence variables, which are the variables that can be observed;
6. the weights for the assumables, which specify the likelihood or relative ranking of the assumable fault modes.

The component modes characterize the ways the component works under an exhaustive set of scenarios. Rx1 has associated assumables, OK and broken, and a quantification as follows: if OK, Rx1 receives data from Tx, but if broken receives no data. In propositional logic, we might write this as:

- IF (Rx1-mode = OK) AND (Tx = transmit) THEN (Rx1 = receive)
- IF (Rx1-mode = broken) OR (Tx = not-transmit) THEN (Rx1 = not-receive).

Figure 3 shows the causal network model with the assumables and quantification defined.

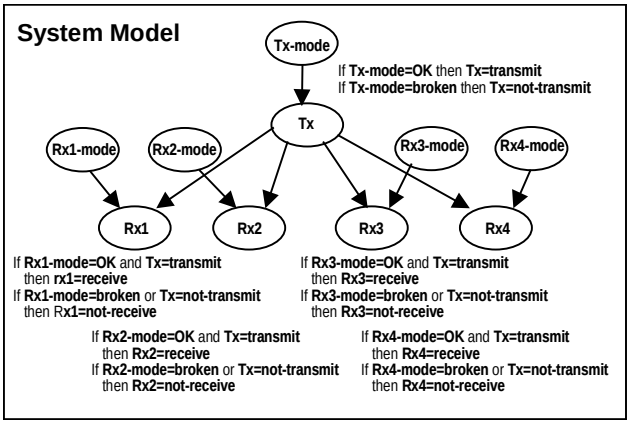

Figure 3: Causal Network for Simple Avionics Example

As described later in the paper, we have implemented a graphical user interface (GUI) that allows a user to specify a system using a simple set of components that are connected based on data flows. For each component, the user needs to specify only the inputs and outputs, the functional relationships between the inputs and outputs, and the operating modes of the component.

Note that the causal network representation can describe not only deterministic systems as in the above example, but also stochastic systems (using probabilities and order-of-magnitude probabilities (OMP) [1,2]), and discrete-event systems [4].

Rockwell Avionics Diagnostic System

Rockwell has developed a distributed maintenance architecture that is based on causal networks and open system standards, and provides the capability to significantly reduce the costs of development and of upgrading and extending the onboard maintenance system (OMS) of the future. The Rockwell avionics diagnostic system has several characteristics that, taken together, distinguish it from competing systems. These key characteristics include:

1. an avionics-oriented user interface for model development, simulation and diagnostic testing;
2. guarantees of diagnostic correctness and completeness given the models;
3. a compiler that allows the user to embed a real-time diagnostic system on-board the aircraft;
4. the deployment of the diagnostic system within a distributed, open architecture.

This section now describes the overall architecture of this diagnostic system.

Diagnostic System Architecture

The general architecture of the tool set, is shown in Figure 4. Diagnostic system development and deployment is divided into two phases: off-line model generation and compilation, and on-line diagnostic evaluation.

The *off-line* phase entails developing and analyzing a model, and then compiling that model so that it can be efficiently evaluated by the on-board avionics system. Model development takes place using a graphical user interface called EASIER (Enhanced Analysis and Simulation Integrated Environment from Rockwell), and compilation is based around a causal network inference system called

CNETS. EASIER supports development of primitive blocks and subsequent interactive model development and simulation for avionics applications. The output of EASIER is a causal network on which inference is performed by CNETS. Models generated in EASIER can be output to a diagnostic compiler in CNETS to generate embeddable diagnostic code.

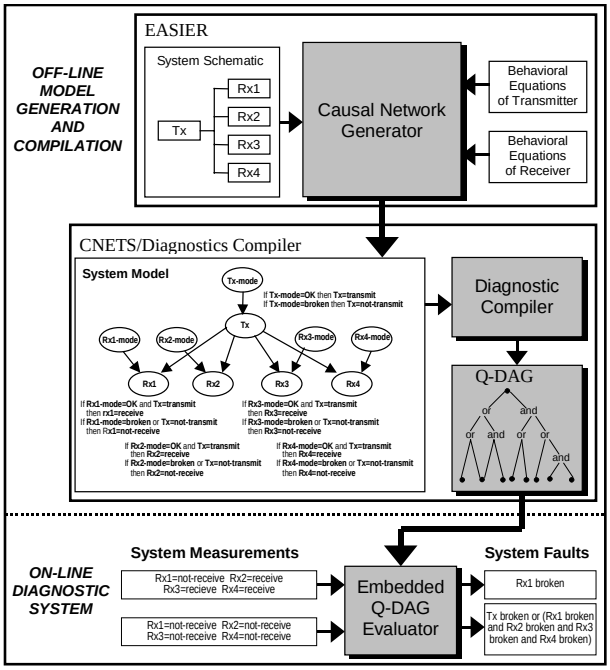

Figure 4: Diagnostic System Architecture: On-Line vs. Off-Line Subsystems

The *on-line* component of this tool-set is a real-time evaluator/interpreter that operates on the compiled diagnostic model or Q-DAG (Query Directed Acyclic Graph) to generate diagnostic results from a set of input variables. A Q-DAG is a compiled representation of the causal network, and is an expression that is optimized for real-time embedded diagnostic inference.

It is important to note that the underlying diagnostic tool, CNETS, as well as the associated compilation system, is a completely general system that can perform domain-independent causal network inference. The other component, EASIER, was designed specifically for the avionics domain, although the concepts of the EASIER interface can be applied to other domains (e.g., communications, digital circuits).

A second important point to note is that the EASIER/CNETS system is meant to hide the details of diagnostic modeling and inference so that engineers can use the system without having to acquire significant new skills. The EASIER/CNETS system has been explicitly tailored to avionics engineers, presenting to these engineers a front end with which they will be familiar. This approach unlocks the potential of causal networks to avionics applications.

Distributed Diagnostics Architecture

For aircraft onboard maintenance system (OMS) applications for the 21st century, the OMS must move to "open systems" and use "industry standards" in an attempt to get away from the proprietary "black box" implementations. The model-based approach is critical to allow model updating and re-use, and to ensure that different types of models can be stored in a single database, thereby leveraging common information. Further, it is widely recognized that to take advantage of current and future technology, the traditional federated avionics system application approach of the past is being replaced with an advanced, multiprocessing, distributed model that provides for future flexibility and expansion. Part of this strategy is to develop a "Distributed Diagnostic Architecture" that does not require specialized systems to be incorporated in the aircraft system to perform the diagnostic function.

Rockwell has defined a distributed maintenance architecture, based on open system standards that can significantly reduce the development costs, as well as the costs for upgrading and extending the OMS of the future. The application of Rockwell's next generation OMS divides the user interface, data storage/retrieval, diagnostic engine, and OMS application code. Separation of these components provides a system design approach that allows flexible applications for the diagnostic

reasoning methods, the data collection and concentration methods, and the maintenance display methods.

Additional benefits of this architecture are improved responsiveness, throughput, and extensibility. Using a distributed system allows for functions to reside on multiple processors within the avionics system. The well-defined object interface allows the CPU loading to be shared over a network, as well as to dynamically change the distribution of tasks for better load sharing and redundancy management. Individual components can simply be upgraded without affecting the other components. If needed, additional functional objects can be added to this open system to allow unplanned options to be incorporated with less risk and cost than previous designs could provide.

A distributed architecture is composed of object-oriented components that are freely distributed across various computers in a network. The key to this distributed environment is developing standardized methods for brokering and communicating objects throughout the network. This is the responsibility of an Object Request Broker (ORB). The two major competing standards are the Common Object Request Broker Architecture (CORBA), developed by the Object Management Group (OMG), and the Distributed Component Object Model (DCOM), developed by Microsoft Corporation. These models effectively separate an object's interface from its implementation by using an Interface Definition Language (IDL). The IDL provides a standardized representation of an object and its methods and attributes, which remain consistent regardless of the language or platform implementation. By using standard IDL interfaces, CORBA and DCOM make distributed programming simple, allowing developers to treat and use any remote object as if it were local to the user.

OMG has also developed an Internet Inter-ORB Protocol (IIOP) standard that guarantees CORBA interoperability over the Internet. IIOP is a transport alternative to the popular HTTP that incorporates an object-oriented communications paradigm instead of the current page-oriented one. The CORBA IIOP approach allows an ORB from one manufacturer to have full interaction with an ORB from another manufacturer without any intervention or special programming.

In order to reuse system information and models (e.g., aircraft system interconnects, data dictionaries) between different organizations (e.g., system engineering, maintainability, reliability, hardware, software) or different tools, the different organizations must either share a common repository or their tools must be able to easily exchange data. Most tool vendors have proprietary formats and databases underneath their tools, so it is virtually impossible to get these tools to share data without some standard data interchange format, such as the CASE Data Interchange Format (CDIF).

CDIF defines a single architecture for exchanging data between modeling tools and repositories, and defines the interfaces of the components to implement this architecture. CDIF also defines a standard way of moving data between tools, without the need for customized interfaces. Each tool only needs one import and one export interface to be able to communicate with other CDIF-conformant tools.

Whether a single repository should be used for storing the data or if each tool should store its own data is still being debated. Our view is that a single common repository should be used to provide a single source for configuration management, version control, model synchronization, etc. Without this single source of control, sophisticated mechanisms must be developed to make sure that a model change in one tool is automatically reflected in all other tools.

Taking into account these standards, a new distributed architecture, shown in Figure 5, is defined for the current development system.

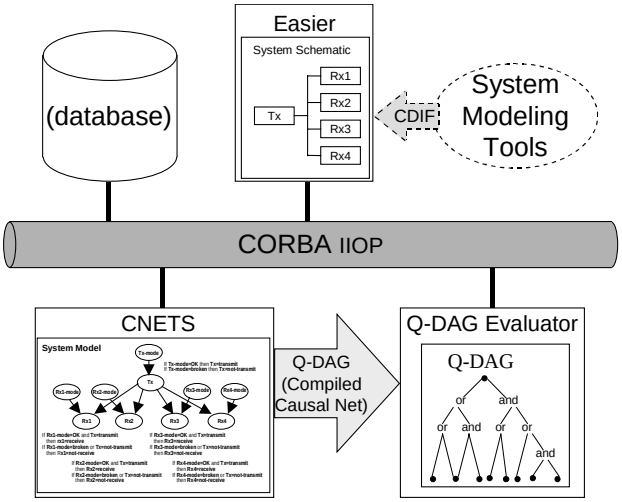

Figure 5: Architecture for Distributed Diagnostic Development System

Similarly, the embedded system, shown in Figure 6, can be distributed over three separate components that communicate over a network using CORBA technology: a data collector, a diagnostic engine (Q-DAG evaluator), and a presenter. EASIER generates the loadable tables necessary for the embeddable components. The *data collector* would monitor the raw input data, perform any filtering on that data, and pass state changes to the diagnostic engine. The *diagnostic engine* would update the Q-DAG based on these state changes and generate the latest diagnosis based on the current state of the system. These results would then be sent (either automatically or on demand) to the *presenter*, which would display the ambiguity group(s) to the appropriate user.

The distributed environment also allows for multiple data collectors and/or presenters, if desired. Separating these components provides a system design that allows flexible applications for the diagnostic reasoning methods, the data collection and concentration mechanisms, and the maintenance display platform. Additional benefits of this architecture are improved responsiveness, throughput, upgradability, and extensibility.

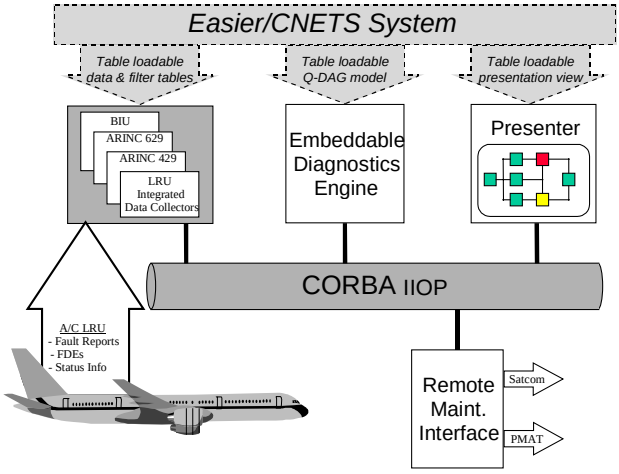

Figure 6: Architecture for Distributed Diagnostic Run-Time System

Diagnostic System Components

This section summarizes the major modules in the diagnostic system, the EASIER interface and the CNETS inference system and compiler.

EASIER User Interface

Diagnostic models developed in EASIER/CNETS are viewed as a collection of interconnected primitives. These primitives, or diagnostic model building blocks, are customized to achieve the appropriate diagnostic "view" for an application. Only information relevant to this diagnostic view (failure characteristics, system variables) is modeled (e.g., general functional characteristics not related to diagnostics are not modeled). Depending on the problem specifications, a primitive could be a logic gate, a bus, or a complex LRU.

Once a set of primitives is identified for a diagnostic problem, causal network structures are compiled and stored for these primitives. These primitives are then available in the EASIER graphical interface—users develop a system model by placing instances of these primitives on the screen and connecting them to define a complete system. The tool then generates a causal network system model from the

specification of these primitives and their interconnections.

The behavior equations (which map into causal network properties) defined for each primitive building block can be simple or complex. Flexible equations are used to capture this behavior. CNETS supports relationships based on both multi-valued propositional logic and conditional probabilities for equations, and OMP or probabilistic weights for modeling component reliabilities.

Diagnostic models developed in EASIER can be debugged and refined through EASIER's simulation utility. EASIER allows the user to simulate normal and abnormal conditions in the system behavior. System users can also simulate and refine the model by changing the state of individual components. In the system, the colors of the blocks indicate the state of the block, e.g., OK or broken. The fully interactive environment provides immediate feedback (e.g., highlighting of affected pathways, changing of color-coded states) to gain understanding of model behavior. In the airplane-level diagnostics domain, EASIER's simulation capabilities provide mechanisms to solve cascade effect removal, fault isolation, fault consolidation, and flight deck effect correlation in one integrated visualization environment.

Causal Networks Inference System (CNETS)

CNETS is a system for representing and reasoning with causal networks, which include probabilistic and symbolic causal networks. CNETS allows the user to create a model by specifying a causal structure and a quantification of this structure using any of the methods described earlier.

Off-Line Inference

The chosen method of quantification depends on the application domain, the reasoning task, and the available information about the system being modeled. Independent of the causal network used, the user will typically engage in the following type of scenario with CNETS: First, the user asserts available evidence about the modeled system. Second, the user makes queries about the behavior of the system, typically simulation, prediction and/or diagnostic queries.

CNETS implements a number of inference algorithms that can be used to process such queries. The fundamental class of algorithms used by CNETS transforms the input causal network into a form that facilitates the particular type of query-processing. The primary algorithm used is a modified version of the clique-tree algorithm [3]. If the queries are probabilistic, then an algorithm like the clique-tree algorithm can be used directly. CNETS also supports a symbolic form of this algorithm for answering symbolic queries for avionics systems and for discrete-event systems [4].

CNETS Compiler for On-Line Inference

CNETS has a compiler that allows the user to generate a run-time version of the system model by using a compiler [2] in conjunction with an algorithm like [3]. Using this novel compilation technique, CNETS is able to compile a causal network into a Boolean expression, called a Query Directed Acyclic Graph, or Q-DAG, that contains all possible diagnoses for the complete system, given inputs for the observable variables of the system model.

The Q-DAG provides the same complete diagnostic information that is available with the original causal network, given that we specify which variables are used as input values and which variables are used for diagnostic conclusions. The main benefit of this compiled approach is an embeddable model that, together with a very simple evaluator, requires minimal computer processing resources and can be ported to other platforms and languages with relative ease.

Causal Network Guarantees

One of the key features of the causal network approach is that, for diagnostic inference, it provides a number of guarantees. First, the causal network approach provides

soundness and completeness guarantees based on the model [1]. In other words, if any diagnosis is computable from the model, CNETS is guaranteed to compute it. This guarantee allows users, when developing models, to test the BIT (built-in test), BITE (built-in test equipment) and diagnostic coverage.

Second, it can provide complexity guarantees in terms of the model parameters. The structure of the network is the most critical aspect that governs the complexity of inference, and this is why causal networks are one of several classes of techniques termed "structure-based" approaches. For example, in the causal graph the maximum number of parent nodes of any node is one of the most significant parameters. Because all significant complexity parameters are ultimately derivable from the system structure, the structure can be manipulated to obtain acceptable inference complexity without altering the diagnostic coverage. Approaches that are not structure-based do not have this capability to alter inference complexity without altering the diagnostic coverage.

Beyond the complexity guarantees, the causal network approach has been optimized in a number of ways. This approach uses a focusing mechanism to focus inference on only the most likely diagnoses [1]; in addition, it makes use of observations to prune and to decompose the system (and hence reduce overall inference complexity) [5].

Conclusions and future work

This paper has summarized a new approach to designing an advanced maintenance system that provides important capabilities both for system design and for on-board diagnostics. This maintenance system provides a graphical user interface that has been tailored for avionics and is designed for ease of use by engineers. This visualization allows engineers to design systems using a schematic-oriented, system-level framework with which they are familiar. The human computer interface (HCI) is connected to the other system software components using CORBA middleware. The on-board diagnostics system can be compiled automatically from the models created by the engineers through the user interface. Together these features constitute a maintenance system that significantly advances the state-of-the-art.

We intend to extend this approach in a number of ways. One important extension is to implement prognostics and remaining functionality algorithms, such that we can predict future faults and define remaining vehicle functionality given the predicted or existing faults, respectively.

References

[1] A. Darwiche: "New Advances In Structured-Based Diagnosis: A Method For Compiling Devices," Proc. Of The 8th International Workshop On Principles Of Diagnosis (Dx97)

[2] Adnan Darwiche and Gregory Provan, "Query-DAGs: A Practical Paradigm for Implementing Belief-Network Inference," *Journal of AI Research*, 1996.

[3] V. Jensen, S. L. Lauritzen, and K. G. Olesen, "Bayesian Updating in Recursive Graphical Models by Local Computation," *Computational Statistics Quarterly*, 4:269--282, 1990.

[4] Cassandras, C. G., *Discrete Event Systems*, IRWIN Inc. 1993.

[5] Adnan Darwiche and Gregory Provan, "The Effect of Observations on the Complexity of Model-based Diagnosis," *Proc. AAAI, pp. 91-94, 1997*.