From knowledge graphs to probabilistic models for system-level diagnostics

T. Nägele*, L. Barbini, G. van den Braak, M. Lipplaa and A. Piedrafita

**High Tech Campus 25, 5656 AE Eindhoven, The Netherlands*
{thomas.nagele, leonardo.barbini, gert-jan.vandenbraak, micha.lipplaa, alvaro.piedrafitapostigo}@tno.nl

Abstract. The increasing complexity of high-tech systems poses a significant challenge on service organizations tasked with the timely identification of the root causes of unexpected downtimes. While data-driven methods are effective for diagnosing frequently occurring issues, or those affecting a large number of systems, rare issues suffer from data scarcity, necessitating alternative approaches. This paper presents a two-step model-based methodology that leverages system architecture information to support diagnosing of systems for which little data is available. Firstly, system design and observability information, including diagnostic tests, is captured in a knowledge graph. Secondly, the knowledge graph is queried and transformed into a probabilistic graphical model. This is fully automated using transformation rules based on the ontology underlying the knowledge graph. The probabilistic graphical model then infers the most likely causes of failure using measurement data, and guides service engineers by suggesting cost-effective diagnostic actions. This paper outlines the proposed methodology, demonstrates its application on a small example system, and reports early-stage validation findings from high-tech system cases.

1. Introduction

Efficiently diagnosing high-tech systems is increasingly difficult. This is not only caused by the increased complexity of such systems, but also by their long lifetime and high variability due to customization (Acur & Hendriks, 2024), resulting in a variety of failure manifestations. Some failures will happen relatively frequently across the installed base, allowing the use of data-driven approaches to recognize and diagnose them quickly, provided that sufficient data is being logged. Other failures will occur more rarely, so these are hard to diagnose with data-driven methods, as such approaches require a vast amount of data for training. The diagnosis of rare failures requires one or more highly experienced service technicians and is often time consuming and costly. Growing numbers of systems in the field, combined with their increasing complexity and the scarcity of skilled personnel demand a technical solution: a digital diagnostic assistant that enables fewer technicians to handle more diagnoses effectively. This paper presents a methodology to assist in the diagnosis of rare hardware failures without depending on vast amounts of data.

Model-based diagnostics (MBD) brings automated diagnostic reasoning to the service technician, using a formal model to perform the reasoning (de Kleer & Williams, 1987). MBD formalisms to perform the reasoning include logic (Poole, 1988) and probabilistic models (Pearl, 1988). The structure and semantics of such models depend on the information they are based on. These models can be created either from expert experience (experience-based) or from the system's design information (design-based). Due to the intrinsic lack of expert experience and data for rare failures, our methodology leverages design information as source of information for these diagnostic models.

The architecture of our methodology is shown in Figure 1. On the left-hand side, system dependencies are manually extracted from design information. These system dependencies are transformed into a *diagnostic network*, which is a knowledge graph according to a fixed ontology. The diagnostic network is transformed into a reasoning model. In our methodology we use probabilistic graphical models as diagnostic reasoners for their ability to deal with arbitrary model structure and perform both deterministic and probabilistic reasoning. The reasoning model can be used in two ways. First, it can be used by an analyzer to assess the observability of failures by means of design-for-diagnostics algorithms (Barbini et al., 2021). Insights generated by the analyzer can be used to improve the system design, represented as the light grey edge back to the design information. Second, the reasoning model can be used by the diagnoser to guide the service engineer in the root cause analysis process, by iteratively suggesting the next best tests (van Gerwen et al., 2022), (van Gerwen et al., 2024). This diagnosis loop is represented on the right-hand side of Figure 1.

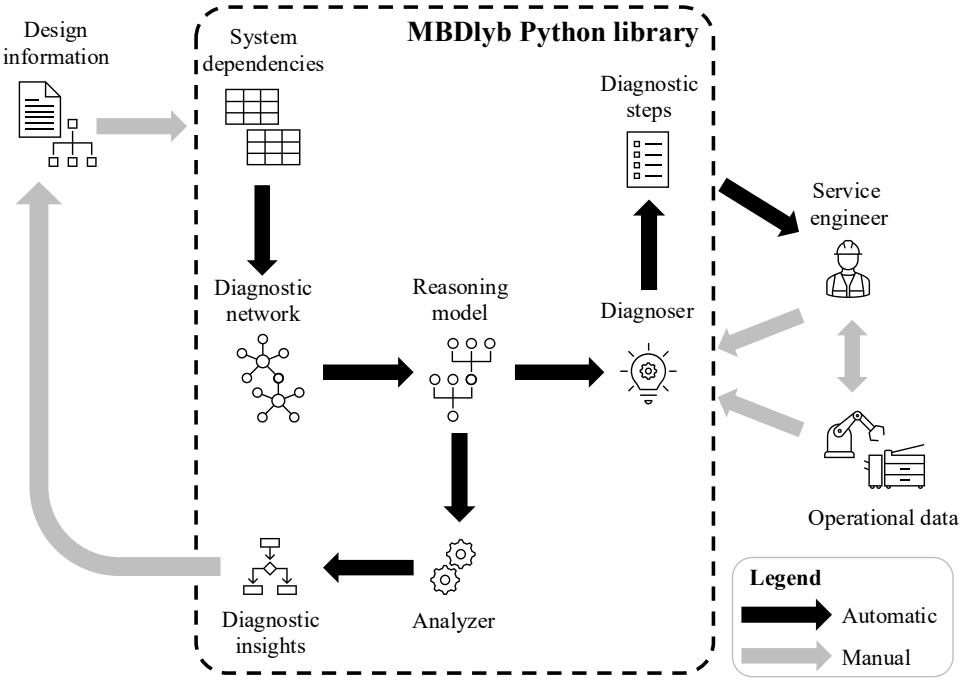

Figure 1. Architecture of our function-based diagnostic methodology, indicating transformations between different knowledge representations. Dark edges correspond to automated transformations, and light edges are (semi-)manual.

The remainder of this paper focuses on the center part of this architecture. For further details on the methodology, see (Nägele et al., 2025), and (Nägele, 2025) for the MBDlyb Python library. Section 2 introduces the ontology of the diagnostic network and describes the types of information sources used to instantiate it. Section 3 specifies the formalism of the reasoning model and explains how the diagnostic network can be transformed into this formalism. Section 4 shows how the model can be used to support the diagnostic process, and the paper concludes in Section 5.

2. Diagnostic network: a knowledge graph

The proposed methodology uses system architectural information for diagnostic purposes. This information is stored in the diagnostic network. The methodology follows a model-based systems engineering (MBSE) approach starting from the system functional decomposition, in which the high-level system functions are iteratively decomposed into subsystem functions, module functions, and so on until an elementary level of functionality is reached. Once the functional decomposition is specified, a structural decomposition is made and system functions are allocated to structural parts, i.e., hardware components. Inter-functional dependencies can be specified to indicate what system functions are required for fulfilling another system function. Figure 2 (right) illustrates how modeling functional dependencies using design information enables automatic derivation of hardware failure observability, eliminating the need for manual expert linking of observations to hardware failures (left).

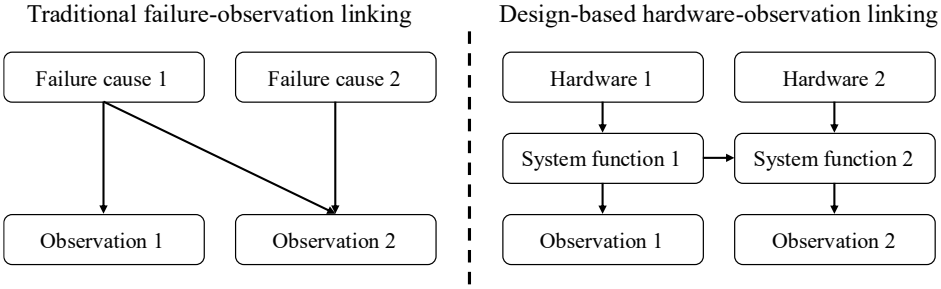

Figure 2. (Left) Traditional constructs in which observations are linked to possible failure causes based on experience. (Right) The proposed idea of a design-based approach: hardware failures are observable due to system functions failing.

As shown in Figure 2, including observations in the diagnostic network is crucial for diagnosing hardware failures. These observations represent system parameters, such as key performance indicators (KPIs) or events logged by software. For example, software may log an error if it is unable to fulfill the function to move a robotic arm to a specified position. Another type of observation is one resulting from an action triggered by the operator, such as taking a physical measurement from the system. We refer to the first type of observation as a *DirectObservable*, and to the latter as a *DiagnosticTest*.

The diagnostic network follows a hierarchical representation of the system according to the structural decomposition. This tree-like structure is organized into elements called *Clusters*. A *Cluster* may contain other clusters, hardware parts, system functions or observables. While the diagnostic network is structured hierarchically, it can be represented as a flat knowledge graph following a strict ontology to support systematic generation of the reasoning model. Table 1 shows the node types in our ontology and a brief description of what these types represent in the domain of high-tech systems. Table 2 shows the relation types in the ontology, as well as the types of nodes the relation connects and a brief description of what the relation represents.

Node	**Represents**
Cluster	Hierarchical layer in the structural decomposition, acts as container for other nodes
Hardware	Hardware component that may fail
Function	System function
DirectObservable	Automated observation, such as KPIs, parameters or event logs
DiagnosticTest	Action (test) that can be executed to acquire additional data from the system
DiagnosticTestResult	One of the findings resulting from conducting a diagnostic test

Table 1. Types of nodes in the diagnostic ontology and a short description of what these represent.

Relation	**Connects**	**Represents**
part_of	Any node to a Cluster	The containment of a specific node in the hierarchy
realizes	Hardware to Function	The functional allocation of the function on the hardware. Indicates that the hardware part is necessary for realizing the system function
subfunction_of	Function (X) to Function (Y)	A functional decomposition of function Y into function X
required_for	Function (X) to Function (Y)	A functional dependency of function Y onto X
observed_by	Function or Hardware to DirectObservable	A dependency relation for an automated observable onto the function or hardware component
results_in	DiagnosticTest to DiagnosticTestResult	A containment relation indicating that a reading for the connected result will be acquired by conducting the test
indicated_by	Function or Hardware to DiagnosticTestResult	A dependency relation for a test observation onto the function or hardware component

Table 2. Types of relations in the diagnostic ontology, including the relation scope and representation.

The ontology connects to the Arcadia method (Voirin, 2017), which has been implemented in Eclipse Capella (Roques, 2017). The logical and physical architectures in this MBSE method contain relevant information, such as the functional decomposition (*subfunction_of*), structural decomposition (*part_of*), functional deployment (*realizes*) and functional exchanges (*required_for*). Since an increasing number of high-tech companies are adopting such model-based approaches to formalize their system's design, Capella models have been used as basis for the diagnostic models in our validation.

3. Transformation to reasoning model

Unlike simulation tasks, the diagnostic task focuses on inferring the health state of unobserved variables – top layer in Figure 2 – from a set of uncertain observations – bottom layer in Figure 2. This formulation of diagnostic reasoning as an inference problem makes probabilistic graphical models (PGMs) a natural choice for its formalization. While many diagnostic approaches use Bayesian networks as their PGM of choice, our methodology uses Markov random field (MRF) (Koller & Friedman, 2009), as it can cope with cyclic relations.

In a MRF, every node represents a probabilistic variable with one or more discrete states. Edges between the nodes represent probabilistic dependencies between the variables. Figure 3 shows an example Markov random field, in which 'Function 1' depends on two hardware components, and 'Function 2' depends on one hardware component and on 'Function 1'. 'Function 1' is testable, and 'Function 2' can be observed automatically. In our approach not all types of nodes in the diagnostic network are transformed into nodes of a MRF. Only the *Hardware*, *Function*, *DirectObservable* and *DiagnosticTestResult* nodes are transformed. The remaining node types are used solely to structure the system knowledge. In the MRF all nodes are random variables with two states. *Function*, *DirectObservable* and *DiagnosticTestResult* nodes can be *Ok* or *NOk*. *Hardware* nodes can be *Healthy* or *Broken*.

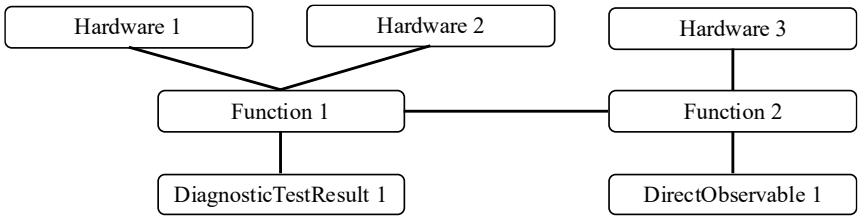

Figure 3. Visualization of an example Markov random field.

Conditional dependencies between nodes in the MRF are expressed as factors. The structure and parameters of each factor are determined by the type of the corresponding node in the diagnostic network and the types of its neighbors in the knowledge graph. Hardware nodes, which do not depend on other variables, are associated with unary factors representing prior probabilities: 0.99 for the *Healthy* state and 0.01 for the *Broken* state. All other factors in the MRF are modeled using noisy AND-gates (Pearl, 1988). To finetune the dependency — e.g., how strongly a function depends on another function—relations in the diagnostic network have weights. These weights are incorporated into the parameters of the corresponding factors in the MRF. Table 3 shows the factor for 'Function 2' as shown in Figure 3 for the default case of all weights set to 1.

Hardware 3	**Function 1**	**Function 2 = Ok**	**Function 2 = NOk**
Healthy	*Ok*	1.0	0.0
Healthy	*NOk*	0.0	1.0
Broken	*Ok*	0.0	1.0
Broken	*NOk*	0.0	1.0

Table 3. Factor for 'Function 2' as shown in Figure 3.

This factor encodes that each Hardware and Function on which 'Function 2' depends must be *Healthy* or *Ok* for 'Function 2' itself to be *Ok*; otherwise, it will be *NOk*. Table 4 shows the factor when different weights are assigned to these dependencies. The specific weights are indicated in the header of the table.

Hardware 3 (0.8)	**Function 1 (0.6)**	**Function 2 = Ok**	**Function 2 = NOk**
Healthy	*Ok*	1.0	0.0
Healthy	*NOk*	0.4	0.6
Broken	*Ok*	0.2	0.8
Broken	*NOk*	0.08	0.92

Table 4. Factor for 'Function 2' if its dependency on 'Hardware 3' would have a weight of 0.8 and its dependency on 'Function 1' a weight of 0.6.

For *DirectObservable* and *DiagnosticTestResult*, false positive and false negative rates can be specified on the corresponding nodes in the diagnostic network. This allows the reasoning model to account for uncertainty in observations.

The diagnoser uses the reasoning model to compute posterior probabilities for all hardware nodes in the model after inserting evidence gathered from the system, see (Nägele et al., 2025) for further details. Evidence is inserted into the MRF by adding unary factors to the *DirectObservable* and *DiagnosticTestResult* nodes,

reflecting their observed values. After every step, these posteriors represent the likelihood of each of the hardware components to have failed. The conditional entropy between each *DiagnosticTest* (as collection of *DiagnosticTestResults*) and suspected hardware nodes is computed and used to rank the utility of the remaining diagnostic tests. This ranking is then presented to the service technician. After a new test is executed, the resulting evidence is added to the model, and updated diagnoses are computed (van Gerwen et al., 2022). This is akin to dynamically generating a fault tree based on the available evidence. As such, it supersedes static fault tree analysis.

4. Application

This section applies the methodology to a simple system: a CD-radio player. For illustrative purposes, we assume that this system consists of four main modules: a power system, a CD reader, a radio receiver, and an audio system (i.e., the speaker). The transformation from the system architecture model to the diagnostic network is depicted in Figure 4. The figure shows only the highest level of the model. Each module has been further detailed out (not shown here), resulting in a final model with depth 5. After this automatic transformation, the diagnostic network does not yet contain any observability information. For this simple system, it was assumed that the lowest-level functions and hardware components could be tested, and *DiagnosticTests* were therefore added at this lowest level. At the higher level, two *DirectObservables* were added: one for the 'ReadCD' function and one for the 'ReceiveRadio' function. The diagnostic network can be automatically transformed into a reasoning model. This resulting reasoning model (not shown here) is topologically very similar to the diagnostic network of the system. Below we show the usage of a reasoning model of the CD-radio player system under an hypothetical scenario. Hypothetical findings in accordance with the scenario will be inserted in the model by following the diagnostic tests suggested by the diagnoser.

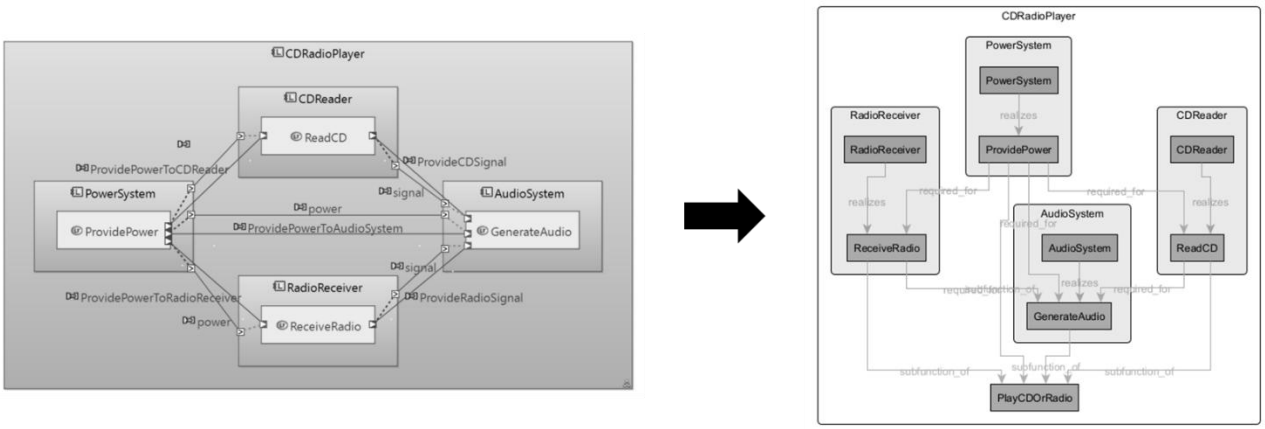

Figure 4. Transformation from the system architecture model in Capella (on the left) to the diagnostic network (on the right).

Let us assume the hypothetical failure to diagnose is a broken motor (in the CD reader). This broken motor causes the listener to be unable to hear music, while attempting to play the CD. In this scenario, it is assumed that the power system, which also supplies all other modules, is functioning correctly, and that there are no issues in the audio system itself. Evidence inserted in the model are according to these assumptions. Table 5 shows the diagnostic steps and their findings for this hypothetical situation. The model correctly identifies the CD reader's motor as being faulty.

Step	New finding	Diagnoses / likely suspects (likelihood)	Suggested diagnostic test
1	CD error showing	All components in the power system and the CD reader modules (~10%)	Check whether the CD reader's motor is spinning
2	CD reader's motor is not spinning	CD motor and all components in the power system (~15%)	Check temperature of the power supply
3	Power supply temperature is Ok	CD motor and all electrical components in the power system (~20%)	Check whether power LED is on
4	Power LED is on	CD motor (~100%)	*Reached a clear diagnosis.*

Table 5. Diagnostic steps to diagnose the hypothetical failure of the CD motor in the CD-radio player.

We applied the proposed methodology also in an industrial setting. Together with ASML we created a diagnostic network based on high-level system architecture information, covering 10 subsystems in various levels of detail. Three solved cases from customer support were selected to be diagnosed retrospectively with the model. This single model was able to reach a satisfactory diagnosis for all three cases, also according to domain experts from ASML, confirming the diagnostic capabilities of this MBD approach.

5. Conclusions and future work

In this paper we have presented a methodology for aiding service technicians in identifying the root cause of system failures. We have shown that diagnostic reasoning can be effectively executed based on available design information in the form of architectural descriptions of the system. The diagnostic models presented here are conceptually very similar to those built by system engineers following MBSE principles, and, as shown, can be partially derived from them. We have identified that a key challenge lies in integrating observability information (KPIs, errors, tests) with MBSE models, a functionality currently not sufficiently provided by MBSE approaches. Using knowledge graphs as a diagnostic knowledge base has proven to be effective. It introduces a clear separation of concerns, in which the knowledge graph is meant to formalize knowledge, while the reasoning model can be of any structure that is relevant for supporting diagnosis. The use of a knowledge graph is not limited to diagnostics: it may also be one central knowledge base, storing much more information than for diagnostic purposes. As long as it also contains the diagnostic-relevant information, the formal transformation to the reasoning model may be done. Together with industry, more large-scale validation is needed to increase confidence in this MBD approach and to identify edge-cases for which extensions or modifications are required. Future research may focus on the inclusion of software behavior into the model, as the current approach is tailored to identify hardware components.

Acknowledgement

The research is carried out as part of the SD2Act program under the responsibility of TNO-ESI in cooperation with ASML. The research activities are co-funded by TKI HTSM via the PPP Innovation Scheme (PPP-I) for public-private partnerships.

References

Acur, S., & Hendriks, T. (2024). *PMC SA / SE Vision and Outlook for Systems Architecting and Systems Engineering in the High-Tech Equipment Industry*.

Barbini, L., Bratosin, C., & Nägele, T. (2021). Embedding Diagnosability of Complex Industrial Systems Into the Design Process Using a Model-Based Methodology. *PHM Society European Conference*, *6*(1), 9.

de Kleer, J., & Williams, B. C. (1987). Diagnosing multiple faults. *Artificial Intelligence*, *32*(1). https://doi.org/10.1016/0004-3702(87)90063-4

Koller, D., & Friedman, N. (2009). *Probabilistic graphical models: principles and techniques*. MIT press.

Nägele, T. (2025). *MBDlyb: Python library for model-based diagnostics*. https://esi.nl/research/output/tools/mbdlyb

Nägele, T., Barbini, L., van den Braak, G.-J., Piedrafita Postigo, A., & Lipplaa, M. (2025). *Guided diagnosis of functional failures in cyber- physical systems*. https://resolver.tno.nl/uuid:52f484ea-2638-4403-88a4-65305b245697

Pearl, J. (1988). Probabilistic Reasoning in Intelligent Systems. In *Probabilistic Reasoning in Intelligent Systems*. https://doi.org/10.1016/c2009-0-27609-4

Poole, D. (1988). Representing Knowledge for Logic-Based Diagnosis. In *Proceedings of the International Conference on Fifth Generation Computer Systems 1988*.

Roques, P. (2017). Systems Architecture Modeling with the Arcadia Method: A Practical Guide to Capella. In *Systems Architecture Modeling with the Arcadia Method: A Practical Guide to Capella*. https://doi.org/10.1016/C2016-0-00854-9

van Gerwen, E., Barbini, L., Borth, M., & Passmann, R. (2024). Efficient Differential Diagnosis using Cost-aware Active Testing. *International Journal of Prognostics and Health Management*, *15*(3), 1–11. https://doi.org/10.36001/ijphm.2024.v15i3.3849

van Gerwen, E., Barbini, L., & Nägele, T. (2022). Integrating System Failure Diagnostics Into Model-based System Engineering. *Insight*, *25*(4), 51–57. https://doi.org/10.1002/inst.12412

Voirin, J. L. (2017). Model-based system and architecture engineering with the Arcadia method. In *Model-based System and Architecture Engineering with the Arcadia Method*. https://doi.org/10.1016/c2016-0-00862-8