Root Cause Analysis in IT Infrastructures using Ontologies and Abduction in Markov Logic Networks☆

Joerg Schoenfisch[a,*], Christian Meilicke[a], Janno von Stülpnagel[b], Jens Ortmann[b], Heiner Stuckenschmidt[a]

[a] *Research Group Data and Web Science, University of Mannheim, Germany*
[b] *Softplant GmbH, Munich, Germany*

Abstract

Information systems play a crucial role in most of today's business operations. High availability and reliability of services and hardware, and, in the case of outages, short response times are essential. Thus, a high amount of tool support and automation in risk management is desirable to decrease downtime.

We propose a new approach for calculating the root cause for an observed failure in an IT infrastructure. Our approach is based on abduction in Markov Logic Networks. Abduction aims to find an explanation for a given observation in the light of some background knowledge. In failure diagnosis, the explanation corresponds to the root cause, the observation to the failure of a component, and the background knowledge to the dependency graph extended by potential risks. We apply a method to extend a Markov Logic Network in order to conduct abductive reasoning, which is not naturally supported in this formalism.

Our approach exhibits a high amount of reusability and facilitates modeling

☆This work has been partially supported by the German Federal Ministry of Economics and Technology (BMWI) in the framework of the Central Innovation Program SME (Zentrales Innovationsprogramm Mittelstand - ZIM) within the project "Risk management tool for complex IT infrastructures".
The most notable extensions compared to the work presented in [1] is the incorporation of ontologies in the modeling step, and the presentation of scalability results on specifically generated datasets.

*Corresponding author
Email addresses: joerg@informatik.uni-mannheim.de (Joerg Schoenfisch), christian@informatik.uni-mannheim.de (Christian Meilicke), janno.stuelpnagel@softplant.de (Janno von Stülpnagel), jens.ortmann@softplant.de (Jens Ortmann), heiner@informatik.uni-mannheim.de (Heiner Stuckenschmidt)

by using ontologies as background knowledge. This enables users without specific knowledge of a concrete infrastructure to gain viable insights in the case of an incident. We implemented the method in a tool and illustrate its suitability for root cause analysis by applying it to a sample scenario and testing its scalability on randomly generated infrastructures.

Keywords: Root Cause Analysis, IT Infrastructure Management, Markov Logic Network, Ontology, Abductive Reasoning

---

1. Introduction

Root cause analysis (RCA) plays an important part in processes for problem solving in many different settings. Its purpose is to find the underlying source of the observed symptoms of a problem. IT plays an important role in processes in a wide area of business, thus a high availability and short response times to failures (e.g., failing e-mail deliveries, inaccessible websites, or unresponsive accounting systems) are crucial [2]. Today's IT infrastructures are getting increasingly complex with diverse direct and transitive dependencies. This makes root cause analysis a time intensive task as the cause for a problem might be unclear or the most probable cause might not be the most obvious one. Therefore, automating the process of root cause analysis and helping an IT administrator to identify the source of a failure or outage as fast as possible is important to achieve a high service level [3].

In this paper we present our approach to root cause analysis that uses Markov Logic Networks (MLN) and abductive reasoning to enable an engineer to drill down fast on the source of a problem. Markov Logic Networks provide a formalism that combines logical formulas (to describe dependencies) and probabilities (to express various possible risks) in a single representation. We focus on abductive reasoning in MLNs and show how it can be used for the purpose of root cause analysis. To our knowledge, the proposed approach is a novel method to root cause analysis that combines probabilistic and logical aspects in a well-founded framework.

Throughout the paper, we illustrate our approach on a small case study. The IT infrastructure in our settings is comprised of a multifunction office printer that offers – amongst others – printing and scanning services via a network. These services use a mail and indirectly an LDAP service. Everything is dependent on the network and the power supply. This small case study already has dependencies that cross several different levels of infrastructure (services, server hardware, network hardware, power supply). We will expand this setting with possible causes for failure and probabilities for their occurrence. Theses risks are described in the "IT-Grundschutz Catalogue" by the German "Bundesamt für Sicherheit in der Informationstechnik" (Federal Office for Information Security) which is based on the ISO 27001 certification[1]. Furthermore, we evaluate the scalability of the approach on infrastructures generated randomly based on the structure observed in real-world environments.

Within our framework, the IT infrastructure is represented as a logical dependency network that includes various threats to its components. When a problem occurs, available observations are entered into the system which then generates the Markov Logic Network from the available observations, the given dependency network, and the general background knowledge related to the components of the infrastructure. Some of these observations might be specified manually, while other observations can be entered into the system automatically, e.g. via constantly running monitoring software. These observations are typically incomplete in the sense that not all relevant components are monitored, or not all problems are recognized. Thus, taking the given observations into account, there might still be a set of several explanations for the problem that occurred. Under the assumption that the modeled dependency network captures all relations present in the infrastructure and all threats are adequately taken into account, the correct explanation will always be contained in that set.

We calculate, via abduction, the most probable cause for the current problem, which is then presented to the user, e.g., the administrator of the IT infras-

[1] `https://www.bsi.bund.de/EN/Topics/ITGrundschutz/itgrundschutz_node.html`

tructure. The user can then investigate if it is indeed the source of the problem. This might require to manually check the availability of some component or to analyze a log file. If the proposed explanation is correct, counter-measures can be introduced immediately. If the additional observations revealed that the calculated explanation is wrong, those new observations are entered into the system as additional evidence and a better explanation is computed. This iterative, dialog-based process is a practicable approach to quickly narrow down on a root cause.

In our approach, we represent the given infrastructure and the possible risks as ontology. This allows us to automatically infer that certain threats are relevant for certain infrastructure components, or add logical constraints ensuring consistency. Relevant background knowledge can easily be maintained and used to generate the Markov Logic Network. Moreover, our approach can take into account known probabilities of risks and failures. These probabilities are derived from expert judgment or statistical data. Instead of computing multiple candidate explanations, which is possible in purely logic based approaches, we are able to generate the most probable explanation with our approach, while still leveraging the full power of an expressive, declarative framework.

This paper is structured as follows. First, we present the theoretical underpinnings of our approach. In Section 2, we give a brief description to Markov Logic, introduce the general notion of abduction, and explain how abduction can be realized in the context of Markov Logic Networks. Furthermore, we give a short introduction to ontologies and their benefit in modeling IT infrastructures. In Section 3, we first present a typical scenario for root cause analysis. Then, we show how to model this scenario in our framework and describe how to apply abductive reasoning to find the most probable root cause. We present a workflow that illustrates how our approach is used in the context of a dialog-based process in Section 4. Furthermore, we conduct the evaluation of the scalability of the approach in Section 5. A tool we implemented to support the user in modeling the infrastructure and running a root cause analysis is presented in Section 6. In Section 7, we show how our approach is related to other works.

Finally, we discuss the drawbacks and benefits of our approach, and point out directions for future work in Section 8.

2. Theoretical Background

This section first describes First-Order Logic and Markov Logic Networks. Then, we explain abduction and its concrete implementation in the context of Markov Logic Networks.

2.1. First-Order Logic

First-order logic (FOL) is used do describe and reason in a domain of discourse. Syntactically, it consists of: *constants* $\mathcal{C} = \{c_1, ..., c_{|\mathcal{C}|}\}$, *variables* $\mathcal{V} = \{v_1, ..., v_{|\mathcal{V}|}\}$, *predicates* $\mathcal{R} = \{r_1, ..., r_{|\mathcal{R}|}\}$, *functions* $\mathcal{F} = \{f_1, ..., f_{|\mathcal{F}|}\}$, and *logical operators* $(\forall, \exists, \wedge, \vee, \rightarrow, \leftrightarrow, \neg, (,), \equiv)$. A *term* t can either be a variable v, a constant c, or a function of terms $f(t_i, ..., t_j)$. An *atomic formula* (or simply atom) is a formula that contains no logical connective, i.e. it consists of a single predicate. A general formula is created by connecting multiple atoms. If an atom contains no variables we call it a *ground atom*; if a formula contains no free variables (i.e. only constants and variables bound by $\forall$ or $\exists$), we call it a *ground formula*.

The semantics of a first-order logic is given by an *interpretation*. Intuitively, an interpretation assigns real-world objects present in the domain to the syntactic constructs described above. This way, every term is also assigned a truth value, e.g. an atomic formula is true if a relation as identified by the predicate exists (or can exist) between the specified constants and variables.

A small example FOL model is the following simple description of relations between persons and their hobbies:

$$\begin{gathered} \text{person}(\mathit{Alice}) \qquad \text{person}(\mathit{Bob}) \qquad \text{person}(\mathit{Eve}) \\ \text{friends}(\mathit{Alice}, \mathit{Bob}) \\ \text{hasHobby}(\mathit{Alice}, \mathit{Football}) \\ \text{friends}(x, y) \wedge \text{hasHobby}(x, z) \rightarrow \text{hasHobby}(y, z) \end{gathered} \tag{1}$$

The model contains the constants *Alice*, *Bob* and *Eve*, variables x, y and z, the predicates *friends* and *hasHobby* and the logical operators $\wedge$ and $\rightarrow$. It states that *Alice* and *Bob* are both persons and friends, *Eve* is a person, *Alice* has *Football* as a hobby, and that individuals who are friends have the same hobbies. From this model it can be inferred that *Bob* also has the hobby *Football*.

2.2. Markov Logic Networks

Markov Logic Networks (MLN) generalize first-order logic and probabilistic graphical models by allowing hard and soft first-order formulas [4]. Hard formulas are regular first-order formulas, which have to be fulfilled by every interpretation. An interpretation is also referred to as a possible world. Soft formulas have weights that support (in case of positive weights) or penalize (in case of negative weights) worlds in which they are satisfied. The probability of a possible world, one that satisfies all hard formulas, is proportional to the exponential sum of the weights of the soft formulas that are satisfied in that world. This corresponds to the common understanding of Markov Networks as log-linear probabilistic models [4].

MLNs are a template for constructing Markov Networks. A formula is called a grounded formula if all variables have been replaced by constants. Given a set of constants, a Markov Network can be generated from the MLN by computing all possible groundings of the given formulas. Due to the closed world assumption, the domain of interest consists of only those entities that are defined by specifying the set of constants. An atom is a formula that consists of a single predicate. A possible world corresponds to a set of ground atoms, which is usually a small subset of all possible groundings.

Furthermore, in some implementations for inference predicates can be defined as being either observed or hidden (unobserved) and get assigned some type. For an observed predicate, only explicitly stated groundings are allowed and considered to be true, whereas for hidden predicates, any grounding with the given constants can be generated. Typing allows to restrict the possible constants for grounding a predicate to subset. Both techniques are used to restrict

the number of possible groundings and thus the effort needed for inference.

Revisiting the previous example about persons and hobbies, if all predicates were untyped and hidden the following is an excerpt of a possible world:

$$\begin{aligned}
\text{person}(\textit{Alice}) \quad & \text{person}(\textit{Bob}) \\
\text{person}(\textit{Eve}) \quad & \text{person}(\textit{Football}) \\
\text{friends}(\textit{Alice}, \textit{Alice}) \quad & \text{friends}(\textit{Alice}, \textit{Bob}) \\
\text{friends}(\textit{Alice}, \textit{Eve}) \quad & \text{friends}(\textit{Alice}, \textit{Football}) \\
& \dots \\
\text{hasHobby}(\textit{Alice}, \textit{Football}) \quad & \text{hasHobby}(\textit{Alice}, \textit{Bob}) \\
& \dots
\end{aligned} \tag{2}$$

With no further restrictions, *Alice*, *Bob*, *Eve* and *Football* would all be persons, friends and have each other as hobby. Some of those groundings, like hasFriend(*Alice*, *Football*), do obviously not make sense. By restricting *person* and *friends* to be observed and typed to only persons, and *hasHobby* to be typed to *person* and the newly introduced predicate *hobby*, the same possible world is reduced to this:

$$\begin{aligned}
\text{person}(\textit{Alice}) \quad & \text{person}(\textit{Bob}) \\
\text{person}(\textit{Eve}) \quad & \text{hobby}(\textit{Football}) \\
\text{friends}(\textit{Alice}, \textit{Bob}) \quad & \text{hasHobby}(\textit{Alice}, \textit{Football}) \\
\text{hasHobby}(\textit{Bob}, \textit{Football}) \quad & \text{hasHobby}(\textit{Eve}, \textit{Football})
\end{aligned} \tag{3}$$

Eve still has the hobby *Football* despite having no friends, but the overall number of groundings is greatly reduced. By making $\textit{friends}(x, y) \wedge \textit{hasHobby}(x, z) \rightarrow \textit{hasHobby}(y, z)$ a soft rule with positive weight and adding the rule $\textit{hasHobby}(x, y)$ with a negative weight we can also discourage worlds with additional *hasHobby* groundings.

Formally, an MLN L is a set of pairs $\langle F_i, w_i \rangle$, where F_i is a first-order logic formula and w_i is a real numbered weight [4]. The MLN L, combined with a finite set of constants $\mathcal{C} = \{c_1, c_2, \dots, c_{|\mathcal{C}|}\}$, defines a ground Markov Network $M_{L,\mathcal{C}}$ as follows [4, p. 113]:

1. $M_{L,\mathcal{C}}$ has one binary node for each possible grounding of each predicate in L. The value of the node is 1 if the grounded atom is true and 0 otherwise.
2. $M_{L,\mathcal{C}}$ contains one feature for each possible grounding of each formula F_i in L. The value of this feature is 1 if the formula is true, and 0 otherwise. The weight of the feature is the w_i associated with F_i in L.

Generally, a feature can be any real-valued function of the variables of the network. In this paper we use binary features, essentially making the value of the function equal to the truth value of the grounded atom.

The description as a log-linear model leads to the following definition for the probability distribution over possible worlds x for the Markov Network $M_{L,\mathcal{C}}$:

$$P(X = x) = \frac{1}{Z} \exp\left(\sum_i w_i n_i(x)\right) \tag{4}$$

where Z is a normalization constant and $n_i(x)$ is the number of true groundings of F_i in x.

When describing the MLN we use the format $\langle \textit{first-order formula}, \textit{weight} \rangle$. Hard formulas have infinite weights. If the weight is $+\infty$ the formula must always be true, if the weight is $-\infty$ it must always be false. A soft formula with weight 0 has equal probabilities for being satisfied in a world or not.

There are two types of inference with Markov Logic: maximum a posteriori (MAP) inference and marginal inference. MAP inference finds the most probable world given some evidence. It does not compute probabilities of variables but the world with the highest overall probability for its variable assignment. Marginal inference computes the a posteriori probability distribution over the values of all variables given some evidence. In other words, it calculates the sum of the probabilities of all the worlds in which a given variable is true. Note that the single most probable event does not have to be included in the overall most probable world. We are interested in MAP inference, as we want to determine the world with the most probable explanation for a failure.

The following example shows a small, simple MLN with some constants and how MAP inference is conducted. First we define all predicates with their types (i.e. predicate(*type*)) – observed predicates are prefixed with an asterisk (*).

$$\begin{gathered} *\text{person}(person) \quad *\text{hobby}(hobby) \\ *\text{friends}(person, person) \\ \text{hasHobby}(person, hobby) \end{gathered} \tag{5}$$

The MLN consists of the following (soft) rules that were introduced before:

$$\langle \text{friends}(x, y) \wedge \text{hasHobby}(x, z) \rightarrow \text{hasHobby}(y, z),\ 1 \rangle \tag{6}$$

$$\langle \text{hasHobby}(x, y), -0.1 \rangle \tag{7}$$

We provide the network with these facts about persons, hobbies and the known relationships between those:

$$\begin{gathered} \text{person}(Alice) \qquad \text{person}(Bob) \\ \text{person}(Eve) \qquad \text{hobby}(Football) \\ \text{friends}(Alice, Bob) \\ \text{hasHobby}(Alice, Football) \end{gathered} \tag{8}$$

In the grounding step all possible combinations of constants are assigned to the hidden predicates (in this case only hasHobby) and it is checked whether the grounding satisfies all hard formulas. Note that not using any constant is also a valid grounding, resulting in an empty world. Those combinations of constants and assignments to hidden predicates that fulfill all hard formulas are the possible worlds. The weight of a possible world is calculated by counting how often a soft formula is fulfilled therein and multiplying the count with the formulas weight. Assignments that where already present are usually not counted (they would however be present in every possible world and thus only add a constant value to each). The four possible worlds and their weights are

shown below:

$$\left.\begin{array}{l}\text{hasHobby}(Bob, Football)\\ \text{hasHobby}(Eve, Football)\end{array}\right\} \quad \sum w_i = \quad 0.8 \tag{9}$$

$$\text{hasHobby}(Bob, Football) \quad \sum w_i = \quad 0.9 \tag{10}$$

$$\text{hasHobby}(Eve, Football) \quad \sum w_i = -0.1 \tag{11}$$

$$— \quad \sum w_i = \quad 0 \tag{12}$$

For example the first world (Formula 9) has a weight of 0.8, as it fulfills Formula 6 once and Formula 7 twice. The second world (Formula 10) fulfills the Formulas 6 and 7 once, resulting in a weight of 0.9. Formula 10 is also the result of MAP inference, as it has the heights weight of all possible worlds.

2.3. Abduction in Markov Logic Networks

Abductive reasoning – or simply *abduction* – is inference to the best explanation. It is applicable to a wide array of fields in which explanations need to be found for given observations, for example plan or intent recognition, medical diagnosis, criminology, or, as in our approach, root cause analysis. According to [5], abduction is usually defined as follows [6]:

Given: Background knowledge B and a set of observations O, both formulated in first-order logic with O being restricted to ground formulae.

Find: A hypothesis H, also a set of logical formulae, such that $B \cup H$ is consistent and $B \cup H \vdash O$.

In other words, find a set of assumptions (a hypothesis) that is consistent with the background knowledge and, combined with it, explains the observation. It is the opposite of deductive reasoning which infers effects from cause.

The relation between root cause analysis and abductive reasoning is rather straightforward. In our approach, the background knowledge is the dependency

network, respectively the Markov Logic Network to which we transform it. The dependency graph and Markov Logic Networks both are based on first-order logic as a formalism and thus conveniently are already in the desired logical representation. The observations, i.e., information about components being available or unavailable, are not part of the model but rather are directly provided as evidence to the MLN. We then try to prove through abduction that a specific threat – the most plausible cause – has occurred.

The inference mechanism in Markov Logic Networks is by default deductive, not abductive. Deductive reasoning draws new, logically sound conclusions from given statements. Kate et al. and Singla et al. [5, 7] proposed methods – Pairwise Constraint (PC) and Hidden Cause (HC) model – that adapt Markov Logic Networks to automatically perform probabilistic abductive reasoning through its standard deductive reasoning mechanism. Their method augments the clauses of the MLN to support abductive reasoning as defined above. In general, the methods first introduce a reverse implication for every logical implication already present in the network. For example, if there are formulas $p_1 \rightarrow q$, ..., $p_n \rightarrow q$ in the MLN, the formula $q \rightarrow p_1 \vee \ldots \vee p_n$ is added to the MLN.

In a second step the model is extended with mutual exclusivity constraints that bias the inference against choosing multiple explanations. The reverse implications and the mutual exclusivity clauses are modeled as soft rules and may occasionally be violated, for example, if multiple explanations provide a better proof for the hypothetical root cause than a single explanation. We follow this basic idea, however, we argue that the mutual exclusivity constraints are not required in the application that we are interested in.

Revisiting the example above, we have to add the following reverse implication to conduct abductive reasoning:

$$\text{hasHobby}(y, z) \rightarrow \text{friends}(x, y) \wedge \text{hasHobby}(x, z) \tag{13}$$

This implications ensures that any additional grounding from hasHobby(y, z) has some corresponding atoms friends(x, y) and hasHobby(x, z) or is forbidden otherwise.

2.4. Knowledge Representation with Ontologies

An ontology can be understood as a logical representation of a domain model. The advantages of such domain models include enabling the sharing of knowledge, the re-use of knowledge, and the better engineering of knowledge-based systems with respect to acquisition, verification and maintenance [8]. Especially for large IT infrastructures all of these task are highly relevant, because an IT infrastructure is often not designed from scratch but has evolved in a dynamic way from legacy IT systems or has been iteratively extended to satisfy growing requirements. Moreover, the knowledge about the whole infrastructure is often distributed across several departments and employees.

There are several logical languages that can be used to define ontologies. One of the most important formalisms is the family of description logics [9] (DL), which is based on a well-defined model-theoretic semantics. The core reasoning problems for DL languages are (usually) decidable, and efficient decision procedures have been designed. A logical formalization that is built on top of a well-defined semantics has several advantages. One of these advantages is the possibility to exploit reasoning capabilities. Reasoning can be used to detect inconsistencies in the model of the IT infrastructure. This helps both the task of acquisition and maintenance by automatically detecting potential mistakes. As shown in [10] it is possible to automatically convert an ontological representation of a basic DL dialect into a set of (weighted) first-order formulas that form the building blocks of a Markov Logic formalization. We will use a similar approach to create a Markov Logic Network from an ontological representation of an IT infrastructure.

The idea to use ontologies for modeling IT infrastructures has been proposed for several reasons. Vom Brocke at el. [11] propose the use of ontologies to model the relationship between IT resources and business processes for the purpose of measuring the business value of IT. Ekelhart et al. [12] provide a security ontology to support small and medium sized businesses IT-security risk analysis. Within our work we are distinguishing between acquisition, verification and maintenance on top of an ontological representation, and the probabilistic rea-

soning using Markov Logic Networks. This means that our work is compatible with the previously mentioned proposals, while we are able to conduct probabilistic reasoning required for root cause analysis. Thus, we can leverage the task-specific benefits of both formalisms.

3. Root Cause Analysis with Markov Logic Networks

Root cause analysis is the task of finding the underlying cause of an event. It is often applied to analyze system failures. System failures are commonly caused by a cascade of events. The goal of a root cause analysis is finding the original reason for the failure, so that a sustainable solution can be provided [13]. Root cause analysis typically comprises two phases: the detection of an event and the diagnosis of the event. In our work, we are concerned with the second phase and assume that a failure has already been detected.

In this section, we first illustrate the infrastructure of our case study. Then we show how to model dependencies and risks as a set of first-order formulas. While this model is the core component for computing the MAP state, which corresponds to the root cause, we also leverage an ontological model to describe and maintain the infrastructure, which is then used to automatically construct some of the relevant dependency and risk assertions as first-order formulas. Then, we explain how we implemented abduction in our Markov Logic Network and show special properties of our settings which simplify the general approach of abductive reasoning. Finally, we explain how the method is integrated in an iterative process.

3.1. Scenario Setting

In the subsequent sections, we discuss our approach with the help of an infrastructure shown partially in Figure 1. This small sample revolving around an office multifunction printer consists of the following components:

- The basic dependency for all components is the *Power Supply*. The only risk that can affect it is a general outage.

- The *Network Switch* connects the other components. It only depends on the power supply; it has multiple risks, e.g. congestion, overheating, or denial-of-service attack, not explicitly depicted Figure 1.

- The two servers *mail.uni-ma* and *cas.uni-ma* each offer one service, i.e. the *Mail Service* and an *LDAP authentication service*. The Mail Service uses the LDAP service to authenticate users. Both servers are subject to various threats, e.g. malicious software, DOS attacks, overloading, or compromise of the system.

- The *Office Printer* offers three services: *Copying*, *Printing*, and *Scanning*. It also has various problem sources, e.g. lack of resources or a technical malfunction.

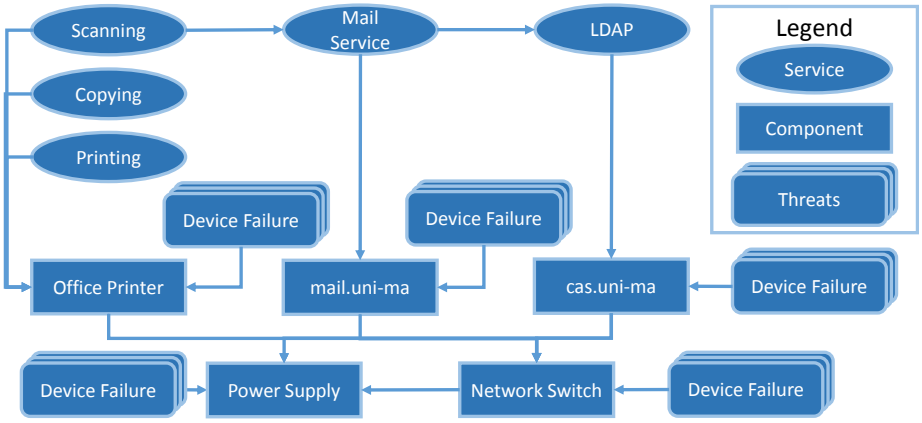

Figure 1: Case Study: Office multifunction printer with multiple risks/threats attached (for brevity risks are grouped as *Device Failure*). In this small example we do not consider redundant components, i.e. all edges represent *specificallyDependsOn* relations.

The threats we are using in our example are defined in the *IT-Grundschutz Catalogues* [14, p. 417ff.]:

- *Disruption of power supply*: Short disruption of the power supply, more than 10 ms, or voltage spikes can damage IT devices or produce failures in its operation.

- *Failure of Devices or Systems*: No equipment runs infinitely and a hardware failure in an IT device will happen if it runs long enough. Beyond the damage of the device, the downtime has an effect on the processes that depend on the device or can even damage other devices, e.g. in the case of a cooling system.

- *Systematic trying-out of passwords*: An attacker can gain access to a system by discovering the password of the system through systematic trial-and-error.

- *Lack of Resources*: If the given resources (for example bandwidth, disk space or personnel) in an area of the operation are smaller than the current demand, a bottleneck occurs. This results in congestion and failure of operation.

- *Malicious software*: Malicious software tries to execute a process that is unwanted or damaging for the owner of the device that runs the software. This includes viruses, worms and Trojan horses.

- *Misuse of spanning tree*: An attacker can use Bridge Protocol Data Units (BPDUs) to initialize the recalculation of the switch topology. This can be used to disrupt the availability of the network.

The IT-Grundschutz Catalogues are a comprehensive collection of threats and safeguards for various parts of an IT infrastructure[2]. They are created and maintained by the German Federal Office for Information Security[3], and compatible to the ISO 27001 certification[4].

3.2. Modeling Dependencies and Risks

The foundation of our root cause analysis is the dependency model. It uses first-order logic to describe various aspects of the IT infrastructure. Our basic

[2] `https://www.bsi.bund.de/EN/Topics/ITGrundschutz/itgrundschutz_node.html`
[3] Bundesamt für Sicherheit in der Informationstechnik (BSI)
[4] `http://www.iso.org/iso/home/standards/management-standards/iso27001.htm`

model uses five predicates:

- **specificallyDependsOn(*x,y*)** specifies that component x is specifically dependent on component y, e.g. the mail service that runs on the mail server. This predicate does not allow for any redundancy of y.
- **genericallyDependsOn(*x,y*)** specifies that component x depends on y. y may be replaced by some other redundant component. An example is a server running on the normal power supply vs. some uninterruptible power source (UPS).
- **redundancy(*x,y*)** states that x and y are redundant, i.e. they offer the same services and can replace each other in the case of failure.
- **affectedByRisk(*x,y*)** assigns the risk y to component x, i.e. y is a threat that endangers the functionality of a component and it can affect x.
- **unavailable(*x*)** designates a component x as unavailable, e.g. offline or not functioning properly.

The distinction between dependencies that allow for redundancy and those that do not helps to improve reasoning performance. For specific dependencies, no further checks for redundancies not to be performed in case of a failure and all dependent components are directly set to be unavailable.

The shown predicates are only one way of modeling the infrastructure; other predicates are possible. For example, a simple modification are additional types of dependencies that distinguish between, e.g., power and network connections, or technical and human errors. This allows for additional constraints to ensure the consistency of the model, for example by requiring that each component has at least one power and one network connection. This change can directly be made and later checked in the ontology that specifies the dependency graph. The subsequent translation to MLN does not need to be adapted. Another, more complex example is to model relationships as individual nodes, which enables the user to specify more details about it, e.g., the probability of a

broken network cable. This also requires changes in the rules of the MLN to account for the additional relationship nodes. However this only needs to be done once, when deciding to model the infrastructure in this way. It is possible to mix both modeling approaches, for example to include detailed information about relationships where available, and ease modeling for the user where it is not. We chose the predicates as presented for reasons of brevity and easier understanding.

Formulae 14a to 14f depict the basic MLN program built from those predicates:

$$\langle \forall x, y\ (\text{unavailable}(y) \wedge \text{specificallyDependsOn}(x, y) \Rightarrow \text{unavailable}(x)), \infty \rangle \tag{14a}$$

$$\langle \forall x, y\ (\text{unavailable}(y) \wedge \text{genericallyDependsOn}(x, y) \wedge \neg\exists z\ (\text{redundancy}(y, z) \wedge \neg\text{unavailable}(z)) \Rightarrow \text{unavailable}(x)), \infty \rangle \tag{14b}$$

$$\langle \forall x, y\ (\text{redundancy}(x, y) \Rightarrow \text{redundancy}(x, y)), \infty \rangle \tag{14c}$$

$$\langle \forall x, y\ (\text{redundancy}(x, y) \wedge \text{redundancy}(y, z) \Rightarrow \text{redundancy}(x, z)), \infty \rangle \tag{14d}$$

$$\langle \forall x, y\ (\text{affectedByRisk}(x, y) \Rightarrow \text{unavailable}(x)), \infty \rangle \tag{14e}$$

$$\langle \forall x, y\ \neg(\text{specificallyDependsOn}(x, y) \wedge \text{genericallyDependsOn}(x, y)), \infty \rangle \tag{14f}$$

Formula 14a forbids any world where infrastructure component y is unavailable and infrastructure component x is available, if there is a specific dependency from x to y. Formula 14b is similar to Formula 14a, but phrased for generic dependencies with redundancies. Provided x is generically dependent on y and y is unavailable, then x is unavailable only if there is no other component z that is redundant to y and available. Thus, a component is only available if every specific dependency is available or if at least one redundant component is available for each generic dependency, respectively. The symmetry and transitivity of *redundancy* is modeled by Formulae 14c and 14d. By adding these two formulas,

we ensure that it is not required to specify redundancy for all pairs in both directions. If we extend an infrastructure with an additional redundant component, we only need to add a single statement instead of specifying the information for all pairs in the group of redundant components. Formula 14e enforces that a component x that is affected by the effects of a risk y becomes unavailable. The predicates *specificallyDependsOn(x, y)* and *genericallyDependsOn(x, y)* are mutually exclusive (Formula 14f).

The known dependencies, risks, and unavailabilities are modeled as evidence as shown below. Note that these formulas are only two examples for all formulas required to describe the infrastructure depicted in Figure 1.

$$\langle \text{specificallyDependsOn}(\mathit{MailService}, \mathit{mail.uni\text{-}ma}), \infty \rangle \tag{15a}$$

$$\langle \text{affectedByRisk}(\mathit{mail.uni\text{-}ma}, \mathit{MaliciousSoftware}), -1.2 \rangle \tag{15b}$$

$$\langle \text{affectedByRisk}(\mathit{mail.uni\text{-}ma}, \mathit{DDOS}), -3.2 \rangle \tag{15c}$$

Formula 15a is a hard fact, which states that the *MailService* depends on the server *mail.uni-ma*. The soft Formula 15b encodes that *mail.uni-ma* can be affected by *MaliciousSoftware*. This formula has a negative weight, i.e. it translates to a low probability. *mail.uni-ma* also has *DDOS* as a second risk (Formula 15c). Generally, there is no upper limit to the number of risks that can be attached to a component

As described before, the dependency relation must hold in every possible world. The soft formula, however, is not fulfilled in most of the worlds due to the negative weight. In fact, if only this evidence is given, the most probable world does not include it, as it lowers the sum of the weights of all formulas.

Achieving high availability, defined as up to 5 minutes unavailability per year, is a longstanding goal in the IT industry [15]. Continuous monitoring of availability is part of IT service management best practices like the Information Technology Infrastructure Library (ITIL) [16]. We define availability as the probability that a system is reachable and working properly. The availability of

a system can be determined as follows:

$$Availability = \frac{Uptime}{Uptime + Downtime} \tag{16}$$

Unavailability is the inverse of availability:

$$Unavailability = 1 - Availability \tag{17}$$

The weights for new threats need to be estimated. We propose involving a domain expert in the estimation of how much the availability of the directly affected infrastructure component should be reduced. This allows us to learn a weight for the new threat and to determine how the new threat indirectly affects the availability of other components.

Determining the correct weight for the evidence is not trivial [17]. However, there exist efficient learning algorithms for MLNs [4]. Those algorithms can either work on collected data, for example from monitoring systems that provide uptime and downtime statistics, or based on estimations from domain experts or vendor specifications. Additionally, approximating the correct weights is sufficient in our use case, as we are not interested in the absolute probability of a specific root cause occurring, but just which root cause is most likely given some evidence.

3.3. Infrastructure Components and Background Knowledge

Our basic dependency model contains relatively simple first-order rules. The dependencies within this basic model are ignorant with respect to the types of the entities that are linked. However, we know that an IT infrastructure is typically a network built from different types of entities. In particular, the dependencies between these entities are restricted with respect to their types. We know, for example, that each server must depend on a power supply, while it makes no sense to have an explicit dependency between a service and a power supply. This dependency is indirectly modeled by the fact that a service must run on a server that depends on a power supply. In our approach, we propose the use of a Description Logics (DL) ontology to model the types of and the

relationships between the components of the IT infrastructure. The formulas of a DL ontology are divided into T-Box axioms and A-Box assertions (see also [18]). The examples given above will be encoded as axioms of the T-Box. A T-Box contains terminological axioms that describe the relations and types that are used (later) in the A-Box to make concrete assertions.

With respect to the A-Box, we have developed a graphical user interface to specify and visualize the concrete infrastructure. It is used to add components to the model of the infrastructure that are typed in terms of the T-Box vocabulary. The user interface, presented in more details in 6, is also used to specify the observations and to compute a root cause whenever a root cause analysis is required. In order to define the T-Box, we have used the ontology editor Protégé to model the T-Box axioms [19]. Protege helps to abstract from the concrete encoding of the axioms and supports views that are also common to users that have only a limited experience in logical modeling.

We use the T-Box to distinguish between the different types of infrastructure components, e.g., `Service`, `Server`, `Switch`, or `PowerSupply`. For these types we add axioms to specify both required and impossible dependencies. For example, we enforce that each service depends (specifically or generically) on a server, while we do not allow a direct dependency between service and power supply. These constraints can be used to check the consistency of an A-Box that uses these axioms. Our user interface can use these reasoning services on the fly to check after each modification whether the resulting ontology is still consistent. This helps to detect both errors and missing dependencies during the knowledge engineering process.

Furthermore, we can use the T-Box to specify concrete subtypes for each of the main types. An example might be the distinction between *FlashMemory*, *OpticalDisc*, *MagneticDisk*, and *MagneticTape* as sub types of *StorageComponent*. *FlashMemory* can again be divided into *FlashDrive*, *MemoryCard* and *SolidStateDrive*, for example. Note that such a fine-grained distinction is not required by our approach. The basic dependency model and our tool for defining the infrastructure works already with very basic types. In the simplest case

we specify only one type called *Component*. However, a fine-grained typology has several advantages. We mentioned already the reasoning capabilities in the paragraph above. Another advantage is the specification of type specific risks and their generic probabilities. For example, we can add information about the failure rate (in the form of a weight) of a specific hard drive model as background knowledge as follows:

$$\langle(\text{SCSIHardDrive}(x) \sqsubseteq \exists \text{affectedByRisk.HeadCrash}), 0.0015\rangle \tag{18}$$

The hard drive model *SCSIHardDrive* is described as hard drive that has a certain risk of a head crash. The probability attached to this formula might have been derived from available failure rates. Note that we can specify directly a probability that will be translated to the corresponding weight in Markov Logic. If required, we can also use the ontology to add further types related to, e.g., the manufacturer of the drive, since it might be known that drives produced by a certain company have a lower failure rate. By defining a concrete drive as instance of this type, it inherits all the properties of this type, i.e. the weighted risk of a head crash. This helps the knowledge engineer to define the components of an infrastructure without explicitly specifying each risk and its probability explicitly.

The ontological representation is automatically translated to the first-order Markov Logic formalization on the fly whenever a root cause analysis is computed. Niepert et al. [10] have shown that such translation is possible in general. For our purpose we have chosen a similar approach, however, type assertions and T-Box axioms are not directly translated but are taken into account when associating risks with their weights to concrete components of the infrastructure.

3.4. Computing Explanations

We now detail our approach and describe how the Markov Logic Network is constructed and extended, and how we use abductive reasoning for root cause analysis. The construction from background knowledge and extension for abduction of the Markov Logic Network is only done once and does not have to be

changed during the root cause analysis. According to the method proposed in [5] we have to add one reverse implication for the Formulae 14a, 14b, and 14e:

$$\begin{aligned}
&\forall x\ (\text{unavailable}(x) \\
&\Rightarrow (\exists y\ (\text{specificallyDependsOn}(x, y) \wedge \text{unavailable}(y))) \vee \\
&\quad (\exists y\ (\text{genericallyDependsOn}(x, y) \wedge \text{unavailable}(y) \\
&\qquad \wedge\ \neg\exists z\ (\text{redundancy}(y, z) \wedge \neg\text{unavailable}(z)))) \vee \\
&\quad (\exists y\ (\text{affectedByRisk}(x, y)))
\end{aligned} \tag{19}$$

Additionally, Kate et al.s' method requires clauses for mutual exclusivity to be added. The purpose of these clauses is to *"explain away"* multiple causes for an observation and prefer a single one [20]. The reverse implications as well as the mutual exclusivity clauses are usually modeled as soft clauses. In general, for each set of reverse implications P_i with the same left-hand side, $(\frac{|P_i|^2+|P_i|}{2}) \in O(n^2)$ mutual exclusivity clauses are added.

However, different from networks in that general method, our approach exhibits a property that simplifies the additional rules needed for abduction: All the weights in the evidence are negative – based on the reasonable assumption that threats and risks only occur rarely, i.e. components are available more than 50% of the time. This property allows us to reduce the size of the Markov Logic Network by leaving out the mutual exclusivity clauses completely: Due to the reverse implication, the MLN solver has to chose one cause to make the clause *true*. However, as all causes have negative weights and thus every cause set to true is lowering the sum of the weights of a possible world, the solver is already biased against choosing multiple explanations. This saves us from generating the quadratic number of mutual exclusivity clauses.

After constructing and extending the Markov Logic Network, we can conduct the root cause analysis. The overall process flow of our approach is depicted in Figure 2. The analysis is a dialog-based and iterative process, with interaction between our system and an administrative user. A fully automatic workflow is desirable, however, not every information can be retrieved directly and sometimes manual investigation of log files or on the status of components

is necessary.

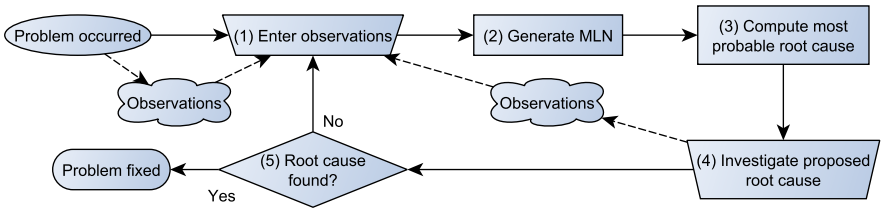

Figure 2: Process flow for our approach on root cause analysis. Rectangles denote automatic action. Trapezoids require manual interaction by an administrative user. Clouds represents observations made and entered by a user.

In its normal state, without any hard evidence about availabilities or unavailabilities, all components are assumed to be available. Thus, when calculating the MAP state, it contains all components as available. When a problem occurs the user is required to provide observations as evidence for the MLN (Fig. 2: Step 1). These observations include any certain information about available and unavailable components. At least one unavailability must be specified to run the root cause analysis, however, providing more information is possible and will increase the accuracy of the analysis. For example, the user can enter that printing (over the network) is not possible, although the network is functional as browsing the internet still works. This results in hard evidence for the printing service being unavailable and network services and hardware required for internet access being available. The presented model only supports hard evidence about available and unavailable components. It is possible to extend this to also allow soft evidence – e.g. when availabilities are checked automatically and there is some probability for error – handled similarly to information about threats. However, this will also have some impact on the performance of the approach, as unavailabilities are not just promoted through the dependency graph, but also influence the weights when calculating the MAP state.

Our approach extends the Markov Logic Network with the new evidence (Fig. 2: Step 2) and uses an MLN solver to run MAP inference on it (Fig. 2:

Step 3). The calculated MAP state contains the evidence provided by the user (this must be always fulfilled), components being unavailable due to a direct or indirect dependency on components observed as not available, and (at least) one root cause that explains the unavailabilities. Components which are not affected by specified observations or the calculated root cause are listed as available. The root cause fulfills the following properties:

- It explains all unavailabilities in the evidence. This is the case due to the additional reverse implications.
- It is not affecting any component stated as available in the evidence. Otherwise a hard rule would be violated.
- It is the most probable cause for all the observations given as evidence and the risk probabilities specified as weights.

We make the assumption that all causes are unlikely (they appear less than 50% of the time). Thus, their weights are negative. As the objective of the MAP state is maximizing the sum of all weights, only the most likely cause that explains all observations is included. A less likely cause has a higher negative weight, causing the sum of the weights to be lower than optimal, and thus getting rejected.

Note that due to the soft formulas used for abduction, our approach only encourages to calculate a single root cause, but does not enforce it. It only presents multiple possible root causes, if the sum of their weights is less than the weight of a single possible cause. If there are two possible root causes with the same weight, only one is presented at random.

The user then has to investigate the presented root cause (Fig. 2: Step 4). If it is the source of the observed problem, the analysis is finished and the cause can be fixed. Otherwise the process starts over from the start where the user enters additional observations (Fig. 2: Step 5). Those new observations can either be gathered while investigating the proposed root cause, or, for example, the user can verify the state of components that should also be affected by this

cause.

In contrast to an approach based for example on Bayesian networks the proposed root cause of the MAP state does not have a probability, but rather a probability is assigned to a state of the infrastructure (possible world). Subsequently, it is also not necessarily the cause with the highest overall probability, but it is the one best explaining all observations. This case most often occurs if an outage is caused by more than one root cause.

3.5. Limitations

Limiting factors for our approach is the expert knowledge required to build the background knowledge, and the worst-case performance of inference. Modeling the background knowledge requires good domain knowledge and experience in modeling ontologies. Furthermore, the model needs to be sufficiently complete to gain valuable results from the root cause analysis, as the approach is very fragile in the case of modeling errors. For example, if some critical, yet subtle, dependency is missed, the analysis might never provide any meaningful results. Many modeling errors can be checked with logical rules (e.g. every hardware component must be connected to a power source), but paying attention to capturing the infrastructure correctly and completely is of high importance. This factor is less relevant in companies that already use semantic technologies for IT infrastructure management, for example as presented in [21, 22]. This data can directly be used, mostly likely with only minor adjustments.

Another limitation is the worst-case complexity of MAP inference, which is NP-complete. However, there are efficient approximation algorithms for MAP inference (cf. [23] for a survey on different approaches), and as shown in Section 5 this is rarely an issue in realistic infrastructures.

4. Exemplary Scenarios

The following section describes the application of our approach to three different scenarios. Those scenarios represent incidents that occurred in our infrastructure, which we then analyzed in hindsight with the presented approach.

4.1. Scenario Analysis

The following three scenarios illustrate failures that occurred in our IT infrastructure. Together with our system administrators, we modeled our infrastructure, analyzed these scenarios in hindsight, and tested the usefulness of our approach in retrospective. We used RockIt [24], a highly optimized and scalable MLN solver, to compute the MAP state.

4.1.1. Scenario 1

The first scenario is the one depicted in Figure 1, revolving around the malfunction of our office multifunction printer. The printer offers three services: copying, printing via the network, and scanning to PDF which is then sent to an email address. A user reported the printer being broken, as scanning to PDF no longer worked. To check the proper functioning of the device, the administrator sent a print job and did a photocopy. Both tests worked successfully. Sending a test mail from his own account, the administrator also found the mail service working correctly. Further investigation finally revealed that the root cause of the scanning problem was a suspension of the account the printer used for the LDAP authentication. However, this cause was only considered after several discussions with two expert administrators involved.

We applied our approach to this scenario. The MLN was constructed automatically from the background knowledge that we maintained as a set of first-order formulas. We enter the observations *available(PrintService)*, *available(CopyService)*, and $\neg$*available(ScanService)* and computed the most probable root cause. The MAP state that was generated as solution contained the root cause *affectedByRisk(cas.uni-ma, Systematic trying-out of passwords)*. While we could not definitely decide, in retrospective, if this risk was the underlying reason for the failure of the server *cas.uni-ma*, an authentication problem related to *cas.uni-ma* was definitely the cause for the problem.

4.1.2. Scenario 2

The second scenario is an outage of our internal Subversion server. It involves more components than the previous scenario and benefits from the iterative approach. The Subversion server is hosted on a virtual machine that is running on a blade server. Subversion was responding slowly and took long time for many operations. Neither Subversion nor other processes on the virtual machine showed considerable resource utilization. Investigating resource usage on the blade server first did not reveal any abnormality. Later, a user discovered that our external website behaved similarly in performance as the SVN. This observation was first attributed to a slow Internet connection in general, but we then discovered that the web server, which was hosted in a different VM but on the same blade server, produced very high network traffic, starving all other services. A member of our group had released a data set of several gigabytes in size, that was downloaded a few hundred times concurrently. That lead to congestion on the network interface of the server. Moving the download to another physical server resolved the problem and the behavior of the Subversion server and our website went back to normal.

Analyzing this scenario with our approach, first, we only entered the observation of the unavailability of the SVN service: $\neg$*available(Service_SVN)*. The computed MAP state proposed *affectedByRisk(VM_Subversion, Overload)* as root cause. After ruling out this cause by adding the observations *available(VM_Subversion)* and $\neg$*available(Service_WebHosting)*, the result of the computation was *affectedByRisk(NetworkInterface_BladeServer, Congestion)* as root cause. This risk has a high probability for that server which is running various other virtual machines, all hosting services sensitive to a high network load. The lack of other resources, e.g. CPU or RAM, is modeled as less probable, because all those services are usually not very computational complex or requiring lots of memory. For this scenario, our approach proposed reasonable root causes which we retrospectively could verify as the reason for the outage. The manual handling of the incident involved more guesswork by the system administrators

and was long winded.

4.1.3. Scenario 3

The third scenario is the most complex of the presented. A very simplified graph of the involved infrastructure is shown in Fig. 3. The infrastructure spans

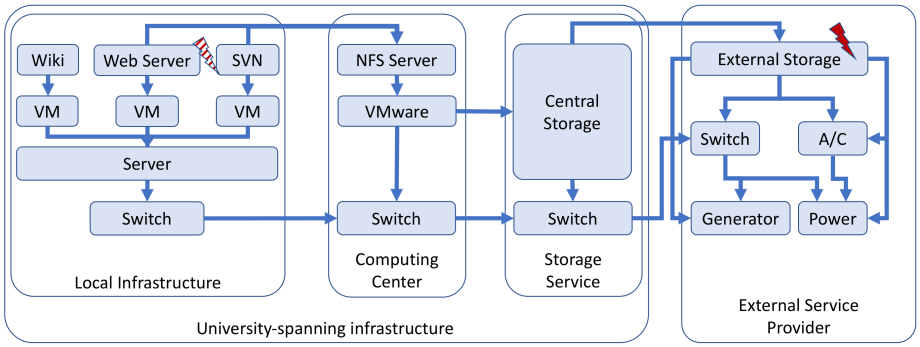

Figure 3: Very simplified depiction of the third scenario. It involves four infrastructures in different physical locations; three of those maintained by universities and one by an external service provider. Arrows indicate some dependency between the components. The red-shaded lightning marks the first observed error at the SVN server; the full-red lightning at the external storage marks the actual root cause.

four physical locations. We will only describe the parts which are relevant for the analyzed scenario at hand. The complete infrastructure is much more complex than can be shown here.

At our chair we directly host some services like an internal Subversion server (SVN) or a Wiki. They rely on a network file system (NFS) as storage, hosted at the computing center of the University of Mannheim, and deployed as one of many virtual machines in VMware. Most of the storage required by VMware is provided in turn by a central storage system located at the University of Heidelberg. Due to external cooperations, some of that central storage is also provided and hosted by an external company. To VMware, the different storage locations are presented as one homogeneous service.

In the analyzed scenario, the power at the external storage provider was switched off due to a short, planned maintenance. Servers and other hardware

critical for the services were running at backup power. The air conditioning required for a sustainable operation of the servers does not have a backup. This was known and should not have created problems during a short maintenance, thus no outage of the service was expected and no warning for the possibility of one given. However, maintenance on the power line took longer than expected and subsequently resulted in the storage system shutting down because of overheating. This caused the first error observed at our chair: the Subversion server being unresponsive.

The diagnosis of the root cause by our administrators was difficult for two main reasons: First because of the overall infrastructure being large and complex in general, and second because they were not aware of the additional storage from the external provider.

The ad-hoc, manual search for the cause of the outage started just locally at our chair. It roughly took the following course:

1. Checking whether the local network was up and the virtual machine of the SVN server running.

   In that course the administrators discovered that our web server was also down; both servers due to their storage not being able to mount.
2. Checking the network connection to the computing center of the University of Mannheim
3. Verifying that the NFS server was running.

   From this point on, more people needed to be involved as it was unclear why the storage could not be mounted albeit NFS running. Additionally, it was unknown to the administrative staff in Mannheim, that there is another external storage provider.
4. Checking the network connection to Heidelberg and inquiring about the central storage system.

   There it was uncovered that there is a problem with the external provider.
5. Investigating at the provider revealed the unforeseen maintenance problems as root cause of the outage.

In general a manual ad-hoc diagnosis involved a breadth-first-like search pattern,

and involvement of multiple people in all locations was required to identify the root cause. Without proper tool support that incorporates infrastructure and dependencies, the search was lengthy and undirected.

After modeling the scenario in our approach and searching for a root cause we found the following advantages over the ad-hoc search:

- Generally, our method favored to directly check services instead of hardware. This is due to services having a slightly higher rate of failure than hardware because of bugs, configuration errors, overload/denial of service, etc.

  This led to a slightly more directed search than the breadth-first pattern before.
- The unified that the modeled ontology provided made it easier to collaborate between the different locations, and it also improved the search by including information about known running services.
- When the maintenance was added as an additional threat, even with an average probability, our method very quickly narrowed it down as a potential root cause, as it explains the various outages in all university locations well. Note that still our local administrators need not have directly be aware of that maintenance being ongoing. Having this information added by staff in Heidelberg would be sufficient.

This scenario also shows well how our approach goes beyond the analysis of network failures, and also covers dependencies between e.g. services.

5. Evaluation of Scalability

We evaluated the scalability of the approach by automatically generating infrastructures of different sizes. The data generator was carefully modeled to inherit properties observed in real-world scenarios. We then simulated different numbers of root causes and observed offline components. We applied our approach to compute the root causes and measured the runtime.

5.1. Data Generation

As manually modeling infrastructures of various sizes to evaluate the scalability of the presented approach is not feasible – due to data not being publicly available and the effort required in modeling – we implemented a data generator that can create random infrastructures of any size. To create models which are close to real-world scenarios, we also discussed general properties of IT infrastructures with the experts when recording the scenarios described above, and include this knowledge in the design of the generator. We made the following observations which are reflected in the implementation:

The dependency graph of an IT infrastructure has usually a limited depth: the basic layers of an infrastructure are power sources, network components, server, and services. Power sources are located at the bottom layer without any dependencies to other components. Network components could technically be layered very deep, however instead of cascading several small switches and routers to connect many devices, generally a bigger switch is used, e.g. resulting in 16 million possible devices with a cascade of 64-port switches of depth 4. In our model servers rely on power sources and switches. Each server can offer services that depend on each other. Dependencies between services are naturally the most complex of the whole infrastructure. They can be layered over multiple services, e.g. starting from a low-level SAN service (storage area network) providing storage, over a virtualization using the SAN, a virtual server offering for example an LDAP service to the mail server using that for authentication. The dependency chain can become slightly longer by having another service providing email notifications and probably one or to more services offering even higher level operations, but in reality there is a limit to that depth. This complex example has a depth of 13; to provide some room for even more complex scenarios we imposed a limit of 16 to the maximum depth for the infrastructure the generator can create.

Another observation that already becomes obvious in this small example, is that there are some services that are very central, like the storage (SAN) or authentication via LDAP which are used by almost all other services, and

services that are used by few others, e.g. printing, which might only be used by persons (which are not part of the model). We factored this into the generation by drawing the possible dependencies from a normal distribution. For example, following from the *three-sigma rule of thumb* that nearly all values are within three standard deviations of the mean ($Pr(\mu - 3\sigma \leq x \leq \mu + 3\sigma) \approx 0.9973$), we draw from a normal distribution with $\mu = 500$ and $\sigma = \frac{500}{3}$ for an infrastructure of size 1000, or more general: for an infrastructure of size n we have $\mu = \frac{n}{2}$ and $\sigma = \frac{n}{6} \equiv \frac{\mu}{3}$. Additionally, the following constraints must hold:

1. Every network component and server is connected to exactly one power source.
2. Every network component and server is connected to one other network component or the root network component (e.g. representing some central switch or external internet connection).
3. Every service is running on exactly one server.
4. Every server is running some service (a server without offering a service is of no use).
5. A switch has $\approx$ 32–64 connections.
6. A server offers $\approx$ 1–3 service.
7. A service directly depends on $\approx$ 1–3 other services.
8. Every component has $\approx$ 1–6 risks attached to it.
9. Risks have a probability $0 < p < 0.5$, following a normal distribution with $\mu = 0.1$ and $\sigma = 0.05$.

Offline components were then defined by first choosing one or two root causes and determining all of their dependent components. Then for different datasets we picked one to five of those and marked them as offline in the model. This way calculating the root cause is more similar to real-world scenarios, where the component observed as offline is often not the real cause of the problem.

5.2. Scalability Results

We ran the approach 10 times for each size of the dataset and different amounts of errors. The average runtime and its standard deviation are shown

in Table 1. Having different numbers of root causes and observed offline components had no significant impact in the runtime, thus we omitted those datapoints and only report the numbers aggregated by size.

Dataset Size	Avg (sec)	StDev (sec)
1000	1.13	0.10
10 000	8.89	1.40
100 000	112.58	15.43

Table 1: Average runtimes on infrastructures of various sizes

The results for different sizes of infrastructures are not calculated in real-time. However, two minutes for the largest dataset is still a reasonable amount of time to wait for results.

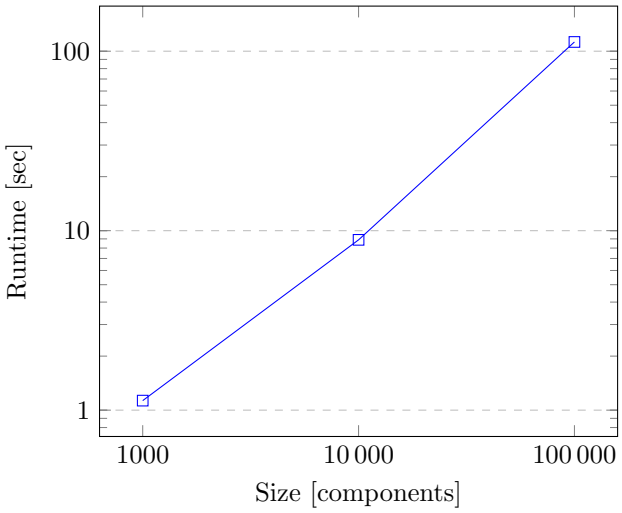

Figure 4: Runtimes on Infrastructures of various sizes

More importantly, the scalability of the approach is linear in the number of components which is a very favorable result, given the complexity of the calculation, as is apparent in Figure 4. This is mainly explained by the relatively simple structure of dependency graphs for IT infrastructures which are mostly tree-like, e.g. without circular dependencies, and do not exhibit the complex

patterns for which MAP inference is NP-complete.

6. Tool Support

We implemented our approach in an open-source tool called RoCA. A screenshot of the user interface showing the clipping of a small dependency graph and the observations provided by the user is presented in Figure 5. It shows a *CMS service* that is running on two redundant *Apache web servers*, the *Mail service* running on some *New server*, and a *Printer*. All of those depend on some not shown network connection and power source. A user provided evidence about the CMS not working (depicted in red), but the printer being functional (green) – for the other components the status is unknown. The graphical user interface vastly increases the usability of our method, as a user does not need knowledge about logical first-order formulas, Markov Logic Networks, or ontologies for entering evidence or running a root cause analysis. The tool and its source code

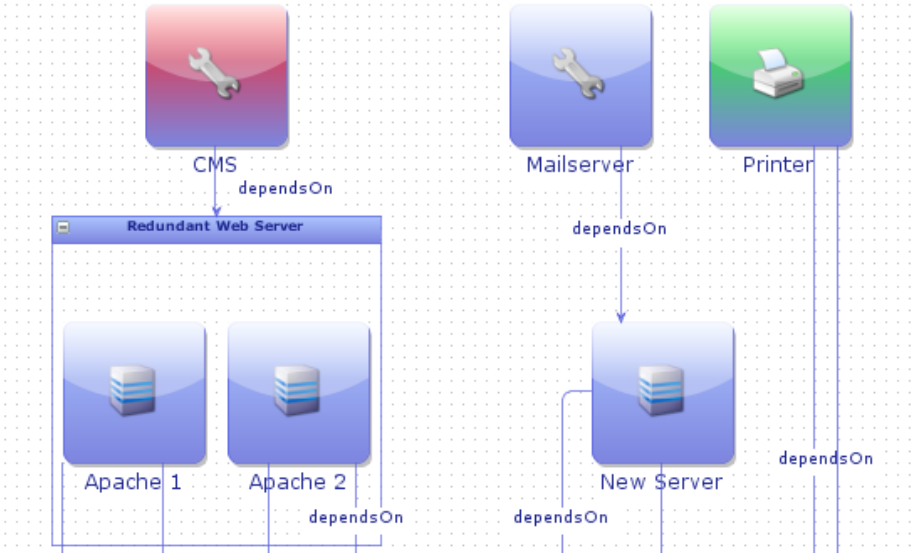

Figure 5: Screenshot of the visual representation of the dependency network and provided evidence (green and red) about the availability of components.

are freely available for download.[5]

6.1. Required Data

The required data to run a root cause analysis is the background knowledge and the dependency graph with evidence about available and unavailable components.

RoCA uses the ontological representation that we described above as background knowledge. Usually, the ontology is defined once and represents the vocabulary used to describe the IT infrastructure. Modeling it is a one-time effort and it requires only infrequent changes, e.g. when new types of components, like solid-state drives in recent years, emerge . In its T-Box, the ontology defines the available types of components and possible relations between those. It also defines the association between types and icons shown in the user interface of RoCA. Whenever a user adds a new component, he has to select one of the types defined in the ontology. Thus, RoCA is highly customizable and not tied to a predetermined vocabulary. This eases the integration with modeling styles that are already in use within the organization, e.g., the terminology of the configuration management databases. As described earlier, relations can be constraint to be only allowed between certain types of components, to enforce a consistent model. A very popular tool for designing and modeling ontologies is Protégé [19]. It provides customizable user interface to design and model ontologies. It also offers connections to reasoner for checking the consistency of the built ontology.

The dependency graph can be stored in the A-Box of the ontology. If no dependency graph is specified, the user is presented with an empty model and can create a new graph there. Extending a graph loaded from the A-Box is also possible.

[5] https://github.com/dwslab/RoCA

6.2. User Interaction

User interaction with RoCA occurs in two different scenarios: when modeling the infrastructure as dependency graph, and when conducting a root cause analysis after an incident occurred.

When modeling the user can choose from components and relations specified in the background knowledge. Components and relations between them can be arrange by drag&drop or an automatic layout algorithm. The user can freely assign names for each component and the a priori weight in a details dialog. A first-order logic reasoner, e.g. Pellet [25] or HermiT [26] can check the finished model for consistency. The model can be saved and also exported as graph.

When an incident occurs, the user can load the previously created model and enter observations about available (marked as green) and unavailable (red) components. There can be entered any number of observations. In our example screenshot (Fig. 5) we have marked the components *CMS* as active (green) and *Printer* as inactive (red), while we have not specified any information related to the component *Mailserver*. If there is no information about some component, it is simply left as unknown. Once these observations have been specified, the user can run the root cause analysis directly from the interface and the most probable root cause is presented. By inspecting the most probable root cause and all assumptions that are entailed, the user might agree on the proposed root cause or might specify additional observations as shown in the workflow shown of Figure 2.

7. Related Work

Related work can roughly be divided into two parts: Approaches also conducting root cause analysis, but using a different method; and approaches using probabilistic frameworks for abductive reasoning, yet not in the context of root cause analysis.

7.1. Root Cause Analysis

In previous work, failure diagnosis is conducted using correlation measures. A specific correlation measure for failure diagnosis is presented in [27]. The approach uses anomalies in the timing of program calls to trace the real root cause of an event. The anomalies are aggregated to give an anomaly score for each component. The scores are correlated within their architectural level to determine an anomaly ranking, which expresses the likelihood that a component is the root cause of a failure. A method for failure diagnosis using decision trees is proposed in [28]. The decision tree classifies the successful as well as failed requests. A correlation of paths in the decision tree with occurred failures indicates the node that represents the likely root cause.

In [29] an approach for requirements-driven root cause analysis for failures in software systems is proposed, wherein a Markov Logic Network is used as knowledge repository for diagnostic knowledge. The approach uses log data as observation information, the Markov Logic Network is used to deal with uncertainty stemming from incomplete log data. Their approach differs from ours in several points: they first model the background knowledge as goal trees and only convert it to first-order logic later; the evidence is solely generated from log data; and most importantly they use marginal inference, different to our approach which uses MAP inference. In [30] marginal inference was also used for the purpose of estimating unavailabilities in an IT infrastructure, where the authors referred to problems when marginal inference is applied to very low probabilities usually attached to the occurrence of risks in an IT setting. These problems are based on the use of sampling algorithms for performing marginal inference, as exact inference is infeasible Our approach is based on solving an optimization problem, which is not affected negatively by very small probabilities.

The Shrink tool [31] uses a Bayesian Network to model the diagnosis problem. It extends previous work on fault diagnosis with Bayesian Networks [32], by proposing a greedy inference algorithm with polynomial running time. Furthermore, Shrink is able to handle noise and small inaccuracies in the observations.

Heiden et al. [33] developed a system with a similar scope to ours. Their implementation uses Prolog or answer set programming. It is inferior in two points: it does not include any probabilities for threats, and it is unclear if it performs well with more than a few dozen possible threats defined in the model, or a too complex model overall.

Weidl et al. [34] present a methodology that uses object-oriented Bayesian networks to determine root causes in complex industrial processes. Although similar to our approach, its graphical model offers less reusability than the ontology we use, and some relations (e.g. symmetry relations for redundancy) are harder and cumbersome to model.

Liu et al. use a simple logical representation in the form of Petri nets [35]. Their method does not include uncertainty and it is unclear how well it scales to large infrastructures.

7.2. A Detailed Comparison

We have tried to apply the approaches referred to in the previous section to model and analyze the scenarios mentioned in Section 4. However, we had to retract from this task for several reasons. In most of the cases the cited literature described an approach with a focus on its theoretical foundations without offering a tool that implements the approach. In other cases it is crucial to understand that we would have to compare apples and oranges because the proposed techniques cannot be applied to solve the task that we try to solve with our tool. In the following we will describe these issues in detail.

As mentioned above Heiden et al. [33] propose an approach that is very close to ours in the sense that it is also based on the idea of abductive reasoning. As we already mentioned, the approach does not support probabilities for threats nor does it support probabilities for any another aspect. This means that in each situation where there are several possible root causes, the approach proposed by Heiden et al. will not find the most probable cause, but only one of the possible causes (or all of them). The probabilistic knowledge used in our approach cannot be expressed within the approach of Heiden et al. While the approach also

supports the dependencies we expressed in Formulas 14a to 14f, risk probabilities that we express in formulas as 15b and 15c, which can be derived from the statistical data gathered by monitoring systems (see the end of Section 3.2) or from background knowledge as described in Section 3.3), cannot be expressed and will thus not influence the computation of the root cause. However, such information is obviously crucial and will help to distinguish between several possible reasons for a failure. While the approach of Liu et al. [35] is based on a completely different formalism, it suffers from the same problems, namely its incapability to model probabilistic knowledge.

This is different when we look at the systems that are based on Bayesian Networks and their extensions. This comprises the works of Weidl et al. [34] as well as the approach implemented in the Shrink tool [31]. The main differences between our approach and theirs are based in the models used which correspond to a relational representation for our approach, in contrast to a propositional representation in Weidl's et al. In a relational representation, we can easily write down formulas like 14a to 14f, 18 or 19. These formulas are general formulas that use variable. This makes it rather convenient to express general dependencies. Even though the general formulas are finally grounded in order to compute a most probable root cause, on the representation level we describe an infrastructure and its dependencies with the help of relations and general formulas. This is much more complicated in an approach that is based on a propositional representation. Here we cannot write down a simple formula as Formula 14. Instead we have to add an explicit edge in the Bayesian Network, for each possible instantiation of the variables in the formula.

Another major difference is based in the directedness of Bayesian Networks. When trying to model an IT infrastructure and its potential root causes, the modeling approach in a directed graphical model will always be guided by potential errors that cause a problem somewhere in the infrastructure. That means that the development of the network will be guided by the attempt to model causal relationships. This is not the case in an undirected model that is based on the observations of correlations without making assumptions about causal

dependencies. In such a model it is much easier to integrate probabilities that are gathered by statistical observations.

7.3. Applications of Abductive Reasoning

In [7], Singla et al. extend the approach presented in [5] and use it in the context of plan and intent recognition. Instead of adding reverse implication, they introduce a hidden cause for all implications with the same left-hand side. In general, this reduces the size of the MLN and subsequently increases performance. However, as detailed above, for our approach the mutual exclusivity clauses are not needed anyway. Nonetheless, if more probable events have to be included in the evidence, their optimization can also be included in our approach.

Most other approaches to abductive reasoning either use first-order logic to calculate a minimal set of assumptions sufficient to explain the hypothesis [36, 37, 38, 39], or Bayesian Networks to compute the posterior probability of alternative explanations given the observations [20]. The former approaches are not able to estimate the likelihood of alternative explanations, as they do not support uncertainty in the background knowledge or evidence. Bayesian Networks, on the other hand, are designed to handle uncertainty. However, as they are propositional in nature, they cannot handle structured knowledge involving relations amongst multiple entities directly [5].

Bayesian Abductive Logic Programs (BALP) [40] are another approach that combines first-order logic and probabilistic graphical models. The main difference to MLNs is that BALPs are based on Bayesian Networks, which are directed. Undirected relations, like the symmetry of redundancy, are thus more complex to model. In [41], Inoue et al. describe a system that uses integer linear programming (ILP) for weighted abduction. They outperform a state-of-the-art abductive engine (Mini-TACITUS [42]). The MLN solver we use also transforms the problem internally to an ILP, which is one of the reasons for its good runtime performance.

8. Discussion and Conclusion

We presented our approach of applying abductive reasoning using Markov Logic Networks to compute the most probable root cause for a failure in an IT infrastructure. Our approach models the infrastructure with the help of ontologies. In particular, we formulated the dependencies of the network as hard formulas. Moreover, we added weighted soft formulas to model the probability of risks that might result in the failure of components and services. We defined these risks in accordance to the taxonomy of the IT Grundschutz Catalogues. Furthermore, we argued how the expressiveness of ontologies can be used to model general, reusable knowledge concerning risks and IT components. Our approach uses the same formalism for both knowledge presentation and abductive reasoning. Thus, all relevant information is readily available to compute the most probable root cause once an incident occurs. To the best of our knowledge, there exists no other approach that combines uncertainty and logical abductive reasoning to solve the problem of root cause analysis. We implemented our approach in RoCA, a tool providing a graphical user interface for modeling the infrastructure and conducting the root cause analysis.

We conducted an evaluation of our approach by analyzing two failures that happened in the infrastructure of our research group. In both cases we were able to determine a root cause (respectively, a sequence of probable root causes) that turned out to be helpful for a system administrator to resolve the problem. Our approach is especially useful when the reasons for the failure are not obvious to the administrator that is in charge of resolving the problem. Thus, our approach will be more beneficial in IT infrastructures, where competences are scattered over the members of different organizational units.

Furthermore, we analyzed the scalability of the approach for various sizes of randomly generated infrastructures, modeled after properties observed in real-world scenarios. The approach proofed to scale linearly in the size of the infrastructure up to hundreds of thousands of individual components.

So far, we have not taken all risks of the Grundschutz Catalogues into ac-

count. Instead, we have focused on a subset relevant for the infrastructure we modeled. To apply our approach to an arbitrary IT infrastructure, we have to create a complete translation of the catalogues to our logical representation.

Finally, we want to stress that the presented approach is not only suited for technical hardware scenarios, but can almost effortlessly be transferred to other settings. One example would be the search for an index case (patient zero) during the outbreak of a disease [43]. Relation between person are modeled similarly, however those will also have weights, representing the uncertainty that people actually know each other or where in contact during a time period. By noting when somebody showed the first symptoms and verifying that the person with the earliest symptoms can have had contact (directly or indirectly through others) with the other patients, the index case can be identified.

References

[1] J. Schoenfisch, J. von Stülpnagel, J. Ortmann, C. Meilicke, H. Stuckenschmidt, Root Cause Analysis through Abduction in Markov Logic Networks, in: Proceedings of the 20th International Enterprise Distributed Object Computing Conference (EDOC), 2016, pp. 1–13.

[2] IBM X-Force, Mid-Year Trend and Risk Report 2013, Tech. rep., IBM X-Force (2013).
URL http://www-03.ibm.com/security/xforce/downloads.html

[3] Ernst & Young, Managing IT risk in a fast-changing environment, Tech. rep., Ernst & Young (2013).

[4] M. Richardson, P. Domingos, Markov Logic Networks, Machine Learning 62 (1-2) (2006) 107–136. doi:10.1007/s10994-006-5833-1.
URL http://dx.doi.org/10.1007/s10994-006-5833-1

[5] R. J. Kate, R. J. Mooney, Probabilistic Abduction using Markov Logic Networks, IJCAI-09 Workshop on Plan, Activity, and Intent Recognition.

URL http://userweb.cs.utexas.edu/users/ml/papers/kate-pair09.pdf

[6] H. E. Pople, On the Mechanization of Abductive Logic., IJCAI (1973) 147–152.
URL http://pdf.aminer.org/000/366/748/two_sided_hypotheses_generation_for_abductive_analogical_reasoning.pdf

[7] P. Singla, R. J. Mooney, Abductive Markov Logic for Plan Recognition, AAAI (2011) 1069–1075.
URL http://www.aaai.org/ocs/index.php/AAAI/AAAI11/paper/viewPDFInterstitial/3615/3992

[8] D. Jones, T. Bench-Capon, P. Visser, Methodologies for ontology development, in: Proceedings of the 15th IFIP World Computer Congress, 1998, pp. 20–35. doi:10.1.1.52.2437.
URL http://www.springerlink.com/index/u3u53033q2508524.pdf%5Cnhttp://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.52.2437&rep=rep1&type=pdf

[9] F. Baader, W. Nutt, Basic Description Logics, in: The Description Logic Handbook: Theory, Implementation, and Applications, Cambridge University Press, 2003, pp. 43–95.

[10] M. Niepert, J. Noessner, H. Stuckenschmidt, Log-Linear Description Logics, in: IJCAI International Joint Conference on Artificial Intelligence, 2011, pp. 2153–2158. doi:10.5591/978-1-57735-516-8/IJCAI11-359.

[11] J. vom Brocke, A. M. Braccini, C. Sonnenberg, P. Spagnoletti, Living IT infrastructures - An ontology-based approach to aligning IT infrastructure capacity and business needs, International Journal of Accounting Information Systems 15 (3) (2014) 246–274. doi:10.1016/j.accinf.2013.10.004.

[12] A. Ekelhart, S. Fenz, M. D. Klemen, E. R. Weippl, Security Ontology : Simulating Threats to Corporate Assets, Proceedings of the 2nd Intl.

Conf. on Information Systems Security (ICISS) (2006) 249–259doi:http://dx.doi.org/10.1007/11961635_17.

[13] J. Rooney, L. Heuvel, Root cause analysis for beginners, Quality Progress 37 (7) (2004) 45–53.
URL https://servicelink.pinnacol.com/pinnacol_docs/lp/cdrom_web/safety/management/accident_investigation/Root_Cause.pdf

[14] Bundesamt für Sicherheit in der Informationstechnik, Threat Catalogues, IT Grundschutz Catalogues 15. EL (2016) 443 – 1338.
URL https://www.bsi.bund.de/EN/Topics/ITGrundschutz/Download/download_node.html

[15] J. Gray, D. P. Siewiorek, High-availability computer systems, Computer 24 (9) (1991) 39–48. doi:10.1109/2.84898.

[16] Cabinet Office, ITIL Service Design, TSO (The Stationery Office), 2011.

[17] D. Jain, B. Kirchlechner, M. Beetz, Extending Markov Logic to Model Probability Distributions in Relational Domains, in: J. Hertzberg, M. Beetz, R. Englert (Eds.), KI 2007: Advances in Artificial Intelligence, Vol. 4667 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2007, pp. 129–143. doi:10.1007/978-3-540-74565-5_12.
URL http://dx.doi.org/10.1007/978-3-540-74565-5_12http://link.springer.com/chapter/10.1007/978-3-540-74565-5_12

[18] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, P. F. Patel-Schneider, The Description Logic Handbook: Theory, Implementation, and Applications, Vol. 32, Cambridge University Press, 2003. doi:10.2277/0521781760.
URL http://portal.acm.org/citation.cfm?id=1215128

[19] M. A. Musen, The Protégé Project: A Look Back and a Look Forward, AI Matters 1 (4) (2015) 4–12. doi:10.1145/2757001.2757003.

[20] J. Pearl, Probabilistic Reasoning in Intelligent Systems, Morgan Kaufmann Publishers Inc., 1988.

[21] W. Chen, C. Hess, M. Langermeier, J. von Stuelpnagel, P. Diefenthaler, Semantic Enterprise Architecture Management, in: 15th International Conference on Enterprise Information Systems (ICEIS), 2013, p. 8.

[22] C. Hess, W. Chen, T. Syldatke, Business-oriented CAx integration with semantic technologies revisited, in: INFORMATIK 2010 - Service Science - Neue Perspektiven fur die Informatik, Beitrage der 40. Jahrestagung der Gesellschaft fur Informatik e.V. (GI), Vol. 2, 2010, pp. 115–120.
URL http://www.scopus.com/inward/record.url?eid=2-s2.0-84874273263&partnerID=40&md5=7ac60c0bb767d63a7ff33c827abb4f1c

[23] A. Kimmig, L. Mihalkova, L. Getoor, Lifted graphical models: a survey, Machine Learning 99 (1) (2014) 1–45. arXiv:1107.4966, doi:10.1007/s10994-014-5443-2.
URL http://dx.doi.org/10.1007/s10994-014-5443-2

[24] J. Noessner, M. Niepert, H. Stuckenschmidt, M. N. Jan Noessner, H. Stuckenschmidt, J. Noessner, M. Niepert, H. Stuckenschmidt, RockIt: Exploiting Parallelism and Symmetry for MAP Inference in Statistical Relational Models, in: Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence, AAAI Press, Bellevue, Washington, USA, 2013, pp. 739–745. arXiv:arXiv:1304.4379v2.
URL http://link.springer.com/chapter/10.1007/BFb0027523

[25] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, Y. Katz, Pellet: A practical OWL-DL reasoner, Web Semantics 5 (2) (2007) 51–53. doi:10.1016/j.websem.2007.03.004.

[26] B. Glimm, I. Horrocks, B. Motik, G. Stoilos, Z. Wang, HermiT: An OWL 2 Reasoner, Journal of Automated Reasoning 53 (3) (2014) 245–269. doi:10.1007/s10817-014-9305-1.

[27] N. Marwede, M. Rohr, W. Hasselbring, Automatic Failure Diagnosis Support in Distributed Large-Scale Software Systems based on Timing Behavior Anomaly Correlation, in: A. Winter, R. Ferenc, J. Kodel (Eds.), Proceeding of the 13th European Conference on Software Maintenance and Reengineering (CSMR 2009), 2009, pp. 47–57.

[28] M. Chen, A. X. Zheng, J. Lloyd, M. I. Jordan, E. Brewer, Failure diagnosis using decision trees, in: International Conference on Autonomic Computing, 2004. Proceedings., IEEE, 2004, pp. 36–43. doi:10.1109/ICAC.2004.1301345.
URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1301345

[29] H. Zawawy, K. Kontogiannis, J. Mylopoulos, S. Mankovskii, Requirements-driven root cause analysis using markov logic networks, in: J. Ralyte, X. Franch, S. Brinkkemper, S. Wrycza (Eds.), Advanced Information Systems Engineering, Vol. 7328 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2012, pp. 350–365. doi:10.1007/978-3-642-31095-9_23.
URL http://dx.doi.org/10.1007/978-3-642-31095-9_23

[30] J. von Stülpnagel, J. Ortmann, J. Schoenfisch, IT Risk Management with Markov Logic Networks, Advanced Information Systems Engineering (2014) 301—-315.
URL http://link.springer.com/chapter/10.1007/978-3-319-07881-6_21

[31] S. Kandula, D. Katabi, J. P. Vasseur, Shrink: A tool for failure diagnosis in IP networks, Proceedings of the 2005 ACM SIGCOMM workshop on Mining network data (2005) 173–178.
URL http://dl.acm.org/citation.cfm?id=1080178

[32] M. Steinder, A. S. Sethi, Increasing robustness of fault localization through analysis of lost, spurious, and positive symptoms, Proceedings

of the Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2002) 1 (2002) 322–331. doi:10.1109/INFCOM.2002.1019274.
URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1019274

[33] E. Heiden, S. Bader, T. Kirste, Concept and Realization of a Diagnostic System for Smart Environments, in: H. J. van den Herik, A. P. Rocha, J. Filipe (Eds.), Proceedings of the 9th International Conference on Agents and Artificial Intelligence, ICAART 2017, Volume 2, Porto, Portugal, February 24-26, 2017., SciTePress, 2017, pp. 318–329. doi:10.5220/0006257903180329.
URL https://doi.org/10.5220/0006257903180329

[34] G. Weidl, A. L. Madsen, S. Israelson, Applications of object-oriented Bayesian networks for condition monitoring, root cause analysis and decision support on operation of complex continuous processes, Computers and Chemical Engineering 29 (9) (2005) 1996–2009. doi:10.1016/j.compchemeng.2005.05.005.

[35] T. Liu, S. Chiou, The application of Petri nets to failure analysis, Reliability Engineering and System Safety 57 (2) (1997) 129–142. doi:10.1016/S0951-8320(97)00030-6.
URL http://linkinghub.elsevier.com/retrieve/pii/S0951832097000306

[36] D. L. Poole, R. G. Goebel, R. Aleliunas, Theorist: A logical reasoning system for defaults and diagnosis, in: N. J. Cercone, G. McCalla (Eds.), The Knowledge Frontier: Essays in the Representation of Knowledge, Springer, 1987, pp. 331–352.

[37] M. E. Stickel, A Prolog-like inference system for computing minimum-cost abductive explanations in natural-language interpretation, Annals of Math-

ematics and Artificial Intelligence 4 (1-2) (1991) 89–105.
URL http://link.springer.com/article/10.1007/BF01531174

[38] H. T. Ng, R. J. Mooney, An Efficient First-Order Abduction System Based on the ATMS, in: Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI-91), University of Texas at Austin, Anaheim, CA, 1991, pp. 494–499.

[39] A. C. Kakas, R. A. Kowalski, F. Toni, Abductive logic programming, J. Log. Comput. 1 (6) (1992) 719–770.
URL http://logcom.oxfordjournals.org/content/2/6/719.short

[40] S. Raghavan, R. J. Mooney, Bayesian Abductive Logic Programs, Stat. Relational Artif. Intell. (2010) 82–87.
URL http://www.aaai.org/ocs/index.php/WS/AAAIW10/paper/viewPDFInterstitial/1974/2487

[41] N. Inoue, K. Inui, ILP-Based Reasoning for Weighted Abduction, Plan, Activity, and Intent Recognition (2011) 25–32.
URL http://www.aaai.org/ocs/index.php/WS/AAAIW11/paper/download/3999/4313

[42] E. Ovchinnikova, N. Montazeri, T. Alexandrov, H. J. R., M. C. Mccord, R. Mulkar-Mehta, Abductive Reasoning with a Large Knowledge Base for Discourse Processing, in: H. Bunt, J. Bos, S. Pulman (Eds.), Computing Meaning: Volume 4, Springer Netherlands, Dordrecht, 2014, pp. 107–127. doi:10.1007/978-94-007-7284-7_7.
URL http://dx.doi.org/10.1007/978-94-007-7284-7_7

[43] D. Mohammadi, Finding patient zero, Pharmaceutical Journal 294 (7845) (2015) 47–49.