IT Risk Management with Markov Logic Networks

Janno von Stülpnagel*and Jens Ortmann†

Softplant GmbH
Agnes-Pockels-Bogen 1
80992 Munich, Germany

Joerg Schoenfisch‡

Research Group Data and Web Science
University of Mannheim, Germany

March 19, 2014

Abstract

We present a solution for modeling the dependencies of an IT infrastructure and determine the availability of components and services therein using Markov logic networks (MLN). MLNs offer a single representation of probability and first-order logic and are well suited to model dependencies and threats. We identify different kinds of dependency and show how they can be translated into an MLN. The MLN infrastructure model allows us to use marginal inference to predict the availability of IT infrastructure components and services. We demonstrate that our solution is well suited for supporting IT Risk management by analyzing the impact of threats and comparing risk mitigation efforts.

Keywords: IT Risk management, IT Infrastructure, Markov logic networks, Availability

This is an author-generated preprint copy of a paper that will be published in the Caise 2014 proceedings in Springer LNCS.

1 Introduction

IT risk management tries to find, analyze and reduce unacceptable risks in the IT infrastructure. Most commonly risk is defined as a set of triplets, each triplet consisting of a scenario, its probability and its potential impact [1]. In the IT environment these scenarios are typically called threats.

*Electronic address: janno.stuelplangel@softplant.de
†Electronic address: jens.ortmann@softplant.de
‡Electronic address: joerg@informatik.uni-mannheim.de

If a new threat surfaces, the IT risk management needs to asses its probability and evaluate its potential impact. Today's IT infrastructure has complex dependencies and a threat to a single component can threaten a whole network. Furthermore, single threats often have a very low probability but the combination of many threats can be a major risk to an IT infrastructure. Therefore, it is not enough to look at infrastructure components individually to determine the possible impact of a threat. While each year a huge number of new threats surfaces, old threats do not vanish [2].

A fast response to a new threat is important to minimize the chance of exploitation. However, a manual threat analysis takes time. The complexity of today's IT infrastructure provides many indirect ways a single threat can affect different IT services and each must be analyzed. At the same time, the number of new threats is increasing, which leaves even less time to evaluate each new threat [3]. A semi-automated approach allows an easier handling of this complexity, but to our knowledge there exists none that incorporates dependencies and combinations of multiple threats. Even the tools to monitor and report risk are still in a premature state and there is a demand for tool supported risk assessment [4]. While the measurement of IT infrastructure availability is already part of IT service management [5], it is only moderately used in IT risk management [4].

Our approach has two major features. First, it employs a reusable top-level model of the infrastructure dependencies. Second, the measured availabilities are added to each infrastructure component. The use of measured availabilities frees us from the need to model each measured threat manually. If a new threat surfaces, it can be added to the model and expected changes to the availabilities can be calculated. In this paper, we focus on impacts that make infrastructure components unavailable.

We start our approach by creating a dependency graph of the central components and services. To predict availabilities of IT components and services we need a way to represent (1) the logic of dependency of IT infrastructure, (2) probabilistic values for availabilities and (3) new threats. The dependencies in IT infrastructure can easily be modeled with first-order logic, but, first-order logic alone has no way to calculate how a threat influences the probability that an infrastructure component is available.

Markov logic networks (MLN)[6] offer a single representation of probability and first-order logic by adding weights to formulas. Together with a set of constants, the MLN specifies a probability distribution over possible worlds. Marginal inference computes the probability for specific values of variables in these possible worlds.

Thereby, we can create a reusable IT infrastructure model that includes infrastructure dependencies and measured availability. The model can be used to calculate a prediction how new threats affect the availability of IT components in an IT infrastructure. This provides a fast and quantitative solution to determine if a new threat is acceptable or needs to be mitigated.

We demonstrate and evaluate our solution in a case study where we implement the MLN for a small part of an IT infrastructure. We show how the change of availabilities through a new threat can be predicted and how to analyze a risk mitigation approach for this threat.

2 Background

2.1 Markov Logic Networks

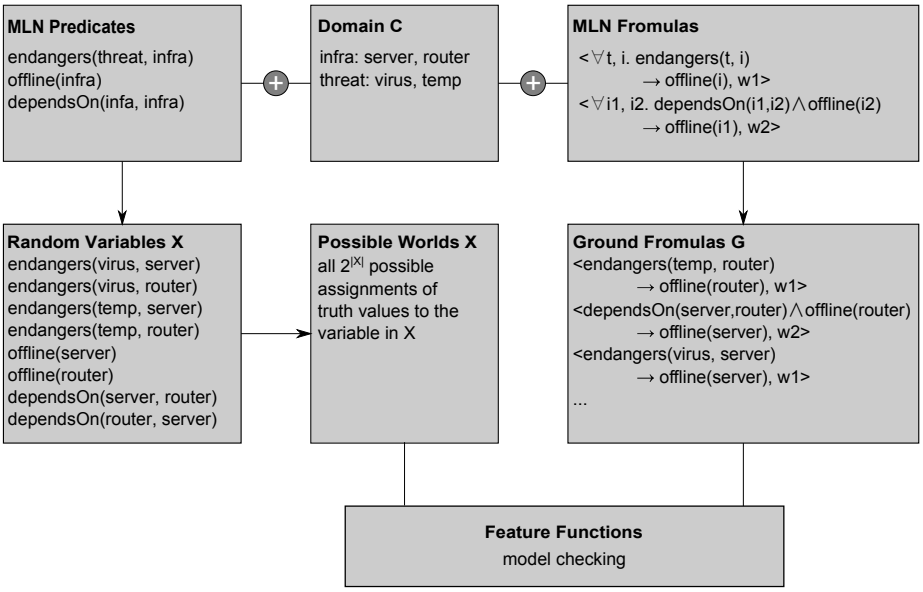


Figure 1: The diagram describes the grounding of a Markov network. The grounded formulas G are generated by substituting each occurrence of every variable in the MLN Formulas with constants of the domain C. The possible worlds X are generated by giving all possible groundings of each predicate. Both the possible worlds X and the grounded formulas G are checked and provided with a value 1 if there are true and 0 otherwise. (Adapted from [7]).

Markov logic networks generalize first-order logic and probabilistic graphical models by allowing soft and hard first-order formulas [6]. Hard formulas are regular first-order formulas, which have to be fulfilled in every possible world. Soft formulas have weights that support worlds in which they are satisfied, but they do not need to, or in case of negative weights even should not, be 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 representation of Markov networks as log-linear model [6].

An MLN is a template for constructing a Markov network. Figure 1 illustrates the structure of a MLN. For each set of constants, there is a different Markov network, following the rules given in the MLN. Markov logic makes the assumption that different constants refer to different objects (unique name assumption) and the domain consists only of those constant and that no other objects (closed world assumption). An atom is a formula that consists of a single predicate and a grounding substitutes each occurrence of every variable in a formula with a constant. A set of grounded atoms is called a possible world.

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 [6]. The MLN L, combined with a finite set

of constants $C = \{c_1, c_2, ...c_{|C|}\}$, defines a ground Markov network $M_{L,C}$ as follows:

1. $M_{L,C}$ 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. $M_{L,C}$ contains one feature for each possible grounding of each formula F_i in L. The value of this feature 1 if the formula is true, and 0 otherwise. The weight of the feature is the w_i associated with F_i in L.

[6, p. 113]

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 $M_{L,C}$:

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

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 $\langle$*first order formula, 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. Marginal inference computes the posteriori probability distribution over the values of all variables given some evidence. We use marginal inference, which allows us to determine the probability distribution of the infrastructure variables for the offline predicate and thereby predict the future availability.

Most query engines require the input to be split into two parts: the formulas (also called program) and the known evidence data.

2.2 Availability

Achieving high availability, defined as up to 5 minutes unavailability per year, is a longstanding goal in the IT industry [8]. Continuous monitoring of availability is part of IT service management best practices like the Information Technology Infrastructure Library (ITIL) [5]. We define availability as the probability that a system is working properly and reachable. The availability of a system can be determined as follows:

$$Availability = \frac{Uptime}{Uptime + Downtime} \tag{2}$$

Unavailability is the inverse of availability:

$$Unavailability = 1 - Availability \tag{3}$$

We distinguish between measured availability and predicted availability. The measured availability of important infrastructure components and services is typically measured over a one-year time-frame. The predicted availability is an estimation how the availability will be, under the assumption of specific changes in the threat-landscape or the infrastructure. We use marginal inference in Markov logic networks to calculate the predicted availabilities in our approach.

3 Predicting Unavailability with Dependency Networks and MLNs

For the threat analysis, we need a dependency model, data on availability and threats. We show how this information can be combined and transformed into evidence for our Markov logic network. Afterwards we discuss the networks function.

3.1 Dependency Network for IT Infrastructures

The basis of our threat analysis is a dependency network. The dependency network models how the availability of one infrastructure component depends on that of other components.

We use this network to calculate how a new threat endangering one component propagates through the network and reduces the availability of other components. The nodes of the network are the high-level infrastructure components and services, for which availability information exists. There are three types of edges: Specific and generic dependencies are modeled as directed edges; redundancies are modeled as undirected edges.

A specific dependency means that a component needs another specific component and cannot be available without it. For example, an email service depends on the router and is unavailable if the router is unavailable.

A generic dependency means that a component needs at least one of a group of redundant components. For example, given two redundant servers realizing an email service, the email service is available if at least one of the servers is running. We model this with an edge between the email service node and each server node.

A redundancy edge means that two components provide the same functionality and one of them can, without manual reconfiguration, provide the work of both. It is possible to have more than two redundant components; in this case, they must be fully connected with redundancy edges. Furthermore, no component should have a redundancy edge to itself. We only consider fully redundant systems; redundancies that are more complex can be modeled with typed redundancies. An example for a dependency network can be seen in Figure 2 in Section 4.

The dependency network can be translated easily into evidence for the MLN. Each edge type translates into a predicate and each node translates into a constant. The symmetry of redundancy and the transitive behavior of the dependencies are modeled as formulas in the MLN program.

3.2 Markov Logic Network for IT Risk Management

Our Markov logic networks has three types of variables and six predicates. The different types of variables are `infra`[1] as the type of all infrastructure components and services, `threat` as the type of all threats and `float` is a numeric value. The predicate `specificDependency(infra, infra)` models that the first infrastructure component depends on the second infrastructure component. `genericDependency(infra, infra)` means that the first infrastructure component depends on the second infrastructure component or a redundant alternative. `redundancy(infra,infra)` models redundant components. `endangered(threat, infra, float)` expresses that a threat endangers an infrastructure component with a weight of `float`. `measuredUnavailability(infra, float)` encodes the measured unavailability as weight `float`, and `offline(infra)` states that a component is not available.

The only hidden predicate, a predicate without full evidence, is `offline(infra)`. It is therefore the only predicate we query and whose variable distribution we determine through sampling. The variable distribution is used to determine the offline probability of an infrastructure component, i.e. the predicted unavailability.

All other predicates are observed, that means all grounded atoms that are not listed in the evidence are false. Each infrastructure component has an availability. If the availability is not specified explicitly the MLN sampling uses a weight of zero, roughly corresponding to an a priori availability of 0.5.

In its basic form, our MLN program comprises five formulas:

$$\langle \texttt{specificDependency(i1,i2)} \wedge \texttt{offline(i2)} \Rightarrow \texttt{offline(i1)}, \infty \rangle \quad (4a)$$

$$\langle \texttt{genericDependency(i1,i2)} \wedge \texttt{redundancy(i2,i3)} \wedge \texttt{offline(i2)} \wedge \texttt{offline(i3)} \Rightarrow \texttt{offline(i1)}, \infty \rangle \quad (4b)$$

$$\langle \texttt{redundancy(i1,i2)} \Rightarrow \texttt{redundancy(i2,i1)}, \infty \rangle \quad (4c)$$

$$\langle \texttt{measuredUnavailability(i1,conf)} \Rightarrow \texttt{offline(i1)}, \texttt{conf} \rangle \quad (4d)$$

$$\langle \texttt{endangered(t,i1,conf)} \Rightarrow \texttt{offline(i1)}, \texttt{conf} \rangle \quad (4e)$$

Formula 4a is a hard formula that invalidates every world where infrastructure component `i2` is offline and infrastructure component `i1` is online, if there is a specific dependency between them.

Formula 4b does the same as Formula 4a for generic dependencies. Provided `i1` is generically dependent on `i2` and `i3`, which are redundant, it invalidates every world in which `i1` is online while `i2` and `i3` are offline.

Formula 4b supports only two redundant infrastructure components, but formulas for more than two redundant components can be constructed analogously. These analogous formulas need their own generic dependency predicate (e.g. `genericDependencyTwo`, `genericDependencyThree`, ...). This prevents that, for example, the formula for a two-component dependency influences the formula for three-component dependency. Though it is possible to create a first-order logic formula that works with any number of redundant components, not

[1] In the following we use type writer fonts like `infra` for statements in MLNs.

all MLN solvers support the full expressiveness of first-order logic. Hence, we chose this workaround for better solver support.

The hard Formula 4c models the symmetry of redundancy: if `i1` is redundant to `i2` then `i2` is also redundant to `i1`.

Formulas 4d and 4e allow giving each evidence a separate weight. An evidence with `measuredUnavailability("Server 1", -3)` translates into ⟨ `offline(" Server 1"), -3`⟩. Formula 4d is used to assign an individual measured unavailability to each infrastructure component.

Formula 4e works analogously. The additional parameter `threat t` allows specifying more than one threat per infrastructure component. The last two formulas can be combined into one formula, but we think modeling is easier when not mixing single threats with estimated probabilities and quantitatively determined availabilities. The source-code for our MLN program is given in Appendix A.

The weights for the measured availabilities can be found manually by iteratively calculating the probabilities and correcting the weights. In some cases, finding the correct weights is not trivial [9], but there are efficient learning algorithms for MLNs [6].

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.

3.3 Summary of the Approach

We start by creating a dependency network of all major infrastructure components and services. We transform this network into evidence for our MLN program. We collect unavailability information for each node and use a learning algorithm to add the corresponding weights to the evidence. By adding threats, the resulting evidence can be used to determine the effect of a local threat to the whole network.

4 Case Study

Table 1: The predicates for the dependency network of the base configuration.

```
genericDependency("Email Service","Server 1")
genericDependency("Email Service","Server 2")
redundancy("Server 1","Server 2")
specificDependency("Server 1","Router 1")
specificDependency("Server 2","Router 2")
specificDependency("WiFi AP 1","Router 1")
specificDependency("WiFi Service","WiFi AP 1")
```

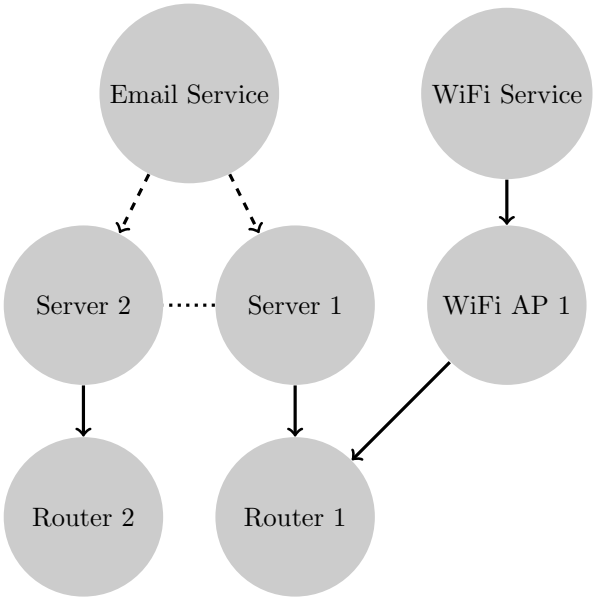


Figure 2: The dependency network of the small IT infrastructure of our case study. Solid arrows indicate specific dependencies, dashed arrows symbolize generic dependencies and the dotted line represents a redundancy.

A small case study demonstrates the usability of our solution. There exist several open-source inference engines for Markov logic networks, e.g. Alchemy[2], Tuffy[3], and RockIt[4]. We use RockIt for the calculations in our study.

The base configuration of the dependency network (see Figure 2) has seven nodes: An `Email Service` is realized by two redundant servers `Server 1` and `Server 2`. Each of the servers depends on its own router (`Router 1` and `Router 2`, respectively). One of the two routers is used by a `WiFi Access Point (AP)`, which offers a `WiFi Service`. The corresponding evidence is listed in Table 1, the MLN program can be found in the Appendix A. The infrastructure components and services have the measured unavailabilities given in column Scenario 1 in Table 2 and the corresponding, learned weights are shown in Table 3.

We can now add a new threat: `Router 1` threatens to overheat because construction work cut off the normal airflow to the room. The predicate expressing this is `endangered("Overheating", "Router 1", 3)`. By using marginal inference, we get the offline probabilities, which give us the predicted unavailabilities shown in column Scenario 2 in Table 2.

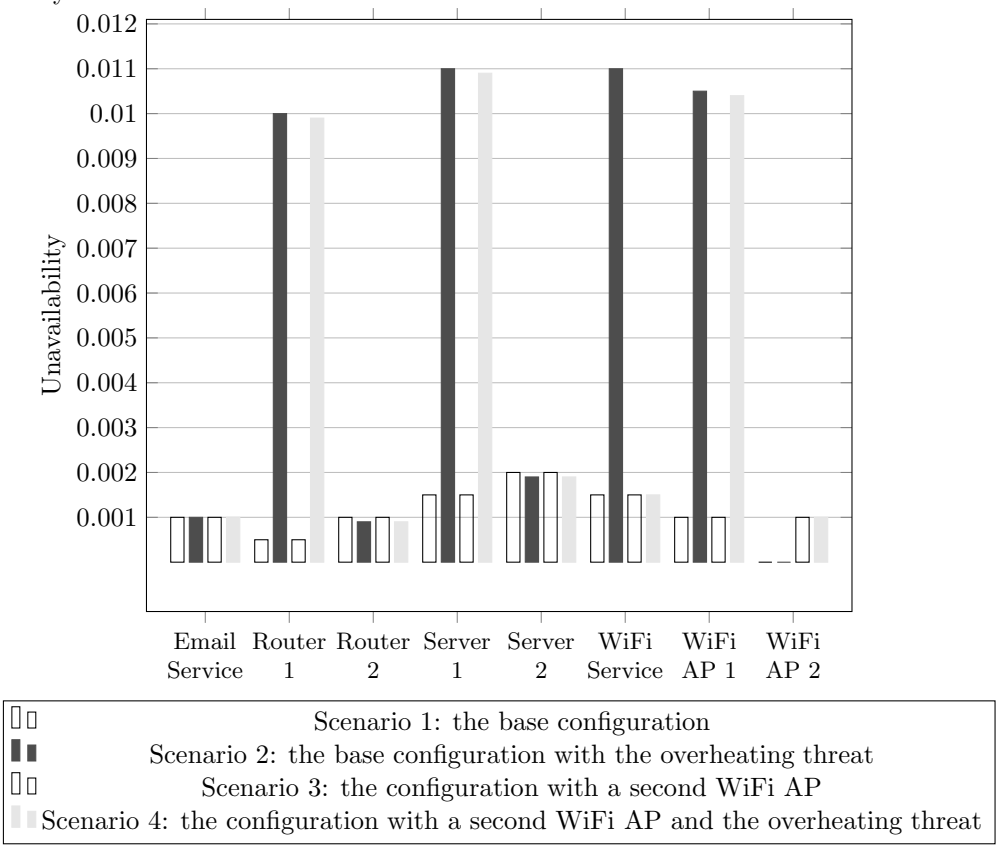
The threat increased the unavailability of `Router 1` by 0.0095. This has the effect that the unavailability of `WiFi AP 1`, `Server 1` and `WiFi Service` also increases by 0.0095. On the other hand, the unavailability of `Email Service` remains unaffected, because it can use `Server 2`, which depends on `Router 2`. The changes can be seen in Figure 3.

[2]http://alchemy.cs.washington.edu
[3]http://hazy.cs.wisc.edu/hazy/tuffy
[4]https://code.google.com/p/rockit

Table 2: The unavailabilities for the different scenarios of the case study. Scenario 1 is the base configuration, Scenario 2 is the base configuration with the overheating threat, Scenario 3 is the configuration with a second WiFi AP, and Scenario 4 is the configuration with a second WiFi AP and the overheating threat.

Component	Scenario 1 (measured unavailabilities)	Scenario 2 (predicted unavailabilities)	Scenario 3 (measured unavailabilities)	Scenario 4 (predicted unavailabilities)
`Email Service`	0.0010	0.0010	0.0010	0.0010
`Router 1`	0.0005	0.0100	0.0005	0.0099
`Router 2`	0.0010	0.0009	0.0010	0.0009
`Server 1`	0.0015	0.0110	0.0015	0.0109
`Server 2`	0.0020	0.0019	0.0020	0.0019
`WiFi Service`	0.0015	0.0110	0.0015	0.0015
`WiFi AP 1`	0.0010	0.0105	0.0010	0.0104
`WiFi AP 2`	-	-	0.0010	0.0010

Figure 3: The change of the unavailabilities in the different scenarios of our case study.



The cooling problem can only be solved through expensive additional con-

Table 3: The learned weights for the measured unavailabilities in the base configuration.

```
measuredUnavailability("Email Service",-6.907250)
measuredUnavailability("Router 1",6.933551)
measuredUnavailability("Router 2",0.001400)
measuredUnavailability("Server 1",-6.908549)
measuredUnavailability("Server 2",-6.905200)
measuredUnavailability("WiFi AP 1",-0.023739)
measuredUnavailability("WiFi Service",-7.590026)
```

Table 4: The predicates for the dependency network with a second WiFi AP.

```
genericDependency("Email Service","Server 1")
genericDependency("Email Service","Server 2")
genericDependency("WiFi Service","WiFi AP 1")
genericDependency("WiFi Service","WiFi AP 2")
redundancy("Server 1","Server 2")
redundancy("WiFi AP 1","WiFi AP 2")
specificDependency("Server 1","Router 1")
specificDependency("Server 2","Router 2")
specificDependency("WiFi AP 1","Router 1")
specificDependency("WiFi AP 2","Router 2")
```

struction work; therefore, other risk mitigation approaches should be investigated.

A cheap risk mitigation approach is to provide a second, redundant `WiFi AP 2`, which uses `Router 2`. We update the infrastructure model (Table 4) and set the unavailability for the new `WiFi AP 2` equal to that of `WiFi AP 1` (see Scenario 3 in Table 2). Again we use the MLN program from Appendix A and learn the corresponding weights (Table 3).

Now we can add again the threat `endangered("Overheating", "Router 1", 3)` and calculate the unavailabilities (see Scenario 4 in Table 2) with the help of marginal inference. The unavailability of `Router 1` increases by 0.0094. The unavailability of `Server 1` and `WiFi AP 1` also increases again but this time the unavailability of the `WiFi Service` remains unaffected. The changes are also shown in Figure 3.

We can see that a second WiFi Access Point would successfully mitigate the risk for the WiFi Service.

The slightly different increase of the unavailability in the two scenarios has two reasons. First, the learning algorithm provides only an approximation of the correct weights for the unavailabilities. Second, the number of evidence (and thereby the number of possible worlds) changed between the two scenarios, but the weight of the threat remained the same. As can be seen in Figure 3, these effects are small enough to not influence the overall result.

The case study also demonstrates that the results of MLN marginal inference

Table 5: The new learned weights for the configuration with a second WiFi AP.

```
measuredUnavailability("Email Service",-6.906300)
measuredUnavailability("Router 1",6.909860)
measuredUnavailability("Router 2",10.333070)
measuredUnavailability("Server 1",-6.905100)
measuredUnavailability("Server 2",-6.888114)
measuredUnavailability("WiFi AP 1",-7.600700)
measuredUnavailability("WiFi AP 2",-10.364995)
measuredUnavailability("WiFi Service",-6.500500)
```

are not exactly the results one would expect from regular probability calculation. However, as we demonstrate, the MLN calculation is well suited for the calculation of threat propagation.

5 Related Work

There exist alternative approaches that combine logic and probability (see [10] for an overview). Here we limit the discussion of related work to Bayesian logic programs (BLP) [11]. Because of the usage of Bayesian networks in combination with risk analysis [12], BLPs seem to be the most prevalent alternative to MLNs.

Bayesian logic programs aim at resolving some limitations of Bayesian networks, among others the essentially propositional nature of their representations. Therefore, Bayesian logic programs unify Bayesian networks with logic programming. Each Bayesian logic program represents a Bayesian network by mapping the atoms in the least Herbrand model, which constitutes the semantics of the logic program, to nodes of the Bayesian network.

A BLP computes the probability for one query from the known probabilities in the resolution to the query statement. In contrast, MLNs infer all probabilities from a given complete set of weights.

Furthermore, there exist efficient solvers for MLNs and they have been applied to a wide range of problems, for instance in requirements-driven root cause analysis [13] and data integration [14].

Another advantages of MLNs over other probabilistic logic approaches is that the Markov networks, constructed through the MLN, are undirected graphs. This has the effect that changing the weights at one node will influence the whole graph and thereby the results. Hence, it is more likely to find new relationships between elements that where not explicitly modeled. On the other side, this has the disadvantage that it is harder to isolate a single variable. Unlike BLPs [11], MLNs have very simple semantics while theoretically keeping the expressive power of first-order logic [6].

There are other approaches that use dependency networks to determine the global impact of a threat. The approach of Zambon et al. [15] focuses on downtime and temporal aspects but unlike our approach it is qualitative and considers only one threat at a time. The approach of Breu et al. [16] uses the number of attacks as the key quantitative concept and models each threat separately but does not take redundant components into account.

6 Discussion

To our knowledge this paper presents the first application of Markov logic networks to risk management. We have shown that MLNs generally allow an automatic calculation of risks, exemplified by the probability of availability, in an IT infrastructure.

As described in the case study, our solution allows us to efficiently calculate how a threat affects an infrastructure network. We have demonstrated how the analysis can be used to compare changes in the infrastructure and thereby support the risk mitigation. Our approach has two major features. First, it employs a top-level model for the dependency network, which can be easily maintained and reused. Second, it uses the measured availabilities of the infrastructure components and thereby creates a quantitative infrastructure model without modeling every threat separately. This also has the advantage that the manual work for risk analysis is greatly reduced.

In most cases, risk assessments are performed regularly, e.g. every six month. At the same time, the link between availability and risk management is seldom used and technological support for IT risk management was identified as one area where improvement is needed [4].

Our approach enhances risk assessment by providing an automation and allowing reuse of earlier work. By using the availability measurements of the IT service management as quantitative input and provide results in the same way, we allow an easy communication of IT risk.

However, there are some limitations, caveats and lessons learned from using Markov logic networks, and in particular marginal inference in MLNs.

It is not possible to simply add or remove infrastructure components or services to or from an existing model. This is so because the weight of the new component or service influences the total weight of the model, which in turn affects the probabilities of all statements. In the current state of our solution, adding or removing a new infrastructure component requires relearning all weights.

Even though MLNs use a quite simple representation, modeling with them is not as straightforward as it may seem [7]. Each variable, every literal that involves a hidden predicate, in the MLN must have a weight. For instance in our case study this means that each node in the dependency network needs a `measuredUnavailability(infra, float)`. This requires an accurate specification of the MLN evidence. However, since the measurement of the availability is a best practice [5], the necessary data should be available.

Without an understanding of the normalization of weights and the log-linear model, the relationship between the weights and the resulting probabilities of the marginal inference can be counterintuitive. Changing the weight of a single formula shifts the relative weights of the possible worlds according to where the formula is true. This results in sometimes unexpected changes in probabilities of statements. Therefore, we usually learn the weights for the measured unavailabilities. While modeling with MLNs it is also important to have in mind that the weight of a soft formula does not directly correspond to a specific universal probability for this formula. The probability depends on the whole MLN and the modeled domain, including the number of individuals in this domain.

A limitation of existing implementations for marginal inference is their use of sampling algorithms to calculate the probabilities. To determine the effect

of threats with very low probabilities, which are quite common in risk management, sampling requires a high number of iterations. Threats with very small probabilities can provide the problem that their frequency is so low that they do not influence the availability measurement of a component. These threats can still be added manually through the predicate `endangered(threat,infra, float)`. To our knowledge, there is no alternative to sampling as the exact computation is too complex to be done for larger networks. However, it has the advantage that it allows us to choose how much time to invest into the accuracy of the probabilities. We reduce the problem of sampling with low probability by using the measured unavailability, as combination of many small threats and thereby are able to aggregate them to larger numbers.

While there are different MLN solvers, to our knowledge none of them supports full first-order logic. Because of the undecidability of first-order logic, this will most likely not change.

7 Conclusion

We have demonstrated how Markov logic networks (MLN) can be used to calculate the expected availabilities of infrastructure components and services. Our solution uses a combination of a dependency network, measured availabilities and a new threat information as basis for this calculation. We obtain the evidence for the MLN by constructing a dependency network and learning the weights for the measured availabilities. By adding a new threat and using marginal inference, we can predict availabilities of IT components under different threat conditions. This allows a faster and more quantitative risk analysis, which is essential for today's threat landscape, than a manual analysis. We have focused our work on how threats indirectly affect the whole network. Future work is needed in differentiating the impact. Not all services are equally important for the organization and there are other kinds of impact than taking an IT component offline, for instance compromising it or reducing its efficiency. We plan to further improve the solution by integrating existing data sources to automatically generate the dependency network.

Acknowledgement

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".

We are thankful to Christian Meilicke for helpful feedback and support.

References

[1] R. Braz, E. Amir, and D. Roth. A survey of first-order probabilistic models. In D.E. Holmes and L.C. Jain, editors, *Innovations in Bayesian Networks*, volume 156 of *Studies in Computational Intelligence*, pages 289–317. Springer Berlin Heidelberg, 2008.

[2] Ruth Breu, Frank Innerhofer-Oberperfler, and Artsiom Yautsiukhin. Quantitative assessment of enterprise security system. In S. Jakoubi, S. Tjoa, and E.R. Weippl, editors, *Third International Conference on Availability, Reliability and Security, 2008. ARES 08.*, pages 921–928. IEEE Computer Society, 2008.

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

[4] Ernst & Young. Managing it risk in a fast-changing environment. Technical report, Ernst & Young, 2013.

[5] European Union Agency for Network and Information Security. Enisa threat landscape mid year 2013. Technical report, European Union Agency for Network and Information Security, 2013.

[6] J. Gray and D.P. Siewiorek. High-availability computer systems. *Computer*, 24(9):39–48, 1991.

[7] IBM X-Force. Trend and risk report 2012. Technical report, IBM X-Force, 2013.

[8] Dominik Jain. Knowledge engineering with markov logic networks: A review. In C. Beierle and G. Kern-Isberner, editors, *Evolving Knowledge in Theory and Applications. 3rd Workshop on Dynamics of Knowledge and Belief (DKB-2011) at the 34th Annual German Conference on Artificial Intelligence, KI-2011, Berlin, Germany, October 4, 2011. Proceedings*, volume 361 of *Informatik-Bericht*, pages 16–30. Fakultät für Mathematik und Informatik, FernUniversität in Hagen, 2011.

[9] Dominik Jain, Bernhard Kirchlechner, and Michael Beetz. Extending markov logic to model probability distributions in relational domains. In Joachim Hertzberg, Michael Beetz, and Roman Englert, editors, *KI 2007: Advances in Artificial Intelligence*, volume 4667 of *Lecture Notes in Computer Science*, pages 129–143. Springer Berlin Heidelberg, 2007.

[10] Stanley Kaplan and B. John Garrick. On the quantitative definition of risk. *Risk Analysis*, 1(1):11–27, 1981.

[11] K. Kersting and L. De Raedt. Bayesian logic programs. *CoRR*, cs.AI/0111058, 2001.

[12] Mathias Niepert, Jan Noessner, Christian Meilicke, and Heiner Stuckenschmidt. Probabilistic-logical web data integration. In Axel Polleres, Claudia d'Amato, Marcelo Arenas, Siegfried Handschuh, Paula Kroner, Sascha Ossowski, and Peter Patel-Schneider, editors, *Reasoning Web. Semantic Technologies for the Web of Data*, volume 6848 of *Lecture Notes in Computer Science*, pages 504–533. Springer Berlin Heidelberg, 2011.

[13] Jan Noessner, Mathias Niepert, and Heiner Stuckenschmidt. Rockit: Exploiting parallelism and symmetry for map inference in statistical relational models. In M. des Jardins and M.L. Littman, editors, *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence, July 14-18, 2012, Bellevue, Washington, USA*. AAAI Press, 2013.

[14] Matthew Richardson and Pedro Domingos. Markov logic networks. *Machine Learning*, 62(1-2):107–136, 2006.

[15] P. Weber, G. Medina-Oliva, C. Simon, and B. Iung. Overview on bayesian networks applications for dependability, risk analysis and maintenance areas. *Engineering Applications of Artificial Intelligence*, 25(4):671–682, 2012.

[16] E. Zambon, S. Etalle, R. J. Wieringa, and P. H. Hartel. Architecture-based qualitative risk analysis for availability of it infrastructures. Technical Report TR-CTIT-09-35, Centre for Telematics and Information Technology University of Twente, Enschede, September 2009.

[17] Hamzeh Zawawy, Kostas Kontogiannis, John Mylopoulos, and Serge Mankovskii. Requirements-driven root cause analysis using markov logic networks. In Jolita Ralyt, Xavier Franch, Sjaak Brinkkemper, and Stanislaw Wrycza, editors, *Advanced Information Systems Engineering*, volume 7328 of *Lecture Notes in Computer Science*, pages 350–365. Springer Berlin Heidelberg, 2012.

A RockIt MLN

For our MLN, we use the syntax of RockIt [17]. RockIt expects first order formulas in conjunctive normal form (CNF). An online version of RockIt is available here: `http://executor.informatik.uni-mannheim.de/systems/rockit/`

```
*specificDependency(infra,infra)
*genericDependency(infra,infra)
*redundancy(infra,infra)
*endangered(threat,infra,float_)
*measuredUnavailability(infra, float_)
offline(infra)

!specificDependency(i1,i2) v !offline(i2) v offline(i1).
!genericDependency(i1,i2) v !redundancy(i2,i3) v !offline(i2)
 v !offline(i3) v offline(i1).
!redundancy(i1,i2) v redundancy(i2,i1).
conf: !measuredUnavailability(i1, conf) v offline(i1)
conf: !endangered(t,i1,conf) v offline(i1)
```