2019 IEEE 23rd International Enterprise Distributed Object Computing Workshop (EDOCW)
©2019 IEEE
DOI 10.1109/EDOCW.2019.0001

Applying Dynamic Bayesian Networks for Automated Modeling in ArchiMate: A Realization Study

Björn Bebensee
RWTH Aachen University
Aachen, Germany
bjoern.bebensee@rwth-aachen.de

Simon Hacks
Research Group Software Construction
RWTH Aachen University
Aachen, Germany
simon.hacks@swc.rwth-aachen.de

***Abstract*—Enterprise Architecture modeling is an approach to manage modern IT infrastructure and landscapes to coordinate a multitude of IT projects in an organization. Enterprise architects apply modeling tools such as ArchiMate to document the enterprise architecture. Because these models have traditionally been created and maintained manually, efforts to manage IT architecture have been both time-consuming and error-prone. We evaluate an approach by Johnson et al. (2016) for automated generation of these models from observed network traffic using Dynamic Bayesian Networks. As inference in large Dynamic Bayesian Network proves computationally infeasible, we propose an alternative approach using a set of Hidden Markov Models to model the current network state, present an implementation, and evaluate its performance in a real-world setting.**

***Index Terms*—Enterprise Architecture, Realization Study, Dynamic Bayesian Network, Hidden Markov Model, Automated Modeling**

1. Introduction

Today's organizations face increasing challenges in managing their IT architecture and landscape as industries are rapidly changing in a global market. As modern business rely on their IT infrastructure for daily operations, the optimal alignment of business and IT has become important. Enterprise Architecture (EA) is a comprehensive approach, which uses models to assist in management and decision-making [1]. Such EA models allow for management of even complex architectures. Over the past two decades EA has become an established approach which is successfully used by many organizations [2].

However, as EA models are mainly created and maintained manually, efforts to manage an organization's IT architecture have been both time-consuming and error-prone. Empirical studies have shown that EA practitioners are not satisfied with quality of the underlying data and actuality of their models [3]. Even in cases where data is gathered automatically the majority of remodeling work is still done manually [4]. These studies have also shown that there is a general interest in automation of the model creation and maintenance process. Further analysis identified main requirements for automated EA modeling and maintenance [5] as well as challenges posed by it [6]. There have been efforts to reduce manual work required for EA model maintenance and creation [7], [8] and in recent years some approaches to automated EA modeling have been presented [9], [10], [11].

As previous approaches had not addressed uncertainty surrounding any data collected from information sources, Johnson et al. [12] have presented a novel approach to automated enterprise IT infrastructure modeling, which considers the task as a probabilistic state estimation problem. They suggest to model the network as a Dynamic Bayesian Network where its state can be estimated using information collected from network traffic.

In this work, we want to evaluate this approach by implementing it. However, as we recognize that the original approach takes a non-acceptable processing time, we propose a similar but significantly faster way of modeling the network using Hidden Markov Models (HMMs). Using HMMs and data collected from a network sniffer, the likelihood for network addresses being occupied by hosts and the likelihood for two hosts to talk to each other can be continuously updated and stored in a graph database. Then, we automatically generate models in ArchiMate notation from the information stored in the graph database. As we are using a network sniffer as our information source, we focus on modeling the technology and application layer. This approach adapts to changes in the IT infrastructure automatically: in the event of a new service joining the network or a host being taken down newly observed evidence will be taken into account and the likelihood quickly adapts to the new situation. The automatically generated ArchiMate models are of high actuality.

The rest of this paper is structured as follows: Next, we present related work to automated EA modeling. Additionally, we explain the approach of Johnson et al. [12] in more detail. In section 3, we present the needed basics to understand Dynamic Bayesian Networks and Hidden Markov Models. Next, in section 4, we explain our implementation of the original work of Johnson et al. As we recognize that the implementation lacks a reasonable computation time, we present further an implementation relying on Hidden

Markov Models. Before we conclude our work in section 6, we check the correct behavior of implementation and evaluate our implementation with respect to the requirements of automated EA modeling according to Farwick et al. [5].

2. Related Work

Back in 2009 Buckl et al. [3] already stated that the majority of EA practitioners create and maintain models manually. They conducted a survey that showed that none of the participants were satisfied with their data quality and that there was a general interest in automating the information gathering process. This is supported by a survey conducted by Winter et al. [4] in 2010, which showed that while a number of participants is attempting to use a semi-automated approach to collection and maintenance of EA data, the majority of remodeling work is still done manually even if information has been gathered automatically.

In an analysis on the requirements for automated EA modeling and model maintenance, Farwick et al. [5] find that there is a need for automated EA modeling as such an automation could reduce EA modeling efforts. Additionally, Farwick et al. investigate how maintenance processes can be automated [7], [8], [13] and propose a meta-model for automated EA model maintenance [14]. Their efforts focus on a semi-automated approach: while data is collected from a variety of sources in an automated fashion, the mapping between different information sources is created manually to maintain high data quality.

Hauder et al. [6] identify challenges in automated EA model maintenance and group them into the categories data, transformation, business and organization. The main challenges mentioned are the quality of collected data and the selection of information sources, mapping of these information sources to a central repository and its maintenance as well as the initial investment and value it adds to an organization. They additionally found EA modeling to be a major challenge for organizations as data quality is not sufficient and the process is very time consuming. The main concerns found with regards to automation of the modeling process are the difficulty in its implementation and the costs it brings with it.

An attempt to automate EA model creation was made by Buschle et al. [9]. In their work they utilize a vulnerability scanner for data collection and, then, create an EA model using a meta-model called CySeMoL for cybersecurity analysis. The mapping from the vulnerability scanner's output to the meta-model is tool-specific, however, Buschle et al. suggest that the output can theoretically be mapped to any other meta-model instead. Similarly, Holm et al. [10] propose using the same vulnerability scanner to populate a model in ArchiMate.

Building on previous work, Välja et al. [11] propose a new approach to automate enterprise IT architecture modeling using multiple information sources for data collection to generate a more complete and higher quality model. In their work, they deal with the added complexity of using multiple information sources by prioritizing more reliable information sources over less reliable information sources. They show that through data cleaning and aggregation limitations can be overcome and that it is possible to automate the process of EA modeling from more than one information source.

Johnson et al. consider the task of estimating the current network state at a given time a probabilistic state estimation problem [12]. They propose modeling the network state using Dynamic Bayesian Networks (DBNs) which are updated for every time step with regards to evidence observed from a passive network sniffer for instance. They suggest that approximate statistical inference to estimate the current network state can be computed in this DBN using particle filters. This state estimate can subsequently be mapped to a meta-model to create EA models in an automated fashion. They further identify different information sources, from which data could be collected automatically and map these sources to entities of the ArchiMate framework [15].

This approach is promising as it captures the uncertainty surrounding the collected evidence and easily adapts to any changes in the architecture and infrastructure. Furthermore, it is easily extendable as the mapping is not specific to any particular tool and evidence can be collected from a multitude of different information sources by feeding these into the model as evidence.

The abstraction provided through the model proposed in [12] is of particular interest for us as it allows utilization of data collected from various different information sources.

3. Basic Concepts

3.1. Dynamic Bayesian Networks

Dynamic Bayesian Networks (DBNs) are a temporal model that extends regular Bayesian networks and can be used to model complex dependencies and uncertainty. Dynamic Bayesian Network approaches have been successfully used to solve the simultaneous localization and mapping (SLAM) problem in robotics [16], to identify gene regulatory networks [17], [18], for speech recognition [19] and to measure network security [20]. They are especially viable for complex stochastic processes and allow for parameters to be learned [16].

3.2. Hidden Markov Models

Another temporal probabilistic model is the Hidden Markov Model (HMM). HMMs have been applied in a variety of contexts, most notably in speech recognition [21], but also in handwriting recognition [22], [23], pattern recognition in molecular biology [24], [25] and others.

Any HMM is solely composed of a single discrete state variable, which represents all the discrete states of the world, and its evidence variable. It is possible to model more than one state variable using a HMM by simply combining all of the state variables into one "superstate" whose possible discrete states are exactly tuples of all the possible state combinations. One can easily see that any HMM is simply a special case of a DBN with a single state variable and a single evidence variable [16].

Formally, given the first-order Markov assumption, the joint probability distribution of a HMM is fully defined by its prior probability distribution $P(X_0)$, a transition model $P(X_t|X_{t-1})$ and a sensor model $P(E_t|X_t)$:

$$P(X_{0:t}, E_{1:t}) = P(X_0)\Pi_{i=1}^{t} P(X_i|X_{i-1})P(E_i|X_i) \quad (1)$$

We write the transition model as a transition matrix T, where T_{ij} is the probability of a transition from state i to state j. To specify the sensor model, we define a diagonal sensor matrix O_t for time slice t, where $O_{t_{ii}} = P(e_t|X_t = i)$, so that the i-th diagonal entry is exactly the probability of state X_t causing evidence e_t to appear.

3.3. A DBN network model

To create an enterprise IT architecture model in ArchiMate, which incorporates data gathered from network traffic, we can process network traffic using a tool such as Tcpdump[1], collect data from IP headers and use the information as evidence in a DBN. The DBN will provide a state estimation of the network that is continuously being updated online. We define the DBN's state space, transition model, and sensor model according to [12] as follows.

State space. Let A be the set of all network address that are to be observed. For a single given time slice the Bayesian Network then contains a boolean variable $\alpha_i \in A$ for each network address. Each of these variables represents the probability that a host is currently using this network address. Then, for each pair of network addresses $(\alpha_i, \alpha_j) \in A \times A$ there is a probability $\mu_{i,j}$ that a message is being transmitted from α_i to α_j. This probability is causally dependent on the existence of hosts at the given network addresses as it is more likely to observe a message between two network addresses, which are being used by hosts, than if one or both of the network addresses were vacant. Therefore, the probability distribution of $\mu_{i,j}$ is given by $P(\mu_{i,j}|\alpha_i, \alpha_j)$ as the probability for a message transmission is conditioned on both addresses' occupancy. X_t denotes the set of all state variables at time t.

Transition model. At each time step the DBN is updated according to a transition model which describes the probability of a variable changing its state. This probability depends on the pace at which the network architecture is changing (which usually is rather slow) and the length of an individual time slice. Given a length of 30 seconds for a time slice for instance, it is very unlikely the physical network will change in the current time slice. Thus, the probability of a variable remaining in its current state $P(\alpha_{t+1}|\alpha_t)$ will be very high.

According to the first-order Markov assumption, the variables in the DBN will only have causal dependencies on the previous time slice. The DBN can then be updated given new evidence e_t for time slice t using online filtering and a method of approximate inference. For our transition model, we specify the transition probability $P(X_{t+1}|X_t)$ from every state possible in X_t to every state in X_{t+1}.

Sensor model. The sensor model describes the probability that a state generates observable evidence. Thus, the sensor model describes the probability $P(\lambda_{i,j}|\mu_{i,j})$ that a message between two hosts is observed by the network sniffer given that a message has or has not been sent. The sensor model captures the uncertainty of indirect measurements and allows for better estimates of the network state. Given $\mu_{i,j}$, we specify the probability of an observation $\lambda_{i,j}$ by the network sniffer.

4. Realization

4.1. DBN-based approach

In order to build a mathematical model of the network infrastructure, we collect data from the physical network through passive measurements that we can process and use as evidence in a DBN. From gathered network packets, we can gain a lot of information by inspecting the IP and TCP/UDP headers respectively.

The source and destination IP addresses from the IP header are of highest interest to us as they are evidence for hosts likely occupying these network addresses. However, we have decided to only use the source IP address as direct evidence for a host in the model. As we observe the entire network traffic and will be able to observe outgoing packets from the destination IP address as well this is sufficient for our purposes. Such outgoing traffic, even for an application that only receives data, is at the minimum going to include acknowledgement messages (ACKs) for TCP traffic and other, higher layer forms of acknowledgements for UDP traffic. Additionally, we are able to obtain the port of the sending application from the TCP/UDP headers. This lets us distinguish traffic from different applications and possibly infer the type of application if some standard port is used or a list of ports is provided, i.e. traffic on port 22 is likely to be SSH traffic. The packet filter we use is built on the *libpcap* library[2], which allows for usage of Berkeley Packet Filters (BPFs) as capture filters and which we use to limit the scope of the model to the local network. BPFs allow for fast and efficient evaluation filters for packet headers [26].

The DBN's state space can be divided into three types of state variables: host nodes (or address nodes) α_i, message nodes $\mu_{i,j}$, and evidence nodes $\lambda_{i,j}$.

Each of these state variables is binary. In the model, a network address will refer to an IP address and (TCP or UDP) port pair. This allows us to approximate separate likelihoods for applications on the same hosts as although the underlying network infrastructure and hosts might remain

1. Tcpdump 4.9.2, Van Jacobson, Craig Leres, Steven McCanne, https://www.tcpdump.org, 2017-09-03

2. Libpcap 1.9.0, The Tcpdump Group, https://www.tcpdump.org, 2018-07-22

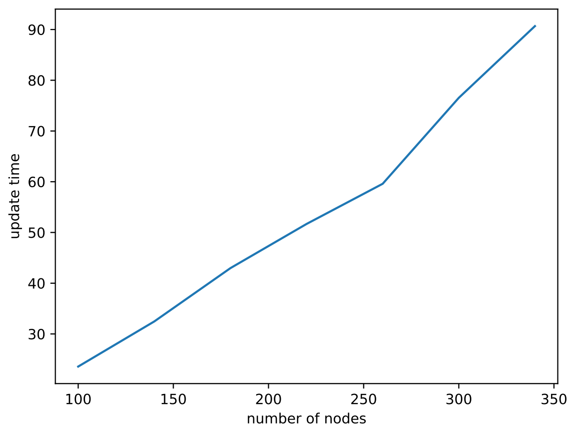


Figure 1. Update time in seconds required in a DBN with n host and message nodes, with 10,000 particles

the same, the applications and services running on them might not. We have a host node for every network address in the subnet that is included in the model.

However, large enterprise architectures in big companies often have thousands of hosts in their network and even more applications and services running on them. According to the model proposed in [12], we have a host node for every network address and a message node for every pair of network addresses. Therefore given n network addresses in the observed subnet we would have n^2 message nodes and the same amount of evidence nodes. Even with a relatively conservative estimate of a subnet like 10.0.0.0/21 that allows for a maximum of 2,046 hosts to connect to the network at once and only a single application communicating on the network per host, the DBN will have 4,186,116 message and again an equal amount of evidence nodes. We find inference in a DBN of this size to be simply intractable as the update time for a few hundred nodes already exceeds one minute (see figure 1) and appears to grow in $O(n)$.

In practice, when testing the model, we unfortunately observe a very poor performance of the model even for smaller networks. Even when continuously presented with new evidence for all network addresses, the fraction of particles indicating that there is a host at all address do not seem to converge to 100%. This behavior could be explained by the structure of the DBN. As we have a large number of evidence nodes, all of which are leaf nodes in the DBN, the particles that explain the evidence best and get assigned the highest weight may in fact not explain all of the evidence. When these subsequently get sampled more often during the resampling process the fraction of samples correctly estimating the state of those addresses, which were not correctly estimated by the highest weight particles, then drops significantly. The model does not seem to converge to the actual state of the network.

Additionally, the computation time for the resampling step grows with the number of network addresses observed as well as the number of particles maintained. Even for a DBN with 100 network addresses and a particle filter with 20,000 particles a single resampling step already takes roughly 39 seconds.

Typically medium-sized organizations may have a few hundred nodes and large organizations can even have multiple thousands of nodes in the EA model [27], [28], [29]. Given that a typical network consists of many more network addresses than hosts as not every network address is actively occupied, we need to observe many more addresses than there are hosts in a typical EA model. Given additionally that a higher number of nodes in the DBN requires an even greater number of particles to maintain accuracy, the computation quickly becomes intractable as computation time appears to be linear in $n \cdot d$ for n particles and d hosts (see figure 1).

4.2. HMM-based approach

As the DBN-based approach has proved to be too slow for practical applications, we propose an alternative approach to model the network state using a set of HMMs, which each of keeps track of the current state of one network address. Each HMM is equivalent to a single-state DBN [16]. These particular HMMs are equivalent to the *supernodes* that result from merging the address and message variables in the DBN-based approach.

The HMMs are updated individually using the forward algorithm allowing much faster inference while still giving us an accurate estimate of the network state at any given time. The forward equation is given by

$$f_{t+1} = \alpha \; O_{t+1} T^{\top} f_t \qquad (2)$$

where f_t is the state at time t, T is the transition matrix, O_{t+1} is a diagonal sensor matrix for time slice $t + 1$ and α is the normalizing constant [30]. Since our state space only consists of two different spaces for each HMM – an address either is or is not occupied by a host – the size of the matrices in the forward equation is 2×2. One step of the forward algorithm can therefore be computed efficiently.

Given one HMM for each network address, we can keep track of the estimated network state with regards to the observed evidence. Just like the DBN, these HMM will be updated using the evidence gathered by the network sniffer. Because we no longer have message variables in our model but still want to keep track of which network addresses are talking to each other, we introduce another HMM for each pair of addresses. Since these HMMs are independent of each other, we can simply add new HMMs as new hosts are observed for the first time instead of keeping track of all network addresses in the target address space. Additionally, the filtering task is inexpensive as each HMM only consists of a single hidden state.

We are using a network sniffer to gather data which can be fed into the model as evidence in order to estimate the current network state. However, the choice of the vantage point is critical in passive network measurements and often

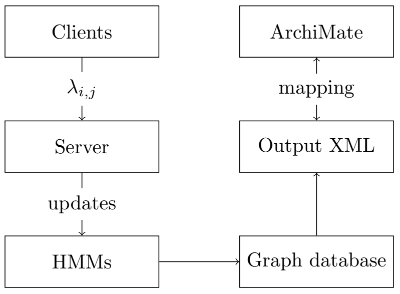


Figure 2. An overview of the architecture of our implementation

proves to be a challenge [31]. On a typical large-scale enterprise network the network architecture will be complex and there is typically a hierarchy of different routers, so it is not possible to observe traffic passively from a single host. In order to observe traffic from many vantage points, we have built a client-server architecture into our modeling tool that allows us to collect evidence from any number of different clients. These clients send any gathered evidence to the server. After each time slice it factors the new evidence into the model and updates the database. The ArchiMate model can be generated directly from the database at any given time.

We only include nodes and relationships in our model that have a high likelihood, i.e. nodes with a likelihood $p > h$ for a threshold h. For any relationship we additionally require both the source and destination nodes' likelihoods to exceed the threshold as well. All the nodes and relationships can then be retrieved from the Neo4j database using a simple Cypher query. We can output this information to an XML file and map its schema to the ArchiMate Model Exchange File Format's XML schema. Using this mapping, the output XML file can easily be converted to the ArchiMate Model Exchange File Format. The model can then be viewed and altered in any EA modeling tool which supports the ArchiMate standard.

For each node, we include the IP address, the port and its likelihood. For relationships the output includes the source and destination's address-port pair as well as their likelihood. The address-port pair will serve as an ID in the ArchiMate model. We can then map this XML schema to the ArchiMate standard's XML schema.

The output XML schema consists of three main sections: the property definitions which include definitions for the source address, destination address and the likelihood, the hosts in the network and the relationships between hosts in the network. The definitions of properties found in the schema of the output are mapped to instances of ArchiMate property definitions. Each host in the output is mapped to an instance of an ArchiMate element where its network address is the unique identifier and the likelihood is the elements property. Finally, each relationship is mapped to an instance of an ArchiMate relationship using the source and destination network addresses as unique ArchiMate element identifiers. The relationship's likelihood property is mapped to an instance of an ArchiMate property.

5. Evaluation

We evaluate the model in a fictitious case study. We set up a network of ten servers, which run applications that are communicating with each other through dummy protocols on the application layer. The underlying transport layer protocols are TCP and UDP on randomly assigned ports. Using this cloud environment, we test our model in a number of ways, evaluate its performance, and whether it met requirements for automated Enterprise Architecture model creation and maintenance found in [5].

5.1. Correctness of the model

To verify the correct behavior of our model, we evaluate it in a number of test cases. We simulate a network of servers with different applications on different ports and use the proposed model to automatically generate up-to-date ArchiMate models (see figure 3).

Unchanged network. For the first test case the network remains unchanged. Therefore, we expect the generated ArchiMate model to remain unchanged as well. Our model behaves as expected and the resulting ArchiMate models include exactly all of the nodes of the default network state.

Arrival of a node. A new host joins the network and we expect it to be included in the generated ArchiMate model after enough time slices have passed by for the likelihood to be higher than the threshold of 50%. The number of time slices required depends on the specified sensor model. As evidence for the new host and its communication is observed, the model adapts and includes the new host and connection in the resulting ArchiMate model.

Departure of a node. A host leaves the network and should no longer be included in the generated ArchiMate model after enough time slices have passed for the likelihood to have adapted and dropped under the threshold of 50%. We observe that the model adapts to the departure of the host and that the likelihood of its network address being occupied drops below the threshold. As a result the host is no longer included in the generated ArchiMate model.

Outage of a node. We want the model to remain unaffected by a brief service outage (such as an application restart, a host reboot or an update). The behavior observed in this test case heavily depends on the chosen time slice length and sensor model and can be adapted to an organization's needs depending on how quickly the architecture changes (see figure 4).

During an outage of a node for a short period of time we observe that the model adjusts the likelihood of the network address being actively occupied according to the chosen sensor model. For our use-case we use the sensor model with sensor matrix O when no evidence is observed and P when evidence is observed

$$O = \begin{pmatrix} 0.7 & 0 \\ 0 & 0.1 \end{pmatrix} \quad P = \begin{pmatrix} 0.3 & 0 \\ 0 & 0.9 \end{pmatrix}$$

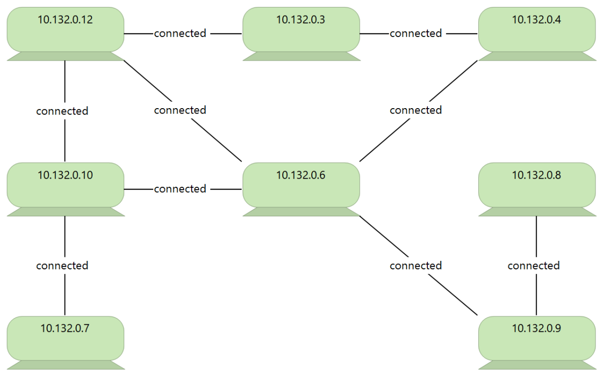
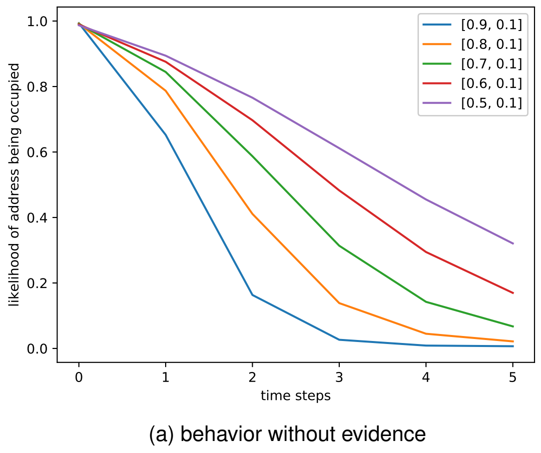


Figure 3. An ArchiMate model generated from the test network using the proposed method



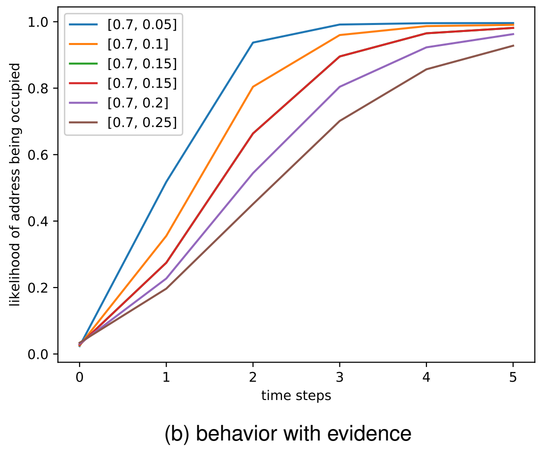


Figure 4. Convergence of the model given evidence or no evidence at every time slice for different sensor models

which results in the likelihood dropping below the threshold of 50% if no evidence is observed for at least three time slices in a row. Given this sensor model and a time slice length of 5 minutes this would mean that any outage resulting in no network communication for 15 minutes or longer will result in the host no longer being included in the ArchiMate model. During testing the model behaved as expected and during short outages the generated ArchiMate model would remain unchanged while a longer outage would lead to the host disappearing from the model and reappearing at a later time. It is however also possible to learn the sensor model from data using parameter learning and the Baum-Welch algorithm [16].

Empty network. In the case of a network without any applications communicating besides the server and clients which update the model continuously, we observe that the generated ArchiMate model only includes the model's server and clients. The model therefore behaves as expected.

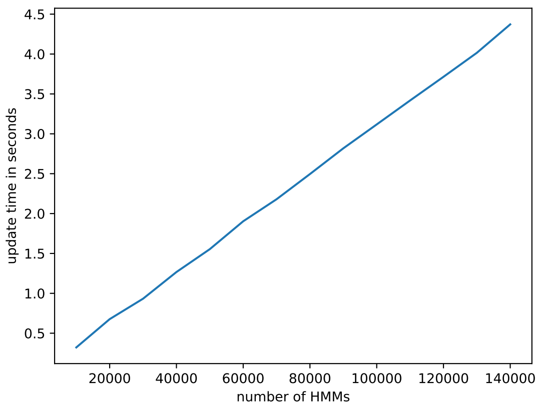
5.2. Performance of the model

To determine the performance of the model, we consider the time complexity of an update for one time slice. As the time required to update all HMMs with the new evidence during the online filtering task can not be longer than the length of one time slice, this makes the time required to update the HMMs the most important factor in its performance. In our model a single HMM can be updated with regards to new evidence using the forward algorithm in linear time.

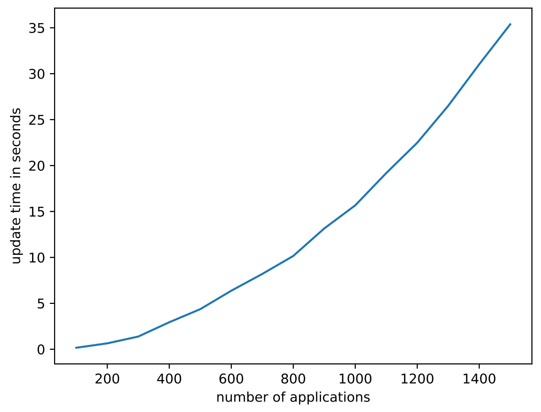
As each HMM is updated individually this gives us an update time in $O(n)$ for n HMMs. However, as we use use one HMM for each application as well as each connection observed between different applications the number of HMMs is in $O(m^2)$ for m applications. This yields an update time of $O(n^2)$ in the worst case of a fully connected network. However, it is unlikely that each application would generate traffic to all other applications in a real-world scenario. Assuming a fully connected network with each application being connected to all other applications the network is equivalent to a complete graph and the number of connections therefore equals $\frac{m(m-1)}{2}$. The model would therefore maintain $m + \frac{m(m-1)}{2}$ HMMs to estimate the network state. Measurements of the time required to update the model for one time step can be seen in figure 5. Despite the worst-case assumption of a fully connected network where each application is communicating with all other applications, the model seems to perform reasonably well. Even in the worst-case it is possible to model the network and create an EA model with 1500 applications while keeping the time for one time slice under one minute.

5.3. Evaluation of requirements

Farwick et al. [5] have previously conducted an analysis on the requirements for automated EA modeling and model maintenance based on literature review and a survey. They



(a) by number of HMMs



(b) Worst-case update time for a fully connected network with n hosts

Figure 5. Required update time for one time slice

found a number of architectural, organizational, integration and data source, data quality functional system and non-functional requirements. We evaluate the proposed model with respect to these requirements in [5]. However, we focus on data source and data quality requirements as these are most relevant to our work which focuses on automated processing of data to generate EA models automatically. Other requirements deal more with the organizational processes (i.e. data ownership) or requirements relevant for a complete EA tool (i.e. it allows for KPI calculations).

Regarding the requirements on the data source, our approach solely does not meet the requirement to integrate information about projects. As it is the focus of our approach to require data from the infrastructure and application layer, this shortcoming is expectable and acceptable.

With respect to the data quality requirements of Farwick et al., our approach implies several shortcomings. First, our approach does not provide a mechanism to help the quality assurance team to ensure data consistency. However, if necessary, one could implement a manual check of the intended changes before the export of the data to ArchiMate. However, this would counteract the idea of automated modeling.

Second, our approach does not provide a mechanism to adjust the granularity of the data. To overcome with this issue, we see two solutions. On the one hand, we can rely on different levels of detail like solely the IP or a certain subnet than on the port. On the other hand, an abstraction can be added to the ArchiMate export so that there is an aggregation to a desired level of abstraction.

Lastly, our approach does not offer an automated process to propagate changes to other systems. However, the approach could be easily extended in the way that there would be a propagation of changes if a certain node passes a defined threshold and, therefore, either would be added to the model or removed from the model.

6. Conclusion

We have evaluated a probabilistic approach to automated Enterprise Architecture modeling using Dynamic Bayesian Networks proposed in [12]. Based on the key ideas behind this approach, we have proposed our own model for the network state, which is faster and can be used to automate the modeling process on the technology layer and parts of the application layer in ArchiMate. This model is continuously updated with evidence from observed traffic and can be used to automatically generate ArchiMate models representing the current network state. The sensitivity of the model can be adapted through its sensor and transition model, which determine how quickly the model adapts to new evidence. Applications running on these hosts can be inferred from the ports they communicate on given a list of common and standard ports and applications that use them. Although this approach does not capture causal dependencies as well as a large DBN because HMMs are updated independently, it performs very well and generates accurate models of the network.

The evidence mechanism of the HMMs provides a valuable abstraction which can be used to extend the model to other information sources. It is possible to combine data from multiple sources as shown in [11] and use it evidence for the HMMs. Our approach focused on passive measurements via network sniffers. As it is possible that some services only communicate very infrequently or are never accessed at all they may not be accurately included in the model if the model's sensitivity is too low or the length of a time slice is too short. The accuracy of the model can be improved by additionally using data from other sources such as an active network scanner, which can identify services by a host's open ports. This information can then be fed into the model as evidence in combination with data from the passive measurements. Since HMMs are updated independently, it may also be possible to achieve further improvements by adapting the time slice length individually according to the uncertainty in different areas of the network.

The model fulfills most requirements for automated EA modeling identified by [5] and provides a foundation that more extensive EA modeling tools can be built upon. More information sources that can be used for modeling on different layers of the ArchiMate model have been identified in [12], although evidence for higher level abstractions such as business layer entities is more difficult to come by. However, some valuable information is very cheap to collect which makes it possible to automate at least parts of the EA model creation and maintenance process. We expect to see approaches which combine different information sources to automatically create a more complete EA model.

Furthermore, the model proposed in this work can be extended in several ways. More information sources for application and business layer modeling can be added. Additionally, applications and services used can also be inferred from network traffic given a list of common ports. A network scanner could be used to improve accuracy and to find applications, which may not be actively communicating on the network. Lastly, information collected from various data sources may not always be reliable. Further study may be necessary on how to best aggregate and filter this data so it can be used for automated EA modeling.

References

[1] J. W. Ross, P. Weill, and D. Robertson, *Enterprise architecture as strategy: Creating a foundation for business execution*. Harvard Business Press, 2006.

[2] R. Lagerstrom, T. Sommestad, M. Buschle, and M. Ekstedt, "Enterprise architecture management's impact on information technology success," in *2011 44th Hawaii International Conference on System Sciences*. IEEE, 2011, pp. 1–10.

[3] S. Buckl, E. M. Ernst, J. Lankes, F. Matthes, and C. M. Schweda, "State of the art in enterprise architecture management 2009, technische universität münchen," 2009.

[4] K. Winter, S. Buckl, F. Matthes, and C. M. Schweda, "Investigating the state-of-the-art in enterprise architecture management methods in literature and practice." *MCIS*, vol. 90, 2010.

[5] M. Farwick, B. Agreiter, R. Breu, S. Ryll, K. Voges, and I. Hanschke, "Requirements for automated enterprise architecture model maintenance," in *13th International Conference on Enterprise Information Systems (ICEIS), Beijing*, 2011.

[6] M. Hauder, F. Matthes, and S. Roth, "Challenges for automated enterprise architecture documentation," in *Trends in Enterprise Architecture Research and Practice-Driven Research on Enterprise Transformation*. Springer, 2012, pp. 21–39.

[7] M. Farwick, B. Agreiter, R. Breu, S. Ryll, K. Voges, and I. Hanschke, "Automation processes for enterprise architecture management," in *2011 IEEE 15th International Enterprise Distributed Object Computing Conference Workshops*. IEEE, 2011, pp. 340–349.

[8] M. Farwick, C. M. Schweda, R. Breu, K. Voges, and I. Hanschke, "On enterprise architecture change events," in *Trends in enterprise architecture research and practice-driven research on enterprise transformation*. Springer, 2012, pp. 129–145.

[9] M. Buschle, H. Holm, T. Sommestad, M. Ekstedt, and K. Shahzad, "A tool for automatic enterprise architecture modeling," in *Forum at the Conference on Advanced Information Systems Engineering (CAiSE)*. Springer, 2011, pp. 1–15.

[10] H. Holm, M. Buschle, R. Lagerström, and M. Ekstedt, "Automatic data collection for enterprise architecture models," *Software & Systems Modeling*, vol. 13, no. 2, pp. 825–841, 2014.

[11] M. Välja, R. Lagerström, M. Ekstedt, and M. Korman, "A requirements based approach for automating enterprise it architecture modeling using multiple data sources," in *2015 IEEE 19th International Enterprise Distributed Object Computing Workshop*. IEEE, 2015, pp. 79–87.

[12] P. Johnson, M. Ekstedt, and R. Lagerstrom, "Automatic probabilistic enterprise IT architecture modeling: A dynamic Bayesian networks approach," in *Enterprise Distributed Object Computing Workshop (EDOCW), 2016 IEEE 20th International*. IEEE, 2016, pp. 1–8.

[13] M. Farwick, R. Breu, M. Hauder, S. Roth, and F. Matthes, "Enterprise architecture documentation: Empirical analysis of information sources for automation," in *2013 46th Hawaii International Conference on System Sciences*. IEEE, 2013, pp. 3868–3877.

[14] M. Farwick, W. Pasquazzo, R. Breu, C. M. Schweda, K. Voges, and I. Hanschke, "A meta-model for automated enterprise architecture model maintenance," in *2012 IEEE 16th International Enterprise Distributed Object Computing Conference*. IEEE, 2012, pp. 1–10.

[15] The Open Group, "Archimate 3.0.1 specification, an open group standard," 2017. [Online]. Available: http://pubs.opengroup.org/architecture/archimate3-doc/

[16] K. P. Murphy and S. Russell, "Dynamic bayesian networks: representation, inference and learning," 2002.

[17] B.-E. Perrin, L. Ralaivola, A. Mazurie, S. Bottani, J. Mallet, and F. d'Alche–Buc, "Gene networks inference using dynamic bayesian networks," *Bioinformatics*, vol. 19, no. suppl. 2, pp. ii138–ii148, 2003.

[18] M. Zou and S. D. Conzen, "A new dynamic bayesian network (dbn) approach for identifying gene regulatory networks from time course microarray data," *Bioinformatics*, vol. 21, no. 1, pp. 71–79, 2004.

[19] G. Zweig and S. Russell, "Speech recognition with dynamic bayesian networks," 1998.

[20] M. Frigault, L. Wang, A. Singhal, and S. Jajodia, "Measuring network security using dynamic bayesian network," in *Proceedings of the 4th ACM workshop on Quality of protection*. ACM, 2008, pp. 23–30.

[21] L. R. Rabiner, "A tutorial on hidden markov models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257–286, 1989.

[22] M.-Y. Chen, A. Kundu, and J. Zhou, "Off-line handwritten word recognition using a hidden markov model type stochastic network," *IEEE Transactions on Pattern Analysis & Machine Intelligence*, no. 5, pp. 481–496, 1994.

[23] A. Kundu, Y. He, and P. Bahl, "Recognition of handwritten word: first and second order hidden markov model based approach," *Pattern recognition*, vol. 22, no. 3, pp. 283–297, 1989.

[24] E. L. Sonnhammer, S. R. Eddy, E. Birney, A. Bateman, and R. Durbin, "Pfam: multiple sequence alignments and hmm-profiles of protein domains," *Nucleic acids research*, vol. 26, no. 1, pp. 320–322, 1998.

[25] A. V. Lukashin and M. Borodovsky, "Genemark. hmm: new solutions for gene finding," *Nucleic acids research*, vol. 26, no. 4, pp. 1107–1115, 1998.

[26] S. McCanne and V. Jacobson, "The bsd packet filter: A new architecture for user-level packet capture." in *USENIX winter*, vol. 46, 1993.

[27] A. Schoonjans, "Social network analysis techniques in enterprise architecture management," Ph.D. dissertation, PhD thesis, Ghent University, Ghent, 2016.

[28] R. Lagerström, C. Baldwin, A. MacCormack, and D. Dreyfus, "Visualizing and measuring enterprise architecture: an exploratory biopharma case," in *IFIP Working Conference on The Practice of Enterprise Modeling*. Springer, 2013, pp. 9–23.

[29] S. Hacks and H. Lichter, "Optimizing enterprise architectures using linear integer programming techniques," *INFORMATIK 2017*, 2017.

[30] S. J. Russell and P. Norvig, *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,, 2016.

[31] N. Chatzis, G. Smaragdakis, J. Böttger, T. Krenc, and A. Feldmann, "On the benefits of using a large ixp as an internet vantage point," in *Proceedings of the 2013 conference on Internet measurement conference*. ACM, 2013, pp. 333–346.