This is the accepted version of a paper presented at 2018 *IEEE 22nd International Enterprise Distributed Object Computing Conference (EDOC 2018).*

Citation for the original published paper:

A Probabilistic Enterprise Architecture Model Evolution
In: (pp. 51-57). IEEE
https://doi.org/10.1109/EDOC.2018.00017

N.B. When citing this work, cite the original published paper.

Permanent link to this version:
http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-255632
A Probabilistic Enterprise Architecture Model Evolution

Simon Hacks and Horst Lichter
Research Group Software Construction
RWTH Aachen University
Aachen, Germany
{hacks, lichter}@swc.rwth-aachen.de

Abstract—Enterprise Architecture (EA) is a widely accepted means to ease the alignment of IS projects with enterprise-wide objectives. One central artifact of EA are EA models, which provide a holistic view on the organization and support EA's stakeholder to create added value. As EA collects its data from different sources, the data can be contradictory. This work contributes to existing research by proposing a novel approach to deal with contradictory data without solving the thereby caused conflicts.

In order to achieve this objective, we refine the Predictive, Probabilistic Architecture Modeling Framework (P²AMF) introduced by Johnson et al., which already incorporates a way to represent uncertainty regarding the existence of modelled entities. To make our technique usable, we generalize P²AMF from its UML/OCL notation to a graph presentation in order to apply it to EA models notated in arbitrary notations like ArchiMate. Furthermore, we add alternative scenarios in different versions along a time series to meet the requirements of a distributed EA evolution. To show the applicability of our approach, we developed a proof of concept prototype by implementing the proposed calculations and guidelines on a Neo4j graph database. Last, we argue that our approach meets the stated requirements of a distributed EA evolution.

Index Terms—Enterprise Architecture Management; Probabilistic Modelling; Quality Management

1. Introduction

Enterprise architecture (EA) is a widely accepted discipline to guide local information system (IS) endeavors through a holistic view on the fundamental structures, design, and evolution principles of the overall organization [1]. EA eases the alignment of IS projects with enterprise-wide objectives, which leads to reduced complexities as well as integration efforts in the overall corporate IS landscape [2], [3].

Since it beginnings in the 1980’s [4], EA has developed to an established discipline in industry and research [5], [6]. A widely accepted definition of the term architecture [7] is given in the ISO 42010:2011 [8], which defines architecture as the “fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution”.

As this definition implies, the EA model, comprised by the elements and relationships of the organization, is one central artifact of EA. It provides a holistic view on the organization and, therefore, supports the EA’s stakeholders to create added value [9]. Consequently, EA has to provide central, important, and up-to-date information to its stakeholders. For this purpose, EA needs to collect data from several sources which may be contradictory.

There are two different origins of contradictory EA data input. First, there are technical sources like network scanners or databases, e.g., the Configuration Management Database (CMDB). Those sources, mostly contained in the technology layer [10], [11], may deliver data on different levels of details or even outdated data. Second, data provided by human EA suppliers may cause conflicts, because there are different ways to model the same aspect or they rely on wrong or outdated data.

To handle such contradictions, so far two different kinds of approaches are proposed: The first approach tries to resolve the contradictions, e.g., by estimating the trustworthiness of the sources [10], [11]. The second approach tries to prevent contradictions before they emerge, for example, by providing a holistic framework to assess the quality of EA models [12].

Both approaches have in common that they try to determine one single truth for the model. However, in some cases it might be useful to keep the contradictory information. For example, there are two contradictory information from different sources and it is not possible to estimate which one is true. Therefore, we want to keep both information and do not want to abolish one. Another example could be that both information describe alternative future EA scenarios and we do not know which one will become true.

Consequently, we formulate our research question:

RQ: How can evolutionary EA scenarios provided with uncertainty information be represented in an EA model?

This research question encloses four facets:

1) The EA model needs to provide uncertainty regarding the existence of any of its entities.

2) The EA model needs to reflect different evolutionary scenarios along a certain time-span, because projects may deliver competing scenarios for different points in time.
3) As interaction with the EA model is needed, e.g., to integrate new scenarios, the proposed approach should define process guidelines, by means of a set of rules, to add and remove scenarios to an EA model as well as how to handle different versions along a certain time series.

4) A version of the EA model without uncertainty is needed, e.g., the management concerns an EA model with less details or the used EA tool cannot represent uncertainties.

In order to develop means to reflect uncertainty in EA model evolution, we opt for Design Science Research (DSR) in accordance to Peffers et al. [13]. Therefore, we generalize the P²AMF [14] to a simple graph representation and add capabilities to represent uncertainty regarding future states of the model. To discuss our work, we create a proof of concept prototype using a graph database containing a small EA model, transform our procedures to Cypher [15] to calculate different EA model states, and, finally, apply Cypher queries to the EA model.

This article is structured as follows: Next, we introduce the DSR methodology and how we applied its activities. In section 3 we present at first P²AMF and discuss its shortcomings related to our research question. Second, we introduce our solution by describing its formal background and we illustrate it with a small example. Third, we demonstrate our solution by implementing the beforehand introduced example using a graph database. Afterwards, we discuss our work by comparing our research objectives with the achieved results. Before we conclude the presented results and sketch future work, we discuss related work elaborating on uncertainty in models.

2. Methodology

DSR offers a systematic structure for developing artifacts, such as models or methods [16]. As our research question indicates the development of a constructive solution, the application of DSR is appropriate. We stick to the approach of Peffers et al. [13], which consists of six activities (A1..A6) and two possible feedback loops. We applied them as follows.

A1 Problem Identification & Motivation: The problem to be solved emerged in the context of a research project, which aims to develop a solution to keep the central EA model and peripheral models created by projects in synchronization. As projects may deliver contradictory information and it is not decided which information is true, the contradictory information need to be kept.

A2 Objectives of our solution: From the identified problem, we derived the objectives for a solution, manifested in the stated research question and its facets, detailed in the introduction.

A3 Design & Development: To develop the proposed constructive solution, we applied an engineering research approach according to Lázaro and Marcos [17]. Facilitating and adapting the findings of Johnson et al. [14], [18], we added a time-related facet to their research.

A4 Demonstration: To demonstrate the developed solution, we performed a single case study to gain a first, in-depth reflection on our solution in a real life scenario [19]. Moreover, single case studies are a feasible instrument to show the applicability of software engineering artifacts [20].

A5 Evaluation: We will discuss our results in 3.3 by comparing them to the objectives stated in section1.

The last activity (A6 Communication) and the feedback loops are not reflected directly in this paper. The feedback loops from Evaluation/Communication to Objectives/Design & Development will influence future research elaborating on this topic.

3. Solution Design

Based on the stated research question, we propose a constructive solution to represent uncertainty in EA models. This solution is embedded in a broader EA process already presented in [21]. The overall aim is to create a roundtrip between the central EA model and peripheral models developed by projects. Hence, the results presented in this paper contribute to a distributed evolution approach for EA models, to be more concrete, to assure the EA model data’s quality.

3.1. An Architecture Modeling Framework for Probabilistic Prediction

In the following we present concepts to represent a probabilistic EA for solving quality issues. Our solution is based on P²AMF, a framework to model uncertainty in class and object diagrams proposed by Johnson et al. [14]. Therefore, they facilitate OCL and add attributes storing existence probability values for objects or relations. Moreover, object attributes can be stochastic. However, we are only interested in objects and relations among them.

The following example illustrates P²AMF. Assume there are two classes (Service and Cloud) connected by an association. As a service might be down or we are not absolutely certain about its presence, its existence is modeled with an uncertainty expressed by a Bernoulli distribution with a probability of 0.98. In other words, a service has a probability of 98% to be existent and 2% to be not existent. Each service is provided by a cloud. As we have no secure knowledge regarding the relation between the service and its respective cloud, the association between Service and Cloud is also fraught with uncertainty. This may lead to a situation in which the defined cardinalities get violated. For more details on this issue and how to solve it, we refer to Johnson et al. [14], especially Section 5.

3.2. A Probabilistic Enterprise Architecture

P²AMF provides a framework to represent uncertainty in class and object diagrams. However, if we apply this approach to solve our research problem, two shortcomings occur: First, it is restricted to EAs modelled with UML class
and object diagrams. Consequently, dedicated EA modeling languages like ArchiMate [22] are not supported. Therefore, a more general representation is needed. Second, the special needs of decentralized performed projects contributing to a distributed EA evolution are not sufficiently reflected neither. Those needs can be satisfied by applying Dynamics Bayesian Networks [23] similar to the idea by Johnson et al. [18].

To get a general representation of EA models, we describe an EA model as a pair of a set of nodes and a set of relations: \( \text{EA} = (N, R) \). A node, \( n \in N \), represents an architectural element of the EA like a business process, an application, or a server. A relation, \( r \), is a tuple of two architectural elements, which are linked somehow to each other:

\[
 r \in R \subseteq \{(u, v) : u, v \in N\}. \tag{1}
\]

To model uncertainty, we introduce a probabilistic existence function

\[
p : N \cup R \rightarrow \{x \in \mathbb{R} | 0 \leq x \leq 1\} \tag{2}
\]

annotated as \( p(e) \) which returns the probability of a node or a relation to be existent. Those probabilities can be determined in different ways. First, the value could be calculated similar to the approach presented in Johnson et al. [18]: The more often a certain element appears within a reported data set, the higher is its probability to be existent. Second, experts could estimate these probabilities.

Unfortunately, calculating the probability whether a certain scenario will be realized, falls short, because this needs a large amount of data, typically produced in an automated way. But, this is not the case in the EA domain due to the following reasons: First, project architects model a future state which could not be captured automatically. Second, projects architects would model only changes they plan to carry out. Consequently, changes would appear seldom. Third, all projects rely on the same data basis. If a project removes an element, other projects will not, because they necessarily do not know that this element does not exist anymore. Hence, the model would never change. Therefore, calculating the probabilities is not applicable in our case.

Expert estimation seems to be a more promising approach to determine existence probabilities of architecture elements. Obviously, estimation is biased and not perfect, but research has shown that even more formal approaches are not necessarily better [24]. Moreover, the estimation effort is low compared to the suggested calculations. Therefore, we opt for expert estimations.

Project architects or project managers are possible experts to estimate these probabilities. There are two different levels of granularity conceivable to estimate probabilities: First, experts can estimate the probability for each node and relation individually. Second, experts can estimate the probability for a complete scenario consisting of many nodes and relations to become existent.

Apart from the question who should estimate, it is also important to answer the question when the estimations should be conducted. Obviously, there is the initial estimation at the beginning of a project when the experts know about the different possible scenarios. This estimation should be updated every time the probabilities of the scenarios change fundamentally, e.g., because one of the scenarios got discarded.

As estimating each node and relation individually is too fine grained, we take estimating the probability of complete scenarios as more feasible. Let \( S \) be the set of all scenarios. To represent a scenario \( s \in S \) we annotate it as a quadruple with \( s = (N^+_s, R^+_s, N^-_s, R^-_s) \). \( N^+_s \subseteq N \) and \( R^+_s \subseteq R \). To differentiate between added and removed elements of the EA, we introduce \( + \) to describe that a node or relation is added and \( - \) to describe that a node or relation is removed.

Additionally, we extend the definition of \( p \) so that it also returns the probability value of a scenario to be existent:

\[
p : N \cup R \cup S \rightarrow \{x \in \mathbb{R} | 0 \leq x \leq 1\}. \tag{3}
\]

Furthermore, it might occur that different projects deliver their results for the same point in time. Therefore, it is necessary to know which scenarios are competitive to each other. Thus, we group all competitive scenarios into one set: \( S_C \subseteq S \), where the sum of all scenario probabilities has to be 1:

\[
\sum_{s \in S_C} p(s) = 1. \tag{4}
\]

In other words, we know every competitive scenario which delivers its results to a certain point in time.

The following short example clarifies this concept. We assume an EA model simplified to nodes and relations as depicted in Figure 1(a). Additionally, there is a project contributing to EA’s evolution by proposing two competitive scenarios (cf. Figure 1(b) and 1(c)). Either scenario 1 or 2 will be implemented. An expert estimates the realization of scenario 1 with \( p = 0.6 \) and scenario 2 with \( p = 0.4 \). Within both scenarios the node \( M \) is added as well as the relation between \( M \) and \( D \). Furthermore, a relation between \( D \) and \( F \) is added in both scenarios. Both scenarios differ in handling the node \( K \). Scenario 1 replaces this node by node \( L \) and additionally links \( L \) to node \( C \). In contrast, scenario 2 keeps node \( K \) and adds a relation to node \( C \) like in scenario 1.

![Figure 1. Possible Evolution Scenarios of an EA Model](image)

To merge an origin EA model with all competing scenarios (leading to \( \text{EA}' \)), the existence probability for each node and each relation has to be calculated. For added nodes and relations, we simply sum up the estimated probabilities.
along all scenarios \( s \in S \). If nodes or relations are removed, we have to subtract the estimated probabilities from 1. If a node or relation is unchanged, it keeps its initial probability. This leads to the following equation to calculate the probability for a node \( n \in N' = N \cup \bigcup_{s \in S} N^+_s \) of EA':

\[
p(n) = \begin{cases} 
\sum_{s \in S} \text{getP}(n, N^+_s) & n \in \bigcup_{s \in S} N^+_s, \\
1 - \sum_{s \in S} \text{getP}(n, N^-_s) & n \in \bigcup_{s \in S} N^-_s, \\
0 & \text{else.}
\end{cases}
\]  

(5)

with

\[
\text{getP}(n, N_o) = \begin{cases} 
p(n) & n \in N, \\
0 & \text{else.} \\
N_o : \text{set of nodes}
\end{cases}
\]  

(6)

The equation for a relation \( r \) of EA' looks similar. To explain and clarify these equations, we merge the origin EA model with scenarios 1 and 2 from Figure 1. The result is depicted in Figure 2(a). The solid lines represent nodes and relations with \( p = 1.0 \), the dashed lines with \( p = 0.6 \), and the dotted lines with \( p = 0.4 \). Two observations can be made. First, nodes and relations occurring in both scenarios get a value of 1. Second, node \( K \) and its relation to \( E \) have a probability value of 0.4, because both are removed in scenario 1 and, therefore, the probability value has to be inverted.

![Figure 2. Merging the Origin EA Model with Two Different Scenarios](image1)

So far, our solution to represent probabilities does not take the time dimension into account. However, this is necessary since projects usually do not deliver their results synchronously. Therefore, we introduce points in time, \( t_i \in T \), and link each point with the current valid EA model.

Figure 3 shows an example. The EA model \( M \) contains only entities with \( p = 1.0 \) and, thus, is considered to have no uncertainty. Starting with \( M \) as a baseline at \( t_0 \), we introduce project A delivering two competitive scenarios \( A_1 \) and \( A_2 \) at \( t_1 \) and project B delivering one non-competitive scenario \( B_1 \) at \( t_2 \). To keep track of the changes of the competitive scenarios, we trace \( A_1 \) and \( A_2 \), until we know which scenario will be realized. At this point in time, we integrate the chosen scenario into the current EA model and discard the others. E.g., at \( t_3 \) we decide to realize scenario \( A_2 \), thus, we incorporate it into \( M \) leading to \( M' \). \( A_1 \) is discarded and \( B_1 \) remains unchanged as it is not competitive to the other scenarios.

![Figure 3. Evolution of an EA Model \( M \) over Time](image2)

If an EA model representation without probabilities is needed, a report can be generated containing the most likely EA model. This report can either base on elements with a probability value greater than a threshold or on the most likely scenarios of each project. Assuming a threshold of \( p \geq 0.6 \), the EA model presented in Figure 2(b) is created consisting of the origin EA model and the changes introduced by scenario 1. If we choose to create the EA model based on the most likely scenario, the resulting one will be the same.

### 3.3. Demonstration

To demonstrate the proposed approach, we implemented the aforementioned examples in the graph database Neo4j\(^1\). Additionally, we created the needed reports utilizing Cypher [15] as query language on the database.

Representing the origin EA model depicted in Figure 1(a) in a graph database was straightforward. We simply added all nodes and edges and initialized their existence property with 1.0. To represent points in time, as a project delivers two competitive scenarios, we added two special nodes. The first node expresses the time value itself (e.g., \( t_0 \) in Figure 3). The second node, linked to the introduced time node via the AT-relations, represents the current EA model (cf., \( M \) in Figure 3). As we also wanted to relate edges to the current EA model, we introduced additional nodes for each edge, because it is not possible to create an edge between an edge and a node. Each special “edge-node” inherits the existence property from its respective edge.

Based on this initial EA model representation, we introduced the two competitive scenarios from Figures 1(b) and 1(c) named \( A_1 \) and \( A_2 \) in Figure 3. First, we added a time node representing \( t = 1 \) (cf., \( t_1 \) in Figure 3) and linked it to the already present time node (\( t = 0 \)). Second, we copied the entire EA model related to the current EA node (i.e., \( M \) in Figure 3) and linked it to time node \( t = 1 \). Third, we added two nodes representing the two scenarios with the existence property equal to 0.6 respectively 0.4. This is necessary to allow for a rollback if a scenario gets dismissed, since we store just the computed existence value at each node and not how it was computed.

Last, we added the edges and nodes introduced by each scenario with their probability to the database, linked them

1. https://neo4j.com
to the related scenario node via the \textit{BELONGS}-relation, and recorded that they are added at the relation to ease a possible rollback. If a scenario retires a node or a relation, we remove it from \textit{M} and relate it to the appropriate scenario with a note that the entity should be retired. Furthermore, we reduce the \textit{existence} property by the probability of the scenario to become existent.

Adding further projects and their scenarios to the database works the same way. More interesting is the rollback of a scenario, because the complementary scenario has to be realized. First, we remove all entities related to the scenario which have been added. Second, we move the entities, which should have been retired, back to the current EA model and restore their \textit{existence} property. Third, we do the opposite with the realized scenario: move the added entities to the current EA model, set their \textit{existence} property to 1.0, and remove the retired entities.

So far, we have presented a concept to enhance an EA model with uncertainty and how to maintain such a model over time. But, how to generate a report representing the EA model in a “classical” way, e.g., to import the model into existing EA tools? This can be achieved by evaluating a Cypher query [15], shown in Listing 1, which generates a report incorporating no respectively a certain degree of uncertainty.

Listing 1. Querying for EA entities at \( t = 1 \) with \textit{existence} \( \geq 0.5 \)

\begin{verbatim}
1 MATCH
2 ( :TIME {time : 1})
3 -[:AT]-
4 ()
5 -[:BELONGS]-
6 (n)
7 WHERE n.existance >= 0.5
8 RETURN (n);
\end{verbatim}

To ensure that we only get EA entities to a specified point in time, the query asks for nodes labelled with \textit{TIME} and a property \textit{time} equal to 1 (cf. line 2). From this start point, we follow the relation \textit{AT} (cf. line 3) to all nodes representing the current EA model and all scenarios (cf. line 4). From these nodes, we follow the \textit{BELONGS} relation (cf. line 5) and end up at all entities of the EA model at the chosen point in time and store them into a variable \( n \) (cf. line 6).

Now, we apply a filter to ensure that we select only entities we are interested in, i.e., entities with a probability of \textit{existence} greater or equal than the defined threshold 0.5 (cf. line 7). Last, we return the collected and filtered entities (cf. line 8).

Another and more sophisticated procedure to create such a report is not to rely on the \textit{existence} probability of each entity, but on the most probable scenario (see Listing 2). The query is comprised of three parts. The first part (cf. lines 1 to 6) retrieves the current EA model at the time point 1. Line 7 passes the results of the first part to the second part (cf. lines 8 to 11), which collects all scenarios at time point 1. Next, the maximum value of all \textit{existence} values along all scenarios is calculated. This value together with the current EA model is handed over to the last part in line 12. Last, all entities related to the most probable scenario (cf. lines 13 to 18) are collected and returned together with the current EA model (cf. line 19).

Listing 2. Querying for EA entities at \( t = 1 \) with most probable scenario

\begin{verbatim}
1 MATCH
2 ( :TIME {time : 1})
3 -[:AT]-
4 ( :CURRENT)
5 -[:BELONGS]-
6 (c)
7 WITH c
8 MATCH
9 ( :TIME {time : 1})
10 -[:AT]-
11 (s : SCENARIO)
12 WITH c, max(s.existance) as max
13 MATCH
14 ( :TIME {time : 1})
15 -[:AT]-
16 ( :SCENARIO {existence: max})
17 -[:BELONGS]-
18 (e)
19 RETURN (c), (e);
\end{verbatim}

4. Discussion

According to Shaw [20] examples are a proper technique to discuss artifacts produced in software engineering. Consequently, we discuss to what extent our proposed approach answers the stated research question represented by its four facets compared to \( \text{P}^2 \text{AMF} \).

The first facet of our research question covers the need to represent uncertainty regarding the entities within the EA model itself. In \( \text{P}^2 \text{AMF} \) this is considered by adding an existence property to the model elements and assigning it a probability. Further, in an instantiation of a \( \text{P}^2 \text{AMF} \) model there is still no uncertainty regarding the existence anymore. Similarly, we have added an existence property to edges and nodes, but we still have uncertainty in our EA model instances.

The second facet expresses the need for a representation of the evolution of the EA model over time. This is not present in \( \text{P}^2 \text{AMF} \), since one model only represents one state. However, the probability may be facilitated to express the behavior of a certain entity along a time series. In contrast, our approach links different evolution alternatives of an EA model along the time to each other. Therefore, we create an actual representation of the EA model at each point in time where it is somehow altered.

The third facet requires the ability to manipulate the model. It covers, on the one hand, adding and removing scenarios to an EA model and, on the other hand, the handling of different versions along a certain time series. The beforehand stated requirement is not explicitly covered within \( \text{P}^2 \text{AMF} \), since it neither contains competing scenarios.
nor different versions. However, there is no need to cover this requirement in P²AMF, because there are no dependencies among the included entities, which would raise the need for such a rule set. Whereas, we described beforehand how to fulfill this requirement both in theory and in practice.

The fourth facet incorporates the necessity to create a report without uncertainty. As the instantiation of P²AMF does not contain any uncertainty regarding the existence of its entities, this instantiation can be utilized as the required report. In opposition, we suggest two different ways to extract a representation of the EA model at an arbitrary point in time. This is more sophisticated, but it is more flexible regarding to the stakeholders’ needs.

5. Related Work

There is still research elaborating on uncertainty in models. A first approach is to deny uncertainty and to merge different variants of one model. This problem has been researched for some years in the context of model driven software development. For instance, Välja et al. [10], [11] try to estimate the trustworthiness of the sources by facilitating techniques from the human-computer interaction [28] and data fusion [29] domain. Therefore, they build their means upon a model of information processing automation. This model is comprised of the phases of data acquisition, data analysis, decision and action selection, and implementation.

Another concept to handle contradictions is to prevent them before they emerge. In the case of a human source, one can try to identify possible weaknesses in the model and tell the modeler about. The authors of [12] develop a holistic framework to assess the quality of EA models. Their methodology can guide enterprise architects in their work and, thus, avert contradictions.

In contrast, Johnson et al. [18] do not neglect uncertainty but represent it in models. They interpret modelling of EAs as a probabilistic state estimation problem [30]. For solving this problem, they propose to facilitate Dynamic Bayesian Networks [23]. Therefore, they observe a computer network and use a Dynamic Bayesian Network to predict the likeliest representation of the EA’s technology layer.

As the beforehand discussed approaches concentrate on representing the as-is state of a model, Johnson et al. [14] elaborate on uncertainty in a predictive manner by introducing the Predictive, Probabilistic Architecture Modeling Framework (P²AMF). P²AMF is based on the Object Constraint Language (OCL) [31] and introduces uncertainty to class and object diagrams in two ways. First, attributes may be stochastic. Second, objects and the relations among them may be uncertain. To realize the second aspect, a stochastic existence attribute for classes and relations becomes mandatory.

6. Conclusion

EA models are a central artifact of EA, which provides a holistic view on the organization and eases the value creation for EA’s stakeholder. As EA collects its data from different sources, the gathered data can be contradictory. This work contributes to existing research by proposing a novel approach which does not try to solve the conflict but keeps the contradictory data.

In order to achieve this objective, we have refined P²AMF [14], which already incorporates a way to represent uncertainty regarding the existence of modelled entities. To ease the use of our technique, we generalized P²AMF from its UML/OCL notation to a graph presentation. Therefore, it can also be applied to EA models notated in arbitrary formats like ArchiMate [22]. Furthermore, we added competing scenarios and different versions along a time series to meet the requirements of a distributed EA evolution. To show the applicability of our approach, we utilized the theoretical described calculations and guidelines on a Neo4j graph database. Following, we argued that our realization meets the stated requirements of a distributed EA evolution.

So far, we have shown the applicability of our approach. Next, the approach should be included into existing tools for EA. If the tool allows defining attributes on model elements and their relations, the existence probability can be depicted easily. Same holds for modelling the competing scenarios and the different versions along a time series if the tool allows altering its meta-model freely, i.e., add new model element types and relationship types. To create the needed report, the tool needs to support free formulated queries. Especially, the last aspect is not easy to accomplish using the tools we know, since often some kind of scripting would be necessary.

Another way to enrich existing tools by the advantages of our approach is to patch a graph database in before the EA tool. The graph database would handle the uncertainty as presented in section 3.3 and an export would be generated which can be handled by the EA tool. In this case the database becomes the data master for the EA model. Therefore, all sources for the EA model have to be linked to the database which processes the data and delivers the results to the EA tool.

Besides the beforehand presented lack of integration into existing EA tools, our research encloses still some more limitation: We implicitly assumed a continuous evolution of the EA model. In other words, we expect that a new state added to our time series belongs to a point in time which is more in the future than all thus far persisted ones. Consequently, there are still no mechanisms which could handle a change of an included state or the addition of a state between two existing states. A possible option could be not to copy the current EA each time but to use symbolic links or to keep only the changes to the model and not the whole model.
References


