Living Modeling of IT Architectures: Challenges and Solutions

Thomas Trojer, Matthias Farwick, Martin Häusler, and Ruth Breu

Institute of Computer Science,
University of Innsbruck,
Innsbruck, Austria
{firstname.lastname}@uibk.ac.at

Abstract. Enterprise Architecture Models (EA Models) are documentations capturing the elements of an enterprise's IT infrastructure, setting these elements in relation to each other and setting them into the context of the business. EA Models are a crucial backbone for any IT management process and activities like analysing IT related risks and planning investments. The more companies depend on reliable IT services and use IT as innovation driver, the more high quality EA Models provide competitive advantage. In this paper we describe core challenges to the maintenance of EA Models based on previously conducted surveys and our longstanding experience in industrial collaborations. This is followed by a sketch of an innovative solution to solve these challenges.

1 Introduction

Enterprise Architecture Management (EAM) is an IT management process to describe, structure and plan complex IT systems in the context of the business. A core task within this process is to document the current state of business and IT infrastructure elements, e.g. business functions, software applications, servers, and to set these elements in relation to each other. The resulting *Enterprise Architecture Model* (EA Model) is usually very large in size, i.e. typically comprising several thousands of elements, and captures distributed knowledge of manifold stakeholders within the organization. There are a variety of tools for Enterprise Architecture Management off-the-shelf [22]. These tools typically support the documentation of architectural elements according to a given meta model and provide a set of representations, both of tree-like, graph-like or chart-like nature.

As we have shown in several surveys conducted with experts from industry [8,9], the quality of EA models in practice is an issue. Parts of this quality issue originate from organizational aspects, others stem from drawbacks of the available tools. One core drawback of available tools concerns the inflexibility of the EA Meta Model which does not adapt to grown terminology in organizations. A second drawback is deficiencies in the currentness of EA Models. EA Models which do not reflect the current state of the IT landscape may lead to wrong decisions on the management level. As two sources of this drawback we have been

R. De Nicola and R. Hennicker (Eds.): Wirsing Festschrift, LNCS 8950, pp. 458–474, 2015.

able to identify lacking automation capabilities and inappropriate user interfaces to maintain the model [7]. In addition, largely static visualizations of the EA Model in current tools are regularly mentioned to not adequately support the tasks of stakeholders like IT architects, operations staff and project managers.

In this paper we will start with a more thorough discussion of the challenges of tool support regarding EA models in the context of large IT infrastructures that are managed by geographically and organizationally distributed teams. These teams involve manifold stakeholders ranging from information officers, enterprise and IT architects to system administrators. This is followed by a presentation of possible corner stones for a solution to these challenges (see Section 3). Overall we call this modeling solution to be *living* in the respect that models and meta models can be maintained and visualized with a much higher degree of flexibility than in state of the art solutions.

We will demonstrate the materialization of the sketched solution within the novel EA modeling tool *Txture*[1] which has been developed by our team in the course of two industrial collaborations. Finally, we reference related work (see Section 4) and draw conclusions in Section 5.

2 Challenges

Typically IT management teams in enterprises have their own distinct terminology and levels of detail to document their IT systems and business functions. Hence, the need for flexible meta models is among the conclusions from conducted surveys. Flexibility means e.g. that the EA Meta Model which is underlying the architecture documentation needs to be customizable in order to fit the current information demand of an organization and its stakeholders. Over time these information demands usually change, e.g. due to the use of new technology or modified catalogues of provided business services. Hence, a properly usable EA Meta Model needs to adapt accordingly. This is in line with the work of Schweda [25], who also describes evolving organization-specific meta models as important and defines them as a requirement for successful EAM.

Once the EA Meta Model is aligned with the documentation requirements of an enterprise, a corresponding EA Model can be developed and describes the current state of the IT architecture and business assets. Modeling the EA is an incremental process and needs collaboration of a diverse set of stakeholders providing their knowledge about different enterprise aspects. While the enterprise transforms over time, the EA Model, in order to stay usable, needs to be adapted. Our experience has shown that enterprise stakeholders are often reluctant to keep the EA Model in-sync with reality, mostly because their documentation tools do not integrate well in their working environments. Recently we have described the use of text-based EA modeling [11], specifically tailored to support stakeholders with technical background. In general we argue that an EAM tool has to provide modeling support tailored towards the needs of its stakeholders. In particular this comprises consideration of both business-level

[1] See `http://www.txture.org`

stakeholders (preferring e.g. forms and charts) and technology-level stakeholders (preferring e.g. programming style or graphical representations).

Up-to-date EA Models are a prerequisite to EAM activities such as the analysis of current IT architectures and planning upcoming developments and projects. These analysis and planning steps require dedicated consideration and are commonly supported by a set of static visualizations providing specific views on the EA models [20]. Feedback received by experts from industry has shown that stakeholder-specific visualizations are to a large extent missing in current EAM tools. By stakeholder-specific visualizations we mean dynamic and navigable views, supporting an individual user to analyze the EA model. This is in line with trends in Business Intelligence, where such visualizations are also known and part of the *self-service* aspect[2].

Finally, an integral challenge to all of the aforementioned aspects is to maintain high computational performance of operations on top of EA Models. When dealing with large-scale EA Models optimized and automated adaptations based on a changed meta model, the browsing and querying of arbitrary EA Model elements and the creation of visualizations from the entire EA Model need to be accomplished in a timely manner. Otherwise, the usability of an EA modeling tool and the tool's acceptance by enterprise stakeholders may be heavily affected.

Overall, we summarize the EAM tool challenges we have outlined in this section as follows:

- Implementation of *flexible IT architecture modeling*
- Need for *stakeholder-centric modeling editors*
- Provision of *dynamic and navigable visualizations*
- *High performance of EAM operations* on top of large-scale EA Models

3 Solutions within the Living Modeling Environment Txture

In 2011 we started a consulting project with a banking data center and subsequently a research project with the data center of a large semiconductor manufacturer. The overall goal of both projects was to make the documentation of EA more efficient and effective. Enhanced flexibility as well as usability features and stakeholder orientation of the implemented tool were generally seen as important. In addition, requirements to support common EAM activities on top of an EA Model were considered.

The key features of the resulting modeling tool *Txture* are as follows:

- Modeling of the architecture via a form-based web-client to support less technically skilled users.
- Textual architecture modeling via a meta-model aware *Eclipse*-based text editor [11] provided to technical staff.
- Dynamic and flexible graph visualizations of EA Models.

[2] BI Survey '13, "The latest market trends", `http://barc-research.com/bi-survey/`

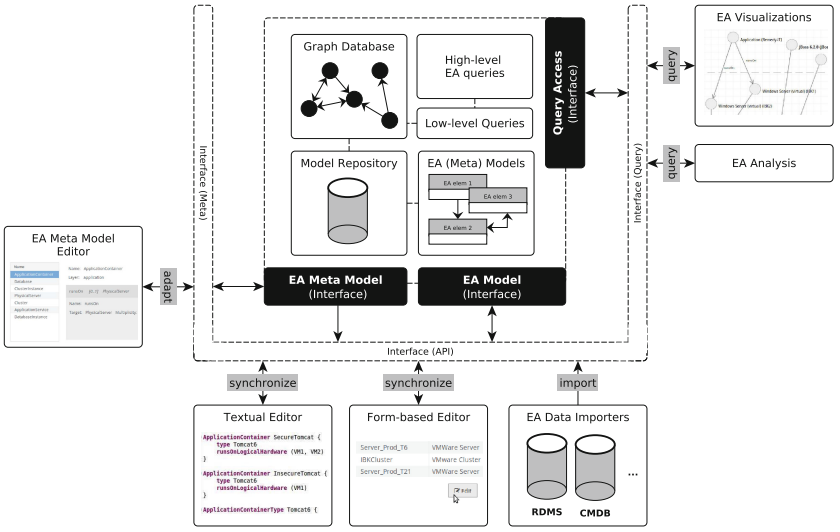

Fig. 1. The *Txture* environment showing the EA model persistence at its core and auxiliary components for EA management purposes

- High performance model queries via optimized persistence of models in a graph database.
- The ability to define and change the EA Meta Model at runtime.
- Configurable import mechanisms to automatically use architectural data contained in external sources such as in *Configuration Management Databases* (CMDB), *Excel* spreadsheets, source code or relational databases.

These key features are reflected in the architecture of the *Txture* modeling environment (see Figure 1).

Figure 2 depicts a graph-based architecture visualization (see top-most screenshot). There, relationships between *application containers*, an *application* and the underlying (clustered) hardware infrastructure are shown. Such visualization is used e.g. to perform impact and risk analysis of application deployments.

Several other key visualization features can be seen in the corresponding part of the figure:

- Architectural elements are assigned to *layers*, hence the visualization automatically shows an intuitive architectural stack.
- The visualization is *navigable* and *dynamic* via a set of modification operations (see the context menu depicted in the screenshot).
- Graph nodes are *styled* based on their type or other attributes, like mission-criticality (cf. the elements *VMWare Cluster* and *VMWare Server*).

Furthermore, Figure 2 shows the meta modeling capabilities via a form-based editor. This editor allows the user to change the EA Meta Model at runtime

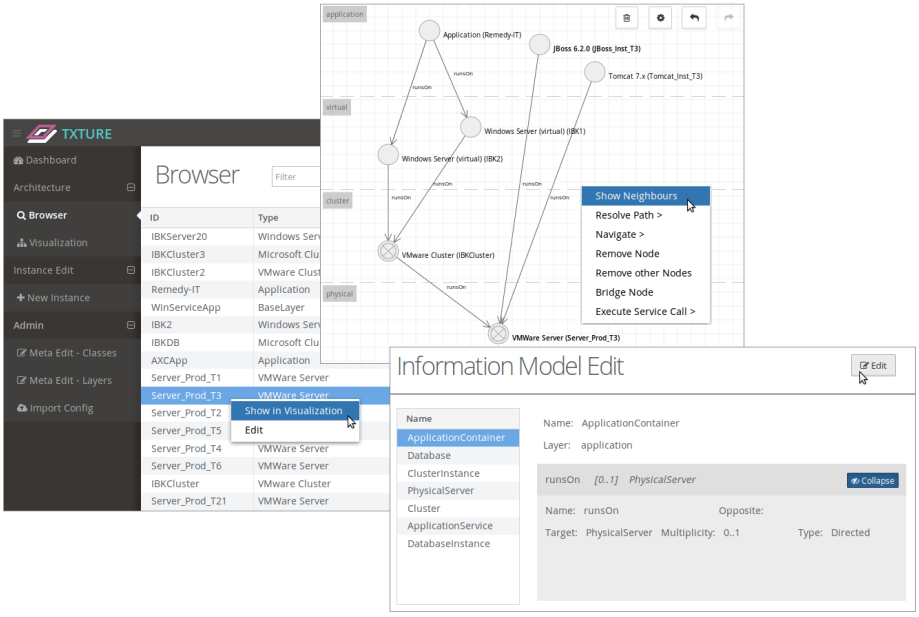

Fig. 2. The *Txture* environment showing the architecture browser (left screenshot), dynamic visualizations (top-most) and the ability to view and change the EA Meta Model (bottom-most)

which in turn directly influences the visualizations and the import configurations for mapping EA data of external data sources.

Finally, basic search and query functionality is implemented via an architecture browser and is indicated on the left side of Figure 2.

3.1 Modeling Framework

In this section we outline *Txture*'s employed modeling framework using a sample model (see Figure 3).

The EA Model in Figure 3 shows documented instances of IT system components. The example describes an application container instance "*JBoss_Inst_T3*" which "*runsOn*" a physical server named "*Server_Prod_T3*". As we have described in the previous section, such a model can be used e.g., to perform impact analysis ("What happens if the specific server crashes?") or to do infrastructure planning ("Is the specific server appropriately dimensioned to run such software?").

Additional to modeling IT component instances and their structural dependencies, a simple notion of *ontology* can be seen on the right side of the figure. Such ontological classifications are modeled as part of the documentation activity and allow responsible persons for EA elements to further describe and categorize their documented instances. In our example case, the application container instance is of *type "JBoss EAP 6.2.0"* which reflects a part of the enterprise's

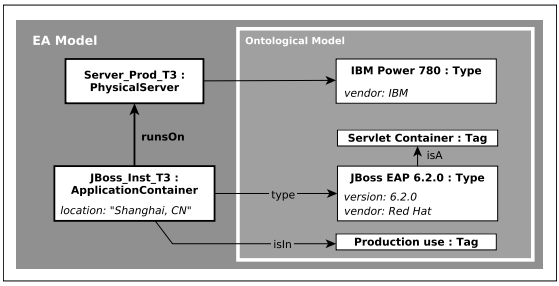

Fig. 3. A simple EA Model showing IT infrastructure elements

modeled ontology. Furthermore it is *tagged* with *"Servlet Container"* to indicate its relatedness (*"isA"*) to *Java servlet* technology. Ontologies in EA Models are established to introduce enterprise-specific terminology (e.g., by means of employed technology), but are also used in *Txture* to enhance browsing, search and filter functionality.

Figure 4 provides an extended view of our example model by including its corresponding meta-model hierarchy. On the EA Meta Model level, the expressiveness of the underlying EA Model is set. At this level the structure of a EA documentation that architects agreed upon is modeled.

The top-level artifact, the meta-meta model, defines all concepts that are needed to properly describe IT infrastructures. The meta-meta model defines the concepts of *class*, *association* (i.e. association classes) and *property* to develop the structure of an organization-specific architecture modeling language and the concepts *type*, *tag* and *mixin* that allow shaping the ontological model.

Classical Hierarchies to Separate Modeling Activities. One of the experiences we gained from modeling workshops with our industry partners is that modeling novices or software developers understand modeling best when using strict and limited hierarchies in which modeling concepts and their instantiations are described. In our case the modeling levels that users have to interact with are manifested by the EA Meta Model and the EA Model as its instantiation.

Besides understandability of concepts, having a clear cut between modeling levels also supports a permission and concern-oriented separation for managing the EA documentation and the meta model it relies on. This separation is important as different modeling activities are performed by individual stakeholders with potentially diverse domain expertise. This is further explained in Section 3.2.

Types to Mitigate Invasive Meta Model Changes. Another experience we made was that adapting the EA Meta Model is typically a recurring activity, triggered by frequent change requests from our partners and driven by adjustments, extensions and simplifications to modeled concepts.

It is common to any modeling activity, that changes to models may involve corresponding changes on dependent models, as part of re-establishing conformance in the model hierarchy. To minimize the efforts and consequences of such

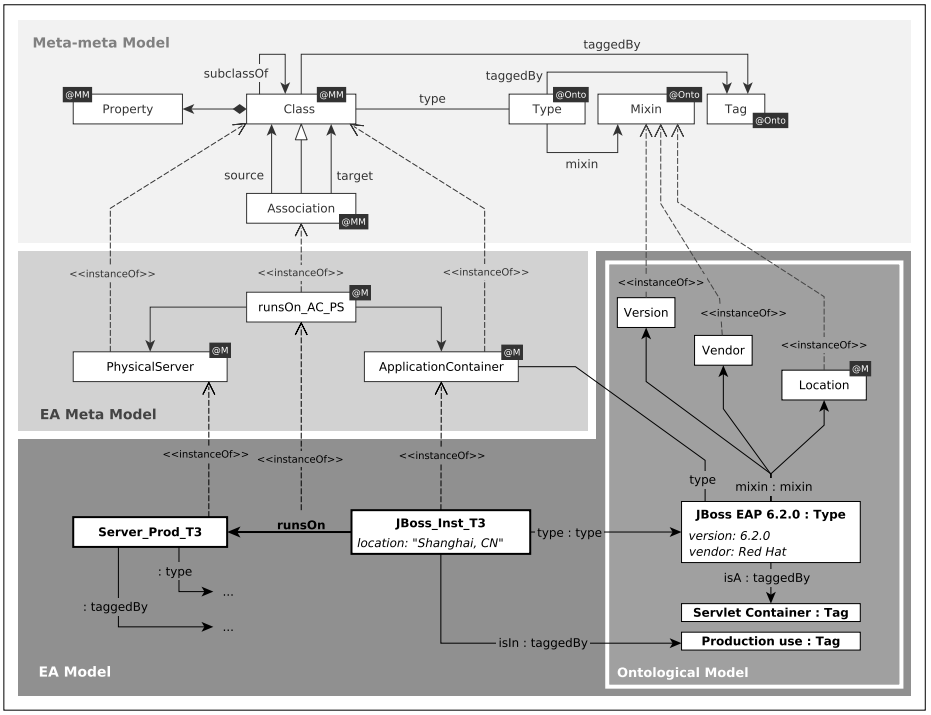

Fig. 4. The *Txture* modeling environment. Annotation boxes (black) reflect where a model element gets instantiated (@MM = EA Meta Model, @Onto = Ontological model and @M = EA Model)

changes, either well-defined automated model refactoring procedures are required or a meta model needs to be realized in a way such that the most common changes to it only minimally interfere.

For our industry partners a manual refactoring after changes to the EA Meta Model was out of question. This is why we settled on a modeling pattern similar to the one of *power types* [23] that allows for creating types at the EA Model level and therefore reduces the need to actually adapt the related meta model.

Our original modeling approach made heavy use of inheritance on the meta model level. For example we applied a deep inheritance structure to model different *Application Containers* according to their *vendor*, *software version* or required *runtime platform*. This rendered the meta model both large in size (i.e. number of model elements) and prone to frequent changes (e.g. on software version changes).

Using *types* greatly helped to reduce the size of the meta model and therefore maintaining comprehensibility and lowering the frequency in which changes to it needed to be applied. A modeling environment that allows types, can rely solely on generic meta model elements like *Physical Server* or *Application Container* and therefore provides stable modeling concepts that are *invariant* to an organization and all of its stakeholders. This means e.g. that no highly-specific vendor-based

product terminology would be described within the EA Meta Model which would only be understood by a minority of the enterprise's stakeholders and which is likely to change over time (cf. *JBoss*-specific server software in the example of Figure 3).

Our understanding of types, as part of the ontological model, is that adjustments to them can be easily applied during the regular EA documentation processes. This is in line with Atkinson and Kühne [3], who describe the need for changes and newly added types that are possible while the system is running. Our type concept delivers a light-weight way for dynamic additions and proved to be intuitively usable in EA documentation practice.

In addition to types, we use *tags* to further categorize model elements. Tags are comparable to *UML stereotypes*[3] and can be applied to types and individual instances. In *Txture* both type and tag elements are modeled by responsible persons for EA elements and are part of the EA Model.

Multi-Level Instantiation to Support Dynamic Extensions. With the introduction of types on the EA Model level, we are able to limit the amount of changes that otherwise are applied to the meta model. While this is beneficial, maintaining an EA Meta Model of only generic concepts bares issues regarding the expressiveness of the documentation: Generic EA Meta Model concepts leave out detail and shift the specification of properties of model elements onto types.

Our documentation activities require that types and instances can be managed by the same stakeholders within the EA Model. For proper architecture documentation, types not only define properties to be instantiated by their related instances, but need to specify values for certain properties themselves.

Figure 4 shows that the *JBoss*-example type defines values for the properties *version* and *vendor*, whereas our example application container defines a text value reflecting its deployment *location* to be "Shanghai". In our example we assume this property to be dependent on the actual type, as e.g., not for all application containers the location is known or relevant to be documented. Because of this, we needed to realize a property-like concept, so called *mixin*s, that can be instantiated on both the level of types and the level of documented instances. This is comparable to the concept of *deep instantiation* [2] or that of *intrinsic attributes* in the *MEMO* meta-modelling language [14].

The mixin concept aligns well with the flexible nature of our type concept and allows the documenting stakeholders to adapt the EA Model to cater for their particular documentation needs.

3.2 Stakeholder-Centric Editors

A key challenge in the context of EAM is to cater for the many different stakeholder types that are typically involved in editing the EA model. These range from database administrators and software developers to enterprise application architects, to process owners, project managers and even the CIO in some cases.

[3] cf. *UML 2.4.1* infrastructure specification, `http://www.omg.org/spec/UML/2.4.1/`

It is clear that each of these stakeholder types has different requirements when it comes to proper user interfaces. As we have described in our previous work [10], one problem in the EA management process is that users are often reluctant to enter data into an EA tool because of the time overhead involved. One reason for this problem are the potentially diverging conceptions between developers of an EA tool's features and its eventual users.

Following from this, we argue that adequate user interfaces for the different stakeholder groups can mitigate this problem by reducing the barriers for stakeholders to document the EA. Along this line we previously presented an approach to enter EA data in a textual way [11]. Our experience has shown that the textual editing approach generally works well for technical staff that is accustomed to work in text-based environments such as they are used for programming (e.g., via *Eclipse*[4]) or systems configuration (e.g., of databases or server applications). In other cases it might be more appropriate to let users enter data via simple form-based applications. Finally there are also users that commonly work with standard office applications like spreadsheets (most commonly *Microsoft Excel*).

In the following, we provide some detail on the different stakeholder-oriented editing functionality that we have implemented in *Txture*. The specific challenges that all EA model editors have in common is that they need to seamlessly cope with a changing underlying meta model and the multi-level modeling concepts like runtime-added types and attributes (cf. Section 3.1).

Textual Modeling Editor. While working with our first industry partner, we implemented textual editing of EA models. In a number of interviews with a variety of technologically educated stakeholders, we learned that text-based tools are commonly used by them. We decided to implement a textual editor for EA management in order to yield a high level of acceptance in this specific user group.

The editor (see Figure 5) was developed to be accessible within the *Eclipse* development environment and builds upon the textual modeling framework *Xtext*[5] in order to offer sophisticated textual editor functionality and user assistance out of the box. Visual support is provided via font- and color-based highlighting of known syntactical elements, including EA Meta Model classes, attribute names and types. The so called outline view (see the right part in the figure) delivers a navigable tree view that lists all described elements in a compact way. Besides the regular in-text search functionality, the outline view can be used to quickly overview the entire documentation and search for specific elements.

Beyond visual appeal and standard text editor functionality, our EA model editor also provides advanced features like automatic text completion for known syntax, error highlighting on failed model validations and the ability to insert placeholder text templates to help documenting new EA elements.

In a previous work [15] we have demonstrated textual modeling challenges, specifically by taking collaborative modeling efforts into account that involve the

[4] `http://www.eclipse.org`
[5] See `http://www.xtext.org`

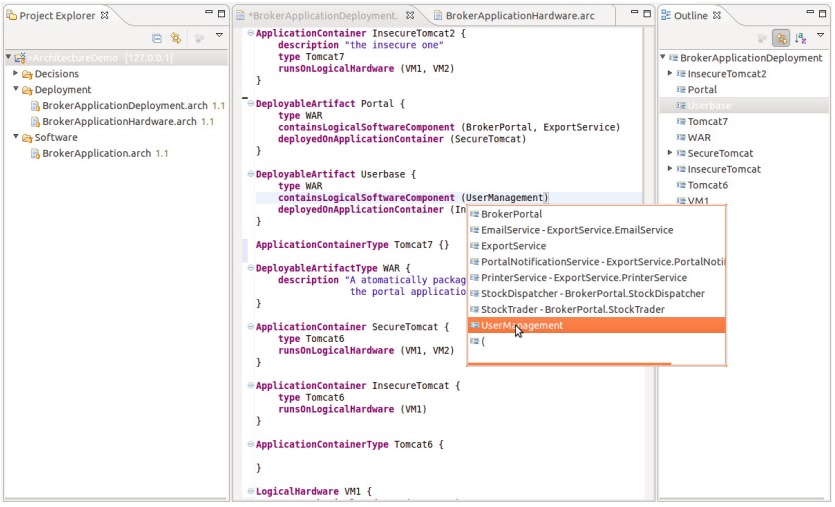

Fig. 5. The textual editor as *Eclipse* plugin with file management, syntax highlighting, automatic text completion and outline support, developed with the *Xtext* framework

use of non-text based modeling editors as well. The main discrepancies between these two natures of editing are regarding the representation and persistence of EA model data. While models are commonly stored in a way so that only dedicated modeling tools can open and modify them (cf. XML-based persistence via e.g., *XML Metadata Interchange* (XMI)), any text editor can be used to work with a textual representation of models. Still, specific methods are required in order to manage the necessary file and folder based persistence of textual model parts and strategies which help to translate back and forth between text and other EA model representation formats. Considerations on e.g. the order of elements in text files or the storing of textual user comments had to be made and led to the requirement of maintaining *extensional information* about EA models.

Form-Based Modeling Editor. In addition to textual editing, *Txture* provides web-based forms to conveniently allow management of data by users with less technical background. There, the typical user interface elements like text fields and combo boxes are used to maintain attribute values of EA elements and cross-references between them.

Similar to the textual editor that provides syntactical keywords based on the currently employed EA Meta Model, the form-based editor is dynamically generated to reflect all available elements and their valid structure.

This type of editor is directly integrated as a web-based application within the *Txture* environment which we have shown in Figure 2.

Other Modeling Editors. To cover the entire range of stakeholder types for EA documentation, additional modeling editors can be considered for implementation. E.g. in a previous work [15] we have described our current efforts about

an easy-to-use modeling extension for *Excel*. By now we have implemented a first prototype which renders documented EA elements together with their attributes and references into the cells of a spreadsheet. Such an editor is helpful to support business stakeholders and to integrate data that pre-exists in spreadsheet tables as external data sources.

Another current effort is to use *Java* code annotations to allow software developers to indicate a rough underlying software architecture. Such annotations get processed by a code analyzer and are fed into the EA documentation as well. The banking data center that we work together with, also operates a large software development department which established the use case of annotation-based modeling capabilities.

Our experience over the last years has shown the importance of first determining typical stakeholder tools and trying to adapt them, prior to making plans for custom tool developments. We have seen that users more easily accept and learn new functionality provided by familiar tools, as opposed to operating entirely new tools. Learning new tools, besides having to execute one's daily working activities, is often perceived as cumbersome and may in turn lead to an abandoned EA documentation.

3.3 Dynamic Architecture Visualizations

Architecture visualizations constitute a key reason why EA models are created. They are the means to reduce the architectural complexity and make potential problems visible to the persons responsible. The main challenge for EA visualizations is to present large models in a way that only the relevant information regarding a specific EA or IT architecture question are shown.

A typical approach is to allow users to pre-configure visualizations in a form-based manner and then generate graphical representations from this view-definition [24]. We argue that the roundtrip between configuration and the generation of the visualization presents a hurdle for the efficiency of creating adequate visualizations. In *Txture*, visualizations can be both created from a selection of EA elements or from a pre-defined view definition, but also edited dynamically from within a given visualization. Some of the editing functionality is shown within the top right screenshot in Figure 2, visible as the context menu that contains several options for editing the current visualization.

During the EA projects with our industry partners we gathered a number of requirements that useful architecture visualizations need to implement. Accordingly, visualizations should

- be able to represent EA model elements in different ways,
- be easily navigable in order to make the architecture's structure understandable,
- implement filter mechanisms to allow simplifications of the visualization and
- be visually extensible (e.g., via visual groups and separators derived from extensional EA model information), hence providing additional meaning to what is depicted.

The actual types of visualizations and the way EA model elements are represented are numerous and need to be adapted to the requirements of certain user groups. For instance, we interviewed system administrators who declared *treemap*-based visualizations as helpful in order to quickly determine *runs on* or *hosted by*-relationships between server applications and virtual systems that run on top of physical hardware. Software developers and IT systems architects felt comfortable with graph-based visualizations or a mixture of treemaps and graphs. With treemaps, the typical containment relationships are reflected, whereas a graph allowed them to determine system communication paths, e.g. implemented via services. By contrast, project managers and business-oriented stakeholders were interested in matrix or list-based representations of EA model elements. These stakeholder groups were mostly only interested in visualizing types that occur in an architecture, but no specific instances. We were told that this would allow them to get an overview of the employed technology stack or to make abstract business processes visible.

In the current version of *Txture*, navigation within visualizations is possible due to a number of operations. For example, the *show neighbours*-operation helps to explore the neighbourhood of a given model element by showing all of its directly related elements (via EA model cross-references). The *navigate*-operation allows to insert directly related model elements into the current visualization, by choosing a specific relationship of interest. Finally the *resolve path* functionality enables a user to resolve arbitrary dependencies of a selected model element to all elements of a specific type or class. This operation is intended to show transitive dependencies between EA elements. E.g. one could select a specific application and resolve all physical hardware that this application relies on.

Filter operations applied to current visualizations are a helpful tool to simplify what is depicted. E.g., we implemented the *removal*, *grouping* and *bridging* of EA model elements. The removal operation, as its name implies, deletes elements from the current visualization in order to simplify them if unnecessarily loaded. Grouped elements are visualized as a single node within the visualization. A label for the replacing group node is either automatically generated or can be defined manually. Groups can also be dissolved via an inverse *ungroup*-operation. Lastly, the bridging of nodes allows to transitively skip arbitrary model elements in the visualization. The skipped nodes are replaced by new relationships that are either labeled automatically or receive custom names. The purpose of bridging is to lower visual complexity by means of raising the abstraction level.

In order initiate new or extend current visualizations, an *adding*-operation enables a user to insert a selection of documented EA model elements. If requested, any direct dependencies to already visualized elements are shown as well.

Additional to these operations, other functionality is planned as well. Current efforts include the implementation of the aforementioned visual extensions by means of visual groups to mark arbitrary collections of elements. This will be done with the help of colors, separating boxes and custom labels.

One of the greatest challenges we encountered while implementing our visualization components was to keep the runtime performance of the described operations high. To us, this highlighted the need to establish an efficient, performance optimized model query framework.

3.4 Efficient Querying of Large EA Models

Querying EA models is especially important to perform analysis and to select EA model elements along with certain criteria. Results of such model queries are typically interpreted by enterprise stakeholders. Furthermore, query results are the basis of *Txture*'s visualizations.

Two main requirements guided our design decisions regarding a query framework. Namely, high performance in obtaining query results and access to a query expression language that is easy and intuitive to use.

In order to find out about the technology that best caters our requirements, we performed a number of benchmarks with different query frameworks. E.g., the *Object Constraint Language*[6] (OCL) and *EMF Model Query*[7] have been used. We were dissatisfied with all of the tested frameworks, regarding performance results or the high complexity as well as the low expressiveness of the query languages they offer.

We finally decided to create a query framework based on a graph persistence to store the structure of an EA model (cf. Figure 1, central part). Regarding performance, this decision reflects the choice of e.g. Barmpis and Kolovos [5], who evaluated graph database to be fastest for querying, out of a number of other model query and persistence approaches. Additionally, a regular indexed data container (in our case a relational database) is employed and holds the actual data of all model elements. As graph databases are typically capable of storing vertices and edges as well as properties for both of these entities, we found that these graphs are able to resemble the nature of EA models well.

The graph database we use is called *Titan*[8] and the query language it supports is *Gremlin*[9] . *Gremlin* is a highly sophisticated graph traversal language that is widely supported by current graph database systems. With it we were able to mitigate any performance issues while expressing queries. Nevertheless, its complexity would have not allowed any regular users of *Txture* to take advantage of its capabilities. Therefore we established an extensible set of high-level queries that build upon complex graph queries, but provide a simple interface to users. E.g., each of the visualization operations described in Section 3.3 is implemented as such a high-level query.

The graph-based mapping of EA models, low-level graph-based querying together with the layer containing the high-level queries is depicted as part of Figure 1 (see the right side of the core part of *Txture*'s architecture).

[6] See `http://www.omg.org/spec/OCL/`
[7] See `http://www.eclipse.org/modeling/emf/?project=query`
[8] See `http://thinkaurelius.github.io/titan/`
[9] See `https://github.com/tinkerpop/gremlin/wiki`

4 Related Work

This paper presents an overview of our experience in EAM, the *Txture* tool as well as a diverse set of challenges in the field. Accordingly, related work is similarily diverse. We start its discussion by naming advancements in three research fields that made the development of *Txture* possible. These are:

Advancements in Model-Driven Software Engineering. Runtime changes of the underlying EA Meta Model and the consecutive adaptation of an EA model is a complex problem. With the increased adoption of model-driven software development this problem has received considerable attention in research literature (see e.g. Favre [12]). In addition, allowing to model on multiple modeling layers, such as it is required in the context of EAM is another challenge. In particular, the work of Atkinson et al. [3] has helped in forming a better understanding of the problems of standard modeling languages such as the UML. Also, work on textual domain-specific languages (like *Xtext* is used for) has contributed to the development of the textual modeling editor of *Txture*.

Proliferation of Graph-databases. The already mentioned size of practical models in the EA context requires efficient methods for querying and storing models. Graph databases have recently gained much attention because of their utility for the use in social media applications and also other areas (an overview is given by Angles and Gutierrez [1]). Fortunately, this resulted in the development of several open-source, quality graph databases that are particularly useful for querying EA model element relationships.

Advancements in Web-Engineering for Visualizations. A key-factor for the utility of EA models are their visualizations. Building flexible client-side visualizations for web-applications was, until recently, limited by the lack of standards and accompanying technologies. With the adoption of new standards (like *HTML 5*[10]) by most modern browsers, major obstacles were removed, leading to sophisticated graphing and drawing libraries for the web.

In the context of EAM it is common that tools provide predefined EA Meta Models that can often only be adapted in a very limited way. For example, the EAM tool *iteraplan*[11] only allows for the extension of existing classes via attributes. As shown in the EAM tool survey by Matthes et al.[22] there exist some configurable tools, their technical foundation, however, is not clear. Other tools work with fixed EA Meta Models based on EA modeling standards such as *The Open Group Architecture Framework* [16] or *Archimate* [21]. We argue that these standards are inflexible as it is difficult to adapt them to the terminology used in an organization or to evolve. Schweda, on the other hand, presents a sophisticated approach to pattern-based creation of organization-specific meta

[10] cf. `http://www.w3.org/TR/html5/`
[11] `http://www.iteraplan.de/en`

models [25]. However, its practical applicability was not shown. With the *MEMO* meta-modeling language, Frank et al. [14] present a language and a tool suite for building modeling languages in the enterprise context. The tool is Eclipse-based and needs code generation steps in order to react on a changed meta model. The proposed language for IT infrastructure modeling, *ITML* [13], provides fixed concepts and can not support organization-specific meta models. Additionally, we found that some of the complex virtualization and clustering patterns that we have witnessed in practice cannot be modeled with this approach. In line with Kattenstroth [17], we conclude that although the need for organization-specific and evolving EA Meta Models has already been identified in literature [10,25], most related work focus on formulating generic and fixed meta models that cannot be adapted to the requirements of specific organizations.

Despite the existence of many commercial EA tools on the market, their capabilities for flexible visualizations are rather limited. A relatively recent development in the area of EA visualizations is to separate the model from the visualization unlike e.g. *Archimate* which makes use of a graphical modeling notation. This separation is suggested in several research works [24,18,6].

In the general model engineering community much groundwork has been laid. Recently, the multi-level modeling paradigm gained more attention due to the criticism of classical (two level) modeling, like done e.g. in the UML which only allows a model and an instantiation at the same time [4,19]. This paradigm has influenced the meta-modeling capabilities of *Txture*, in particular, by providing mechanisms to model types and mixins. Still, multi-level modeling mainly discusses requirements from software engineering and does not necessarily consider modeling techniques from other domains. Regarding our work on *Txture* we use a mixture of classical and multi-level modeling approaches and unified them in a novel way to contribute a usable EA documentation method.

5 Conclusion and Outlook

In this paper we have described the EA modeling framework underlying our research prototype *Txture*. It provides a unique feature set including classical meta-modeling, type-based modeling and mixins and tackles some of the pressing problems of EA and IT systems documentation.

As our research elicites requirements from practical experience, we believe that our work can be useful for other EA researchers as well, but also for vendors of existing EA tools.

Challenges we specifically discussed are:

1. The difficulty to adapt EA Meta Models at runtime which we tackle with a combination of multi-level modeling techniques and classical approaches like stereotyping and power typing.
2. Issues regarding dynamic and navigable visualizations that entail the problem of efficient queries over large EA Models. We solve this by using a graph-based model persistence together with a layer of high-level EA Model queries.

3. The requirement to be able to edit EA Models by considering preferences of different stakeholder groups. We solve this by implementing model editors that either extend existing tools or align custom editors with the requirements named by their prospective users.

In our future work we aim to further evaluate our approach in practice and conduct empirical studies that will assess to what extent our approach assists and motivates different stakeholder groups to contribute to EA documentation processes. So far, textual editing and dynamic visualizations have already shown their usefulness at work for both of our industry partners. In the banking data center enterprise architects, software developers and DevOps teams document their work in the EA model, without having to change tools. The semiconductor manufacturer uses pre-defined architecture visualizations as a starting point for impact analysis of systems in their data center. In discussions, users have confirmed the value for them to be able to define their own custom visualizations that support their daily working activities.

Recent developments in the fields of Model-driven Software Development, graph databases and web-engineering have made the development of the presented framework and prototypical tool implementation possible.

References

1. Angles, R., Gutierrez, C.: Survey of graph database models. ACM Computing Surveys (CSUR) 40(1) (2008)
2. Atkinson, C., Kühne, T.: The essence of multilevel metamodeling. The Unified Modeling Language. Modeling Languages, Concepts, and Tools (2001)
3. Atkinson, C., Kühne, T.: Model-driven development: a metamodeling foundation. IEEE Software 20(5) (2003)
4. Atkinson, C., Gerbig, R.: Harmonizing Textual and Graphical Visualizations of Domain Specific Models Categories and Subject Descriptors. In: Proceedings of the Second Workshop on Graphical Modeling Language Development. ACM (2013)
5. Barmpis, K., Kolovos, D.: Evaluation of contemporary graph databases for efficient persistence of large-scale models. Journal of Object Technology, JOT (2014)
6. Buckl, S., Ernst, A.M., Lankes, J.: Generating Visualizations of Enterprise Architectures using Model Transformations.. Enterprise Modelling and Information Systems Architectures 2(2) (2007)
7. Farwick, M.: A Situational Method for Semi-automated Enterprise Architecture Documentation. Ph.D. thesis, University of Innsbruck (2014)
8. Farwick, M., Berthold, A., Breu, R., Ryll, S., Voges, K., Hanschke, I.: Requirements for Automated Enterprise Architecture Model Maintenance. In: International Conference on Enterprise Information Systems (ICEIS). SciTePress (2011)
9. Farwick, M., Breu, R., Hauder, M., Roth, S., Matthes, F.: Enterprise Architecture Documentation: Empirical Analysis of Information Sources for Automation. In: Hawaii International Conference on System Sciences (HICSS). IEEE, Wailea (2013)
10. Farwick, M., Schweda, C.M., Breu, R., Hanschke, I.: A situational method for semi-automated Enterprise Architecture Documentation. SoSyM (2014)
11. Farwick, M., Trojer, T., Breu, M., Ginther, S., Kleinlercher, J., Doblander, A.: A Case Study on Textual Enterprise Architecture Modeling. In: Enterprise Distributed Object Computing Conference Workshops (EDOCW), IEEE (2013)

12. Favre, J.M.: Meta-model and model co-evolution within the 3d software space. In: Workshop on Evolution of Large-scale Industrial Software Applications (2003)
13. Frank, U., Heise, D., Kattenstroth, H., Ferguson, D.F., Hadar, E., Waschke, M.G.: ITML: A Domain-Specific Modeling Language for Supporting Business Driven IT Management. In: Proceedings of the 9th OOPSLA workshop on domain-specific modeling (DSM). ACM (2009)
14. Frank, U.: The MEMO meta modelling language (MML) and language architecture. 2nd Edition. Tech. rep., Institut für Informatik und Wirtschaftsinformatik (ICB) Universität Duisburg-Essen (2011)
15. Haeusler, M., Farwick, M., Trojer, T.: Combining textual and web-based modeling. Submitted to 16th IEEE/ACM MODELS (2014)
16. Haren, V.: TOGAF Version 9.1. Van Haren Publishing (2011)
17. Kattenstroth, H.: DSMLs for enterprise architecture management. In: Workshop on Domain-specific modeling (DSM). ACM Press (2012)
18. Kruse, S., Addicks, J.S., Postina, M., Steffens, U.: Decoupling models and visualisations for practical ea tooling. In: Service-Oriented Computing. ICSOC/ServiceWave 2009 Workshops (2010)
19. Kühne, T.: Matters of (Meta-) Modeling. SoSyM 5(4) (2006)
20. Lankes, J., Matthes, F., Wittenburg, A.: Softwarekartographie: Systematische darstellung von anwendungslandschaften. In: Wirtschaftsinformatik (2005)
21. Lankhorst, M.: Enterprise Architecture at Work, 3rd edn., vol. 36. Springer, Heidelberg (2012)
22. Matthes, F., Buckl, S., Leitel, J., Schweda, C.M.: Enterprise Architecture Management Tool Survey 2008. Tech. rep., Technische Universität München, Chair for Informatics 19, sebis (2008)
23. Odell, J.J.: Power Types. Journal of OO Programming (1994)
24. Roth, S., Hauder, M., Zec, M., Utz, A., Matthes, F.: Empowering Business Users to Analyze Enterprise Architectures: Structural Model Matching to Configure Visualizations. In: International Enterprise Distributed Object Computing Conference Workshops (EDOCW). IEEE (2013)
25. Schweda, C.M.: Development of Organization-Specific Enterprise Architecture Modeling Languages Using Building Blocks. Ph.D. thesis, Technical University of Munich (2011)