Visualizing Differences of Enterprise Architecture Models

Sascha Roth
Software Engineering for Business Information Systems (sebis), TU München,
Boltzmannstr. 3, Garching bei München
roth@tum.de

Florian Matthes
Software Engineering for Business Information Systems (sebis), TU München,
Boltzmannstr. 3, Garching bei München
matthes@in.tum.de

***Abstract*—Enterprise Architecture (EA) models are structured, object-oriented models that typically conform to an organization-specific, i.e. customized, meta-model that evolves over time. Enterprise architects commonly use different branches of such an EA model to plan future states of an EA with respect to their origin–the current state of an EA. It is particularly interesting for enterprise architects to analyze differences of planned states to this current state of an EA. Based on these differences they can derive projects as means to carry out changes in order to realize planned states as temporally limited end. In our previous work, we focus on the evolution of models and meta-models, model differencing, merging, and conflict detection and resolution. A particular challenge we observed is the communication of model differences. We diagnose that as of today, no common standard to visualize and analyze differences in models and in particular Enterprise Architecture (EA) models has been established. In this paper, we present a four-layered conceptual design of an interactive visualization to drill down and analyze model differences in meta-models (schema) and respective models (data). Our design copes with the complexity of an EA model and provides mechanisms to filter particular parts of an EA model. We reveal implementation details of the concept, discuss end-user feedback on our prototype and point out some known limitations of the approach.**

Keywords—visual differences; models; differencing; interactive visualization; Enterprise Architecture

I. Introduction and Motivating Example

Visualizations provide means to communicate complex facts [1, 2]. In Enterprise Architecture (EA) management they have been established as a common means to communicate the current state of an EA as well as transitions towards a desired target state [3]. Although many visual variants have been applied to software engineering problems such as software maintenance and understanding of refactoring [4], in particular 3 dimensional visualizations seem to be unsuitable to communicate information in EA management [5]. At the same time model conflicts [6] and concurrent evolution [7] play an important role in EA management as well as in general software models [8].

Fig. 1 depicts different states of an EA over time. Within this paper, let us assume that an EA model always conforms to an EA meta-model, also called EA information model. We further assume that both, the model as well as the meta-model co-evolve. EA management commonly uses branches of an EA repository (model, i.e. data, and its meta-model, i.e. respective schema) to derive planned states that are later consolidated, may get executed, or disregarded, etc. Prior to merging a modified state into the EA repository, EA experts want to analyze changes made to a state carefully with respect to 1) the current state to see how the plan would affect the currently captured reality, 2) its origin, or 3) other planned states of an EA that might intervene.

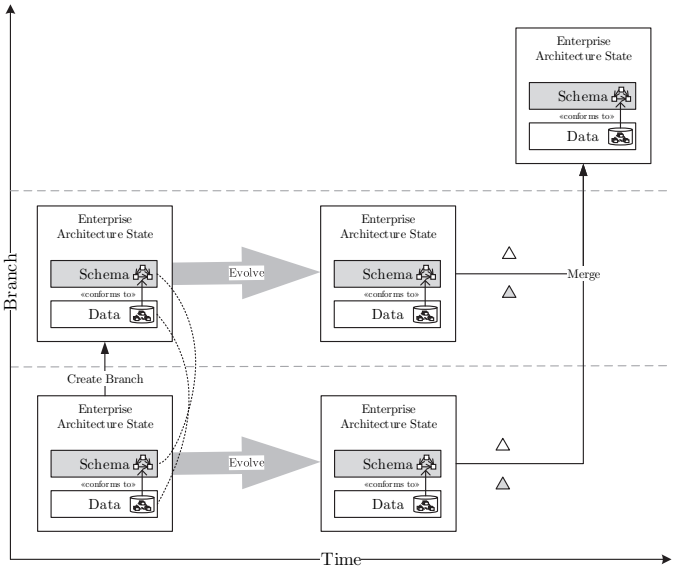

Fig. 1. Co-evolution of EA states, which are commonly stored and managed in an EA repository

This is a long-lasting process and should not be considered similar to software merging as it involves many political and social aspects, too. In this process, EA management serves as a mediator to communicate changes made to an EA. This discipline is regarded as highly collaborative and needs a high degree of stakeholder involvement. Visualizations help to facilitate communication with stakeholders. Although differently with respect to temporal and collaborative aspects, the technical difficulties to compare models and the cognitive challenge to communicate model differences are very similar for both, software models and EA models. We diagnose that as of today, no standard to visualize differences in models and in

particular EA models has been established and conclude to the following research question:

'How to visualize differences of co-evolving models and their respective meta-models?'

Addressing this issue, the remainder of the paper is structured as follows. Subsequently, we revisit related work briefly and present our conceptual design. Thereby, we assume that model differences already have been calculated, e.g. by an algorithm like presented by Kelter et al. in [26]. We continue with a feedback from EA experts. The paper concludes with an overview of known limitations and gives an outlook on future research.

II. A Brief Overview of Related Work

Due to space limitations, we point the interested reader to the body of knowledge on the topic instead of presenting figures of visual solutions to display model differences. Literature focusing on mere visualizations of model differences seems to be scarce. However, the modeling community uses visual concepts often to explain their concepts. We revisit these publications and highlight interesting points.

Wenzel presents an interesting concept employing a graph layout, so-called polymetric views [16] as a scalable approach to visualize huge models. Vertices in polymetric views encode information (metrics) in up to five dimensions, i.e. width, height, position (x and y), and color of a vertex. Wenzel proposes metrics for differences to quantify their properties and distinguish relevant from irrelevant changes. Van den Brand et al. [20] also follow the polymetic approach for reasons of scalability. Schmidt et al. [17] use a revision graph to display model conflicts. For the revision in conflict, they display the changes directly in the model via color coding. Kehrer et al. [9, p.14], Wenzel [12] as well as Krause et al. [11] apply a colored coding to present differences in models in a visual manner. Most differences visualizations, e.g. [9, 11, 12, 14, 16, 17], uses green for new concepts and red for deletions. For our work, we consider color coding as an essential technique to communicate change visually. Girschick [19] even uses seven different colors to display differences.

"The standard advice for using color to encode categories is to limit your selection to ideally about six—hopefully no more than 12, and absolutely no more than 20—colors […]"[25, p. 66]. We conclude that too many colors can also become a source for cognitive overload. The authors of [21, 24] use iconification to visualize conflicts between models. Other approaches are very specific for one particular language, i.e. meta-model, e.g. [22].

To our best knowledge, tools as well as literature in the discipline of EA management do not provide any means that go beyond the concepts presented above (see also [5]).

At the same time we perceive a lack of approaches that address stakeholder needs, i.e. prevent information overload (cf. also [13]). "Information overload occurs when the amount of input to a system exceeds its processing capacity. Decision makers have fairly limited cognitive processing capacity. Consequently, when information overload occurs, it is likely that a reduction in decision quality will occur" [15]. Our concept presented subsequently addresses this issue when displaying model differences visually.

III. Conceptual Design

We start to explain the conceptual design by outlining the information demand for our interactive visualization. The latest meta-meta-model of a prototypical implementation of our design and details of core concepts of the respective modeling language employed can be found in [7]. This meta-meta-model stores the meta-model of a model explicitly. Changes to the meta-model do not directly affect its instance. Hence, the meta-model and model are loosely coupled and altering the meta-model does not trigger a migration, cf. [7]. Fig. 2 shows the information demand that is required to generate the shown four-layered differencing visualization. We assume that each Model is made up of

- a schema consisting of Object Definitions (conceptually similar to classes) and Attribute Definitions as well as
- data consisting of Objects and Attributes that carry values.

Each of these Model Elements including the Model itself may have Differences. Differences on the other hand are just a set of references to existing Model Elements in different branches. Note the loose coupling between Objects, Attributes and their definitions (cf. [7]) that allows a considerable degree of freedom, i.e. the model does not have to conform to its meta-model all the time, is disregarded in the meta-model depicted in Fig. 2.

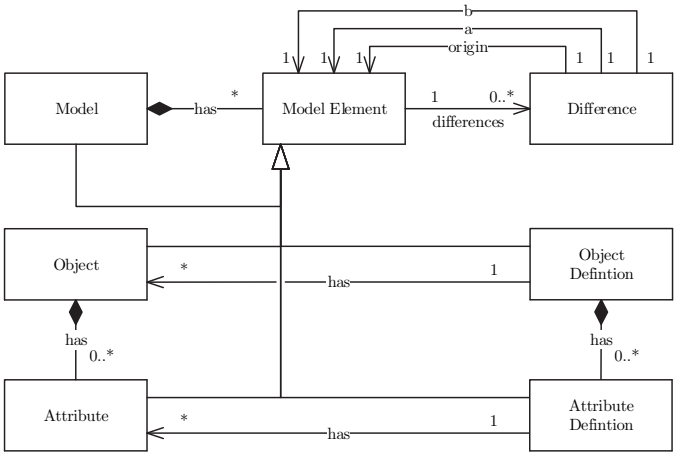

Fig. 2. Meta-model of the information demand for the difference visualization

In our interactive difference visualization, the end-user can access information on model as well as meta-model differences at four layers via various interactions. Thereby, filtering (cf. Section III.E) is essential as showing all the differences of two models (and their instances) is regarded too complex and the result often ends in an information overload (cf. [13, 15]).

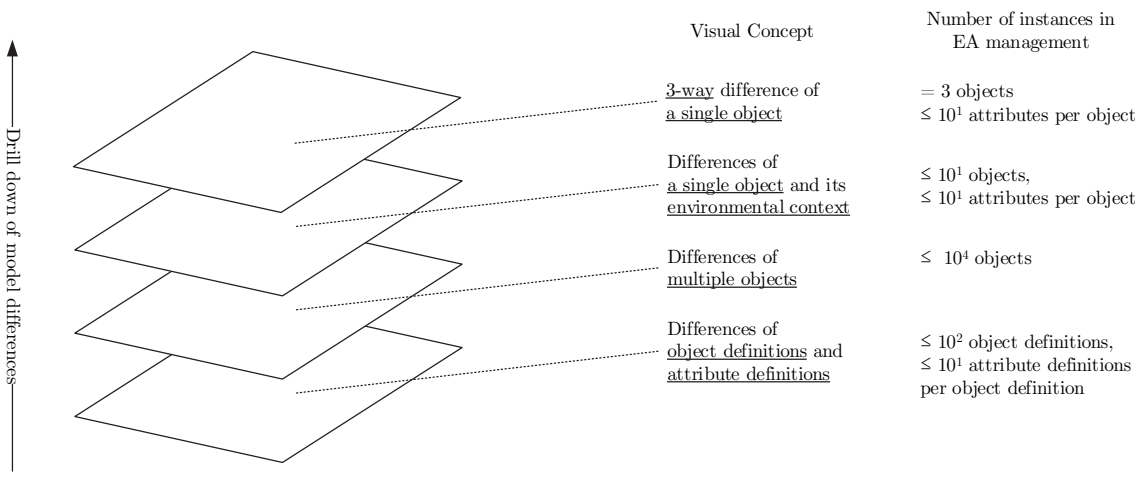

Fig. 3. Conceptual design of an interactive visualization to display and navigate through (meta-)model differences

Fig. 3 illustrates the relationships of the four layers, the visual concept applied at each layer, and the typical number of instances we regard to interact with in an EA model and respective meta-model. Each layer serves as a means to navigate for the user.

A. *Meta-model Differences*

In the first layer (see Fig. 4), we visualize the differences of two meta-models denoted *A* and *B*. The entire meta-model is shown to the user as a graph. A graph layout algorithm is used to arrange the vertices and edges visual appealingly. An important design criteria to choose an algorithm is its resistance to change, i.e. the layout must be relatively stable w.r.t. small changes in the underlying data, i.e. the structure of the meta-model. A spring graph layout algorithm for instance commonly rearranges all the vertices such that a user ends-up with an entirely new layout each time the graph is rendered. New OBJECT DEFINITIONS[1] are displayed in the background color green, altered OBJECT DEFINITIONS in orange and deleted ones in red. In case the name of an OBJECT DEFINITION has been altered, a two-way string diff is applied to the respective class names (cf. *Business Application* in Fig. 4).

ATTRIBUTE DEFINITIONS are displayed at this layer as part of the OBJECT DEFINITION. The number of instances of every ATTRIBUTE, i.e. its actual usage, and the cardinality are given. New ATTRIBUTE DEFINITIONS are displayed in green, deleted ones in red, and altered ATTRIBUTE DEFINITIONS are displayed 1) using a two-way textual differences algorithm, 2) showing version A and B, and 3) showing their origin[2]. RELATIONSHIPS are illustrated as edges between the vertices in the graph. Newly created RELATIONSHIPS (stored as a special kind of an ATTRIBUTE DEFINITION) are displayed green, deleted ones red, and modified, i.e. the name, source, or target in orange. Each modification is annotated with the version a modification has been made. The respective versions of a modification (A, B, or A+B) is displayed in magenta colored text. This holds true for OBJECT DEFINITIONS, ATTRIBUTE DEFINITIONS as well as RELATIONSHIPS.

Besides differences between OBJECT DEFINITIONS, their ATTRIBUTE DEFINITIONS and RELATIONSHIPS to each other, the first layer also displays aggregated information of model differences, i.e. number of differences between OBJECTS of each OBJECT DEFINITION. Absolute and relative values of OBJECTS that have no differences (green, right part of the progress bar at the bottom of each OBJECT DEFINITION), and OBJECTS that have differences in the other branch (red, left part of the progress bar of each OBJECT DEFINITION) are shown. This way, an end-user gets a good overview of model differences (per OBJECT DEFINITION) just by looking at layer 1 of the difference visualization. The user can either navigate to the respective OBJECT DEFINITION by clicking on its name or the red *A* or *B*. or can click anywhere else on an OBJECT DEFINITION to open the next layer which shows differences between OBJECTS that conform to the clicked OBJECT DEFINITION.

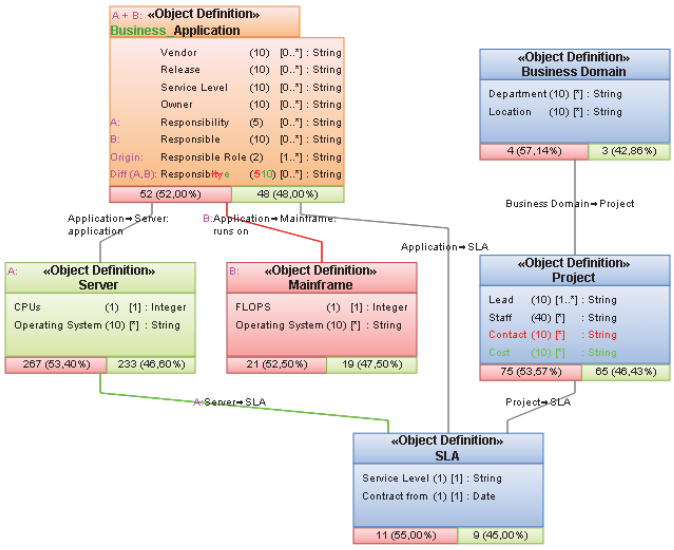

Fig. 4. Layer 1: Meta-model differences (object definitions and attribute definitions) of two meta-models (schemas) A and B with aggregated information of differences in respective models (data)

[1] OBJECT DEFINITIONS can be considered UML classes
[2] aka. provenance, base version/revision, or common anchor

B. Instance Overview

The second layer gives an overview of the meta-model instances, i.e. OBJECTS and ATTRIBUTES (cf. Fig. 5). The end-user already narrows the differences down to OBJECTS of a single OBJECT DEFINITION by the choice made in layer 1. As depicted in Fig. 3, the number of instances can grow considerably large in layer 2. For this reason, we add an intuitive filter to our interactive visualization. This way, users can define sophisticated range filters on multiple attributes (see Section III.E).

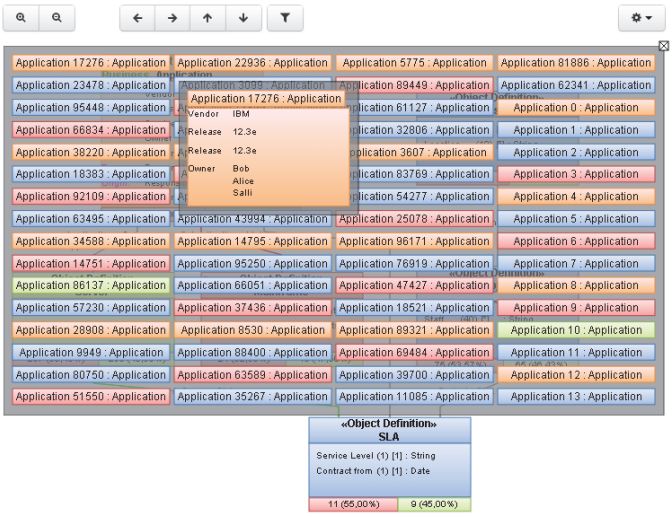

Fig. 5. Layer 2: Differences in OBJECTS with OBJECT details on mouse hover

At this layer our design goal has been to prevent information overload. We chose to hide any unnecessary details including ATTRIBUTES. As illustrated just the name and the state of an OBJECT (via color coding) is shown to the user. The latter denotes whether it has been altered in A, B or both branches. When the user hovers over an OBJECT, its details are shown. Fig. 5 further illustrates the controls available in any of the four layers. Besides basic facilities to zoom and navigate, the filter can be activated or the visualization can be downloaded as PowerPoint (.pptx) presentation for further manipulation.

C. Instance Neighborhood

In EA management, not only the single OBJECT is of interest, but also any implied impact by changes when considering the bigger picture, i.e. an OBJECT's environment. In layer 3, we show this environment to an end-user. Although EA management commonly analyzes aggregated information [27], more fine grained information such as servers and routers also could be subject of analysis [28]. Considering to visualize all instances and relationships among them, displaying 10^8 visual objects at once might not be unrealistic. For this reason, we propose a consecutive drilldown of differences. Fig. 6 shows the n^{th}-neighborhood of an OBJECT which is again, arranged as a graph. Thereby, the number of neighborhood OBJECTS to traverse (the n) is configurable. The same color coding and semantics like on the other layers are applied on this layer (orange: an object has been modified in two branches; blue: no changes; red: object deleted, green: new object created). Layer 3 picks up expert feedback on an earlier version of a prototypical implementation of the interactive difference visualization. The expert states that LINKS (instances of RELATIONSHIPS) between OBJECTS are far more interesting for an analysis than changes of ATTRIBUTES. This is also 'well-known' fact in EA management.

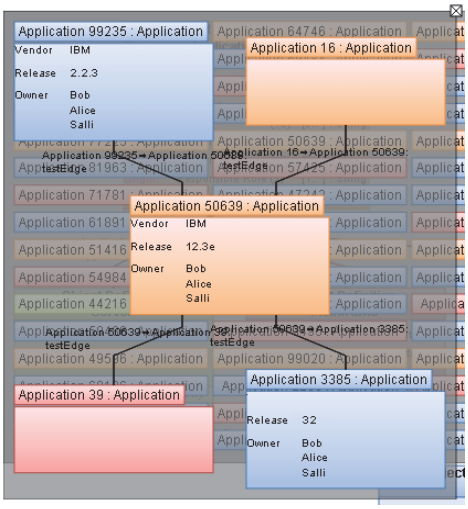

Fig. 6. Layer 3: Graph representation an OBJECT and its environmental context (1st-neighborhood)

D. Three-Way Differences

Another click brings the user to the last layer of the interactive visualization. At this layer, a three-way difference is shown, i.e. version A, B and (if existing) their origin (cf. Fig. 7). Since only three OBJECTS are shown no further (string) differencing is needed. We consider layer 4 the entrance point to navigate to the respective OBJECT within the system if any further details, such as access rights or modification date, are missing. Note that each of the layers 2 to 4 can be opened for multiple OBJECTS and the different views may overlap each other and can be switched by the user that analyzes the model differences.

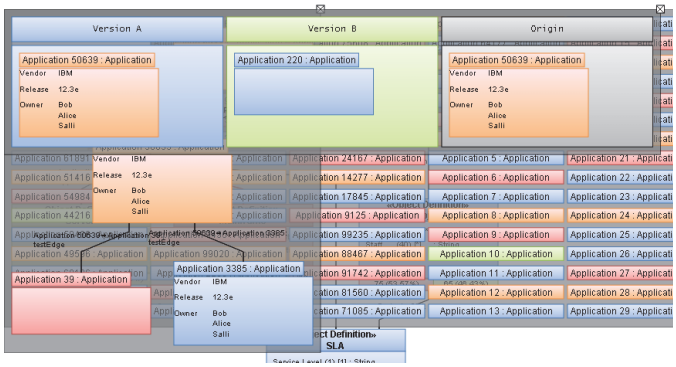

Fig. 7. Layer 4: Three way difference view of an OBJECT (top) and other layers (bottom/background).

E. Filtering

We facilitate the analysis of larger EA models, i.e. more instances, by employing a filter. A conceptual mock-up of the filter is depicted in Fig. 8. It is meant to define a query for OBJECTS to be compared and can be applied in layer 1 and 2 preferably. The design was inspired by the iTunes playlist fil-

ter. Our design goal was to empower EA experts to configure filters without the need to program or adapt sophisticated search scripts. The outcome of the filter is a generated JavaScript Object Notation (JSON), which is depicted in Fig. 9. This JSON object is processed as a server-sided query.

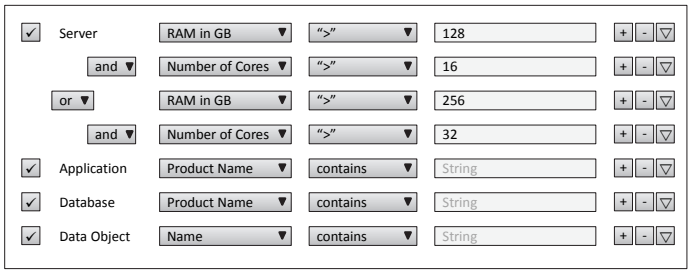

Fig. 8. End-user ready model filter inspired by the playlist user interface of iTunes

As depicted, filters can be concatenated on the same level or nested recursively (cf. Fig. 8 and Fig. 9). This empowers EA experts to define sophisticated model queries. The filter can be applied to different OBJECT DEFINITIONS and depending on the type of ATTRIBUTE DEFINITION that is subject to the filter, different comparators can be used, e.g. *after* and *before* are just accessible for the type DATE whereas *contains*, *starts with*, and *ends with* can be applied to the type STRING. Greater than ('>') and less then ('<') serve to compare NUMBERS whereas *not null* and *equals* ('=') are generic operators for any type. Although the filter is defined on their definition (schema), the filter is applied to OBJECTS and ATTRIBUTES (data).

```
{"object definition" : [{
          "conjunction" : "and | or",
          "attribute" : "attribute name",
          "comparator" : "contains | starts with | ends with | = | < |
          > | before | after | not null",
          "predicate" : "value",
          "filter" : [{"conjunction" : "...",
                    "filter" : [{"conjunction" : "..."}]
                    }]
          },{
          "conjunction" : "and | or",
          "attribute" : "attribute name",
          "comparator" : "contains | starts with | ends with | = | < |
          > | before | after | not null",
          "predicate" : "value",
          "filter" : [{"conjunction" : "...",
                    "filter" : [{"conjunction" : "..."}]
                    }]
          }]
}
```

Fig. 9. Recursive JSON filter for OBJECTs of one OBJECT DEFINITION

IV. FIRST END-USER FEEDBACK ON THE PROTOTYPE

The conceptual design has been implemented in an existing visualization framework initially presented in [3]. This framework has been considerably extended and incorporated in a wiki-based system [10] that allows to structure its content. In an iterative approach we showed this prototype to EA experts. Thereby, we imported their EA model such that they could give us feedback based on their productive data reflecting the real-world to them.

First interviews with EA experts confirm that the overall concept for our two interviewees seems sensible. In particular to traverse the model via classes, i.e. OBJECT DEFINITIONS, to drill down to the instances (OBJECTS) is regarded as a very intuitive way. However, minor changes are still to be done. In this section we give a brief overview on the gathered feedback.

Link changes over attribute changes: Changes of the RELATIONSHIP between OBJECTS, i.e. structural changes, are more important than changes to ATTRIBUTES in EA management.

Apply color coding sparingly: EA experts had the opinion that the color orange is very similar to red. This may be improved in another version. However, color coding always copes with visual cognition and we discussed that using additional colors should be considered carefully.

Necessity of a filter: Filters are of high relevance to view differences in EA models. Although planned states of an EA have a specific purpose, one might not view at the changes but at the affected areas of the current state of the EA model.

Exporting and printing visualizations: Another very important topic in the discussion with our EA experts has been the ability to export a visualization. This not only plays an important role to communicate results (and differences) via e-mail but also to print huge posters advertising an EA initiative.

Integration with other platforms: currently, the solution has a relatively strong cohesion to the wiki-based platform. The EA experts think that it is definitively worthwhile to offer an independent component with a meta-meta-model featuring a similar expressive power, e.g. to visualize differences of excel spreadsheets without the need to setup an entire platform or to integrate the difference visualization in other tools.

V. CURRENT LIMITATIONS

Our implementation does not build on the ecore/EMF meta-meta-model and thus the expressive power is different. Some limitations we foresee in this and other contexts are subject of this section.

Currently, our solution does not address to visualize any form of inheritance. This is a direct consequence of the underlying meta-meta-model which does not allow inheritance, cf. [7]. When introducing inheritance, in particular the model operation 'move' must be addressed by a visual concept that intends to communicate differences of this operation.

The interaction extensions made to the visualization framework [3] contradict with the requirement to print a visualization. This issue addresses application programming interfaces (API) design as well as it raises the question whether visualizations are rendered best client or server-sided. From our experience in EA visualization development and tool analyses [3, 5, 29, 30], we learnt that data will always be queried from servers. However, we believe that the more interaction is required client-sided, the more code will shift towards the client. The object graph must then be transformed to an export format on the client.

Another pain point is the performance of our solution. Meant as an ad-hoc analytical tool, we want to be able to deliver differences within < 5 seconds. First steps are to use two kind of caches, however, the illustrated visualization took a considerable large amount of time (~10s) and included 'only' 500 instances with synthetic data that can be regarded highly interlinked to have realistic data for development purposes in place. Currently, we query data and calculate the layout server-sided. The resulting visualization is then rendered in JavaScript and does not invoke any callbacks on user interaction. This requires to calculate any interaction ex-ante. A concept to avoid the long-lasting pre-calculation of course are server callbacks. On the other hand, the user-experience during a difference analysis could suffer considerably from network latencies.

Finally, future versions could follow strict visual design guidelines, e.g. [18]. Experiments could reveal if such a 'guided' design brings considerable new means to practitioners.

VI. Conclusion

The contributions of this paper was a holistic concept for a model difference visualization featuring interaction to drill down model differences. We detailed the conceptual design of a novel, four-layered visualization of model and meta-model differences embracing presentation, layout, filtering, and interaction. We advocate that EA models share many similarities with software models regarding complexity, number of instances, etc. In further research we are interested to see if and how the presented concepts can be applied to general software models.

Acknowledgment

We thank our industry partners for their valuable feedback to our design. In particular we would like to thank Martin Baumann from Kassenärztliche Vereinigung Bayerns (KVB) as well as Karsten Hahn and Erich Wohnig from HUK Coburg.

References

[1] Spence, R. Information visualization Addison-Wesley, 2001.

[2] Ware, C. Information visualization: perception for design Morgan Kaufmann Publishers, 2012.

[3] Schaub, M.; Matthes, F. & Roth, S. Towards a Conceptual Framework for Interactive Enterprise Architecture Management Visualizations Modellierung, 2012.

[4] Wettel, R.; Lanza, M. & Robbes, R. Software systems as cities: a controlled experiment Proceedings of the 33rd International Conference on Software Engineering (ICSE), 2011.

[5] Roth, S.; Zec, M. & Matthes, F. Enterprise Architecture Visualization Tool Survey 2014 Technical University Munich, 2014 (to appear).

[6] Roth, S.; Hauder, M. & Matthes, F. Facilitating Conflict Resolution of Models for Automated Enterprise Architecture Documentation Nineteenth Americas Conference on Information Systems (AMCIS), 2013.

[7] Roth, S.; Hauder, M. & Matthes, F. Collaborative Evolution of Enterprise Architecture Models. 8th International Workshop on Models at Runtime (Models@run.time), 2013.

[8] Kehrer, T.; Kelter, U.; Ohrndorf, M. & Sollbach, T. Understanding model evolution through semantically lifting model differences with SiLift Software Maintenance (ICSM), 2012 28th IEEE International Conference on, 2012.

[9] Kehrer, T.; Kelter, U. & Taentzer, G. Consistency-Preserving Edit Scripts in Model Versioning, Technical Report, University of Siegen, 2013.

[10] Matthes, F. & Neubert, C. Enabling Knowledge Workers to Collaboratively Add Structure to Enterprise Wikis Proceedings of the 12th European Conference on Knowledge Management, 2011.

[11] Krause, C.; Dyck, J. & Giese, H. Metamodel-Specific Coupled Evolution Based on Dynamically Typed Graph Transformations Theory and Practice of Model Transformations, Proceedings of the 6th International Conference on Model Transformation (ICMT), Springer, 2013.

[12] Wenzel, S. Scalable visualization of model differences Proceedings of the 2008 international workshop on Comparison and versioning of software models, ACM, 2008.

[13] Toffler, A. Future shock Random House Digital, Inc., 1970.

[14] Ohst, D.; Welle, M. & Kelter, U. Differences between versions of UML diagrams ACM SIGSOFT Software Engineering Notes, 2003.

[15] Speier, C.; Valacich, J. S. & Vessey, I. The influence of task interruption on individual decision making: An information overload perspective Decision Sciences, Wiley Online Library, 1999.

[16] Lanza, M. & Ducasse, S. Polymetric Views-A Lightweight Visual Approach to Reverse Engineering IEEE Trans. Softw. Eng., IEEE Press, 2003

[17] Schmidt, M.; Wenzel, S.; Kehrer, T. & Kelter, U. History-based merging of models Comparison and Versioning of Software Models, 2009. CVSM '09. ICSE Workshop on, 2009.

[18] Caire, P.; Genon, N.; Heymans, P. & Moody, D. L. Visual notation design 2.0: Towards user comprehensible requirements engineering notations RE, IEEE, 2013.

[19] Girschick, M. Difference detection and visualization in UML class diagrams Technical University of Darmstadt, 2006.

[20] van den Brand, M.; Protić, Z. & Verhoeff, T. Generic tool for visualization of model differences Proceedings of the 1st international workshop on model comparison in practice, 2010.

[21] Brosch, P.; Kappel, G.; Seidl, M.; Wieland, K.; Wimmer, M.; Kargl, H. & Langer, P. Adaptable Model Versioning in Action Modellierung, 2010.

[22] Pietsch, P. & Wenzel, S. Comparison of BPMN2 Diagrams Business Process Model and Notation, Springer, 2012.

[23] The SiDiff Project, http://pi.informatik.uni-siegen.de/Projekte/sidiff/index.php,. Last-accessed: Dec 28, 2013.

[24] Langer Wieland, S. W. K. B. K. Representation and Visualization of Merge Conflicts with UML Profiles Proceedings of the International Workshop on Models and Evolution (ME 2010) @ MoDELS 2010, Online Publication, 2010.

[25] Iliinsky, N. & Steele, J. Designing Data Visualizations: Representing Informational Relationships O'Reilly Media, 2011.

[26] Kelter, U.; Wehren, Jü. & Niere, Jö. A Generic Difference Algorithm for UML Models. Software Engineering, Software Engineering, 2005.

[27] Winter, R. & Fischer, R. Essential Layers, Artifacts, and Dependencies of Enterprise Architecture Journal of Enterprise Architecture, 2007.

[28] Buschle, M.; Ekstedt, M.; Grunow, S.; Hauder, M.; Matthes, F. & Roth, S. Automated Enterprise Architecture Documentation using an Enterprise Service Bus Proceedings of in Americas conference on Information Systems (AMCIS), 2012.

[29] Roth, S.; Hauder, M.; Zec, M.; Utz Alexej & Matthes, F. Empowering Business Users to Analyze Enterprise Architectures: Structural Model Matching to Configure Visualizations 7th Workshop on Trends in Enterprise Architecture Research (TEAR 2013), 2013.

[30] Hauder, M.; Matthes, F.; Roth, S. & Schulz, C. Generating dynamic cross-organizational process visualizations through abstract view model pattern matching Architecture Modeling for Future Internet enabled Enterprise (AMFInE), 2012