Historization of Enterprise Architecture Models Via Enterprise Architecture Knowledge Graphs

Robin Bråtfors, Simon Hacks, and Dominik Bork

To appear in:

15th IFIP WG 8.1 Working Conference on the Practice of Enterprise Modelling (PoEM'2022)

Final version available soon:

www.model-engineering.info

Historization of Enterprise Architecture Models Via Enterprise Architecture Knowledge Graphs

Robin Bråtfors[1], Simon Hacks[2], and Dominik Bork[3]

[1] KTH Royal Institute of Technology, Stockholm, Sweden
bratfors@kth.se
[2] University of Southern Denmark, Odense, Denmark
shacks@mmmi.sdu.dk
[3] TU Wien, Business Informatics Group, Vienna, Austria
dominik.bork@tuwien.ac.at

Abstract. Enterprise Architecture (EA) is the discipline that aims to provide a holistic view of the enterprise by explicating business and IT alignment from the perspectives of high-level corporate strategy down to daily operations and network infrastructures. EAs are consequently complex as they compose and integrate many aspects on different architecture layers. A recent proposal to cope with this complexity and to make EAs amenable to automated and intuitive visual analysis is the transformation of EA models into EA Knowledge Graphs. A remaining limitation of these approaches is that they perceive the EA to be static, i.e., they represent and analyze EAs at a single point in time. In the paper at hand, we introduce a historization concept, a prototypical implementation, and a performance analysis for how EAs can be represented and processed to enable the analysis of their evolution.

Key words: enterprise architecture, historical analysis, knowledge graph

1 Introduction

IT and Communication Technology (ICT) is a crucial part of most modern organizations, and many resources are invested to make them more efficient. However, to achieve the full benefits of those investments, the business strategy and the ICT of the organization need to be aligned, which can be achieved by means of Enterprise Architecture (EA) [1]. EA is a discipline that aims to give a comprehensive view on the architecture of an organization, i.e., the structure, processes, and infrastructure of the organization [2]. An EA provides a holistic view of the enterprise and lays out the insight to balance different requirements and turn theoretical corporate strategy into real daily operations.

There is no agreement on an exact definition of EA, the literature shows varying definitions [3]. This uncertainty can add challenges when it comes to the quality assessment of EA, as different definitions naturally have different metrics for what is good and what is bad. Providing a standardized way to measure the quality of EA would therefore help with improving that quality and clarify the

definition of EA. EA debt is a concept introduced by Hacks et al. [4], which depicts how much the current state of an EA deviates from a hypothetical ideal state. The extent of that deviation naturally indicates the quality of the EA.

But to accurately ascertain that deviation one needs to be able to measure the EA debt. A way to do that was proposed by Salentin and Hacks [5] with the concept of EA smells. These smells are derived from the concept of code smells and, in similar fashion, highlight specific flaws within an EA. Providing additional ways to detect these smells would help with measuring EA debt, which in turn would help with discerning the quality of EA.

Analyzing EA models as graphs is one out of four main categories of EA analysis proposed by Barbosa [6]. The graph's nodes and edges represent the components and relationships of an EA, respectively, and this sort of graph representation enables more ways to assess the quality of the EAs through graph-based analysis methods and algorithms. Graph-based analysis does not need to be overtly complicated to achieve some effect. Just the transformation of EAs into graph structure grants some visual analysis capabilities that a stakeholder can manually utilize [7, 8]. That is not to say that graph-based analysis cannot be more complex or automatized. For example, further analysis of graphs can be implemented with the use of filter and search methods [9].

Smajevic et al. [10] created the CM2KG platform [11] that is capable of transforming conceptual models into Knowledge Graphs which enable graph-based analysis of EA models [10, 12]. It transforms EA models into graphs, which enable automatic analysis to detect flaws and points of improvement within the EAs. Further work on the CM2KG platform allowed it to automatically detect EA smells in transformed EA models [12] and to realize a plugin to the widely used open EA modeling platform Archi [13, 14]. The transformation of EA models to graphs (e.g., using the CM2KG platform) enables the realization of a generic solution in the sense that model quality measures, like EA smells, can be defined on the generic level of graph structures. The generic metrics can then be evaluated on any EA representation, including ArchiMate. As the field of EA is packed with different modeling languages and frameworks, a generic and modeling language agnostic solution is favorable. This is also the value of the approach compared to the analysis functionality offered by EA tools which are always bound to the specific EA modeling language at hand.

While the previous works, particularly the CM2KG platform, show the feasibility of transforming EA models into graph structures and using these EA graphs for analysis, they also come with severe limitations. Most importantly for the scope of our work, they considered only one state of the EA (i.e., the graph) [8, 9, 15]. In other words, the changes resulting in an evolution of an EA are neither considered not amenable to analysis. Even professional tools often just provide solely basic capabilities to analyze/query the evolution of EA and then are bound to their proprietary model. Thus, the identification of EA smells, that can be identified by comparing different states of the EA, is not possible. For example, to identify a *Big Bang*, a strategy where large changes to a system are made at once instead of in several smaller steps, can be just identified

by analyzing the delta between different states of an EA. Adding historization to graphs can provide the needed information by enabling an evolutionary perspective on EA analysis. Such a perspective allows for historical analysis and gives enterprise architects automated and programmatic insight into the evolution of an EA. To be able to analyze the evolution of EA models, we deduce the following two research questions:

RQ1: What is an appropriate means to represent EA models for historical analysis?

RQ2: How performant is the historical analysis of even large EA Knowledge Graphs?

The rest of this work is structured as follows: Background and related works are discussed in Section 2. Section 3 proposes a concept for the historization of EA graphs. An implementation of a platform for historical EA graphs is introduced in Section 4. Performance and efficacy of the platform are evaluated in Section 5. Eventually, we close this paper with a discussion of contributions and limitations in Section 6 and concluding remarks in Section 7.

2 Background and Related Work

2.1 EA debts and Smells

In software development, Technical Debt was introduced by Cunningham [16] as a metaphor for the potential cost caused by refactoring code that is not quite right when it is created. The metaphor is inspired by financial debt, where the debt is the future work that has to be done to improve the code.

Technical Debt allows organizations to handle quality issues related to their application landscape, but it does not extend to other domains of an organization. Hacks et al. [4] proposed to combine the Technical Debt metaphor with EA into what they coined EA debt. EA debt aims to provide a more holistic view of the organization by not only measuring the quality of software but also measuring the quality of all the other domains and thus exposing faults within the enterprise. They demonstrated the use of EA debt by giving examples of how it can highlight problems that would be costly to solve at the moment but need to be solved at some point, such as dependency issues [4]. The EA debt can make management aware of the problem so that it can be solved as soon as the dependency is met, instead of potentially forgetting about it.

Schmid [17] presents some shortcomings of Technical Debt, e.g., that the debt needs to be considered with respect to future evolution, since the development might be impacted by the current implementation and structure. Another shortcoming is the fact that a certain debt might not have an actual impact on the development, this potential debt is just structural issues and needs to be differentiated from effective debt that actually impacts the development.

Thus, a goal of EA debt is to provide relevant factors for estimating the architecture's quality to increase awareness and communication about its improvement [5]. However, to accomplish that, EA debt would need to be measured. Salentin and Hacks [5] have proposed to transfer the concept of Code Smells [18] to the EA domain in what they call EA smells. Accordingly, Technical Debt can be seen as a subset of EA debt [4] and thus many EA smells are just direct translations from already existing Code Smells [5]. EA smells can be considered a metric to measure the amount EA debt in an EA [5], representing the symptoms of EA debt and thus increasing awareness of potential deficits in an EA. We can differentiate two basic categories of EA smells regarding their input. The first and biggest category are EA smells, that analyze an EA based on a single state such as *Bloated Service* or *Cyclic Dependency*. The second category demands information about the evolution of an EA to be able to analyze if the EA is flawed. Those EA smells are for example *Big Bang* or *Data-Driven Migration*. For the latter class of EA smells, we are missing means to effectively identify them yet. To close this gap, this work proposes an underlying graph structure that will enable the identification of such EA smells.

2.2 Graph-based Analysis of EA Models

Graph-based analysis is one of four main categories of EA analysis proposed by Barbosa et al. in [6]. In this approach the architectures are represented as graphs where EA components and relations are transformed into nodes and edges, respectively. Once transformed, such a graph enables the application of many existing graph-based algorithms and metrics [10].

Panas et al. [8] introduced an architecture that dealt purely with the visualization of a model using a graph-based approach. While its visualization does provide some clarity, the user has to process most of the information by their own, which does not scale well for large and complex models. Chan et al. [9] presented a visual graph-based tool to analyze an enterprise, but this tool allows for more interactive exploration of the data. The user can use the tool to navigate, filter, and search the graph to process complex graph structures. Naranjo et al. [15] implemented another visual analysis tool, called PRIMROSe. They wanted to utilize visualization on the basis that the human visual system is a naturally great analysis tool, with the caveat that the complexity of EA models requires additional aid to properly inspect and find information about them.

The common trend through these works is that none of them looked at the history of graphs. Implementing a historization of the graphs on top of the benefits of graph-based analysis would increase the value for enterprise architects. All discussed works are further constrained to a specific kind of data input, which could be improved. Creating a generic solution that allows for several kinds of EA models to be used and analyzed increases its value.

First steps toward graph-based analysis of conceptual models were proposed in [10, 11] with the CM2KG platform, which has been recently specialized for the model-based construction of EA Knowledge Graphs (EAKG) [13] and integrated as a plugin to the Archi platform [14]. Their solution transforms EA models into

EAKGs to enable the execution of graph analysis metrics like betweenness and centrality. The usage of KGs also allows for automatic, efficient detection of EA smells even in large EA graphs with the help of semantic KG queries [12]. What was lacking in the original proposal of CM2KG, and what is the focus of this work, is the representation of the evolution of EAs and how this evolution can be represented in an EAKG and made accessible to historical analysis. We plan to incorporate that feature also in the newly developed EAKG toolkit [14].

2.3 Graph Historization

Nuha [19] evaluated two different approaches to store the historical data of graph databases: a *delta-based* approach and a *snapshot-based* approach. A combination of the two approaches was then implemented to utilize the strengths of each approach. This combination only created a snapshot whenever the difference between versions got too large and the reconstruction time of a version took too long. The evaluation of the solutions was performance-based and covered the execution time of storing new data, the checkout time of reconstructing and retrieving a past version, and the storage cost.

Other graph versioning solutions also utilized the delta-based approach [20, 21]. Castellort and Laurent [22] aimed to separate the versioned data from the operational data of graphs by storing the versioned data in a *revision graph*. Every node and relation in the *data graph* is represented by a node in the revision graph called revision element. These revision elements are a part of linked lists that represent the history of each data graph element at certain points in time. Any transaction will create a new revision and any data graph element that is modified during that transaction will get a new revision element in their linked list. Conversely, unmodified data graph elements keep their last revision element.

3 Toward Historization for EA Models

The purpose of this work is to conceptualize the transformation of EA models into a KG that preserves and represents the historical information found in EA models over time (cf. RQ1). With such a conceptualization, we aim to develop an IT artifact that enables users to view and interact with the historical graph through visual analysis and queries in a performant manner (cf. RQ2). Such an artifact will thus provide practitioners in the field of EA a generic solution to track, store, and analyze their EA models over a period of time by means of EA smells. The artifact might also create avenues for new research in the field that could further improve architects' ability to enhance their EAs. Accordingly, we have identified the following objectives. Any implementation of the historization would need to:

O1 ... be *performing* well for realistic model sizes. If a system is not responsive enough users might opt to stop using it. Research shows that users maximally tolerate a loading time between 2 and 3 seconds [23, 24]. Consequently, our solution should not exceed a querying time of 2 seconds.

O2 ... be *generic* in the sense that it is independent from any particular EA modeling language. We therefore aim to conceptualize a graph format that can store the historical data of an EA independently of the concrete EA modeling language.

O3 ... *integrate* into the existing CM2KG platform to benefit from its existing capabilities such as import from different conceptual models, transformation into graph structures, and graph-based analysis [10, 12].

O4 ... provide users with an *intuitive* way to represent, interact, and query historical states of an EA model.

3.1 Graph Structure

To realize Objective 2, we rely on a GraphML representation of the EA model which has been used by the CM2KG platform since its conception [10]. This design decision also eases the integration of the historization with CM2KG (Objective 3). GraphML is standardized and modeling language-agnostic. Since the graph transformation of the tool converts the EA models into GraphML, there was already a large incentive to stick with that format.

The most important extensions from the original structure are related to the question, how to represent historical EA versions in the graph, to keep track on the *micro evolution* of a single element of the EA model (e.g., adding, removing, or altering an element). We propose to add properties to every element of the graph (cf. "Additional properties" of a_i, b_j, t_k in Fig. 1). The purpose of these properties is to track the modifications of an EA element (i.e., *when?* and *how?*), which is a necessary information to be able to identify history-based EA smells. Since these properties are not inherent to the EA models themselves, they have to be inserted during the model to graph transformation. The properties are further defined in Section 3.3, their implementation is reported in Section 4.

3.2 Storage

The design choices when it comes to the storage of the transformed graph are not inherently important to the problem at hand since they do not overly affect the requirements of the design. The only vital part is that the historical data is stored in some way that differentiates each version of the model, since that is of course mandatory for historization.

The two major ways to store graph histories are the *snapshot-based* and the *delta-based* approaches. In this work, a design that implements both approaches is satisfactory by storing the change date and the actual changes (cf. "Additional properties" in Fig. 1) as well as each complete graph representation of the EA model's version and their evolutionary relation (cf. m_i in Fig. 1). This is necessary to keep track on the *macro evolution* of the EA model (i.e., the aggregated set of micro evolutions performed in a single iteration). The only drawback of implementing both is the increased storage space. This was weighed towards the benefit of having fast access to any requested version and the capability to easily show differences between versions of the entire model.

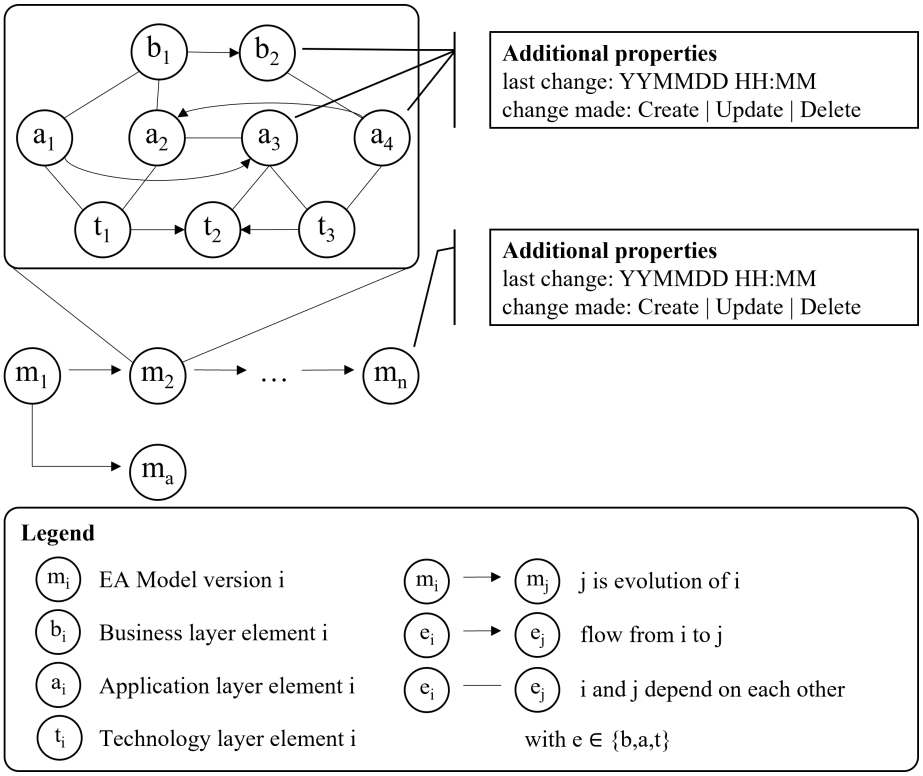


Fig. 1: Conceptual representation of realizing historization for EAKGs

3.3 Historization properties

To sort and catalog different chronological versions of a model, which is necessary to realize historization and create useful insights for EA smell analysis, those versions need a standardized property to differentiate and contextualize them from each other. The simplest way to implement this is by adding a date property to every element of the model, including the model itself. This property should then be updated each time its respective element gets modified. A property that marks the kind of modification that was performed on the element is also important to be able to efficiently sort and query the history. For simplicity reasons, all modifications to the model are identified and tracked each time the model gets uploaded to our artifact. Of course, a more detailed differentiation between the time of the actual change and the upload of the changes can be made. However, to demonstrate our approach, it is solely important to identify the order of the changes. Consequently, it makes no difference from a conceptual point of view what the concrete representation of the time is and, moreover, the implementation (see Section 4) can be easily adapted in future versions.

4 Implementation

During the transformation of a model, the currently selected version is fetched for comparison purposes. The previously last addition to this version is of course especially important to establish what kind of changes have been made during the last upload; we will henceforth refer to it as the *parent* model of the new uploaded model, and the new uploaded model as the *child* model.

The content of the child is sorted into HashMaps that represent the nodes and edges of the graph. These HashMaps are then used to compare the content of the child to the content of the parent by looking at their GraphML properties. Since GraphML is based on XML, each model follows a similar structure, and a generic comparison can be realized by matching the element's ID in the models. If an ID exists in the child but not in its parent, we conclude that this is a newly *added* element. If an ID instead exists in the parent but not in the child, then that element has been *deleted*. If an ID exists in both models but has diverging content, then that element has been *modified*. An element that is identical in parent and child is accordingly the *same*.

A summary of the differences between parent and child is compiled and stored in a comprehensive file that tracks the whole history from the first upload. After the transformation is completed, the relevant historization data is converted into session attributes to be used by Thymeleaf templates which are then presented to the user to interact with.

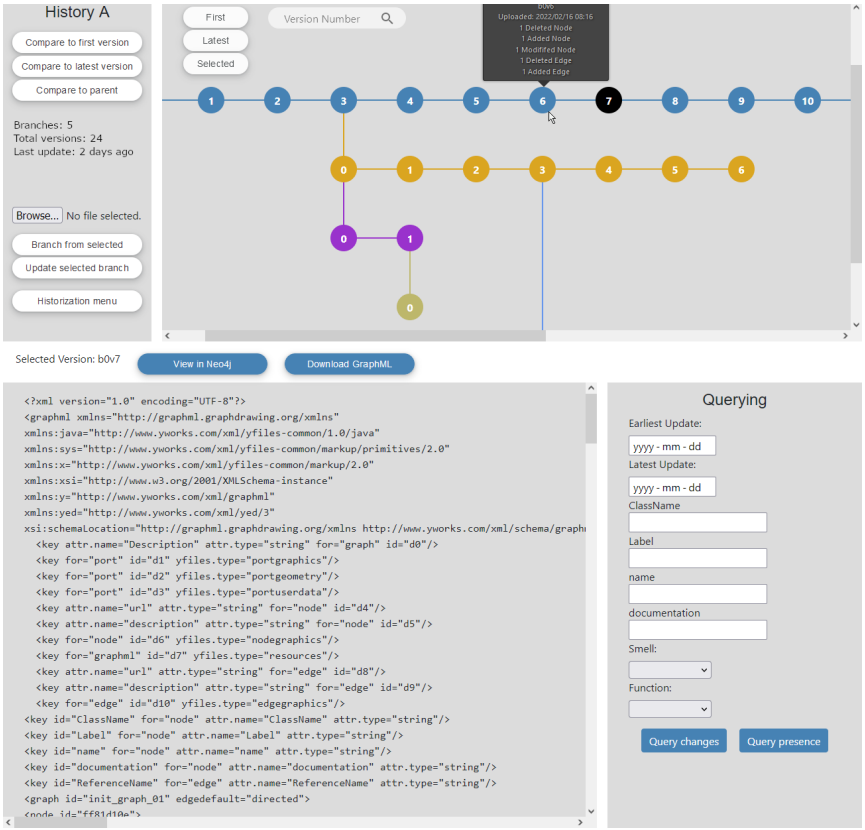
Interface The interface of the artifact consists of four areas: *history menu*, *history visualization*, *graph content*, and *querying*.

The *history menu* shows basic information about the history as a whole such as the number of branches, the total number of versions, and the date of the last update to the history. It also offers the option to upload a new version of the model as a new branch or as a continuation of an existing branch. The new graph gets added as a child to the currently selected graph. There are also options to visually compare the currently selected graph to either its parent, the first graph of its branch, or the last graph of its branch (cf. Figure 2a).

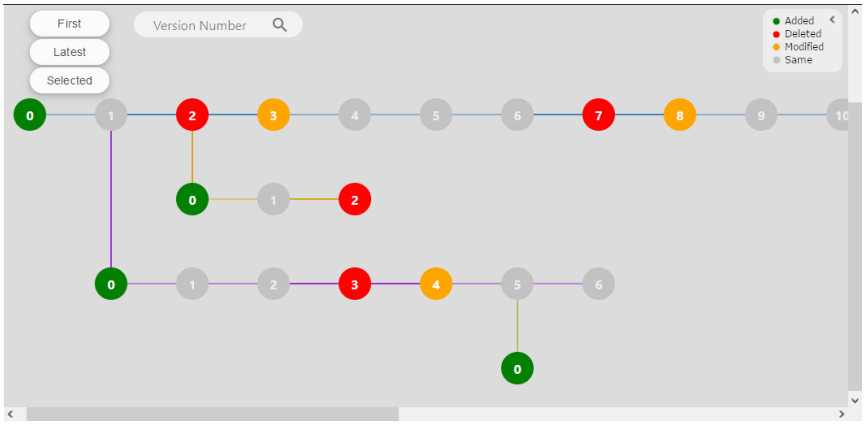
The *history visualization* area shows a visual representation of the history, enabling the user to view and interact with the model history. Each node represents a version of the model and in the baseline view they all have a color to indicate if they belong to the same branch. When hovering over a node, a tooltip appears and provides some information about the version in question, such as the differences to its parent and when it was uploaded. There are some buttons to ease the navigation through large histories. When clicking the buttons, the view instantly adapts to either the first graph, the last graph, or the currently selected graph. A graph can be selected by clicking on its node. The user can also view any version of the history by entering its ID in the search bar.

The *graph content* area shows the GraphML content of the selected graph. This content can also be downloaded as a GraphML file or viewed visually with the aid of Neo4j.

The *querying area* provides the user with the ability to query the history on some model attributes, such as the update time or name of graph element. The



(a) Artifact interface



(b) Delta-based query of a history

Fig. 2: Screenshots of the developed artifact

available queries show either the attributes' presence or how they have changed through the course of the historization. Moreover, the user can analyze the EA's evolution by means of pre-defined EA smells loaded from the central EA smells catalog. A more detailed explanation of how the queries work is provided in the next section.

Queries Two implemented queries have been realized to explore the model history. There is a *delta-based* query (cf. Figure 2b) and a *presence-based* query. The two queries are functionally similar since checking for the presence of something can be seen as a sub-task of checking if that something has changed. If a previously present object is not present any longer, then that object has obviously been removed, which essentially summarizes how the delta-based query determines if a model element has been deleted. Checking if an element has been added is the same process but reversed, namely if it is present in the child but not the parent then it is an added element.

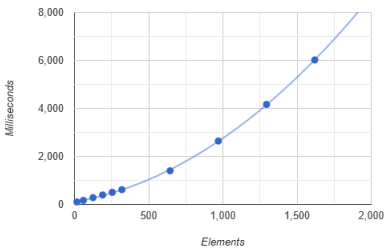
5 Analysis

Hitherto, we have presented the implementation to realize a historization of EA models for further analysis with EA smells addressing the Objectives 2 to 4. To prove that the presented solution meets the performance requirements of Objective 1, we conducted a set of experiments, that will be discussed next.

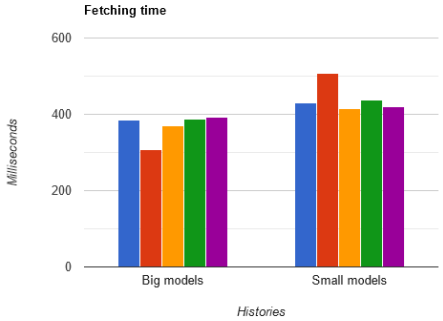
Transformation time. The time required to transform the model into a history graph is shown in the scatter plot of Figure 3a. Each point represents an average time of five different transformations of a certain version of a model, indicating a polynomial growth. Taking the average of just five executions was deemed sufficient as the transformation time never deviated more than 5% from each run. Each version differed in the number of elements it contained (16, 59, 124, 189, 254, 319, 644, 969, 1294, 1619). The elements had a fairly even split of nodes and edges.

Fetching time. The fetching time, the time to initially load an existing history, for two histories with similar overall sizes but different make-up is shown in Figure 3b. The histories both have around 46,000 elements but one history only represents 32 total models while the other consists of 506 smaller ones. The chart shows the results for five fetches for each history, measured in milliseconds. We can see that the fetching time for several small models is slightly higher than the fetching time for fewer large models.

Querying time. Figure 3c compares two implemented queries when executed on a history with around 140,000 elements. The *delta query* explored the difference between two evolutions of the EA model, i.e., it identified all changes between the two model evolutions. The *presence query* queried a single EA model evolution for the presence of a certain element. Therefore, each element in a single EA model evolution was assessed for a certain property and returned if found. The chart indicates that the two queries have essentially identical execution times.



(a) Transformation time of models



(b) Fetching time of histories

(c) Query comparison

(d) Querying differing history compositions

Fig. 3: Transformation and Querying analysis

Next, we elaborate the influence of the model size on the query times. Therefore, we use the same query for different sizes of models. This is illustrated in Figure 3d. Both queried histories had about 46,000 elements in total, but one history only represented 32 models while the other consisted of 506 smaller ones. The chart shows that the history with more and smaller graphs had longer execution times when queried.

Figure 4 shows the execution time of the delta-based query and how it increases based on the element count of the history it is querying. We can derive from the figure a linear relationship between the query execution time and the element count of the queried model.

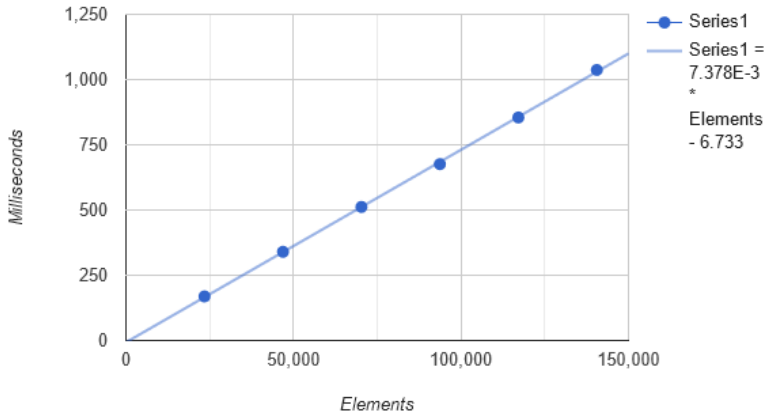


Fig. 4: Query execution time

6 Discussion

We believe even a rudimentary conceptualization of graph historization provides advantages when it comes to the analysis of EA models. Just the means of being able to store a history of models provides an easier way to access past versions of a model in a coherent and organized way. Simpler and faster access to past versions makes it easier to revert any mistakes by simply returning to a previous version of the model. Adding the capability of visualizing the history grants further benefits in terms of giving the user an overview of the history and a greater understanding of how each version of the model relates to one another. Future empirical research with our artifact will of course attest these claims.

The currently implemented queries in this work provide visual clarity of how the status of an attribute or node changed over time to ease the analysis via EA smells. Furthermore, the added functionality of the historization being generic means that a greater number of users might find the artifact helpful, especially if they are locked into a certain modeling platform and cannot for some reason switch.

The performance could be improved in a number of areas, especially the transformation time of models. The transformation time has a polynomial growth in relation to the number of model elements. The bulk of the time is spent on sorting the content of the child and then comparing that content to the content of its parent. Given a proper sorting of the content, the time complexity of the comparison should be no higher than linear since HashMaps are used to store the data, and the complexity for lookups in HashMaps is constant.

It can be argued that this polynomial growth can cause issues, due to long waiting times for large models. However, classically, the transformation of a single EA model evolution will only happen once, as afterwards the graph is already stored in the database. With that in mind, the transformation time

should be tolerable for users, especially as querying on the graph representation will be performed significantly more often.

From the results, we see that the number of graphs in a history had a slightly bigger impact on the execution time of fetching and querying compared to the individual graph sizes. Accordingly, the artifact is in its current state more suitable for large models with few updates than for smaller models that might be updated more frequently. Update frequency and model size are not necessarily linked, but it might be something to take into consideration from a storage perspective since both approaches to a history will result in similar history sizes.

The functionality that should see the most use is most likely the querying of the history since varying the attribute input of the query will grant different insights. With that in mind, it is beneficial, that the query execution time grows linearly as it should stay within a tolerable range for any history with less than 500,000 elements (i.e., models far exceeding the size of typical EAs). Since the implemented queries inspect every element in the history for a certain property, it would be difficult to improve the execution time any further as long as the data structure becomes not optimized towards certain expected queries, e.g., building sets of nodes with same properties.

7 Conclusion

With this work, we aimed to contribute toward historical analysis of EA models and to provide a solution that allows a satisfactory analysis with respect to performance and efficacy.

This overall aim has been detailed into four different objectives. Objective 1 demanded that the solution should not exceed a loading time of 2 seconds. This objective has been partly realized, depending on the size of the input models. If the size of the model exceeds approximately 750 elements, the transformation time will be larger than 2 seconds with polynomial growth. However, this is still acceptable, as a single EA model evolution only needs to be transformed once and then can be queried several times by the enterprise architects. Thus, the analyst will query the model more often and our solution performs well for querying. Even for large models with more than 100,000 elements the execution time remains around 1 second.

Objective 2 is realized by the graph-presentation that is shown in Figure 1, and Objective 3 by the integration of the solution into the existing CM2KG platform. Finally, we developed a graphical user interface that eases the exploration of the solution (cf. Section 4).

We believe this research establishes a foundation for several streams of future research. Firstly, it enables us in future to detect EA smells that we were not able to detect before, due to missing information. For instance, we can now detect the *Big Bang* EA smell by analyzing the number of changes between different evolutions of an EA. From a theoretical viewpoint, we want to involve enterprise architects in an empirical evaluation of our solution. This will add insights with respect to the usability, ease of use, and intention to use of our solution by

practitioners. From a technical point of view, we aim to further experiment with the artifact and different implementations of transformation, fetching, and querying algorithms in order to further improve the usability. Hitherto, we have solely conducted a technical evaluation of our approach. A deeper evaluation of the functionality is missing, especially we aim for a discussion with practitioners about their concrete needs and foreseen questions towards the historization of EA models. Eventually, we intend to deploy the artifact publicly to ease testing and use by the enterprise modeling and enterprise architecture community. Besides, the source code is available on github[1].

Finally, we want to make our research easily integrate-able into enterprise architects daily working live. Therefore, we aim to integrate our approach into wide-spread tooling such as Archi [14]. Thus, the analysis capabilities will be available directly at the place in which the architect is conducting the modeling. Moreover, due to our agnostic approach relying on Knowledge Graphs, our approach can be easily integrated with existing tooling and also enables analysis of EA smells in proprietary EA model notations of existing tool vendors.

Acknowledgements

This work has been partially funded through the Erasmus+ KA220-HED project Digital Platform Enterprise (project no.: 2021-1-RO01-KA220-HED-000027576) and the Austrian Research Promotion Agency via the Austrian Competence Center for Digital Production (contract no.: 854187).

References

1. Olsen, D.H.: Enterprise architecture management challenges in the norwegian health sector. Procedia Computer Science **121** (2017) 637–645
2. Lankhorst, M., Iacob, M.E., Jonkers, H., van der Torre, L., Proper, H., Arbab, F., Boer, F., Bonsangue, M., Hoppenbrouwers, S., Veldhuijzen van Zanten, G., Groenewegen, L., Buuren, R., Slagter, R., Campschroer, J., Steen, M., Stam, A., Wieringa, R., Eck, P., Krukkert, D., Janssen, W.: Enterprise architecture at work: Modelling, communication, and analysis. Springer (2017)
3. Saint-Louis, P., Morency, M.C., Lapalme, J.: Defining enterprise architecture: A systematic literature review. In: 2017 IEEE 21st International Enterprise Distributed Object Computing Workshop (EDOCW). (2017) 41–49
4. Hacks, S., Höfert, H., Salentin, J., Yeong, Y.C., Lichter, H.: Towards the definition of enterprise architecture debts. In: 2019 IEEE 23rd International Enterprise Distributed Object Computing Workshop (EDOCW). (2019) 9–16
5. Salentin, J., Hacks, S.: Towards a catalog of enterprise architecture smells. In: 15th International Conference on Wirtschaftsinformatik (WI). (2020)
6. Barbosa, A., Santana, A., Hacks, S., von Stein, N.: A taxonomy for enterprise architecture analysis research. In: 21st International Conference on Enterprise Information Systems. (2019)

[1] https://github.com/rbratfors/CM2KG

7. Garg, A., Kazman, R., Chen, H.M.: Interface descriptions for enterprise architecture. Sci. Comput. Program. **61** (2006) 4–15
8. Panas, T., Lincke, R., Löwe, W.: Online-configuration of software visualizations with vizz3d. In: Proceedings of the 2005 ACM Symposium on Software Visualization. SoftVis '05 (2005) 173–182
9. Chan, Y.H., Keeton, K., Ma, K.L.: Interactive visual analysis of hierarchical enterprise data. In: 2010 IEEE 12th Conference on Commerce and Enterprise Computing. (2010) 180–187
10. Smajevic, M., Bork, D.: Towards graph-based analysis of enterprise architecture models. In: Conceptual Modeling - 40th International Conference, ER 2021, Virtual Event, October 18-21, 2021, Proceedings, Springer (2021) 199–209
11. Smajevic, M., Bork, D.: From conceptual models to knowledge graphs: A generic model transformation platform. In: ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion, MODELS 2021 Companion, IEEE (2021) 610–614
12. Smajevic, M., Hacks, S., Bork, D.: Using knowledge graphs to detect enterprise architecture smells. In: The Practice of Enterprise Modeling - 14th IFIP WG 8.1 Working Conference, PoEM 2021, Proceedings. (2021) 48–63
13. Glaser, P.L., Ali, S.J., Sallinger, E., Bork, D.: Model-based construction of enterprise architecture knowledge graphs. In: 26th International EDOC Conference (EDOC'2022). (2022) in press
14. Glaser, P.L., Ali, S.J., Sallinger, E., Bork, D.: Exploring enterprise architecture knowledge graphs in archi: The eakg toolkit. In: 26th International EDOC Conference (EDOC'2022) – Tools and Demos. (2022) in press
15. Naranjo, D., Sánchez, M.E., Villalobos, J.: Primrose: A graph-based approach for enterprise architecture analysis. In: International Conference on Enterprise Information Systems. (2014)
16. Cunningham, W.: The wycash portfolio management system. OOPS Messenger **4** (1992) 29–30
17. Schmid, K.: On the limits of the technical debt metaphor some guidance on going beyond. In: 4th International Workshop on Managing Technical Debt. (2013) 63–66
18. Fowler, M.: Refactoring: Improving the Design of Existing Code. Addison-Wesley (2018)
19. Nuha, M.U.: Data versioning for graph databases. Master's thesis, TU Delft Electrical Engineering (2019)
20. Gómez, A., Cabot, J., Wimmer, M.: Temporalemf: A temporal metamodeling framework. In: 37th International Conference, ER 2018, Xi'an, China, October 22–25, 2018, Proceedings. (2018) 365–381
21. Vijitbenjaronk, W.D., Lee, J., Suzumura, T., Tanase, G.: Scalable time-versioning support for property graph databases. In: 2017 IEEE International Conference on Big Data (Big Data). (2017) 1580–1589
22. Castelltort, A., Laurent, A.: Representing history in graph-oriented nosql databases: A versioning system. In: Eighth International Conference on Digital Information Management (ICDIM 2013). (2013) 228–234
23. An, D.: Find out how you stack up to new industry benchmarks for mobile page speed https://www.thinkwithgoogle.com/intl/en-ca/marketing-strategies/app-and-mobile/mobile-page-speed-new-industry-benchmarks/ Accessed: 2022-06-12.
24. Nah, F.: A study on tolerable waiting time: How long are web users willing to wait? Behaviour & Information Technology - Behaviour & IT **23** (01 2003) 285