Using Knowledge Graphs to Detect Enterprise Architecture Smells

Muhamed Smajevic, Simon Hacks, Dominik Bork

Using Knowledge Graphs to Detect Enterprise Architecture Smells

Muhamed Smajevic[1], Simon Hacks[2,3][0000−0003−0478−9347], and Dominik Bork[1][0000−0001−8259−2297]

[1] TU Wien, Business Informatics Group, Vienna, Austria
e11742556@student.tuwien.ac.at, dominik.bork@tuwien.ac.at
[2] KTH Royal Institute of Technology, Division of Network and Systems Engineering, Stockholm, Sweden
shacks@kth.se
[3] University of Southern Denmark, The Maersk Mc-Kinney Moller Institute, Odense, Denmark
shacks@mmmi.sdu.dk

Abstract. Hitherto, the concept of Enterprise Architecture (EA) Smells has been proposed to assess quality flaws in EAs and their models. Together with this new concept, a catalog of different EA Smells has been published and a first prototype was developed. However, this prototype is limited to ArchiMate and is not able to assess models adhering to other EA modeling languages. Moreover, the prototype is not integrateable with other EA tools. Therefore, we propose to enhance the extensible Graph-based Enterprise Architecture Analysis (eGEAA) platform that relies on Knowledge Graphs with EA Smell detection capabilities. To align these two approaches, we show in this paper, how ArchiMate models can be transformed into Knowledge Graphs and provide a set of queries on the Knowledge Graph representation that are able to detect EA Smells. This enables enterprise architects to assess EA Smells on all types of EA models as long as there is a Knowledge Graph representation of the model. Finally, we evaluate the Knowledge Graph based EA Smell detection by analyzing a set of 347 EA models.

Keywords: Enterprise Architecture · Model transformation · ArchiMate · Knowledge Graph · Analysis.

1 Introduction

With the increasing complexity of today's enterprises and enterprise ecosystems, creating, using, and maintaining a model representation thereof becomes increasingly challenging. Enterprise Architecture Management (EAM) with the de-facto industry standard modeling language ArchiMate [28] provides a high-level view of different enterprise domains (e.g., business, application, and technology) as well as their interrelationships. However, ArchiMate has also limitations, especially with respect to its semantic specificity [30] and the capabilities it offers to process the modeled information [7].

Naturally, with the increasing complexity of the modeled system under study, also the complexity of the model itself increases. Although Enterprise Architecture (EA) modeling is widely adopted in industry and much research is conducted in the field, the analysis of EA models is surprisingly underrepresented [2, 19]. Generally, two analysis approaches can be distinguished: manual and automated. Given the discussed complexity of EA models, manual analysis *"can be complicated and omissions or miscalculations are very likely."* [10] Automated model analysis can mitigate this problem by scaling well and by providing interactive analysis means that extend static ones [22]. Aside from first attempts to equip EA modeling by advanced visualization and analysis techniques [5, 12, 17, 21, 31, 40], automated analysis of EA models is still underdeveloped.

The value of EA models is of course threatened by the shortcomings stressed at the outset. To mitigate parts of these problems, in the paper at hand, we concentrate on the use and maintenance of EA models. In particular, we want to automatically and efficiently analyze even large EA models with the aim to detect *EA Smells*. EA Smells have been recently proposed as a novel and promising research direction [33]. EA Smells are inspired by Code Smells, which are a common means to indicate possible Technical Debts [8]. Generally, a smell describes a qualitative issue that effects future efforts (e.g., maintenance) and not the functionality. While Code Smells analyze source code, EA Smells analyze an organization from a more holistic point of view and go beyond a technical scope. Hitherto, first EA Smells and tool prototypes have been proposed aiming to detect possible flaws in EA models [24, 33, 41].

To also allow the analysis of other EA models than ArchiMate and to realize a scalable approach, we generalize the EA model to a Knowledge Graph (KG) [9] and provide queries representing respective EA Smells. Hence, the detection of EA Smells can be applied to all EA models, which can be represented as a KG – which is not uncommon in EA research [2]. We propose a *generic* and *extensible* platform that facilitates the transformation of EAs into KG representations. The platform can be easily extended to support further modeling languages. Once a transformation is realized, the existing EA Smells queries can be efficiently executed even on very large models and model corpora.

Combining the discussed challenges with the sketched solution characteristics mentioned at the outset, the research presented in the remainder of this paper aims to contribute to the following research objectives:

i) *Transforming Enterprise Architecture models into Knowledge Graphs*
ii) *Using Knowledge Graphs to automatically detect EA Smells*

The rest of the paper is organized as follows. Background information on graph-based analysis of EA models and EA Smells is presented in Section 2. The transformation of EA models into KGs is then discussed in Section 3. Section 4 reports on how the KG can facilitate the automated detection of EA Smells. A comprehensive evaluation of our approach is presented in Section 5 where we report on the transformation and analysis of a huge corpora of openly available ArchiMate models. We conclude this paper in Section 6 with a discussion and some directions for future research.

2 Background

In this section, we will first introduce the foundations and related works on graph-based analysis of enterprise architecture models (Section 2.1) before introducing the backgrounds of EA Smells (Section 2.2).

2.1 Graph-based Analysis of EA Models

Recently, the concept of Knowledge Graphs (KG) was proposed [9], which is continuously gaining more attention – also driven by the prominent use by Google in presenting the search results to its users. KGs realize an integrated representation on heterogeneous data that is ready for automated and efficient reasoning starting from (complex) graph queries toward the application of machine learning algorithms (e.g., Graph Neural Networks). At the core, a Knowledge Graph is a labelled graph that connects nodes by edges. More generally, a Knowledge Graph is *"a large network of entities, and instances for those entities, describing real world objects and their interrelations, with specific reference to a domain or to an organization."* [4, p. 27]

Interpreting EA models as graphs is a common approach in EA research [2]. For example, Garg et al. [13] propose a 3-tier architecture that allows defining EAs and their transformation into a graph structure to enable stakeholders with different visual analysis capabilities. Aier [1] propose the *EA Builder* tool that supports the identification of clusters in graphs which can then be considered as candidates for services in a service-oriented architecture. Similarly, Iacob et al. [19] quantitatively analyze layered, service-oriented EA models.

Santana et al. [36] propose to combine manual inspection by enterprise architects with automated analysis of graphs. Johnson et al. [20] interpret modeling of EAs as a probabilistic state estimation problem. They propose to facilitate Dynamic Bayesian Networks and to observe a computer network in order to predict the likeliest representation of the EA's technology layer. This was later implemented and refined in [3]. Similarly, Hacks and Lichter [16] use the graph representation to plan for future evolutions of the EA by considering different scenarios with underlying probabilities to become reality.

Taking a step further, several efforts have been taken to use graphs for maintaining and optimizing EAs. Giakoumakis et al. [14] replace existing services with new services while aiming not to disrupt the organization. Therefore, they formalize the EA as a graph and solve the resulting problem by means of multi-objective optimization. Similarly, Franke et al. [11] use a binary integer-programming model to optimize the relation between IT systems and processes based on needed functionalities. In contrast, MacCormack et al. [26] use Design Structure Matrices to analyze the coupling between the EA components. Moreover, they consider future states of the EA and generate measures that can be used to predict performance.

Further, there are also works using graph structures in the background without naming it explicitly. Österlind et al. [29] extend selected ArchiMate concepts with variables that are computed for structural analysis of the EA. Alike, Singh

et al. [38] develop seven metrics to measure criticality and impact of any element in an EA model. Holschke et al. [18] perform failure impact analysis with Bayesian Belief Networks and Buschle et al. [7] adapt ArchiMate by fault trees to analyze the availability of EA components.

2.2 EA Smells

Previously, Hacks et al. [15] proposed to combine the concept of Technical Debt [8] with the concept of EA to so called *EA Debts*. EA Debts do not solely cover technical aspects but provide a more holistic view on the entire organization including for example flaws related to organizational structures. However, their proposal lacks an effective means to measure EA Debts. Therefore, Salentin and Hacks [33] facilitated the concept of Code Smells, which is popular to measure Technical Debt, and adapted it to EA models. They started with 56 Code Smells and ended up with a catalogue of 45 EA Smells [34]. This catalog was further extended by Lehmann et al. [24] and Tieu and Hacks [41], who took inspiration from process anti-patterns and software architecture smells, respectively.

In the aforementioned catalog [34], each EA Smell is documented in the same manner [33]: First, each EA Smell has an associated *name*, possible *synonyms*, and a *description*. The *context* provides further information such as the underlying concept from other domain smells (e.g., Code Smells). An *example* is provided to ease the understanding of the EA Smell. Second, the *cause* describes the reasons for an EA Smell, while the *consequences* illustrate the negative influence on the organization. Additionally, a short description is provided how the EA Smell could be *detected*. Third, a *possible solution* is proposed to solve the EA Smell. Finally, for each EA Smell *meta-information* is provided, which eases the searching for certain EA Smells, e.g., by filtering for EA layers.

An example for an EA Smell is *Weakened Modularity* [33]. It is adapted from the Code Smell with the same name that desires each module for high cohesion and low coupling. To detect this EA Smell, for each element and all successive sub-elements the modularity ratio is calculated by dividing the number of internal references (cohesion) by the number of external references (coupling).

Fig. 1 illustrates three examples of EA Smells in an EA model. Firstly, a *Cyclic dependency* in which three services build on each other. Secondly, a *Dead Component*, which has no relations to the rest of the model. Lastly, a *Strict Layers Violation* where elements of the business layer are directly linked to an element of the technology layer. Fig. 1 contains even more flaws such as other EA Smells, underlying issues behind the smells themselves (i.e., EA Debts [15] causing EA Smells to arise), or issues with ArchiMate. However, as the focus of this work is to automate the identification of EA Smells using Knowledge Graphs, we do not elaborate further on these aspects.

To achieve an automated detection of EA Smells, Salentin and Hacks [33] developed a prototype [32] that is capable to detect 14 EA Smells listed in the catalog. Therefore, it takes an ArchiMate Exchange File as input and prints the found EA Smells. Accordingly, the prototype can only analyze ArchiMate

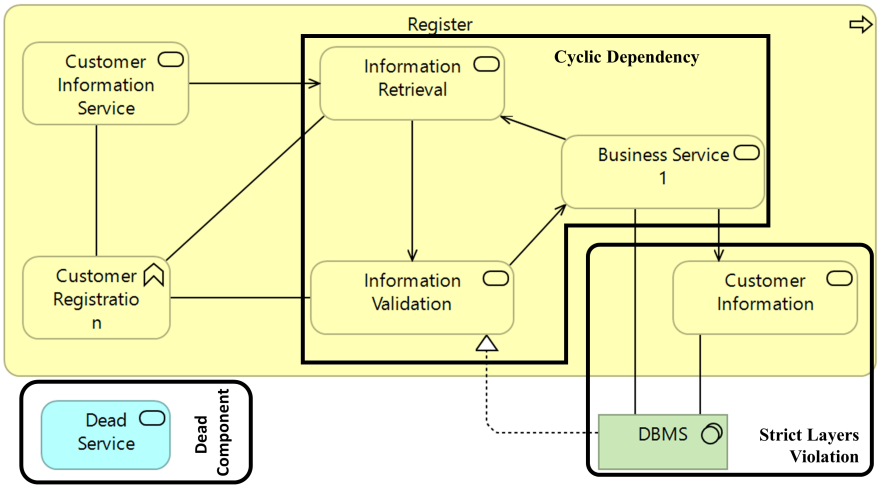

Fig. 1: Three Examples for EA Smells [33]

models and integration with other tools is not possible. Moreover, the detection of EA Smells is implemented in Java and, thus, the tool needs to be compiled every time an EA Smell is added or removed and the scalability of the detection mechanism for large EA graphs is limited.

In this paper, we therefore aim to develop a *generic*, *extensible*, and *scalable* approach that supports *i*) the transformation of EA models into KGs and *ii*) the automated detection of EA Smells by means of KG queries. In the following, we will first elaborate on the transformation in Section 3. Section 4 will then report on the realization of EA Smells detection by means of KG queries. Eventually, applicability and scalability of our approach will be evaluated in Section 5.

3 Transforming EA Models into Knowledge Graphs

In order to analyze the EA in a graph-based manner, we propose an enhancement of the *extensible Graph-based Enterprise Architecture Analysis (eGEAA)* platform (see Fig. 2). The core platform, initially proposed by Smajevic and Bork [40, 6], allows the transformation of ArchiMate models into graph structures. In this paper, we enhance this platform with the capability to transform EA models conforming to the Open Group Exchange format to a KG. In contrast to the initial proposal, which only comprised basic graph analysis metrics like *Centrality* and *Betweenness*, we furthermore enhance eGEAA by means of semantic queries to automatically detect EA Smells (detailed in Section 4).

Compared to the related works (see Section 2), the eGEAA platform is generic and extensible in two ways: First, it builds upon the conceptual models produced by state-of-the-art metamodeling platforms like Ecore and ADOxx instead of being realized on – and being thus constrained to – one modeling language or tool. This enables the transformation of any conceptual model created

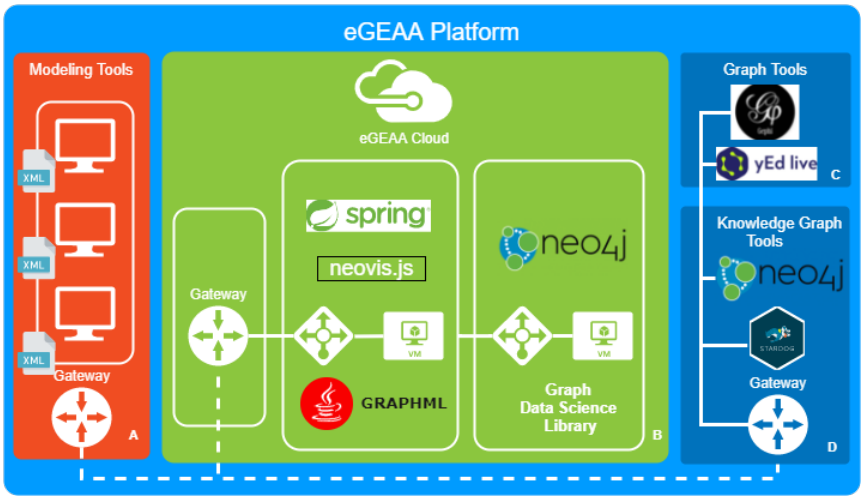

Fig. 2: eGEAA platform architecture – adapted from [40].

with these platforms into a KG. Second, we transform the conceptual model into GraphML, a standardized graph representation format [27] that enables interchangeability with many graph analysis (e.g., Gephi, yEd) and KG tools (e.g., neo4j, Stardog). Consequently, eGEAA builds a bridge between powerful modeling (and metamodeling platforms) and graph analysis and reasoning tools.

Listing 1 presents the pseudo-code for the transformation of EA models into KGs. The transformation combines two parts, the generic part responsible for transforming any conceptual model derived from the Ecore metamodel into a graph structure, and the second part, that takes care of the specifics of the ArchiMate modeling language and the specific implementation of the Ecore metamodel in Archi.

The first rule transforms a *Grouping*, *Folder*, or *View* element into a nested *Graph* thereby overriding the generic transformation that would have resulted in a Node. All contents connected with that grouping through a *nested element* relation in the ArchiMate (ArchiMate) model will be added as *Node*s in the nested graph. Secondly, since Archi stores the ArchiMate relationships as entities (i.e., *IArchimateRelationshipEntity*s), the generic transformation rule needs to be overridden to transform an IArchimateRelationshipEntity into an *Edge* with an additional edge data to store the relationship endpoints.

Fig. 3 visualizes the conceptual view on the model transformation approach. It shows that the transformation itself is specified on the metamodel-level, i.e., using elements of the Archi metamodel and elements of the GraphML metamodel when specifying the transformation rules. This enables the execution of the transformation on any source Archi-based EA model which will transform the conceptual model into a KG representation. From a technical point of view, the source and target model are stored as xml documents – the former in the Open Group Exchange format, the latter in the GraphML format.

Algorithm 1: Archi ArchiMate model to GraphML transformation.

```
   Input: Archi-based ArchiMate model instance in Open Group Exchange xml format.
   Output: Knowledge Graph serialized in GraphML xml format.
 1 for EObject package : input.eAllContents().getPackages() do
 2 |   Graph g ← transformPackage(package)
 3 |   for EObject eo : package.eAllContents() do
 4 |   |   if eo instanceof Grouping, Folder, View then
 5 |   |   |   Graph subg ← createSubgraph(eo)
 6 |   |   |   for EObject eo : subg.getEAllNestedElements() do
 7 |   |   |   |   Node n ← transformNode(eo)
 8 |   |   |   |   for EAttribute a : eo.getEAllAttributes() do
 9 |   |   |   |   |   n.addAttribute(transformAttribute(a))
10 |   |   |   |   end
11 |   |   |   |   subg.add(n)
12 |   |   |   end
13 |   |   |   g.add(subg)
14 |   |   else
15 |   |   |   Node n ← transformNode(eo)
16 |   |   |   for EAttribute a : eo.getEAllAttributes() do
17 |   |   |   |   n.addAttribute(transformAttribute(a))
18 |   |   |   end
19 |   |   |   g.add(n)
20 |   |   end
21 |   end
22 |   for EObject eo : package.eAllContents() do
23 |   |   if eo instanceof IArchimateRelationshipEntity then
24 |   |   |   Edge edge ← transformEdge(eo)
25 |   |   else
26 |   |   |   for EReference ref : eo.getEAllReferences() do
27 |   |   |   |   Edge edge ← transformEdge(ref)
28 |   |   |   end
29 |   |   end
30 |   |   edge.source ← findNode(eo)
31 |   |   edge.target ← findNode(eo.get(ref))
32 |   |   for EAttribute a : ed.getEAllAttributes() do
33 |   |   |   edge.addAttribute(transformAttribute(a))
34 |   |   end
35 |   |   g.add(edge)
36 |   end
37 |   output.add(g)
38 end
39 return output
```

4 Knowledge Graph based EA Smells Detection

Once the transformation of ArchiMate models into GraphML is achieved, enterprise architects can visually explore the graph structure or apply basic graph analysis techniques as e.g., reported in [40]. More advanced analysis by means of reasoning is required for larger models and where the interests go beyond basic structural questions. This latter case is followed upon in the remainder of this paper, where we use KG queries to automatically detect EA Smells.

Table 1 lists already in Java implemented EA Smells [33, 32] and maps them to KG queries that are capable to automatically detect them. The presented smells hereafter emphasize structural characteristics such as circular dependencies while semantic aspects like the relation of data elements to service elements are also considered. Of course, future extensions of the catalogue will most likely be dominated by even more complex semantic smells (cf. [35]).

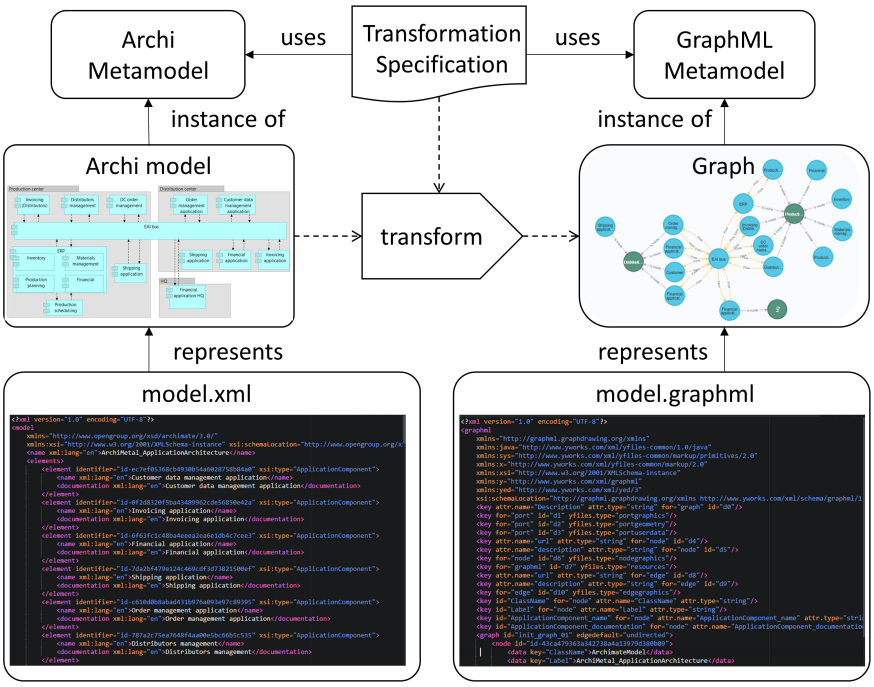

Fig. 3: Archi to Knowledge Graph transformation example.

Table 1: KG queries resolving EA Smells

EA Smell [34]	**KG Query**

Chatty Service
A high number of operations is required to complete one abstraction. Such operations are typically rather simple tasks that needlessly slow down an entire process.
	`MATCH (a)-[r]-(b)`
	`WHERE a.ClassName contains 'Service'`
	`and b.ClassName contains 'Service'`
	`and not r.Label contains 'Composition'`
	`with a,count(r) as cnt`
	`where cnt>4`
	`MATCH (a)-[r1]-(b1)`
	`WHERE a.ClassName contains 'Service' and b1.ClassName contains 'Service'`
	`return a,b1, cnt`

Cyclic Dependency Two or more abstractions directly or indirectly depend on each other.	`MATCH (a)-[r1]->(b)-[r2]->(c)-[]->(a)` `return a,b,c`

Data Service
A service that exclusively performs information retrieval and typically provides only simple read operations.
	`MATCH (a)-[r1]-(b1)`
	`WHERE a.ClassName contains 'Service'`
	`and (b1.ClassName = 'BusinessObject'`
	`or b1.ClassName = 'DataObject'`
	`or b1.ClassName = 'SystemSoftware')`
	`with a,r1,b1`
	`match (a)`
	`where not (a)--(:BusinessService)`
	`return a`

Dead Component
A component is no longer used or used to support potential future behavior.

```
MATCH (n)
WHERE not (n)--()
return n
```

Dense Structure
An EA repository has dense dependencies without any particular structure.

```
MATCH (p)
RETURN CASE WHEN avg(apoc.node.degree(p))
      >1.75 THEN 1 ELSE 0 END AS result;
```

Documentation
A lengthy documentation often points to unnecessary complex structures.

```
MATCH (n)
where size(n.documentation)>256
RETURN n
```

Duplication
Two or more abstractions with highly similar functionality exist.

```
MATCH (a),(b)
where a<>b and  a.ClassName = b.ClassName
      and apoc.text.jaroWinklerDistance(a.
      Label, b.Label)>0.8
RETURN a,b,apoc.text.jaroWinklerDistance(a.
      Label, b.Label) as similarNameScore
```

Hub-like Modularization
This smell arises when an abstraction has dependencies (both incoming and outgoing) with a large number of other abstractions, being a single point of failure.

```
match (a)-[r]-(b)
where  (r.Label contains 'Aggregation' or r.
      Label contains 'Realization' or r.Label
       contains "Composition" or r.Label
      contains "Assignment")
     and a.ArchimateLayer = b.ArchimateLayer
with a, collect(r) as rels,  a+collect(b) as
      cluster
match (m)-[r1]-(n)
where not (r1.Label contains 'Aggregation'
      or r1.Label contains 'Realization' or
      r1.Label contains "Composition" or r1.
      Label contains "Assignment") and
(m in cluster and not n in cluster)
with a, cluster, collect(r1) as fanout
match  (m)-[r2]-(n)
where not (r2.Label contains 'Aggregation'
      or r2.Label contains 'Realization' or
      r2.Label contains "Composition" or r2.
      Label contains "Assignment") and
 (not m in cluster and n in cluster)
with a, cluster,  fanout, collect(r2) as
      fanin
where size(fanout) > 7 and size(fanin)>7
return a, cluster, size(fanout), size(fanin)
```

Lazy Component
A component that is not doing enough to pay for itself should be eliminated. Those components often only pass messages on to another.

```
MATCH (n)
where n.Label contains 'controller' or n.
      Label contains 'manager'
RETURN n
```

Message Chain
A number of services that rely on each other, while providing similar functionality.

```
MATCH (a)-[r1]->(b)-[r2]->(c)-[]->(d)-[]->(e
      )
where a.ClassName contains 'Service'
       and b.ClassName contains 'Service'
       and c.ClassName contains 'Service'
       and d.ClassName contains 'Service'
       and e.ClassName contains 'Service'
return a,b,c,d,e
```

Shared Persistency
Different services access the same database. In the worst case, different services access the same entities of the same schema.

```
MATCH (a)-[r]-(b)
WHERE a.ClassName='SystemSoftware' and (r.
      Label='AssociationRelationship' or r.
      Label='RealizationRelationship' or r.
      Label='AssignmentRelationship')
with a,count(r) as cnt
MATCH (a)-[r1]-(b1)
where cnt>1 and (r1.Label='
      AssociationRelationship' or r1.Label='
      RealizationRelationship' or r1.Label='
      AssignmentRelationship')
return a,b1
```

Strict Layers Violation
An element skips the EA layer directly beneath and accesses a layer further below instead.

```
MATCH (a)-[r]-(b)
where a.ArchimateLayer contains 'Business'
      and b.ArchimateLayer contains '
      Technology' //example
return a,b,r
```

Weakened Modularity
Each module must strive for high cohesion and low coupling. This smell arises when a module exhibits high coupling and low cohesion.

```
match (a)-[r]-(b)
where  (r.Label contains 'Aggregation' or r.
      Label contains 'Realization' or r.Label
       contains "Composition" or r.Label
      contains "Assignment")
     and a.ArchimateLayer = b.ArchimateLayer
with a, collect(r) as rels,  a+collect(b) as
       cluster
match (m)-[r1]-(n)
where m in cluster and n in cluster
with a, cluster, collect(r1) as internal
match  (m)-[r2]-(n)
where not (r2.Label contains 'Aggregation'
      or r2.Label contains 'Realization' or
      r2.Label contains "Composition" or r2.
      Label contains "Assignment") and
(not m in cluster and n in cluster)  or (m
      in cluster and not n in cluster)
with a, cluster,  internal, collect(r2) as
      external
where size(internal) < size(external) and
      size(internal)>3
return a, cluster, size(internal), size(
      external)
```

5 Evaluation

For evaluating our approach, we refer to the openly available model corpora of the MAR search engine [25]. In summary, we found 369 ArchiMate models which were created with the Archi modeling tool. In average, a model in the corpus comprised 51.41 ± 97.04 Nodes and 47.14 ± 70.23 Edges. We transformed these models using the eGEAA platform and executed the EA Smells queries defined in Table 1. The evaluation aimed to respond to the following research questions:

RQ.1 – Feasibility Is our approach feasible to automatically detect EA Smells in ArchiMate models? If yes, how often do specific smells (co-) occur?

RQ.2 – Performance How efficient is our Knowledge Graph based approach in detecting EA Smells?

RQ.1 – Feasibility For evaluating the feasibility, we collected a set of openly available ArchiMate models [25] and transformed them initially into the Open Group Exchange format using Archi. From the resulting set of 369 models, the eGEAA platform was able to automatically transform 347 of them (94%) into a KG stored in GraphML. The few models that were not transformed had some encoding issues or the source model was corrupt. As can be derived from Fig. 4 (left), we found all implemented EA Smells in the data set. In future research, we plan to extend the data set and to also involve real models form practitioners. In an action design research setting, we could then investigate, how practitioners value our EA Smell detection approach.

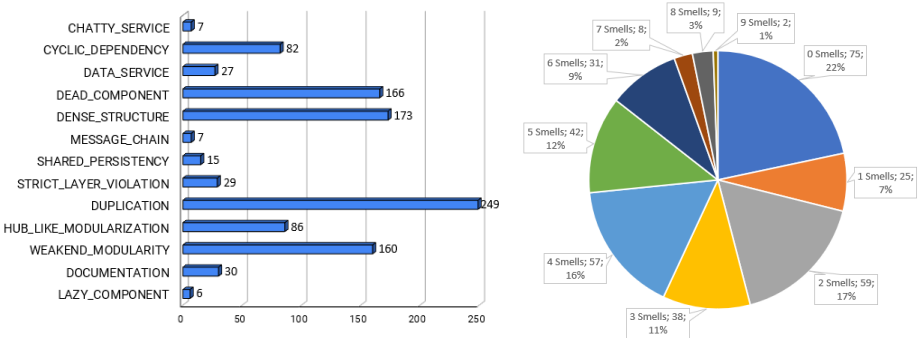

Fig. 4: Detected EA Smells.

The results of applying the EA Smells queries of Table 1 are summarized in Fig. 4. The detailed analysis showed that 78.38% of the EA models had at least one smell. Fig. 4 (right) shows, how many EA Smells have been found in how many of the analyzed EA models. It can be derived, that 45.82% of the models had at most two smells, whereas the majority of the EA models had three or more smells. Noteworthy, this is only an indicator of the smell's existence in a model, not the number of incarnations of the smell in a model.

We then were also interested to see, which smells occur most often and which smells co-occur most often. The data showed, that DUPLICATION (249 hits), DENSE_STRUCTURE (173 hits), DEAD_COMPONENT (166 hits), and WEAKEND_MODULARITY (160 hits) together make up for almost 75% of all detected EA Smells. When analyzing the relationships between the detected EA Smells, the following three co-occurrences were most frequent in the data set: 162 co-occurrences (46.68%) of DENSE_STRUCTURE and DUPLICATION, 155 co-occurrences (44.66%) of DUPLICATION and WEAKEND_MODULARITY, and 153 co-occurrences (44.09%) of DEAD_COMPONENT and DUPLICATION.

RQ.2 – Performance Past research has indicated that Knowledge Graph queries can be executed highly efficiently also on huge graphs with many thousands or even millions of nodes [4]. With our experiments, we can confirm this observation also for our Knowledge Graph based EA Smell detection. Fig. 5 plots the relationship between the size of the Knowledge Graph (x-axis) and the model transformation time (blue) and the KG query time (orange) on the y-axis, both measured in milliseconds.

It can be derived, that the performance is stable even when analyzing KGs with more than 1000 elements (nodes and edges). In reality, it is unlikely to see larger EA models (cf. [23, 37]), we can therefore conclude, that the performance of our solution is promising to tackle the complexity of the task at hand. While the model transformation time remains stable even for large KGs with an average time of 7.46ms ±23.85ms, the query execution time naturally increases with the size of the KG with an average of 786.31ms ±1044.09ms.

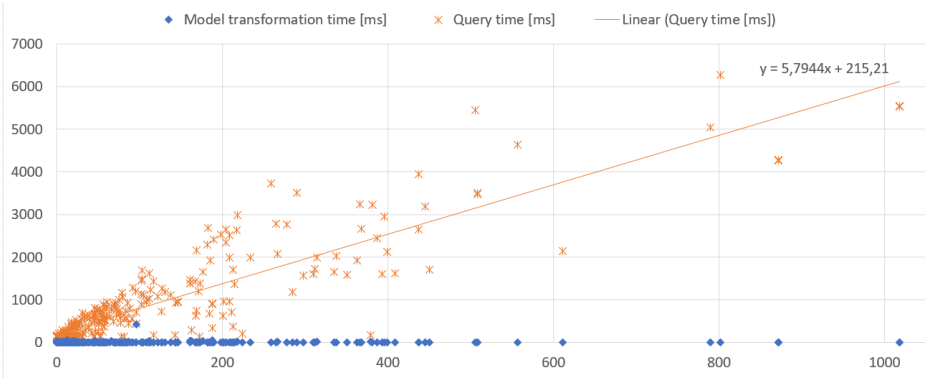

Fig. 5: Performance of the EA Smells detection.

To show also the relative performance of our solution compared to the prototype presented in [32], we used the same set of experimental models and compared the query execution times. These early investigations yielded interesting insights and confirmed, that our approach is stable with respect to time in executing smell detection while it is a bit slower in detecting EA Smells than the previous solution. However, the results vary depending on the smell (i.e., the complexity of the query itself). Moreover, these first results may not tell the true story since the tracked time in our experiments involved setting up a connection to a database and transporting from and to a database server instead of the pure query execution time as for the previous solution. In our future research we will, amongst others, therefore conduct further experiments to investigate the performance of our approach more rigorously and comparatively.

6 Conclusion

In this paper, we showed, how we enhanced the extensible Graph-based Enterprise Architecture Analysis (eGEAA) platform to automatically transform Enterprise Architectures modeled with the ArchiMate modeling language into Knowledge Graphs. We furthermore showed, how this Knowledge Graph structure can facilitate the efficient detection of EA Smells. For this, we transformed a representative set of EA Smells into corresponding semantic KG queries.

For evaluating our approach, we created a data set comprising 347 ArchiMate models. After transforming them into KGs, we applied the EA Smell queries and analyzed the results. This elaborated quantitative evaluation proved feasibility and the performance of the KG based EA Smell detection approach. Compared to existing solutions, our proposal is *generic*, i.e., it can be easily adopted for different EA modeling languages and tools [39], and *extensible*, i.e., further EA Smells or different EA analysis techniques can be easily realized by means of KG queries. However, a qualitative assessment of the smell identification is missing

and needs to be addressed to ensure that it is complete and correct. Such an evaluation would require a curated set of EA models which still needs to be developed.

In our future work, we aim to further extend the catalog of EA Smells. Furthermore, we plan to conduct empirical experiments with enterprise architects. We expect that through such experiments, we can not only evaluate the ease of use and usability of the platform, but also the intention to use EA Smells in practice. Eventually, we perceive this work as a foundation for an entire stream of research that concerns adding knowledge to the Knowledge Graph (e.g., KG embeddings) and applying advanced reasoning (e.g., Graph Neural Networks).

The presented platform is available open source [6], enabling the enterprise modeling and enterprise architecture communities to use the platform for realizing their specific EA analysis purpose and to provide valuable feedback. We are also currently working on a plug-in that enables the execution of our KG based EA Smell detection directly within the Archi modeling tool.

References

1. Aier, S.: How clustering enterprise architectures helps to design service oriented architectures. In: 2006 IEEE International Conference on Services Computing (SCC'06). pp. 269–272. IEEE (2006)
2. Barbosa, A., Santana, A., Hacks, S., Stein, N.v.: A taxonomy for enterprise architecture analysis research. In: 21st International Conference on Enterprise Information Systems. vol. 2, pp. 493–504. SciTePress (2019)
3. Bebensee, B., Hacks, S.: Applying dynamic bayesian networks for automated modeling in archimate: A realization study. In: 23rd IEEE International Enterprise Distributed Object Computing Workshop, EDOC Workshops 2019, Paris, France, October 28-31, 2019. pp. 17–24. IEEE (2019)
4. Bellomarini, L., Fakhoury, D., Gottlob, G., Sallinger, E.: Knowledge graphs and enterprise ai: the promise of an enabling technology. In: 2019 IEEE 35th International Conference on Data Engineering (ICDE). pp. 26–37. IEEE (2019)
5. Bork, D., Gerber, A., Miron, E., van Deventer, P., van der Merwe, A., Karagiannis, D., Eybers, S., Sumereder, A.: Requirements engineering for model-based enterprise architecture management with archimate. In: Proceedings EOMAS 2018. pp. 16–30. Springer (2018)
6. Bork, D., Smajevic, M.: Source code repository of the eGEAA platform. `https://github.com/borkdominik/eGEAA` (2021)
7. Buschle, M., Johnson, P., Shahzad, K.: The enterprise architecture analysis tool - support for the predictive, probabilistic architecture modeling framework pp. 3350–3364 (2013)
8. Cunningham, W.: The wycash portfolio management system. SIGPLAN OOPS Mess. **4**(2), 29–30 (Dec 1992)
9. Fensel, D., Simsek, U., Angele, K., Huaman, E., Elias, K., Panasiuk, O., Toma, I., Umbrich, J., Wahler, A.: Knowledge Graphs – Methodology, Tools and Selected Use Cases. Springer, Heidelberg, Germany (2020)
10. Florez, H., Sánchez, M., Villalobos, J.: A catalog of automated analysis methods for enterprise models. SpringerPlus **5**(1), 1–24 (2016)

11. Franke, U., Holschke, O., Buschle, M., Narman, P., Rake-Revelant, J.: It consolidation: An optimization approach. In: 2010 14th IEEE International Enterprise Distributed Object Computing Conference Workshops. pp. 21–26 (2010)
12. Gampfer, F., Jürgens, A., Müller, M., Buchkremer, R.: Past, current and future trends in enterprise architecture—a view beyond the horizon. Computers in Industry **100**, 70–84 (2018)
13. Garg, A., Kazman, R., Chen, H.M.: Interface descriptions for enterprise architecture. Science of Computer Programming **61**(1), 4 – 15 (2006)
14. Giakoumakis, V., Krob, D., Liberti, L., Roda, F.: Technological architecture evolutions of information systems: Trade-off and optimization. Concurrent Engineering **20**(2), 127–147 (2012). https://doi.org/10.1177/1063293X12447715
15. Hacks, S., Hofert, H., Salentin, J., Yeong, Y.C., Lichter, H.: Towards the definition of enterprise architecture debts. In: IEEE 23rd International Enterprise Distributed Object Computing Workshop (EDOCW). p. 9–16. IEEE (2019). https://doi.org/10.1109/EDOCW.2019.00016
16. Hacks, S., Lichter, H.: A probabilistic enterprise architecture model evolution. In: 22nd IEEE International Enterprise Distributed Object Computing Conference, EDOC 2018, Stockholm, Sweden, October 16-19, 2018. pp. 51–57. IEEE Computer Society (2018)
17. Hinkelmann, K., Gerber, A., Karagiannis, D., Thoenssen, B., Van der Merwe, A., Woitsch, R.: A new paradigm for the continuous alignment of business and it: Combining enterprise architecture modelling and enterprise ontology. Computers in Industry **79**, 77–86 (2016)
18. Holschke, O., Närman, P., Flores, W.R., Eriksson, E., Schönherr, M.: Using enterprise architecture models and bayesian belief networks for failure impact analysis. In: Feuerlicht, G., Lamersdorf, W. (eds.) Int. Conference on Service-Oriented Computing. pp. 339–350. Springer (2009)
19. Iacob, M.E., Jonkers, H.: Quantitative analysis of enterprise architectures. In: Interoperability of Enterprise Software and Applications, pp. 239–252. Springer (2006)
20. Johnson, P., Ekstedt, M., Lagerström, R.: Automatic Probabilistic Enterprise IT Architecture Modeling: A Dynamic Bayesian Networks Approach. In: Franke, U., Lapalme, J., Johnson, P. (eds.) 20th International Enterprise Distributed Object Computing Workshop (EDOCW). pp. 123–129 (2016)
21. Jugel, D.: An integrative method for decision-making in EA management. In: Zimmermann, A., Schmidt, R., Jain, L.C. (eds.) Architecting the Digital Transformation - Digital Business, Technology, Decision Support, Management, pp. 289–307. Springer (2021)
22. Jugel, D., Kehrer, S., Schweda, C.M., Zimmermann, A.: Providing EA decision support for stakeholders by automated analyses. In: Digital Enterprise Computing (DEC 2015). pp. 151–162. GI (2015)
23. Lagerström, R., Baldwin, C., MacCormack, A., Dreyfus, D.: Visualizing and measuring enterprise architecture: an exploratory biopharma case. In: IFIP Working Conference on The Practice of Enterprise Modeling. pp. 9–23. Springer (2013)
24. Lehmann, B.D., Alexander, P., Lichter, H., Hacks, S.: Towards the identification of process anti-patterns in enterprise architecture models. In: 8th International Workshop on Quantitative Approaches to Software Quality in conjunction with the 27th Asia-Pacific Software Engineering Conference (APSEC 2020). vol. 2767, pp. 47–54. CEUR-WS (2020)

25. López, J.A.H., Cuadrado, J.S.: MAR: a structure-based search engine for models. In: Syriani, E., Sahraoui, H.A., de Lara, J., Abrahão, S. (eds.) MoDELS '20: ACM/IEEE 23rd International Conference on Model Driven Engineering Languages and Systems, Virtual Event, Canada, 2020. pp. 57–67. ACM (2020)
26. Maccormack, A.D., Lagerstrom, R., Baldwin, C.Y.: A methodology for operationalizing enterprise architecture and evaluating enterprise it flexibility. Harvard Business School working paper series# 15-060 (2015)
27. Messina, A.: Overview of standard graph file formats. Tech. Rep. RT-ICAR-PA-2018-06 (2018), http://dx.doi.org/10.13140/RG.2.2.11144.88324
28. OMG: ArchiMate® 3.1 Specification. The Open Group (2019), http://pubs.opengroup.org/architecture/archimate3-doc/
29. Österlind, M., Lagerström, R., Rosell, P.: Assessing modifiability in application services using enterprise architecture models – a case study. In: Proceedings TEAR 2012. pp. 162–181. Springer (2012)
30. Pittl, B., Bork, D.: Modeling digital enterprise ecosystems with archimate: a mobility provision case study. In: International Conference on Serviceology. pp. 178–189. Springer (2017)
31. Roelens, B., Steenacker, W., Poels, G.: Realizing strategic fit within the business architecture: the design of a process-goal alignment modeling and analysis technique. Software & Systems Modeling **18**(1), 631–662 (2019)
32. Salentin, J., Hacks, S.: Enterprise architecture smells prototype (2020), https://git.rwth-aachen.de/ba-ea-smells/program
33. Salentin, J., Hacks, S.: Towards a catalog of enterprise architecture smells. In: Gronau, N., Heine, M., Krasnova, H., Poustcchi, K. (eds.) Entwicklungen, Chancen und Herausforderungen der Digitalisierung: Proceedings der 15. Internationalen Tagung Wirtschaftsinformatik, WI 2020, Potsdam, Germany, March 9-11, 2020. Community Tracks. pp. 276–290. GITO Verlag (2020)
34. Salentin, J., Lehmann, B., Hacks, S., Alexander, P.: Enterprise architecture smells catalog (2021), https://swc-public.pages.rwth-aachen.de/smells/ea-smells/
35. Sales, T.P., Guizzardi, G.: Ontological anti-patterns: empirically uncovered error-prone structures in ontology-driven conceptual models. Data & Knowledge Engineering **99**, 72–104 (2015)
36. Santana, A., Fischbach, K., Moura, H.: Enterprise architecture analysis and network thinking: A literature review. In: 2016 49th Hawaii International Conference on System Sciences (HICSS). pp. 4566–4575. IEEE (2016)
37. Schoonjans, A.: Social network analysis techniques in enterprise architecture management. Ph.D. thesis, PhD thesis, Ghent University, Ghent (2016)
38. Singh, P.M., van Sinderen, M.J.: Lightweight metrics for enterprise architecture analysis. In: International Conference on Business Information Systems. pp. 113–125. Springer (2015)
39. Smajevic, M., Bork, D.: From conceptual models to knowledge graphs: A generic model transformation platform. In: MoDELS'21: ACM/IEEE 24th International Conference on Model Driven Engineering Languages and Systems (MODELS) – Tools & Demonstrations. p. in press. ACM/IEEE (2021)
40. Smajevic, M., Bork, D.: Towards graph-based analysis of enterprise architecture models. In: Proceedings of the 40th International Conference on Conceptual Modeling. p. in press (2021)
41. Tieu, B., Hacks, S.: Determining enterprise architecture smells from software architecture smells. In: 23rd IEEE International Conference on Business Informatics Workshops (to be published). IEEE (2021)