Using Machine Learning Techniques for Evaluating the Similarity of Enterprise Architecture Models

Technical Paper

Vasil Borozanov[1], Simon Hacks[1], and Nuno Silva[2]

[1] Research Group Software Construction, RWTH Aachen University, Germany
borozanov@gmail.com hacks@swc.rwth-aachen.de
[2] Department of Computer Science and Engineering, Technical University of Lisbon, Lisbon, Portugal
nuno.silva@inov.pt

Abstract. Enterprises Architectures (EA) are facilitated to coordinate enterprise's business visions and strategies successfully and effectively. The practitioners of EA (architects) communicate the architecture to other stakeholders via architecture models. We investigate the scenario where accepted architecture models are stored in a repository. We identified the problem of unnecessary repository expansion by adding model components with similar properties or behavior as already existing repository components. The proposed solution aims to find those similar components and to notify the architect about their existence.
We present two approaches for defining and combining similarities between EA model components. The similarity measures are calculated upon the properties of the components and on the context of their usage. We further investigate the behavior of similar architecture models and search for associations in order to obtain components that might be of interest. At the end, we provide a prototype tool for both generating requests and obtaining a result.

Keywords: Enterprise Architecture · Model · Graph · Machine Learning

1 Introduction

Enterprises without centralized management of their strategic plans and business processes lack the possibility of providing consistency and direction for their activities. To avoid this, a mechanism for coordinating the integrated development and use of shared information systems and data is required [15]. Enterprise Architecture (EA) refers to both definition and representation of a high-level view of the business processes and IT systems within an organization [22]. It is a well-defined practice that applies architecture principles and practices in a standardized way to execute their strategies, in order to determine how an organization can most effectively achieve its current and future objectives. Their goal is to optimize across processes into an integrated environment that is responsive to change and supportive of the delivery of the business strategy [23].

The Enterprise Architecture Management (EAM) is a discipline consisting of functions related to EA, such as maintenance or providing information gathered from the EA [24]. EAM is a management practice that sets and maintains a set of guidelines and architecture principles that guide the design and development of EA to achieve its vision and the strategy [2].

The practitioners of EA, called enterprise architects, are inter alia responsible for modeling the EA using architecture *models*. The fundamental building blocks of the models are the *components* and the *relations* between them. For easier understanding and communicating of the architecture to the stakeholders, the architects can develop a set of representations of the overall architecture called *views*.

We investigate the scenario of a company that uses a *repository* of all accepted models. In time, the repository can grow into one complex structure of components and relations. Adding a new model can cause unnecessary expansion in the repository if they are not checked beforehand. Components with similar attributes, or components used in the same context but with different names will be treated as newly introduced components and added again in the repository. Elaborating on this problem of repository pollution, we state our research question:

How can machine learning techniques enable enterprise architects to avoid adding duplicates to the repository?

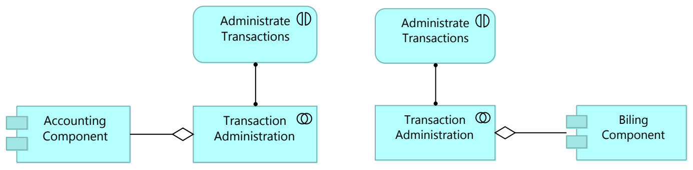
The pollution of the repository is critical for two reasons. First, the repository expanses with respect to its total number of contained components. This leads to a higher complexity of the whole repository and, thus, makes it for the enterprise architects harder to understand. However, the second reason is much more critical: all reports on the repository are distorted, probably resulting in wrong decisions of EA's stakeholders.

To elaborate on our research question, next we give a concrete example, which allows the reader to better understand the problem. Afterwards, we describe the theoretical background, which is needed to answer our research question. In section 5, we show how we apply these theories on our concrete problem, before we discuss our implementation. Section 7 outlines our conducted evaluation, followed by related work and our conclusion.

2 Exemplary Problem

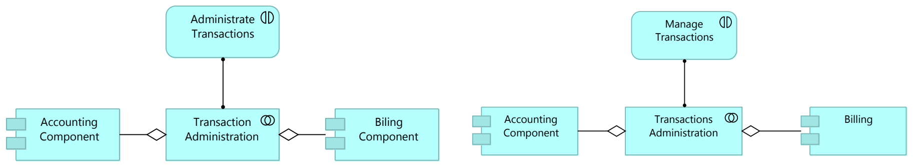
To enable a better understanding of the problem of repository pollution and how it occurs, we take a simple illustration. The models provided in Fig. 1a and Fig. 1b are developed independently. After the acceptance of both, the two models form the initial state of the repository, depicted in Fig. 2a.

In the repository, *Transaction Administration* aggregates both *Accounting* and *Billing* components from different models, while retaining only one instance of *Transaction Administration*. An architect decides to implement the similar


Administrate Transactions
Accounting Component
Transaction Administration
Administrate Transactions
Transaction Administration
Biling Component

(a) (b)

Fig. 1: Different EA models example



(a) Simplified scenario of the state of the repository (b) Model containing redundant components

Fig. 2

financial scenario in a different model (Fig. 2b). The system proceeds to integrate this model in the repository, which results in several replicated components (see Fig. 3):

– *Billing*: The architect supplied a shorter name for the component *Billing Component*.
– *Transactions Administration*: Providing behavior for administrating several transactions can possibly be replaced by the already existing *Transaction Administration*.
– *Manage Transactions*: Although the name differs from the *Administrate Transactions*, they provide same functionality.

Currently, there is no mechanism for redundancy check. Components with similar attributes and purpose can be stored multiple times, which in turn introduces management issues due to the increased complexity of the repository data. This also makes it difficult for architects to reuse certain components.

3 Research Method

To formalize our research method, we use the Design Science Research Methodology (DSRM) [14]. According to DSRM, there are several steps to apply this process successfully. Having defined and formulated the problem, we proceed with the following:

Objectives and solution: The objective was to develop a solution that will identify all new components, evaluate the model against the repository, and

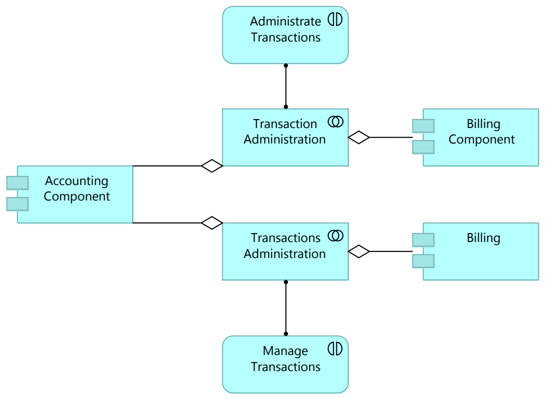

Administrate Transactions
Transaction Administration
Billing Component
Accounting Component
Transactions Administration
Billing
Manage Transactions

Fig. 3: Simplified scenario of the state of the repository formed by adding the model in Fig. 2b to the repository in the state as in Fig. 2a

return a list of components already stored in the repository that can be reused in the given project. The final decision whether the suggested components will be incorporated or not is left to the architect. This ensures that the architecture model retains its correctness.

Design and development: The solution we proposed was a machine learning model. The data on our disposal was unlabeled - we did not have any information on what the correct substitution for the specific component should be. Therefore, we focused on the unsupervised approaches, combining them into one suitable model.

Demonstration and Evaluation: We tested our solution with a simulated repository and architecture models where we knew in advance the correct substitutions for every newly introduced component. This allowed us to evaluate the correctness of the recommended components on labeled data.

Communication: The solution was distributed to the architects as a software service. It consisted of two parts: a server-side module where we performed the evaluation, and a client-side module which allowed the architects to select the architecture model that they wanted to evaluate.

4 Theoretical Background

This section describes the theory behind our approach. First, we define similarity and distance in the scope of our research. Then, we present the fundamental aspects of graph theory used in our proposed solution. Finally, we formalize association rules and a set of parameters used for association analysis.

4.1 Similarity and Distance

With the use of complex objects, we identify the need of fundamental operation for similarity assessment between two objects. If we consider the following spaces: $\mathbb{F}$, which denotes the feature space of an object and $\mathbb{R}^{\mathbb{F}}$, the space of all feature representations, then such function maps the feature space to a score $s : \mathbb{R}^{\mathbb{F}} \times \mathbb{R}^{\mathbb{F}} \to \mathbb{R}$.

Similarity function is the measure which determines how closely related two objects are based on their representation, following the given properties [17]:

- Symmetry: $\forall x, y \in \mathbb{R}^{\mathbb{F}} : s(x, y) = s(y, x)$ - the order of the objects in the input should not affect the output score
- Maximum self-similarity: $\forall x, y \in \mathbb{R}^{\mathbb{F}} : s(x, x) \geq s(x, y)$ - nothing can be more similar than the object itself

If two objects are similar, then the similarity function will have a high positive score.

In contrast to similarity, the *dissimilarity* is a measure defined by a distance function that quantifies how different two objects are. For function $d : \mathbb{R}^{\mathbb{F}} \times \mathbb{R}^{\mathbb{F}} \to \mathbb{R}$ to qualify as distance, in needs to fulfill the following constraints [6]:

- Non-negativity: $\forall x, y \in \mathbb{R}^{\mathbb{F}} : d(x, y) \geq 0$
- Reflexivity: $\forall x \in \mathbb{R}^{\mathbb{F}} : d(x, x) = 0$
- Symmetry: $\forall x, y \in \mathbb{R}^{\mathbb{F}} : d(x, y) = d(y, x)$

Both dissimilarity and similarity models express the closeness between objects. The conversion from the first to latter is essentially converting from distance to similarity function. Since they are negatively correlated, any monotonically decreasing transformation can be applied to convert similarity measures into dissimilarity. Consequently, any monotonically increasing transformation can be applied to convert the similarity to distance. If the similarity values are normalized in the range from 0 to 1, then the corresponding dissimilarity (distance) can be expressed as:

$$d(x, y) = 1 - s(x, y) \tag{1}$$

4.2 Graph Theory

In order to search for similarities between the EA models, a proper representation is needed. We find that representing the models as labeled graphs is the most acceptable solution [5].

A **labeled graph** is defined by a the tuple $G = (V, E, r_V, r_E)$ such that:

- V is a finite set of vertices,
- E is a finite set of edges between the vertices,
- $r_V \subseteq V \times L_V$ is the function that assigns labels to vertices,
- $r_E \subseteq V \times V \times L_E$ is the function that assigns labels to edges.

In this manner, we can represent the EA components as vertices and the connections between them as graph edges.

To calculate the similarity between vertices in a graph, both the labels of the vertices and the edges in the graph need to be considered. The SimRank [10] algorithm takes this into consideration. It accepts a labeled graph G as input and compares each vertex of the graph with the rest. Two vertices are considered similar if they are referenced by other similar vertices in the graph, or formally expressed with by Eq. 2:

$$s^{SR}(p,q) = \frac{C}{|I(p)||I(q)|} \sum_{i=1}^{|I(p)|} \sum_{j=1}^{|I(q)|} s^{SR}(I_i(p), I_j(q)) \tag{2}$$

The constant $C \in \mathbb{R}$ is a user given value from 0 to 1 called the decay factor, $I(p)$ and $I(q)$ are the set of all predecessor vertices of p and q (other nodes who point to p and q) with the total count of $|I(p)|$ and $|I(q)|$ accordingly, and $I_i(p)$ and $I_j(q)$ are the i-th and j-th predecessor of the nodes p and q. Dividing by the total number of predecessors pairs allows us to obtain normalized value: a range between 0 (maximum dissimilarity) to 1. For any vertex v that has no predecessors ($I(v) = \emptyset$), the similarity is set to zero. Alternatively this can be expressed using the set of successors $O(p)$ and $O(q)$ of the nodes p and q (vertices pointed by p and q), or (as shown later in our approach) both $I(p)$ and $O(p)$ combined.

SimRank is calculated recursively: two score between two vertices is dependent of the pre-calculated similarity of their neighbours. The initial similarity score is calculated using the binary similarity:

$$s_0^{SR}(p,q) = \begin{cases} 1 & \text{if } p = q \\ 0 & \text{else} \end{cases} \tag{3}$$

4.3 Association Analysis

Association rule mining [9] searches for recurring relationships in a given data set. More specific, it discovers the associations and correlations between two set of items (item set).

An association rule is indicated as $\{A_1, A_2, ..., A_m\} \Rightarrow \{B_1, B_2, ..., B_n\}$, where $\forall i, j \mid i \leq m, j \leq n$, A_i and B_j are non-empty *item sets*. In order to select only the rules that are interesting for evaluation, we use the *support* and *confidence* parameters. Support *supp* for the rule $A \Rightarrow B$ indicates the fraction of transactions that contain both A and B:

$$supp(A \Rightarrow B) = P(A \cup B) \tag{4}$$

Support filters out rules that occurred by chance. Confidence $conf$ for the rule $A \Rightarrow B$ indicates the fraction of items contained in B that are also contained in A. It measures the reliability made by a rule:

$$conf(A \Rightarrow B) = P(B|A) \tag{5}$$

Rules with $supp \geq minsup$ and $conf \geq minconf$ are called *strong rules*. For generating rules that satisfy the minimal support and confidence, we use the *Apriori* algorithm [1]. This approach considers only the frequent items as basis for generating candidates and extends them to larger item sets with other frequent items.

5 Research Proposal

In this section we give an overview how the similarity models were applied to suit our needs.

5.1 Prerequisites

For a given EA project model, we make the following assumptions:

– The architecture model is complete: the response we are calculating is based on the assumption that all the necessary components and relations are there.
– The project architecture is correct: the types of the components are correct, and the relations between the components are all present and correct.
– The views in the architecture projects are clearly defined: this allows successful application of the association mining. Since there is no explicit notion for a transaction, we rely on the views - every component that is a part of the view belongs in the same transaction.

To categorize a component from the model as a newly introduced, we check if a component with the exact name and type does not exist in the repository. We ignore the description attribute since it is not a required field.

5.2 Feature Extraction and Similarity Models

The underlying representation of the given EA model and the repository is a labeled directed graph. Each node of the graph presents a component with the features: *name* (or title), *type* and *description*, whereas the relations between them are modeled as edges. The same model applies to the repository data. For two labeled directed graphs, we combine **attribute based** and **structure based** similarities.

Attribute-based similarities calculate the similarity score between two features while ignoring the graph structure. For evaluating the similarity between the names of the components, we apply the *String Edit Distance*(d^{EDIT}) [12]. The edit distance is very robust when it comes to comparing single words, but results in large distance score if the strings contain words in different order. To overcome this, we apply the distance measure on pairs of words [11]. We introduce word tokenization, remove any digit characters from the words and discard the words that do not contribute to semantic meaning, such as personal pronouns and definite or indefinite articles.

Let p and q be two components and *p_words* and *q_words* be the tokenized and processed titles of the first and second component respectively. For each word from *p_words*, we apply the String Edit distance to measure the difference with every word from *q_words*, convert it to a similarity score and record only the highest wordwise similarity that occurred for that given word. In the case of difference in the number of words, we compare each word from the input string that has more words to avoid any loss of information. To achieve normalization between 0 and 1, we divide the similarity value by the larger the number of words for the given titles. This is shown in Alg. 1.

Algorithm 1 Name-based similarity

```
 1: procedure $s^{NAME}$(p,q)
 2: p_words = tokenize(p)
 3: q_words = tokenize(q)
 4: less_words = min(p_words, q_words)
 5: more_words = max(p_words, q_words)
 6: total_similarity $\leftarrow$ 0
 7: for each m in more_words:
 8:     word_similarity $\leftarrow$ 0
 9:     for each l in less_words do:
10:         current_similarity = 1 − $d^{EDIT}$(m,l) / max(size(m),size(l))
11:         if current_similarity > word_similarity then
12:             word_similarity = current_similarity
13:     total_similarity = total_similarity + word_similarity
14: return total_similarity / size(more_words)
```

For evaluating the similarity between the types of components, we apply the binary similarity. This is a very restrictive similarity that suits the assumption that the EA models are correct and the components have the right type. For two given components p and q with their respective types t_p and t_q, the type similarity is:

$$s^{TYPE}(p,q) = \begin{cases} 1 & \text{if } t_p = t_q \\ 0 & \text{else} \end{cases} \tag{6}$$

For calculating the similarity based on the description of components, we rely on the semantic meaning of the text. The description of the text is converted to a *Term Frequency - Inverse Document Frequency* (TF-IDF) vector [13]. This metric was motivated by the length of the text: the String Edit Distance will not be able to perform well since it relies on the syntactic matching of the words. TF-IDF emphasizes the importance of a word based on how frequent it appears for the given component's description (term frequency) and how rarely it appears throughout other component's description (document frequency).

Let t be a word from the description d, N the total number of documents and D the number of documents where t appears. The TF-IDF score for t is calculated as:

$$tf\_idf(t, d, D) = word\_count(t, d) \log \frac{N}{1 + D}, \tag{7}$$

where *word_count(t,d)* returns the number how many times the word t has appeared in the document d.

After obtaining the TF-IDF vector for each word in the description $desc_p$ and $desc_q$ of the components p and q, the description similarity is calculated using the cosine distance:

$$s^{DESC}(p, q) = 1 - d^{COS}(tf\_idf(desc_p, D), tf\_idf(desc_q, D)) \tag{8}$$

where $tf\_idf(desc_p, D)$ returns a TF-IDF vector for every word from $desc_p$.

To combine all the different similarities into one, we use a weighted average function from the similarity models:

$$s^{ATTR}(p, q) = \frac{w_1 s^{NAME}(p, q) + w_2 s^{TYPE}(p, q) + w_3 s^{DESC}(p, q)}{w_1 + w_2 + w_3} \tag{9}$$

The **context-based** similarity is calculated based on the graph structure of the EA models. For this, we change the input to accept two graphs: $s^{SR}(p, q)$. For the initial cases $s_0^{SR}(p, q)$, we assign maximal similarity if the titles are a complete match and the components have the same type.

To come up with **combined** context– and attribute–based similarity, we integrate the attribute similarity in the SimRank approach. We call this model *Extended SimRank* (Eq. 10). To achieve this, we relax the initial similarity s_0^{ESR} by assigning a score calculated from the attribute similarity model, if the score is above a given threshold t (Eq. 11). We also check both the predecessors and the successors of any node v for the context-based similarity: $D(v) = I(v) \cup O(v)$.

$$s^{ESR}(p, q) = \frac{C}{|D(p)||D(q)|} \sum_{i=1}^{|D(p)|} \sum_{j=1}^{|D(q)|} s^{ESR}(D_i(p), D_j(q)) \tag{10}$$

$$s_0^{ESR}(p, q) = \begin{cases} s^{ATTR}(p, q) & \text{if } s^{ATTR}(p, q) > t \\ 0 & \text{else} \end{cases} \tag{11}$$

Inspecting every pair of vertices between graphs with n and m nodes leads into generating $n * m$ candidates. To speed up the process of evaluation, we skip the nodes which do not have any incoming and outgoing edges, since such nodes cannot contribute to the structural similarity when using s^{ESR}. For such nodes, we rely only on the s^{ATTR}.

6 Implementation

The implementation of our solution is dependent on several technologies. The **server side** is built using the Java technology. The acceptance of the EA models is realized with POST requests with the following parameters:

- file: mandatory field which contains the ArchiMate (XML) file that needs to be evaluated.
- k: a number of maximum returned components for a single query component (optional).

The ArchiMate files and the repository are read and converted to directed labeled graphs using the JGraphT library [20] and its *DirectedGraph* class. For the repository we are interested in the content of the *Architecture* section and the *Types* section. Each XML node represents either an ArchiMate component, if located under the “Components” section, or an ArchiMate relation, if located under the “Relations” section.

To get a better understanding of the structure of the repository as a graph, we use *Gephi* - a tool for analytics and detailed visualization of graphs [4]. The total size of the graph is 3922 nodes with 9657 edges. Out of those, 1147 are isolated nodes (no incoming and outgoing edges). The diameter is 7, and the average path length is 4.79. The average degree per node (both incoming and outgoing considered) is 4.93. Given graph size, we consider the repository graph as a weakly connected. This is also confirmed by the low value of 0.001 for the density.

Before calculating any similarity, we filter out any unnecessary replicated information which might affect the prediction outcome at the end. The repository is cleaned up by discarding all the replicated components, i.e. components with the same *name* and *type* (the *description* feature is not mandatory, therefore not considered a factor). This results in a reduction of 28 components.

The description of every component is converted into a TF-IDF vector. For this, a corpus needs to be built where each description of a component is considered as a document. Every word (term) gets evaluated using Eq. 7. For tokenization of the title of the components, we use the *WordTokenizer* class from the WEKA library[3] for Java. For the association mining we use the statistical programming language R and the package *arules*[4]. The chosen value for minimal support was 0.17 was the largest value where rules were still generated. Combined with the minimum confidence value of 0.75, the algorithm resulted in the generation of 77 rules.

The **client side** was realized as a plug-in for the Archi tool[5], which is an Eclipse-based IDE. The plugin provides a button on the toolbar of the IDE that allows the architect to select the desired ArchiMate model file. Afterwards, the file is uploaded to our server and the module waits for a response back. The response contains a JSON list of components that are not part of the repository and their most similar components from the repository. The result is presented as a dialog with a tabular view inside Archi.

[3] https://www.cs.waikato.ac.nz/ml/weka
[4] https://cran.r-project.org/web/packages/arules/index.html
[5] https://www.archimatetool.com

7 Evaluation

The similarity models we applied are unsupervised, meaning we do not have the true output available. To successfully evaluate them, we manually created a simulation and architecture model. The repository data consisted of 327 nodes and 275 edges and the model of 35 nodes and 23 edges. The model components were provided from two sources chosen pseudo-randomly from the repository and inserted without any relations to the repository.

A subset of 16 model components were subject to manual change of the attributes, so that different scenarios for similarity can be tested based on a title, description, type, structural similarity and combination of all. Only the nodes that did not appear in the repository were tested. There were 20 unidentified nodes in total, out of which four did not have any substitution. For every evaluation test, the k value was set to the lowest value of 1, which evaluates the shortest result list.

We also set up a simulation of an ArchiMate model for creating transactions. The simulation transaction file consisted of the same number of components and edges as the previous simulation model file, distributed in the same number of views. We replaced each unidentified component with its counterpart from the repository if such existed. Using the approach of creating transaction per view level, we created a set of five transactions with 18 items. Finally, we discarded newly introduced components without substitution as well as the components that did not belong to any view.

For evaluating the correctness of the similarity models, we compared three different metrics: accuracy, precision, and recall [19]. For the association rule mining, we focused only on the accuracy, since the results were returned in the form of "if the components on the leftside of the rule exist, then the components on the rightside of the rule might be of interest". Therefore we could not make the connection which result components belong to which query components. All metrics range from 0 (the lowest value) to 1 (highest value).

We evaluated the similarity models each one separately, as well as the combination between them. The overview is given in Fig. 4. The first evaluation was performed on each feature similarity model separately. We set the number of returned components to one (k=1). The title and type similarity showed poor performance in every metric. The description similarity model showed maximum precision value and better values for accuracy and recall. However, since the description is optional for the components, it cannot be taken as a single metric. The usage of a single feature similarity model is not recommended as they do not provide an effective recommendation service.

Next, for the evaluation of the weighted similarity combination s^{ATTR} we configured the similarity using the values 0.5, 0.1 and 0.4 for the weights w_1, w_2, and w_3 respectively. The weights were provided from a domain expert and reflected the importance of each feature. To avoid results with components with low similarity score, we introduced a threshold t = 0.5. The weighted combination performed better than any separate feature similarity model if the number of suggested components is taken into account as well, which was 17 for the given threshold.

Next, we evaluated the SimRank approach, using the combined in- and out-degree. We noticed the improvement in the score compared to the weighted combination, so we incorporated the two methods together as the *SimRank Extended* s^{ESR} with a threshold of 0.5. The *SimRank Extended* showed the higher accuracy, recall, and the highest F1 score. Increasing the number of returned components k to higher number did not affect the score in our simulation.

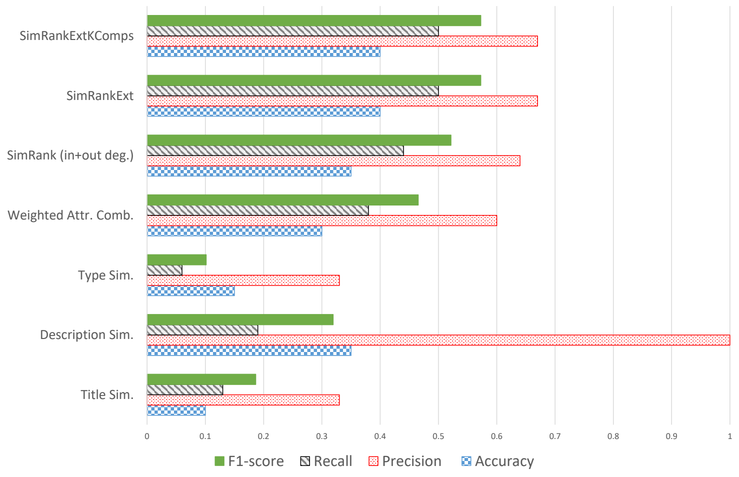


Fig. 4: Comparison of all similarity models

We performed the association mining with the following input: minimal support of 0.2 (the highest support value that still returns results), and confidence of 1 (the maximal possible value). This generated a set of 293 rules. Out of those, for the model that we evaluated, 13 components were returned. We identified a total of 7 correctly suggested components that were a suitable replacement, with an accuracy of 0.54. The rules had a low support value, which means that the component sets did not often appear in transactions. However, confidence had the highest value, thus giving a high level of certainty concerning the truth of the association rules. This method is highly dependent on the data set provided by the architects.

The threats of validity are mainly situated in our prerequisites. First, we assume that the basic model is complete. If we relax this assumption, our approach might propose components which are wrong or even no components. Nonetheless, our approach proposes only components and the final decision is taken by a human. Therefore, the consequences of relaxing this prerequisite are manageable. The same argumentation holds for our second prerequisite that the project architectures are correct. Last, we assume that views represent the changes made by the projects. However, this prerequisite is based only on the technical issue that we need to know what has been changed and, consequently, does not threat our results.

8 Related Work

Previous research [8, 16, 21, 18] was motivated by analyzing EA models as a network graph and applying different ML concepts on EA models by providing decision support for enterprise architects. The work of [8] focuses on representing the complexity of information systems architecture in social network terms and then capturing insights from the graph representation, where components of the architecture are interpreted as nodes, and the dependencies between the components as links.

Different similarity approaches, as well as metrics, have been proposed in the literature to identify the similarity and the differences between models to be matched. The work of Dijkman et al. [7] presents three similarity metrics in order to investigate matching of similar business process models in a given repository namely (i) structural similarity that compares element labels by considering topology structure of business process models; (ii) behavioural similarity that compares element labels by considering behavioural semantics of process models; and (iii) label matching similarity that compares elements based on words in the labels of business process model elements (string edit distance).

Compared to Dijkman et al. [7], our work elaborates on structural similarity (i) and label matching similarity (iii). Dijkman et. al. use Graph Edit Distance as an underlying method for (i). We find that this approach fails when comparing graphs of significantly different sizes (e.g. a model and a repository), as the returned score will always be of high value. We also expand the knowledge in (iii) as we not only consider the labels of the elements but also their attributes. To research behavioural similarity (ii) did not make sense in our case, because our model did not contain elements with behavioural semantics. Nonetheless, e.g. ArchiMate contains elements to model processes and, consequently, future research can take behavioural similarity for EA models also into account.

The work of Aier and Schönherr [3] presents a clustering approach in determining the structure of Service Oriented Architectures (SOA). The paper shows the application of clustering algorithms in supporting the design of a SOA. However, their approach does not present evaluation criteria in comparing different clustering methods. We try to apply different community detection algorithms on EA models which can group similar kinds of connected components.

9 Conclusions

Modeling an EA can result in a complex graph-like structure with many components (vertices) and relations between them (edges) in a repository. We recognized the problem that adding a model to the repository which contains components with similar attributes and behavior as some other repository components, but with different name, leads to repository pollution. To solve this, we inspected two approaches. The first approach relied on finding patterns between two enterprise architecture models. The second approach adopted a collaborative way of recommending components that might be of interest.

For evaluating the architecture models, similarity and collaborative approach were used. The result was formatted to give a list of k closest components for each component that cannot be found in the repository. The evaluation showed that the similarity models have high precision and low recall characteristics. The generated association rules have low support and high confidence value, which means that the item sets of the rules appear rarely, but we are confident that the generated rule will be true.

Our research was realized in two parts: as a a server solution able to perform the evaluations and as a plug-in for the Archi tool able to generate requests and notify the architects for the results. Although our implementation is targeted for EA, the solution we propose is generic and can be applied to any domain that can be modeled as a graph with nodes and edges.

As we rely on a graph-like presentation of the EA model, our approach can be generalized and applied also to other models which can be presented as graphs. We assume that, for instance, our approach might also work for UML models as ArchiMate and UML class diagrams are quite similar.

The current limitations are the constraints we impose before evaluating the EA models. The EA models have to be complete and correct, which means that the tool cannot be used as a recommendation system (suggesting components as the model is in the process of creation). This limitation may be solved in future by incorporating techniques from the model recommendation domain. Also the approaches are not optimized: as the size of the repository increases, recommending a list of components takes more time. Consequently, we will elaborate on this point in future. Lastly, our approach cannot be fully automatized since we rely on a human expert to confirm that the recommended components are the right substitution. We do not see any possibilities to overcome with this issue.

References

1. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules in large databases. In: Proceedings of the 20th International Conference on Very Large Data Bases. pp. 487–499. VLDB '94, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1994)
2. Ahlemann, F., Stettiner, E., Messerschmidt, M., Legner, C.: Strategic enterprise architecture management: challenges, best practices, and future developments. Springer Science & Business Media (2012)
3. Aier, S., Schoenherr, M.: Integrating an enterprise architecture using domain clustering. In: Lankhorst, M.M., Johnson, P. (eds.) Proceedings of the Second Workshop on Trends in Enterprise Architecture Research. pp. 23–30 (Juni 2007)
4. Bastian, M., Heymann, S., Jacomy, M.: Gephi: An open source software for exploring and manipulating networks. In: ICWSM (2009)

5. Champin, P.A., Solnon, C.: Measuring the similarity of labeled graphs. In: Proceedings of the 5th International Conference on Case-based Reasoning: Research and Development. pp. 80–95. ICCBR'03, Springer-Verlag, Berlin, Heidelberg (2003)
6. Deza, M.M., Deza, E.: Encyclopedia of Distances. Springer Berlin Heidelberg, Berlin, Heidelberg (2009)
7. Dijkman, R., Dumas, M., Van Dongen, B., Käärik, R., Mendling, J.: Similarity of business process models: Metrics and evaluation. Information Systems **36**(2), 498–516 (2011)
8. Dreyfus, D., Iyer, B.: Enterprise architecture: A social network perspective. In: Proceedings of the 39th Annual Hawaii International Conference on System Sciences (HICSS'06). vol. 8 (2006)
9. Han, J.: Data Mining: Concepts and Techniques. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2005)
10. Jeh, G., Widom, J.: SimRank: A Measure of Structural-context Similarity. In: Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 538–543. ACM, New York, NY, USA (2002)
11. La Rosa, M., Dumas, M., Uba, R., Dijkman, R.: Business process model merging: An approach to business process consolidation. ACM Trans. Softw. Eng. Methodol. **22**(2), 11:1–11:42 (2013)
12. Navarro, G.: A guided tour to approximate string matching. ACM Comput. Surv. **33**(1), 31–88 (2001)
13. Neto, J.L., Santos, A.D., Kaestner, C.A., Alexandre, N., Santos, D., A, C.A., Alex, K., Freitas, A.A., Parana, C.: Document clustering and text summarization (2000)
14. Peffers, K., Tuunanen, T., Rothenberger, M., Chatterjee, S.: A design science research methodology for information systems research. J. Manage. Inf. Syst. **24**(3), 45–77 (2007)
15. Rood, M.A.: Enterprise architecture: definition, content, and utility. In: Proceedings of 3rd IEEE Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises. pp. 106–111 (1994)
16. Santana, A., Souza, A., Simon, D., Fischbach, K., De Moura, H.: Network science applied to enterprise architecture analysis: Towards the foundational concepts. In: 2017 IEEE 21st International Enterprise Distributed Object Computing Conference (EDOC). pp. 10–19. IEEE (2017)
17. Santini, S., Jain, R.: Similarity measures. IEEE Trans. Pattern Anal. Mach. Intell. **21**(9), 871–883 (1999)
18. Schoonjans, A.: Social Network Analysis techniques in Enterprise Architecture Management. Ph.D. thesis, Ghent University (2016)
19. Shani, G., Gunawardana, A.: Evaluating recommendation systems. In: Recommender systems handbook, pp. 257–297. Springer (2011)
20. Sichi, J., Kinable, J., Michail, D., Naveh, B., Contributors: Jgrapht - Graph Algorithms and Data Structures in Java (Version 1.1.0). http://www.jgrapht.org (2017)
21. Simon, D., Fischbach, K.: It landscape management using network analysis. In: Enterprise Information Systems of the Future, pp. 18–34. Springer (2013)
22. Tamm, T., Seddon, P., Shanks, G., Reynolds, P.: How does enterprise architecture add value to organisations? Communications of the Association for Information Systems **28**, 141–168 (2011)
23. The Open Group: TOGAF Version 9.1. Van Haren Publishing, Zaltbommel (2011)
24. van der Raadt, B., van Vliet, H.: Designing the Enterprise Architecture Function. In: Becker, S., Plasil, F., Reussner, R. (eds.) Quality of Software Architectures. Models and Architectures, Lecture Notes in Computer Science, vol. 5281, pp. 103–118. Springer (2008)