From Gaps to Transformation Paths in Enterprise Architecture Planning

Philipp Diefenthaler[1,2(✉)] and Bernhard Bauer[2]

[1] Softplant GmbH, Munich, Germany
[2] Institute for Software & Systems Engineering, University of Augsburg, Augsburg, Germany
philipp.diefenthaler@softplant.de,
Bernhard.Bauer@informatik.uni-augsburg.de

Abstract. Planning changes in an enterprise and its supporting IT can be supported by enterprise architecture (EA) models. The planned changes result in gaps which can be derived by a gap analysis. But, knowing the gaps is not enough. Also important is to know in which sequence gaps are to be closed for transformation path planning. In this paper we show how gaps are identified and reused for detailing a model of the target architecture. Based on this refinement further gaps become visible. Furthermore, we describe how it is possible to create with a transformation model and an action repository transformation paths towards a desired and detailed target architecture. Afterwards, we give a use case example and propose a technical realization of the solution.

Keywords: Enterprise architecture planning · Gap analysis · Transformation model · Graph transformation

1 Introduction

Enterprises nowadays face challenges like changing markets, security threats, evolving technologies and new regulations that drive the need to adapt the enterprise. Enterprise architecture management (EAM) supports this change in a structured manner. An enterprise architecture (EA) is the "fundamental organization of a system [the enterprise] embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution" [1].

Models of this architecture can support decision making for planning purposes. Such EA models cover aspects from business, processes, integration, software and technology [2]. To cope with the complexity of an EA it is crucial for enterprises to use a managed approach to steer and control the redesign of the enterprise. The complexity arises from the level of abstraction, the number of stakeholders involved, and the change of internal and external conditions inherent to EAs.

S. Hammoudi et al. (Eds.): ICEIS 2013, LNBIP 190, pp. 474–489, 2014.
DOI: 10.1007/978-3-319-09492-2_28

To plan the change it is necessary to have a plan basis, i.e. the current architecture, and to know the goal of planning activities, i.e. the target architecture. According to [3,4] the planning activities take place at different decision levels. Each of them varies in detail and levels of abstraction seem to be inevitable [4]. The need to change and the resulting moving target are challenges for EA planning, as part of the EAM, has to meet [5,6]. EAM and particularly EA planning is supported by tools which allow the creation of visualizations, automated documentation and analysis of EA models.

In this paper we describe how gaps can be derived from two EA models for different points in time. Furthermore, we introduce the transformation model by Aier and Gleichauf [7] to connect architectural building blocks from the models of the current and target EA. With the results from gap analysis and the information contained in the transformation model we introduce an action repository for the creation of different transformation paths. We exemplify the solution to get from gaps to transformation paths based on a model of a current and target architecture of an application architecture within a use case for a master data consolidation challenge. Furthermore, we propose a technical realization based on semantic web technologies and graph transformations.

2 Foundations

This section gives an introduction to the foundations of EA models and their usage for planning purposes. Furthermore, we introduce semantic web technologies and graph transformations for planning purposes, as they are of relevance for our proposed technical realization of the solution.

2.1 Enterprise Architecture Models

According to Buckl and Schweda [8] EAM follows a typical management cycle that consists of the phases plan, do, check and act. The plan phase is concerned with developing change proposals that are implemented in the do phase. Within the check phase differences between intended and actually achieved results are controlled. Based upon the results from the check phase the act phase provides input to the plan phase by supplying information for the next plan phase. Models, as an abstraction mechanism, of an enterprise, can support the plan phase as part of an EAM approach [9,10].

EA models can be used to describe an EA for different points in time [8]. The model of the current architecture of the enterprise is a documented architecture at the present point in time and serves as a starting point for defining a model of a target architecture. In contrast the model of the target architecture represents a desired architecture in the future which can be used to guide the development of an EA from the current towards a target architecture. The development of a target architecture depends on the enterprises' EA goals. It is influenced by business requirements, strategic goals and IT objectives like master data consolidation, improving the flexibility of IT and drive the coverage of standard platforms [11].

Which factors and how exactly they influence the target architecture depends on the architecture method applied and how it is integrated into the enterprise‘s governance processes.

A gap analysis, sometimes also referred to as delta analysis, is the comparison between two models of an EA that is used to clarify the differences between those two architectures. Different models of architectures that can be compared are current to target, current to planned, planned to target and planned to planned [8].

2.2 Semantic Web Technologies in a Nutshell

Semantic web technologies are used to integrate heterogeneous data sets and formalize the underlying structure of the information to allow a machine to understand the semantics of it [12]. The World Wide Web Consortium (W3C) provides a set of standards to describe an ontology and to query it. An ontology "is a set of precise descriptive statements about some part of the world (usually referred to as the domain of interest or the subject matter of the ontology)" [13].

Two standards are of relevance for a proposed technical realization: firstly, the Web Ontology Language (OWL) [13] for making descriptive statements and secondly, the SPARQL Query Language for RDF (SPARQL) [14], which allows querying these statements.

The Resource Description Framework (RDF) [15] is a basis for both standards, as OWL ontologies can be serialized as RDF graphs and can be accessed via SPARQL. An RDF graph consists of triples of the form 'subject, predicate, object', whereas subjects and objects are nodes and predicates are relations. Every resource in an ontology is identified by a resource identifier which allows for example distinguishing between a bank in a financial context and a bank of a river. Information from the ontology is queried via SPARQL, which provides the resources that match patterns specified within the query.

Semantic web technologies have already been applied to domains of interest that range from semantic business process modeling [16] to diagnosis of embedded systems [17]. First implementations based upon semantic web technologies for EAM already exist from TopQuadrant with its TopBraid Composer[1] and Essential Project[2].

2.3 Graph Transformations for Planning Purposes

Several different approaches, techniques and representations to planning problems have been developed over the last decades [18,19]. These approaches range from state space model based planning to task networks, where tasks for reaching a goal are decomposed and sequenced. A state space based approach is preferable, because models of the current and target architecture are used in many EAM approaches [5,11,20,21] and are present in tools used in practice [22].

[1] www.topquadrant.com/docs/whitepapers/WP-BuildingSemanticEASolutions-withTopBraid.pdf
[2] www.enterprise-architecture.org/

Graph transformations for AI planning purposes solve a planning problem by applying graph transformations to a model until a solution for the planning problem is found. The result of such a planning process is a sequence of actions changing a model into another model.

However, graph transformations have the disadvantage that they provide a huge state space regarding the states, which have to be examined when all states in the graph are computed. As a consequence this influences the computation time of all possible worlds created through the transformations. With graph transformations a planning problem can be solved by searching for graph patterns in a state represented by a graph and applying graph transformations to change the state [23]. Graph transformations have the benefit that they have a sound theoretical foundation [24].

3 From Gaps to Transformation Paths as Sequences of Actions

The goal of the proposed approach is to deliver a more detailed model of the target architecture by making suggestions to a domain expert how a detailed target architecture could look like. Afterwards, we describe how these gaps are related to each other to generate a transformation path which allows to structure change activities, which close gaps, in sequence of actions.

3.1 Modeling Current and Target Architecture

First, a current architecture is modeled and afterwards, a target architecture is modeled, at the same level of detail. We reuse the model of the current architecture and change it to the desired target architecture. The same level of detail is necessary to ensure the comparability of the models.

The current architecture may be more detailed, but can be aggregated in a way which restores the comparability [25]. Business support maps, which relate applications to supported processes and organization units, are an example for such a model with the same level of detail [11].

Results of the Modeling. The result of this phase are the two sets:

$currentArchitecture =$ model of the current architecture of the EA
$targetArchitecture =$ model of the target architecture of the EA

In our solution the core of an EA model is a set which consists of three different types of elements. The EA model contains the architecture building blocks *(B)* of the EA, relations between architecture building blocks *(R)* and attributes of architecture building blocks *(A)*. In this sense an EA model can be defined as:

$M := \{B \cup R \cup A\}$
$B := \{x \mid x \text{ is an architecture building block}\}$
$R := \{x \mid x \in B \times B\}$ and $A := \{x \mid x \in B \times V\}$

Architecture building blocks stand for the elements of the EA, for instance a Customer Relationship Management application within the application architecture. Relations hold between these architecture building blocks, for example when an application depends on another application the respective building blocks are connected by a dependency relation. Attributes are values associated with architecture building blocks that characterize measurable and observable characteristics of the architecture building block, e.g. the release number of an application or the uptime of an service.

3.2 Performing Gap Analysis

Gap analysis is performed to compare the modeled current and target architecture. It compares the differences between *currentArchitecture* and *targetArchitecture*. In terms of a set operation this comparison corresponds to a intersection of the two compared sets. As a result three subsets are identified: *onlyCurrentArchitecture*, *onlyTargetArchitecture* and *stable*.

Results of Gap Analysis. *onlyCurrentArchitecture* is the set of building blocks, relations and attributes which only exist in the model of the current architecture.

$$onlyCurrentArchitecture := \{x \mid x \in currentArchitecture \\ \wedge\ x \notin targetArchitecture\}$$

In contrast, *onlyTargetArchitecture* is the set of building blocks, relations and attributes which only exist in the target architecture.

$$onlyTargetArchitecture := \{x \mid x \notin currentArchitecture \\ \wedge\ x \in targetArchitecture\}$$

The third set *stable* is the set of building blocks, relations and attributes which the current and target architecture have in common.

$$stable := \{x \mid x \in currentArchitecture \wedge x \in targetArchitecture\}$$

3.3 Setting the Successor Relationships for Building Blocks

The successor relationships are modelled within a transformation model [7]. The transformation model is defined as follows: $transformationModel := \{x \mid x \in currentArchitecture \times targetArchitecture\}$

With the successor relationships at hand it is possible to identify the successor type for building blocks which can be divided into *noSuccessor*, *noPredecessor*, *oneToOne*, *oneToMany*, *manyToOne*, and *manyToMany*. The inverse of the successor relation is the predecessor relation.

All building blocks in *onlyCurrentArchitecture* that do not have a successor belong to the set *noSuccessor*. All building blocks that belong to *onlyTargetArchitecture* and do not have a predecessor belong to the set *noPredecessor*. The set *oneToOne* consists of the pairs of building blocks that have exactly one successor and this successor has only one predecessor.

oneToMany is the set of building blocks that have several successors in the target architecture whereas the set *manyToOne* is the set of building blocks which have the same successor in the target architecture. *manyToMany* is the set of building blocks which have common successors, which in turn have several predecessors. By querying the models we can determine to which set a building block belongs.

A successor relationship is part of exactly one of the above subsets. Within the six different sets disjoint subsets exist. For the *noSuccessor* and *noPredecessor* set each building block represents a disjoint subset and are planned independently in contrast to the other successor sets. This is an implicit information of the transformation model, as we do not model self-directed relations for this information.

3.4 Creating Suggestions for a Detailed Target Architecture

In order to make suggestions the model of the current architecture considers applications, services and business building blocks. Business Building Blocks are in a tight relationship with the business activities of an enterprise but implementation independent. With the detailed information of the current architecture and the successor relationships at hand for applications it is possible to generate suggestions how a model of a detailed target architecture could look like.

Each application belongs to exactly one subset of the transformation model. Different suggestions are made for the subsets how to detail the target architecture. By following a suggestion the target is stepwise getting more detailed, as all sets of successor relationships are getting processed. A suggestion may be inappropriate for a domain expert she can overrule it by modeling different details. The result is a model of a detailed target architecture. At first all services are transferred to the model of a detailed target architecture. Then the dependencies can be added to the model of the detailed target architecture.

Suggestions for Provided Services.

1. *noSuccessor* set: for each provided service in the current architecture check if it is used by an application that is part of the target architecture or the consuming application has a successor relationship.
   (a) If there are any applications it is necessary to check if they still can work properly without consuming the service.
   (b) Otherwise, no information from the current architecture is added to the target architecture.
2. *noPredecessor* set: it is not possible to suggest a detail for the target architecture as there exists no detail in the current architecture. A manual addition of provided services and their business building blocks in the target architecture is necessary.

3. *oneToMany* set:
   (a) If the predecessor is part of *onlyCurrentArchitecture* all provided services of the predecessor, including their business building blocks, are suggested to be provided by one of the successor applications.
   (b) Otherwise, all provided services and business building blocks of the predecessor are suggested to be provided by one of the successor applications or the remaining part of the predecessor in the target architecture.
4. *manyToOne* set:
   (a) If the successor is part of *onlyTargetArchitecture* it is suggested to provide each service of its successors, but only one per business building block.
   (b) Otherwise, it is suggested that the successor provides the services already provided in the current architecture, i.e. by itself, and provide all services of the other predecessors, but only one per business building block.
5. *manyToMany* set: All provided services are suggested to be provided by one of the successors. If more than one predecessor provides a service with the same business building block the suggestion is to provide only one service in the target architecture with such a business building block. Further suggestions were not identified as this type represents a complex type of restructuring. Nevertheless, the domain expert should be supported with information about applications changing business support and assigned customer groups. Furthermore, information which applications belong to *onlyCurrentArchitecture* and *onlyTargetArchitecture* needs to be presented to the domain expert.
6. *oneToOne* set: all services, including their business building blocks, provided by the predecessor are suggested to be provided by the successor.
7. Furthermore, the domain expert can model additional services or let suggested services be provided by an application that is not a successor of the application that provided it in the current architecture.
8. For each service information is stored if it is the successor of one or more services in the current architecture. This is necessary to allow a sound transformation planning [9].

As a result all provided services have been modeled in the target architecture including their business building blocks. Furthermore, the information about successor relationships of the services is available.

Suggestions for Used Services.

1. *manyToMany* set: all used services of predecessors are suggested to be used by at least one successor. The domain expert can choose if more than one successor uses the service of a predecessor.
2. *oneToOne* set: all services used by the predecessor are suggested to be used by the successor.
3. *manyToOne* set: used services of the predecessors are suggested to be also used in the target architecture.

4. *oneToMany* set:
   (a) If the predecessor is part of *onlyCurrentArchitecture* all used services of the predecessor are suggested to be used by one of the successor applications.
   (b) Otherwise, all used services of the predecessor are suggested to be used by one of the successor applications or the remaining part of the predecessor in the target architecture.
5. *noPredecessor* set: which services are used by the application need to be modeled manually as no information from the model of the current architecture is available.
6. *noSuccessor* set: as the application does not exist in the model of the target architecture no information about used services needs to be added to the target architecture.
7. Furthermore, the domain expert can model additionally used services for every application.

Results of the Guided Refinement. The result is a model of a detailed target architecture including provided and used services with related business building blocks. Consistency checks can be performed on the model to check whether services exist which are provided but no longer used by any application. Gap analysis can be performed again and the detailed gaps between the models of the current and target architecture are available.

With the results of gap analysis and a detailed current architecture it is possible to assist a domain expert in modeling a detailed target architecture by making suggestions how to detail it based on the current architecture. The variety of suggestions that can be provided is limited to the information available in the EA model. For example, technical information about the services can be added to allow more sophisticated suggestions, like to prefer web service technology for services of applications that have to be build.

3.5 Creating an Action Repository

Before the transformation path from the current to the target architecture can be created, it is necessary to describe possible changes in a way which allows the sequencing of actions. This is realized with an action repository where abstract actions are modeled. An abstract action consists of two parts. One part specifies the preconditions for an action to be applicable. The other part is the effect part, which specifies the changes to an architecture model if an (abstract) action is applied to it.

The creation of the action repository is only done once as the actions are described on an abstract level. However, if the meta-model of the EA changes the actions in the action repository need to be checked if they are impacted by these changes.

In a technical sense the abstract action matches via a graph pattern into the concrete model of the different states. Concrete actions relate to concrete entities and relationships in an architecture model and concrete changes to the state of architecture models. The application of a concrete action to an architecture model, may enable the application of several other concrete actions.

Abstract actions are either atomic or composed. An atomic action changes exactly one element of either *currentArchitecture* or *targetArchitecture*. Composed actions are a composition of other actions, regardless if atomic or composed. To create a transformation path it is necessary to model at least abstract actions for shutting down and developing building blocks and abstract actions that take care of the relationships between the building blocks and the attributes of the building blocks.

Logical Order of Abstract Actions. The abstract actions are modeled in a logical order, which means that it is only possible to apply the action if the preceding actions were already applied. For example, it is not possible to change the dependencies from a service to its successor service if it has not yet been built. Furthermore, it may be necessary to build the application first to allow the creation of a new service. After the dependencies of a service have been changed to a successor it is possible to shutdown the service.

If all services of an application have been shutdown it is possible to shutdown the application. The logical order prevents the creation of loops in the transformation path, i.e. to shutdown and create the same application several times. It may be the case that it is not necessary to enact the *develop application* action. For example, if a service which has to be developed for an application that already exists. In this case it is not necessary to develop that application again since it already exists in the current architecture. The logical order prevents the shutdown of the predecessor services, until the successor service is developed.

3.6 Creating the Transformation Path

With the action repository, the transformation model, the models of current architecture and target architecture at hand it is possible to start the creation of possible transformation paths.

We derive all applicable concrete actions by checking which preconditions of abstract actions match in

$$planningKnowledgeBase := \{transformationModel \cup currentArchitecture \cup targetArchitecture\}$$

This corresponds to a breadth search of applicable actions for a possible change from the current towards the target architecture. If a concrete action is applied to *planningKnowledgeBase* it changes the state of the *planningKnowledgeBase*. In contrast if we apply a depth search we receive a transformation path changing the EA in a sequence of concrete actions from the current to the target architecture. If no such transformation path exists the more exhaustive breadth search can be omitted and we are informed that no transformation path was found. By applying the breadth search on each state recursively and we get the whole state space.

With the state space it is possible to determine all possible transformation paths from the current to the target architecture. By selecting concrete actions we create the transformation path, change the *planningKnowledgeBase* and get each time a list of concrete actions which we now can apply. When the transformation path is complete, i.e. all necessary changes have been applied, no further actions are applicable and the transformation path is saved. If gaps are not to be closed it is possible to stop the creation of the transformation path.

The selection process for choosing concrete actions can be enhanced by providing development costs for proposed applications and services, and maintenance costs for applications and services which are to be retired. Furthermore, the consideration of desired benefits, anticipated risks and resource constraints could be considered if available to allow for a weighting of favorable sequences of actions.

4 Use Case - Development Master Data Management

In the past, applications were often developed to address the specific business needs that a part of the organization had at a certain moment. However, considering the whole enterprise it is not effective to store redundant data in several applications as this increases the risk of outdated and inconsistent data. This is the basis for the master data management (MDM) challenge [26]. In our use case we show a typical (and simplified) example for the introduction of master data management in the research and development division of an organization. Figure 1 shows a part of the model of the current architecture of the organization's IT landscape. There has already been placed a development master data management (DMDM) system in the organization which provides services (MasterData_v1 and _v2) to other applications. However, not all existing applications use the master data provided by DMDM: the application DevManager provides similar data that is still used by existing applications such as the product planning tool and the quality tests planning tool. Other applications such as the virtual quality test result database store the master data themselves and are not connected to DMDM. For the modification of products (from one test to another) there exist two applications for the different product classes the organization provides to their customers. Additionally, applications to plan the product, the quality tests and store the results that have been gathered during the (physical or virtual) quality tests, exist. In the model of the target architecture the functionality in the different applications shall be united and all other tools will use the data provided by DMDM. There will be only one planning tool that includes planning for the product as well as the quality tests. All quality tests (including the results) will be managed by one quality test assistance and result management tool (cf. Figure 2).

Please note that Figs. 1 and 2 already contain the services, which may not be considered in the first place for planning purposes.

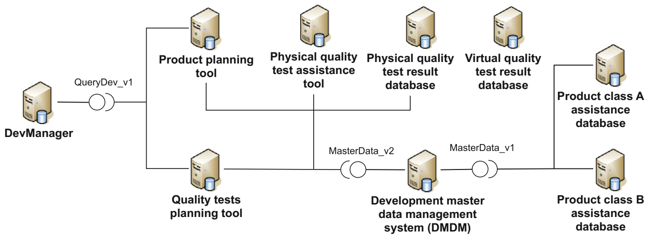
QueryDev_v1
DevManager
Product planning tool
Physical quality test assistance tool
Physical quality test result database
Virtual quality test result database
Product class A assistance database
Quality tests planning tool
MasterData_v2
Development master data management system (DMDM)
MasterData_v1
Product class B assistance database

Fig. 1. Master data management: current architecture.

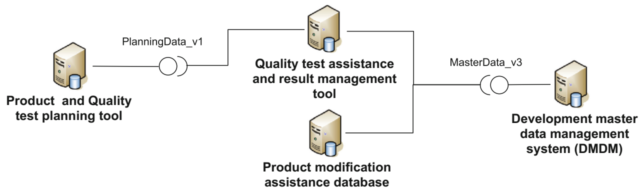
PlanningData_v1
Product and Quality test planning tool
Quality test assistance and result management tool
Product modification assistance database
MasterData_v3
Development master data management system (DMDM)

Fig. 2. Master data management: target architecture.

Solution Applied to the Use Case

At first *currentArchitecture* and *targetArchitecture* are created by modelling both architectures. Applying gap analysis it is possible to derive that *onlyCurrentArchitecture* contains: DevManager, Product Planning Tool, Quality tests planning tool, Physical quality test assistance tool, Physical quality test result database, Virtual quality test result database, Product class A assistance database, Product class B assistance database, QueryDev_v1, MasterData_v1 and _v2.

The set *stable* contains only Development master data management system (DMDM) whereas *onlyTargetArchitecture* contains Product and Quality test planning tool, Quality test assistance and result management tool, Product modification assistance database, MasterData_v3 and PlanningData_v1.

Within the transformation model information on the successor relationships is kept: Product planning tool and Quality tests planning tool have the same successor (Product and Quality test planning tool). Physical quality test assistance tool, Physical quality test result database and Virtual quality test result database have the Quality test assistance and result management tool as a common successor. DMDM is a successor of itself, which is in accordance with [7], and DevManager has no successor. Product modification assistance database is the successor of Product class A assistance database and product class B assistance database.

Regarding the services the following successor relationships are contained in the transformation model: MasterData_v3 is a successor of MasterData_v1 and _v2. The QueryDev_v1 has no successor and PlanningData_v1 has no predecessor. Based upon this information the action repository can show that it is possible to develop MasterData_v3 in the first place or one of the successor applications.

If for example as the first action the development of MasterData_v3 is selected it is possible to take care of the dependencies of applications to the predecessors of the service. After removing the dependencies and creating the new ones to the successor service it is possible to shutdown the predecessors. The development of the new applications are to be selected as the next steps in the transformation path. The remaining actions are not described in detail, however their sequence is constrained by the logical order of the abstract actions.

5 Discussion

The discussion is divided into two parts. At first we discuss the results of solution and its application to the use case. After that, the limitations of the solution are presented.

The solution describes how it is possible to derive gaps between the models of a current and target architecture for planning purposes using a set theoretic description. With the gaps at hand and information regarding the successor relationships of elements the solution reuses existing information to aid in the detailing the model of the target architecture. Afterwards, an action repository aids in the creation of possible transformation paths, which are sequences of actions. Overall, the solution considers a domain expert as an important part of the activities and assists her in the decision making process.

Creating suggestions for detailing the model of a target architecture is only possible if business building blocks are available. However, the mechanism of gap analysis, the transformation model and the creation of transformation paths using the action repository are not impacted by this limitation.

Furthermore, requirements regarding the meta-model are posed by the solution. If the EAM approach does not concern application architectures, and as a consequence the models of applications and their dependencies to services, the solution would in its current shape not be suitable. However, the mechanisms as described in the solution can be adapted to aid in the modelling and creation of transformation paths which address the concerns of the stakeholders. From our point of view, applications and their provided services are an important part of an EA.

Currently, we create the connection of the models of the current and target architecture manually, which is prone to errors and time consuming. The model of the target architecture does currently not consider information which transformation paths, taking technology architecture aspects into account, are possible.

6 Proposed Technical Realization

Using semantic web technologies for formalizing information sources yields a number of advantages, starting with having a formal, unambiguous model to the possibilities of reasoning and consistency checking. The knowledge base containing, the current and target EA models, as well as the transformation model, can be consulted at run time by humans as well as by applications.

Identifying gaps can be realized using standard tools like Protégé[3] for modeling and OWLDiff[4] for comparing the modeled EAs. For detailing the model of the target architecture we suggest the usage of SPARQL as it allows querying and adding information in a semi-automated manner.

Regarding the creation of transformation paths we suggest to use a more sophisticated graph transformation approach, as it provides the expressiveness necessary for the creation of transformation paths. This requirement exceeds the current capabilities of SPARQL. A promising World Wide Web Consortium standard is the Rule Interchange Format[5] (RIF), which initial purpose was the exchange of rules. The second edition of RIF provides an action language which can be used to express the actions necessary for transformation path planning. However, we were not able to test the proposed solution as no free implementations are available yet. Therefore, we propose to use a mature graph transformation tool like GROOVE[6].

However, this proposed technical realization requires a model to model (M2M) transformation of the ontologies to a model which is interpretable for a graph transformation approach.

7 Related Work

In this section related work is introduced. As a starting point the technical report 'On the state of the Art in Enterprise Architecture Management Literature' [8] was taken, as they consider gap (delta) analysis as part of the different EAM approaches. Besides the listed approaches in the technical report an approach from the University of Oldenburg and a technical standard from The Open Group was identified as relate work.

7.1 University of Oldenburg

The Institute for Information Technology of the University of Oldenburg presents a tool supported approach for performing a gap analysis on a current and ideal landscape [27]. The approach is tightly coupled to the Quasar Enterprise approach, which can be used to develop service-oriented application landscapes.

In order to be able to perform their gap analysis it is necessary to model the current application landscape consisting of current components, current services, current operations and business objects. The ideal landscape is modeled with ideal components, ideal services, ideal operations and domains. Based on these two models the tool is capable to generate a list of actions that would, if all were applied, result in the ideal landscape. Within the paper the suggested procedure

[3] http://protege.stanford.edu/
[4] http://krizik.felk.cvut.cz/km/owldiff/
[5] http://www.w3.org/TR/2013/NOTE-rif-overview-20130205/
[6] http://groove.cs.utwente.nl/

for selecting actions is to allow an architect to select certain actions that result in a target. Furthermore, the tool is capable to provide metrics for quantitative analysis of the application landscape.

Gringel and Postina state that gap analysis needs a "detailed level of description when it comes to modeling both landscapes" ([27], p. 283) and as a result the "data necessary to perform gap analysis on the entire application landscape on a detailed level considering operations is overwhelming" ([27], p. 291). How the different actions interfere with each other is not considered and actions can only be provided if an ideal landscape with all details has been modeled.

7.2 Strategic IT Managment by Hanschke

The 'Strategic IT Management' [11] approach is intended to serve as a toolkit for EAM by providing best-practices derived from work experience. After a target architecture has been modeled and agreed upon gap analysis is used to detect differences between the current and target architecture. Gap analysis is performed on the basis of process support maps visualizing which applications support which business processes (x-axis) and which customer group (y-axis) the applications are assigned to. For a more fine grained gap analysis Hanschke suggests to additionally add information about services and information objects of the applications. Afterwards, for each gap possible actions to close the gap are considered.

The actions range from introducing a new application, adding or reducing functionality of an existing application, changing or adding services to the shut down of applications and services. Based upon the results of gap analysis and derivation of appropriate actions it is necessary to clarify dependencies between the actions, bundle the actions and create planned architectures as recommendations for change. As far as we were able to verify the limitations of the tool and approach it is not possible to create suggestions for a detailed target architecture.

7.3 ArchiMate

ArchiMate ([21], Chap. 11) introduces an Implementation and Migration Extension including a Gap element. A gap can be associated with any core element of the ArchiMate meta-models, except for the Value and Meaning element. In general, a gap links several elements of two EA models and contains elements to be removed (retired) and to be added (developed). The linkage of the differences between the EA models and the resulting gap is not described.

8 Future Work

Creating transition architectures as plateaus (see [21]) between the current and target architecture should be supported by actions. A plateau is a stable state of the EA. The current and target architecture are also plateaus according to ArchiMate. However, we need to identify at first in which situations actions are of relevance for transition architecture creation and if it is possible to provide meaningful support for a domain expert.

A value based weighting for different transformation paths is currently elaborated to support a domain expert with information which paths seem to be more promising than others. This ranking will take into account different factors relevant for transformation planning.

The methodology how to create, use and maintain the action repository is currently extended to cope with different EA models and different concerns which need to be addressed during transformation planning.

9 Conclusions

We have shown how it is possible to get from identified gaps to transformation paths by creating a transformation model, detailing a target architecture and using an action repository to create possible sequences of actions for transformation paths.

An use case for parts of an application architecture was presented and the solution was applied to it. Furthermore, we presented a proposition for a technical realisation to allow for tool support.

We discussed the results and limitations of the solution and clarified its connection to related work. Future work to be addressed was also presented.

References

1. International Organization for Standardization. ISO/IEC 42010:2007 Standard for systems and software engineering - recommended practice for architectural description of software-intensive systems (2007)
2. Winter, R., Fischer, R.: Essential layers, artifacts, and dependencies of enterprise architecture. In: 2006 10th IEEE International Enterprise Distributed Object Computing Conference Workshops (EDOCW'06), IEEE, p. 30 (2006)
3. Pulkkinen, M., Hirvonen, A.: Ea planning, development and management process for agile enterprise development. In: Proceedings of the 38th Annual Hawaii International Conference on System Sciences, IEEE, p. 223c (2005)
4. Pulkkinen, M.: Systemic management of architectural decisions in enterprise architecture planning, four dimensions and three abstraction levels. In: Proceedings of the 39th Annual Hawaii International Conference on System Sciences (HICSS'06), IEEE, p. 179a (2006)
5. Niemann, K.D.: From Enterprise Architecture to IT Governance: Elements of Effective IT Management. Vieweg, Wiesbaden (2006)
6. Aier, S., Gleichauf, B., Saat, J., Winter, R.: Complexity levels of representing dynamics in EA planning. In: Albani, A., Barjis, J., Dietz, J.L.G. (eds.) Advances in Enterprise Engineering III. LNBIP, vol. 34, pp. 55–69. Springer, Heidelberg (2009)
7. Aier, S., Gleichauf, B.: Towards a systematic approach for capturing dynamic transformation in enterprise models. In: Sprague, R.H. (ed.) Proceedings of the 43rd Hawaii International Conference on System Sciences 2010 (HICSS-43). Los Alamitos, IEEE Computer Society (2010)

8. Buckl, S., Schweda, C.M.: On the State-of-the-Art in Enterprise Architecture Management Literature (2011)
9. Aier, S., Gleichauf, B.: Application of enterprise models for engineering enterprise transformation. Enterp. Model. Inf. Syst. Archit. **5**, 56–72 (2010)
10. Buckl, S., Ernst, A.M., Matthes, F., Schweda, C.M.: An information model capturing the managed evolution of application landscapes. J. Enterp. Archit. **5**, 12–26 (2009)
11. Hanschke, I.: Strategisches Management der IT-Landschaft: Ein praktischer Leitfaden für das Enterprise Architecture Management, 1st edn. Hanser, München (2009)
12. Shadbolt, N., Hall, W., Berners-Lee, T.: The semantic web revisited. IEEE Intell. Syst. **21**(3), 96–101 (2006). (IEEE Computer Society)
13. Motik, B., Patel-Schneider, P.F., Horrocks, I.: Owl 2 web ontology language: Structural specification and functional-style syntax (2009)
14. Prud'hommeaux, E., Seaborne, A.: SPARQL Query Language for RDF. World Wide Web Consortium (2008)
15. Manola, F., Miller, E., McBride, B.: RDF Primer. World Wide Web Consortium (2004)
16. Lautenbacher, F.: Semantic Business Process Modeling: Principles, Design Support and Realization. Shaker, Aachen (2010)
17. Grimm, S., Watzke, M., Hubauer, T., Cescolini, F.: Embedded $\mathcal{EL}^+$ reasoning on programmable logic controllers. In: Cudré-Mauroux, P., Heflin, J., Sirin, E., Tudorache, T., Euzenat, J., Hauswirth, M., Parreira, J.X., Hendler, J., Schreiber, G., Bernstein, A., Blomqvist, E. (eds.) ISWC 2012, Part II. LNCS, vol. 7650, pp. 66–81. Springer, Heidelberg (2012)
18. Russell, S.J., Norvig, P.: Artificial Intelligence: A Modern Approach, 3rd edn. Prentice Hall, Upper Saddle River (2010)
19. Ghallab, M., Nau, D.S., Traverso, P.: Automated Planning: Theory & Practice. Morgan Kaufmann/Elsevier Science, San Francisco/Oxford (2004)
20. The Open Group: TOGAF Version 9.1. TOGAF Series, 1st edn. Van Haren Publishing, Zaltbommel (2011)
21. The Open Group: Archimate 2.0 Specification. Van Haren Publishing, Zaltbommel (2012)
22. Matthes, F., Buckl, S., Leitel, J., Schweda, C.M.: Enterprise Architecture Management Tool Survey 2008. Technische Universität München, München (2008)
23. Edelkamp, S., Rensink, A.: Graph transformation and AI Planning. In: Edelkamp, S., Frank, J. (eds.) Knowledge Engineering Competition (ICKEPS), Rhode Island, USA (2007)
24. Rozenberg, G.: Handbook of Graph Grammars and Computing by Graph Transformation, vol. 1. World Scientific River Edge, NJ, USA (1997)
25. Binz, T., Leymann, F., Nowak, A., Schumm, D.: Improving the manageability of enterprise topologies through segmentation, graph transformation, and analysis strategies. In: 2012 16th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2012), pp. 61–70 (2012)
26. Loshin, D.: Master Data Management. Elsevier/Morgan Kaufmann, Amsterdam/Boston (2009)
27. Gringel, P., Postina, M.: I-pattern for gap analysis. In: Engels, G., Luckey, M., Pretschner, A., Reussner, R. (eds.) Software Engineering 2010. Lecture Notes in Informatics, pp. 281–292. Gesellschaft für Informatik, Bonn (2010)