HAL open science

Planning Support for Enterprise Changes

Florian Lautenbacher, Philipp Diefenthaler, Melanie Langermeier, Mariana Mykhashchuk, Bernhard Bauer

To cite this version:

Florian Lautenbacher, Philipp Diefenthaler, Melanie Langermeier, Mariana Mykhashchuk, Bernhard Bauer. Planning Support for Enterprise Changes. 6th The Practice of Entrprise Modeling (PoEM), Nov 2013, Riga, Latvia. pp.54-68, 10.1007/978-3-642-41641-5_5 . hal-01474755

HAL Id: hal-01474755

https://inria.hal.science/hal-01474755v1

Submitted on 23 Feb 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License

Planning Support for Enterprise Changes

Florian Lautenbacher[1], Philipp Diefenthaler[1,2], Melanie Langermeier[2], Mariana Mykhashchuk[1] and Bernhard Bauer[2]

[1] Softplant GmbH, Munich, Germany
firstname.lastname@softplant.de
[2] University of Augsburg, Germany
firstname.lastname@informatik.uni-augsburg.de

Abstract. Enterprises have to react to changes with an increasing speed in order to stay competitive. Many approaches support the modeling of enterprise architectures but lack an evolution of enterprise architectures through demonstrating a transformation path from one architecture state to another. Enterprises know their strategic goals and are able to model them, but are not supported towards achieving these goals in terms of developing their architecture. We want to improve the current manual creation of the transformation paths in enterprise architecture planning by providing possible and sound sequences of actions as part of a roadmap from the current to a desired target architecture.
Therefore, we present a solution that supports the enterprise architect with proposals for a transformation path from the current to the target state considering dependencies to be taken into account during the enterprise transformation.

Key words: enterprise architecture management, transformation planning, transformation modeling, application architecture

1 Introduction

Changing laws and regulations, a growing number of competitors, upcoming customer channels and, hence, an adapted business strategy require that enterprises need to react flexibly. Nowadays, an IT landscape that efficiently supports the business and can easily be adapted is a key element for the success of enterprises.

Enterprise architecture management (EAM) assists to develop the IT landscape so that it efficiently supports current and future business needs. Several EAM frameworks were standardized (e.g. The Open Group Architecture Framework (TOGAF) [1] or the Zachman framework [2]) and are utilized and adapted in several companies.

Enterprise architecture (EA) models describe the enterprises in an abstract way and allow for a goal-oriented information systems development. They serve as information basis for discussing and deciding about transformations of the overall enterprise [3]. Models of the EA provide a holistic view on an enterprise by aggregating information about the business strategy, business processes, the applications and interfaces, the data exchanged between the applications as well

as the underlying infrastructure [4]. The relation of those elements provides enterprises with a clear picture of the current business support by IT systems.

Driven by the business strategy a desired target state can be modeled for the business, the application as well as the technological perspective. In order to close the gaps between the current and the target state, transformations from the current to the target architecture are planned by enterprise architects.

Many companies face an IT landscape which has grown over years and which comprises hundreds of systems using various technologies with complex dependencies between them. Considering all these dependencies when planning the enterprise transformations is a difficult task. Especially when focusing on the business, these dependencies are often neglected. These inherent dependencies are often revealed quite late, when implementing the transformation projects and then lead to changed project plans, higher costs and delayed deadlines.

The process of building a roadmap from a current state to a target state of an enterprise considering the EA dynamics is evaluated in the research area EA planning. Spewak [5] defines EA planning as "the process of defining architectures for the use of information in support of the business and the plan for implementing those architectures". The main goal of EA planning is to enhance and maintain the mutual alignment of business and IT [6]. The different approaches in this research area are heterogeneous and cover different topics [7]: from methodologies for EA planning [8, 9] to modeling the transformation [10, 11, 12], describing possible actions for transformations [13] or focusing on key performance indicators to compare the current and the target architecture [14].

The contribution of this paper focuses on the creation of a transformation path from a current to a target architecture (using planning techniques from artificial intelligence, AI) and describes a solution how the enterprise architect is supported by proposed sequences of transformations, which can be performed in the enterprises' IT landscape. These sequences consider the dependencies between existing and planned IT applications (i.e. one provides an IT service that is used by others). Our solution proposes four phases to compute the proposed sequences and supports the enterprise architect during the creation of the transformation path.

The presented solution assumes that the current IT landscape as well as the target IT landscape have already been modeled and are available for the planner. Hence, the solution focuses on ***how to achieve*** the target rather than on ***what*** the target should look like. This allows the enterprise architect to stimulate alternative thinking of how the target can be reached. In this paper we describe the planning process with focus on the application architecture only, i.e. the IT applications and IT services used to exchange data. Other architecture layers (such as business or technology architecture, compare [15, 16]) are outside the focus of this paper.

2 Foundations

One of the first contributors in the field of EA planning are Spewak and Hill [5] that introduce an EA planning model (wedding cake model), which describes how the blueprints for the target state are developed from the analysis of the current state, and how the changes are structured in an implementation and migration plan. More recent research on EA planning addresses the different levels which have to be considered in the planning process and how decisions in different architectures, i.e. business, application, data and technology, may affect the others [9]. Aier et al. [7] derive an EA planning process from the work of Spewak and Hill [5], Niemann [17] and Pulkkinen [9]. The derived process consists of the steps: (1) define vision, (2) model current architecture, (3) model alternative target architectures, (4) analyze and evaluate target alternatives, (5) plan transformation from current to target and, before a new planning cycle is initiated, (6) implement transformation.

The current architecture describes the status quo and the target architecture describes a desired state in the medium-dated future (approximately 3 to 5 years). A vision or ideal architecture is a blueprint of a desired architecture which will possibly never be reached, but serves as guidance for defining a target architecture.

2.1 Changes in Application Landscapes

In the context of the transformations of application landscapes the entities that are changed consist at least of the applications and the services they provide and use (c.f. [18] and [19]). Several sources in literature consider different types of changes in application landscape transformation. All of these sources distinguish between changes that *create* and *delete* entities ([20], p.95; [21], p. 172; [22]; [23], p. 11; [24], p. 59). Furthermore, an *update* of an entity is considered as a change in several sources ([22]; [23], p. 11; [24], p. 59). Sousa et al. [22] take additionally a *read* dependency for changes into account. This type of dependency is used to denote that an entity which creates and deletes certain entities needs another entity which it does not actively change by itself.

According to Aier and Gleichauf [25] the models of the current and target state can be linked through a transformation model which contains information about the successor relationships of entities. A successor relationship always links exactly one element in one state with an element in another state. It is possible that one element has zero or more than one outgoing or incoming successor relationship.

2.2 Graph Transformations for Planning Purposes

Several different approaches, techniques and representations to planning problems have been developed over the last decades in the research field of artificial intelligence planning and scheduling. 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 EA approaches and are present in many tools used in practice.

Graph transformations for AI planning purposes solve a planning problem by applying graph transformations on a model until a solution for the planning problem is found. The result of such a planning process can be 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, and as a consequence influence the computation time. With graph transformations a planning problem can be solved by searching for graph patterns in the state represented by a graph and applying graph transformations to the state [26]. Graph transformations have the benefit that they have a sound theoretical foundation.

By reusing the knowledge from existing contributions in the field of enterprise architecture planning and the existing techniques from AI planning we can create a solution which supports the enterprise architect in gaining an overview on alternative transformation paths.

3 Solution for Transformation Planning

Our solution comprises the steps analyze models and plan transformation from current to target architecture of EA planning as defined by Aier et al. [7].

The planning of the transformation takes place in four steps: (1) the connection of the architectures, (2) the segmentation analysis, (3) the creation of an action repository and (4) the creation of the transformation path. As prerequisite the current and target architecture have to be determined. The current architecture consists of the applications and their used and implemented services at present. In contrast, the target architecture contains all applications and their used and implemented services at future. As a result of the process a partial plan will be created, which describes how to reach the target. Figure 1 shows an overview of the concept which was modeled and tested.

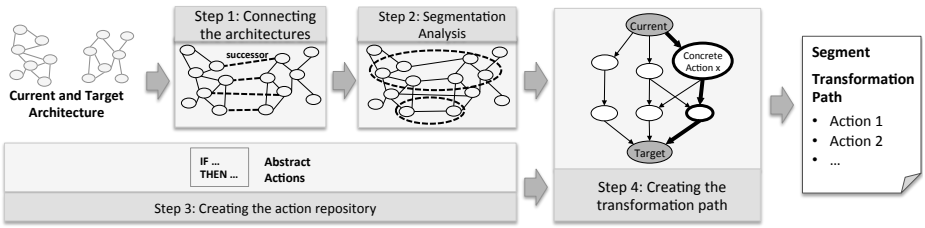

Fig. 1. The proposed solution with its four phases.

The first step of our solution supports the linking of the current and target architecture and thereby the definition of the transformation model. With the segmentation analysis in the second step, we narrow the scope for the following transformation path creation. Independent of those steps is the creation of the action repository, where the abstract actions that are taken into account are modeled. Creating these abstract actions is done only once and can be applied on all variations of current and target architectures that satisfy the metamodel criteria specified in an abstract action. Metamodel entities must exist for the elements of the architecture models. Such metamodel entities represent the type of the element in order to apply the changes to a model. Furthermore, the metamodel restricts the relationships and attributes of the elements.

After finishing these three phases the creation of the transformation path can be started. Thereby, different possible plans, consisting of sequences of concrete actions, are generated. Alternatively the enterprise architect can create those paths interactively through selection of the next concrete action. Based on the preferred sequence of action a project proposal can be determined. In the following sections each of those phases will be described in more detail.

3.1 Connecting the architecture

The first step of the proposed solution supports the enterprise architect in connecting the current architecture with the target architecture. To enable the creation of a transformation path the target architecture has to be semantically connected with the current architecture in the sense of successor relationships. Similar to Aier and Gleichauf [25] we use successor relationships to link current elements to the appropriate target elements. A special unchanged successor relationship indicates those links, where the element of the current architecture exists unchanged in the target architecture. To support the enterprise architect in finding the successor relationships, we analyze the similarity of the elements in the current and target architecture. The automatic derivation of successor relationships for applications based on business support maps and suggestions for successor services is described by Diefenthaler and Bauer [27]. However, this approach is limited to architectures, where applications are localized in the cells of a business support map.

To support the enterprise architect in finding the successor relationships independently from a localization, we analyze the similarity of the elements in current and target architecture.

The similarity of two elements c, t is defined as the total number of such shared relationships in relation to the overall number of relationships: $Sim(c)_t = \#sharedRelationships(c, t) / \#Relationships(c)$. A shared relationship between an element c in current and an element t in target exists, if there is a relationship (c, r, c′) in the current architecture and a relationship (t, r, t′) in the target architecture with *c'* as an element of the current architecture, which has the successor *t'* in the target architecture; r is an arbitrary architectural relationship. The similarity measure has a range from 0 to 1, whereas 0 means that there are no shared relationships and 1 means that all relationships are shared.

Before determining the shared relationships, unchanged elements have to be identified. The enterprise architect has to confirm each mapping, because name-equality does not always conclude an unchanged successor relationship. After that, there are already some successor relationships given. Based on those, the shared relationships and similarity measures can be calculated iteratively until no more suggestions can be found.

For the final decision about the successor relationships the enterprise architect should not only rely on this measure but also take the total number of relationships and the expert knowledge into account. Moreover, he has to specify, whether the relationship is an unchanged successor or not. Further measures can be determined depending on the context and enterprise specific details.

3.2 Segmentation Analysis

The segmentation analysis addresses the problem that enterprise architectures typically have a huge scope and include a lot of elements. To narrow the scope, a segmentation analysis can be done before determining the transformation path. According to Aier and Gleichauf [25] architectural elements can be grouped to segments, if they do not have relationships to other elements outside the group. As there are typically no isolated cells in enterprise architectures we do this segmentation by using different successor relationships. They enable the identification of parts in the architecture where changes occur but also where no changes occur.

A segment is defined as a set of interrelated elements in the current architecture with their successor elements in the target architecture including all implement- and use-relationships between those elements. To ensure the independence of the different segments in an enterprise architecture, each group of changed elements must be surrounded by a set of unchanged elements. Then these groups of changed elements are independent of each other in the context of planning, although the segments overlap at unchanged elements. An example for a segmentation is shown in Figure 2.

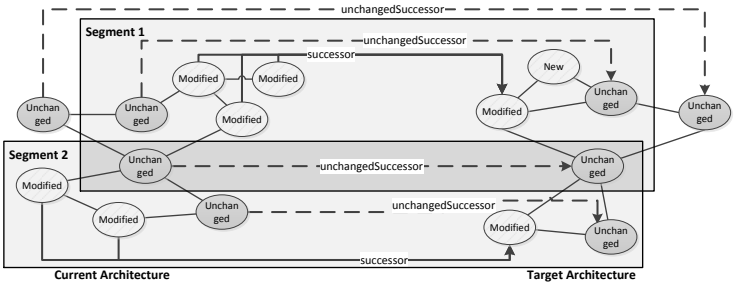

Fig. 2. Segmentation of the connected current and target architecture for narrowing the scope.

The figure shows on the left side a current architecture and on the right side a target architecture. Both consist of interrelated changed and unchanged elements. Furthermore, the successor-relationships between the current and target architecture are modeled. Using the segmentation analysis two segments can be identified in this architecture. They overlap at one unchanged element. There is also one unchanged element which is neither in segment 1 nor in segment 2.

3.3 Creating an Action Repository

Before the transformation from the current to the target architecture can be planned, an action repository with abstract actions has to be 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 if an (abstract) action is applied to it. 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 and concrete changes to the state of architecture. The application of a concrete action to an architecture, may enable the application of several other concrete actions.

Logical Ordering 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 ordering 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 segment contains a service, which has to be developed for an application that already exists, it is not necessary to develop that application again since it already exists in the current architecture. However, in this case the logical ordering would prevent the shutdown of the predecessor services, if present, until the new service is developed.

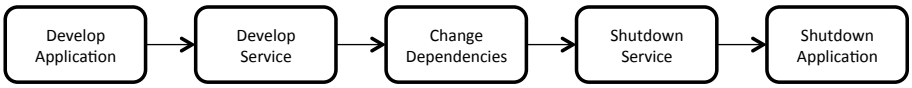

Fig. 3. The logical ordering of abstract actions

3.4 Creating the Transformation Path

With the action repository and a segment at hand it is possible to start the creation of a transformation path. It is also possible to start the creation without segments, but we advise to utilize the segmentation, because the number of possible transformation paths for an whole IT landscape is hard to grasp for a human.

We derive all applicable concrete actions for a segment by checking which preconditions of abstract actions match in a segment. This corresponds to a breadth search of applicable actions for a segment. If a concrete action is applied to a segment it changes the state of the segment. In contrast if we apply a depth search on a segment we receive a transformation path changing the segment 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. We apply the breadth search on a segment recursively and we get the whole state space.

With the state space it is possible to determine all possible transformation paths of a segment. By selecting concrete actions we create the transformation path, change the segment 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.

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.

4 Use case and implementation details

In the past, IT applications were often developed to address the specific business needs that a part of the organization had at that moment. However, considering the whole enterprise it is not effective to store redundant data in several IT applications as this increases the risk of outdated and inconsistent data. This is the basis for the master data management (MDM) challenge [28]. In our use case we show a typical (and simplified) example for the introduction of master data management in the research department (R & D) of an organization. Figure 4 shows a part 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 interfaces (*MasterData_v1* and *_v2*) to other IT applications. However, not all existing systems use the master data provided by **DMDM**: the long lasting **DevManager** provides similar data that is still used by existing systems such as the product planning tool and the quality tests planning tool. Other IT 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

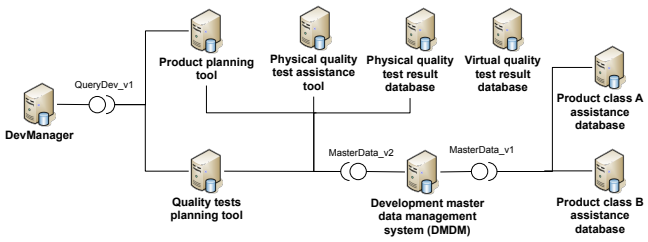

Fig. 4. Master data management: current architecture

exist two IT applications for the different product classes the organization provides to their customers. Additionally, IT 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 target architecture the functionality

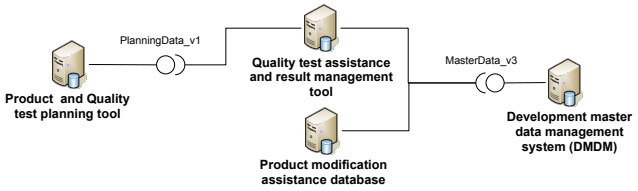

Fig. 5. Master data management: target architecture

in the different IT application 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 5).

4.1 Building the transformation path for the use case

In this section we describe the transformation path created by an enterprise architect. Action prioritization is based on the principle of subject-specific importance of corresponding applications and services, the principle of redundant applications avoidance as well as the principle of using new interfaces wherever possible.

Initially, the enterprise architect has done the mapping between applications in the current and the target architecture. This mapping showed which applications and services had successor relationships. Grouping applications from current architecture by their successors in the target, the enterprise architect was able to identify three groups of applications, which had to be consolidated. One group consisted of the **Product planning tool** and **Quality tests planning tool**, both of which had the successor **Product and Quality test planning**

tool. **Physical quality test assistance tool**, **Physical quality test result database** and **Virtual quality test result database** have built the second group. The third group comprised of **Product class A assistance database** and **Product class B assistance database**.

After getting an overview on forthcoming changes, the enterprise architect had to decide in which order he would perform concrete actions. Due to the high operation costs for **DevManager**, the enterprise architect had decided to start with the shutdown of this application. However, it was not possible to perform this action immediately since **DevManager** provided the service *QueryDev_v1* to **Product planning tool** and **Quality tests planning tool** and was still in use. It was also impossible to stop using the service by these systems before their successor-application was developed. Taking this into account, the enterprise architect decided to develop the **Product and quality test planning tool** together with its service *PlanningData_v1* at first. Next steps comprised the removal of connections between service *QueryDev_v1* and applications **Product planning tool** and **Quality tests planning tool**. Not until then *QueryDev_v1* and **DevManager** could be shut down. After that, once connections of **Product planning tool** and **Quality tests planning tool** to another service *MasterData_v2* have been removed, the enterprise architect could shut down these applications. In this way, the enterprise architect created the transformation path for the first group of applications.

For the remaining applications the enterprise architect decided to proceed with the second group, whilst taking into account that isolated solutions were not desirable in the organization and quality test result management had a high strategic importance for the R & D department. In the end, the transformation path for applications of the third group was created.

An excerpt of the list of actions can be seen below.

1. Develop application **Product and Quality test planning tool**
2. Develop service *PlanningData_v1* of application **Product and Quality test planning tool**
3. Remove connection between application **Product planning tool** and service *QueryDev_v1*
4. Remove connection between application **Quality tests planning tool** and service *QueryDev_v1*
5. Shut down service *QueryDev_v1*
6. Shut down application **DevManager**
7. Remove connection between application **Product planning tool** and service *MasterData_v2*
8. Remove connection between application **Quality tests planning tool** and service *MasterData_v2*
9. Shut down application **Product planning tool**
10. Shut down application **Quality tests planning tool**

4.2 Implementation details

For the implementation of our solution we use model query languages and GROOVE [1]. Models of the segment are transformed via a model-to-model transformation into GROOVE models. The current version of GROOVE allows to import and export Ecore [2] conform models, which can for example be UML models.

The current and target architecture have to be modeled using a formally defined metamodel. The connection of the architecture as well as the segmentation analysis, are done by using a model query language. We modeled the abstract actions, depicted in Figure 3, with GROOVE. Abstract actions change the lifecycle phases of applications and services or change the usage dependencies between those. Besides theses abstract actions, also abstract actions for debugging purposes were modeled to allow the detection of states which are incorrect from an expert viewpoint. For example it should not be possible that an application consumes a service it also provides.

A segment serves as a starting state for GROOVE. It consists of the applications and services from the current and target architecture, the successor relationships between them, the implementation relationships between applications and services of the current and target architecture and, moreover, the usage dependencies of the current architecture. Upon this state concrete actions can be applied in GROOVE via graph pattern matching and graph transformation.

To be able to have a criterion when the transformation path is ready we define an action, which has no effect on the state. This action consists of the usage dependencies of the target architecture, all applications and services which have to be *shutdown* are in the corresponding lifecycle, and all applications and services which have to be built are in the lifecycle phase *live*. With this action and the initial state at hand it is possible for GROOVE to compute a transformation path. Furthermore, it is possible to create a transformation path in interaction between an enterprise architect and GROOVE, by selecting concrete actions and computing the resulting states.

5 Evaluation

The evaluation for the use case was conducted by an enterprise architect and a knowledge engineer. The knowledge engineer is the creator of the action repository, who models the actions according to certain requirements. The enterprise architect is the creator of the current and target architecture and has functional knowledge about the concrete changes.

Conducting the Evaluation The goal of the evaluation was to determine the differences between the expectations of the enterprise architect and actual

[1] GRaphs for Object-Oriented VErification (GROOVE) http://groove.cs.utwente.nl/
[2] http://www.eclipse.org/modeling/emf/?project=emf

information provided by GROOVE. Therefore we first checked if the desired transformation path from the enterprise architect could be reproduced using the interactive alternative of our solution. Second, we verified the functional correctness of the transformation path, if it was created by GROOVE without interaction.

For the first case the knowledge engineer explained the procedure and the tool setting to the enterprise architect. Then the enterprise architect was asked to explain the concrete changes of her transformation path. The knowledge engineer performed each action in GROOVE until the transformation path was finished.

In the second case the transformation path, created by GROOVE, was extracted to a text file and relevant parts for the enterprise architect were kept in it. This means that it contained actions like Create Application: **Product and Quality test planning tool** and their ordering in the transformation path. The enterprise architect used prints of the current and target architecture with their successor relationships between the applications and services to keep track of the actions applied by the proposed transformation path.

Results of the Evaluation It was possible to reproduce the transformation path of the enterprise architect in GROOVE. The knowledge engineer had to execute additional actions, because not every step of the path was exactly one step in the tool. For example, the change of a dependency is considered by the enterprise architect as one step, but in GROOVE this change was modeled as two steps (one delete and one create). Furthermore, the corresponding action for shutting down applications is modeled in a way that it shuts down all possible applications in one step. The enterprise architect considered the shutdown of the applications **Product planning tool** and **Quality tests planning tool** as two steps.

The enterprise architect confirmed the transformation path created from GROOVE as valid. However, other transformation paths were considered as more optimal from the viewpoint of the enterprise architect. Furthermore, it was necessary to explain to the enterprise architect that not every step in her path was exactly one step in GROOVE‘s path.

One insight of the evaluation was that it was possible to give a hint to the enterprise architect that one service was no longer in use and can be shutdown. Additional actions will be modeled that were considered as useful during the evaluation. Moreover, the enterprise architect asked for the possibility to specify priorities for the development and shutdown of applications. For example, it should be possible to prefer applications with a high strategic importance for development and to prefer applications to shutdown with high maintenance cost. Additionally, the possibility to take resource constraints, like available budget and staff, into account was uttered.

6 Related Work

We summarize three publications related to our solution in the following:

Postina [24] presents a method to manage service and process oriented enterprise architectures. He uses a case based reasoning approach and a case repository to provide information for the evolution of the enterprise architecture. The target state can be reached from the current state through several evolutionary steps. A case is defined by the type of an evolutionary step (create, update, delete), the element and type involved, e.g. organizational unit billing, and the viewpoints attached to it. The cases help to provide views for future evolutionary steps with the same combination of element type and evolutionary step type. However, the creation of different transformation paths is not considered, which a benefit of our approach is.

Postina and Gringel [29] present a prototypical tool for creating a target architecture by selecting gaps between an ideal and current application landscape. The target landscape is interactively designed by an enterprise architect and the tool. The term ideal landscape is tightly coupled to the Quasar Enterprise approach which considers domains, ideal interfaces, ideal components and ideal operations for interfaces. Based on the structural differences between the current and ideal landscape the tool can identify the gaps and provides an action list to close the gaps. The modeling of an ideal landscape is a prerequisite in order to create the target architecture and the resulting action list, which consists of actions to close the gaps. In contrast, our approach provides actions for sequencing changes within the transformation path and needs no ideal landscape, which makes it less approach-dependent. Furthermore, our actions are defined on an abstract level and thus allow determining the different dependencies between the necessary changes.

Sousa et al. [22] describe an approach to reconstruct enterprise architecture models from existing project information and the artifacts influenced by them. Furthermore, the Blueprint Management System (BMS) is introduced which allows to detect the temporal dependencies of projects on certain artifacts and can generate different viewpoints. Time is explicitly taken into account by providing a timeline bar in the BMS, which enables the user to browse through the different points in time. The tool is capable of providing different states as each project has a list of artifacts it creates and deletes. This implies that projects are already on the run and the information is derived afterwards. However, our solution can interactively create the transformation path including alternative paths and then allows the creation of proposals for projects, their change activities and the possible synchronization between them.

7 Conclusion and Future Work

In this paper we proposed a solution of how to close the gap between a current and a target architecture. We identify the successor relationships between those architectural states and determined transformation paths in terms of sequences of concrete actions. Thereby the dependencies between the architectural elements are considered. If necessary the architecture can be divided into smaller segments. For the analysis we use model query languages, for the definition of

abstract actions and the creation of the transformation path we use graph transformation.

The solution is designed to support the enterprise architect in the task to find a way ***how*** to reach a defined target, starting with the current architecture. It enables the consideration of the complex dependencies between the architectural elements and thus reduces changes in project plans later on because of overlooked dependencies. The outcome of our solution, the transformation path, can be used to define the projects which will implement the changes.

The next step is to extend our solution to all architectural layers in enterprise architecture and start a broader field study. Future work will comprise a refinement analysis and actions to enable a more abstract target architecture as a starting point. Furthermore, providing support for the consideration of resource constraints and value-based weighting of the transformation steps is part of our future research. The presented solution provides a stable basis for these further extensions.

References

1. The Open Group: TOGAF Version 9.1. 1 edn. TOGAF series. Van Haren Publishing, Zaltbommel (2011)
2. Zachman, J.: A framework for information systems architecture. IBM SYSTEMS JOURNAL **26**(3) (1987)
3. Rouse, W.B.: A theory of enterprise transformation. Systems Engineering **8**(4) (2005) 279–295
4. Lankhorst, M.M., ed.: Enterprise architecture at work: Modelling, communication and analysis. 2 edn. Springer, Berlin (2009)
5. Spewak, S.H., Hill, S.C.: Enterprise architecture planning: Developing a blueprint for data, applications, and technology. John Wiley & Sons, New York (1992)
6. Luftman, J.N.; Lewis, P.O.S.: Transforming the enterprise: The alignment of business and information technology strategies. IBM Systems Journal **32**(1) (1993) 198–221
7. 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. Springer Berlin Heidelberg, Berlin and Heidelberg (2009) 55–69
8. Aier, S., Gleichauf, B.: Towards a systematic approach for capturing dynamic transformation in enterprise models. In Sprague, R.H., ed.: Proceedings of the 43rd Annual Hawaii International Conference on System Sciences. IEEE Computer Society, Los Alamitos and Calif (2010)
9. 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 (2006) 179a
10. Aier, S., Gleichauf, B.: Applying design research artifacts for building design research artifacts: A process model for enterprise architecture planning. In Winter, R., Aier, S., Zhao, J.L., eds.: Global Perspectives on Design Science Research. Volume 6105 of Lecture Notes in Computer Science. Springer, Berlin (2010) 333–348

11. Buckl, S., Ernst, A., Matthes, F., Schweda, C.M.: An information model for landscape management – discussing temporality aspects. In Feuerlicht, G., Lamersdorf, W., eds.: Service-oriented computing, ICSOC 2008 workshops. Volume 5472 of Lecture Notes in Computer Science. Springer, Berlin and and New York (2009) 363–374
12. Buckl, S., Dierl, T., Matthes, F., Schweda, C.M.: Complementing the open group architecture framework with best practice solution building blocks. In: 2011 44th Hawaii International Conference on System Sciences, IEEE (2011) 1–9
13. Buckl, S., Ernst, A.M., Matthes, F., Schweda, C.M.: An information model capturing the managed evolution of application landscapes. Journal of Enterprise Architecture **5**(1) (2009) 12–26
14. Nissen, V., Rennenkampf, A.v., Termer, F.: Agile it-anwendungslandschaften als strategische unternehmensressource. In Hofmann, J., Knoll, M., eds.: Strategisches IT-Management. Volume 284 of HMD Praxis der Wirtschaftsinformatik. dpunkt.verlag, Heidelberg (2012) 24–33
15. The Open Group: TOGAF Version 9. Van Haren Publishing, Netherlands (2009)
16. Hasselbring, W.: Information system integration. Communications of the ACM **43**(6) (2000) 32–38
17. Niemann, K.D.: From enterprise architecture to IT governance: Elements of effective IT management. Vieweg, Wiesbaden (2006)
18. Winter, R., Fischer, R.: Essential layers, artifacts, and dependencies of enterprise architecture. Journal of Enterprise Architecture **3**(2) (2007) 7–18
19. Matthes, F.: Softwarekartographie. Informatik-Spektrum **31**(6) (2008) 527–536
20. Keller, W.: IT-Unternehmensarchitektur: Von der Geschäftsstrategie zur optimalen IT-Unterstützung. 1 edn. dpunkt.verlag, Heidelberg (2007)
21. Hanschke, I.: Strategisches Management der IT-Landschaft: Ein praktischer Leitfaden für das Enterprise Architecture Management. 1. edn. Hanser, München (2009)
22. Sousa, P., Lima, J., Sampaio, A., Pereira, C.: An approach for creating and managing enterprise blueprints: A case for it blueprints. In Albani, A., Barjis, J., Dietz, J.L.G., eds.: Advances in Enterprise Engineering III. Springer, Berlin (2009) 70–84
23. Simon, D.: Application landscape transformation and the role of enterprise architecture frameworks. In Steffens, U., ed.: MDD, SOA and IT-Management. Gito, Berlin (2009)
24. Postina, M.: Evolutionsmanagement prozess- und serviceorientierter Unternehmensarchitekturen. PhD thesis, OlWIR Verlag für Wirtschaft, Informatik und Recht, Edewecht and Oldenburg, Germany (2011)
25. Aier, S., Gleichauf, B.: Application of enterprise models for engineering enterprise transformation. Enterprise Modelling and Information Systems Architectures **5**(1) (2010) 56–72
26. Edelkamp, S., Rensink, A.: Graph transformation and ai planning. In Edelkamp, S., Frank, J., eds.: Knowledge Engineering Competition (ICKEPS), Rhode Island, USA (2007)
27. Diefenthaler, P., Bauer, B.: Gap analysis in enterprise architecture using semantic web technologies. In: Proceedings of 15th International Conference on Enterprise Information Systems (ICEIS 2013). (2013) 211–220
28. Loshin, D.: Master data management. Morgan Kaufmann (2010)
29. 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. Gesellschaft für Informatik, Bonn (2010) 281–292