Context-Sensitive Impact Analysis for Enterprise Architecture Management

Melanie Langermeier, Christian Saad and Bernhard Bauer
Software Methodologies for Distributed Systems, University of Augsburg, Germany
{*langermeier, saad, bauer*}*@ds-lab.org*

Keywords: Enterprise Architecture Analysis, Impact Analysis, Change Propagation, Data Flow Analysis.

Abstract: Since Enterprise Architecture (EA) models are typically very large, it is often difficult for humans to fully grasp their contents. Due to this inherent complexity, the task of generating additional value from these models is very challenging without a suitable analysis method. Impact analysis, which is able to determine the effects which changes have on other architectural elements, can therefore provide valuable information for an enterprise architect. Whether an element is affected by a change depends on its context, i.e. its (transitive) connections to other elements and their status with respect to the analysis. In this paper we propose a context-sensitive approach to the implementation of impact analyses. This method relies on the technique of data-flow analysis to propagate the effects of changes throughout the model. As a consequence, the specification can be defined in a very generic fashion, which only relies on relationship classes. Therefore it can be easily adapted to organization-specific EA meta models as only the relationship types have to be mapped to the respective classes.

1 INTRODUCTION

Enterprise Architecture Management (EAM) provides methods for managing the inherent complexity of the large IT infrastructures encountered in many organizations. As a result, Enterprise Architecture (EA) models usually contain many elements which are connected through complex relationships. It is therefore vital to provide suitable methods for (semi-)automatically analyzing their contents to be able to benefit from this methodology once it has been successfully established in an organization.

Although much research has been done in the EA domain, most of this work focuses on methodologies for the development and the representation of enterprise models. By contrast, approaches and techniques which explore possible applications scenarios are very rare (Närman et al., 2012; Niemann, 2006). Regarding the analysis of EA models, a major focal point exists in their quantification. This encompasses the definition and computation of quality attributes such as application usage and service availability. (Närman et al., 2012) Furthermore, it is possible to evaluate the performance and cost aspects in the different layers of enterprise models (Jonkers and Iacob, 2009). Finally, (Matthes et al., 2012) establish a catalog of KPIs to measure EA management goals.

One of the most important analysis methods however, is the so-called impact analysis which allows to simulate the effects of changes (e.g. the modification of a CRM system) and to assess risks in the current architecture (e.g. which business operations would be affected if a specific server goes offline) (de Boer et al., 2005). To generate this information, an impact analysis has to evaluate the dependencies between the architecture's constituents. However, in order to make proper assertions about these relationships, it is necessary to evaluate each element in its respective context. This means, that its relationships with other elements in the model have to be taken into consideration. For example, to examine the impact of a server failure on business processes, one has to determine which applications rely on this server. This requires a careful evaluation of indirect and transitive paths in the model to ensure that all necessary information is retrieved, while at the same time excluding irrelevant relationships.

Existing approaches and tools for the creation and analysis of EA models usually rely on a static meta model structure. This can be a problem since each organization tends to employ its own meta model, making the adaption of existing analyses very difficult (Kurpjuweit and Aier, 2009). To rectify this situation, more flexible methods for handling structural depen-

Langermeier M., Saad C. and Bauer B.
Context-Sensitive Impact Analysis for Enterprise Architecture Management.
DOI: 10.5220/0005424200470055
In *Proceedings of the Fourth International Symposium on Business Modeling and Software Design (BMSD 2014)*, pages 47-55
ISBN: 978-989-758-032-1

dencies are required.

In this paper we present a technique which supports the context-sensitive impact analysis of EA models. It is based on the principle of data-flow analysis, a method which originates from the field of compiler construction. Using this approach, it is possible to derive context-sensitive information by propagating contextual information along the model's edges. Since the developed analysis distinguishes between different semantic relationship classes it can be easily adapted to the conventions in different organizations by mapping the relationship types in the respective target domain to the proposed categories. Furthermore, it is possible to extend the analysis with individual impact propagation rules. To demonstrate the viability as well as the generic applicability of this approach, we implement multiple impact analyses for different EAM languages.

2 IMPACT AND DEPENDENCY ANALYSIS

According to (Bohner, 2002), determining the effects of a change requires an iterative and discovery-based approach. Change impact analysis can be performed for a single software system, but also on an architectural level for a full application landscape or an enterprise architecture. A related topic which is also of interest in this context is the analysis of dependency relationships.

Typically, any change which is made to a model element also affects its neighboring elements (*direct impact*). However, as these changes may in turn affect other elements (*indirect impact*), the effect propagates throughout the model. Consequently, even a small change in a single element can cause ripple-effects, resulting in non-trivial consequences. While the direct impact can be derived from the connectivity graph, the computation of indirect impacts (*n-level impacts*) requires reachability information. However, since this method approximates potential impacts, it tends to overestimate the result by generating false-positives. The precision of the analysis can be improved by using a constraint mechanism or by incorporating structural and semantic information (Bohner, 2002).

Most of the work regarding impact analysis of software focuses on the code level (Lehnert, 2011). Approaches which evaluate architectures usually only regard concepts such as components, packages, classes, interfaces and methods. Due to the limited amount of supported types and the domain-specific characteristics, these approaches are not suitable for use in EAM.

Nevertheless, some techniques which target the UML are more closely related to the EAM domain. (Briand et al., 2003) propose a methodology for subjecting analysis and design documents to an impact analysis to detect side effects of changes in the context of UML-based development. To restrict the set of affected model elements they propose the use of a coupling measure and a predictive statistical model. The impact analysis itself is specified using the OCL. (von Knethen and Grund, 2003) developed an approach which supports traceability by providing requirements engineers, project planers and maintainers with the ability to monitor the effects that changes have on software systems. They differentiate between three types of relationships to define the traces: *representation*, *refinement* and *dependency*. To determine the change impact, they (semi-)automatically analyze requirement traces using these three categories.

(Kurpjuweit and Aier, 2009) and (Saat, 2010) propose techniques for EA dependency analysis. Saat focuses on time-related aspects (org. "zeitbezogene Abhängingkeitsanalysen") by considering for each element its life time, the status (current or proposed) as well as the life cycle phase with its duration. However, no execution or implementation details are provided for this approach. Kurpjuweit and Aier developed a formal method for flexible and generic dependency analysis. To determine dependent elements, they use the transitive closure of a set of relations. They also define an expansion function, which allows to consider special relation semantics, e.g. hierarchical refinement or reflective relation types.

(Holschke et al., 2009) as well as (Tang et al., 2007) propose the use of Bayesian Belief Networks (BBN) for EA modeling. These approaches rely on causal dependencies as well as inference methods for BBN and a diagnosis analysis to determine the impact. The former realizes a failure impact analysis, theoretically described in the pattern catalogue (Buckl et al., 2008), using the diagnostic analysis[1] and the modeling tool GeNIe. As a result, architectural components can be ranked with respect to their criticality for a business process. However, this approach focuses on availability, not on changes. Tang et al. employ a combination of predictive reasoning to determine affected elements and diagnostic reasoning to determine the cause of a change. Prior to the analysis, the architect has to assign a probability to each root node and a conditional probability table to each

[1] Jagt, R.M.: Support for Multiple Cause Diagnosis with Bayesian Networks. Vol. M. Sc. Delft University of Technology, the Netherlands and Information Sciences Department, University of Pittsburgh, PA, USA, Pittsburgh (2002)

non-root node.

Propagation rules are another method for determining the impact of changes. This technique allows to define effects that depend on structural and semantical properties. An iterative application of those rules to a model yields the direct and indirect impacts. (de Boer et al., 2005) present such rules for the most important relationships in ArchiMate models, differentiating between the *removal*, the *extension* and the *modification* of an architectural element. However, the definitions are given in an informal and textual manner and no technical realization is supplied. (Kumar et al., 2008) propose rules that encode the dependency relationships of the attributes of entities. Changes are thereby propagated to determine the impact on a defined set of element types, namely business goals, processes, services and infrastructure components as well as the relations *runs on*, *provides*, *executes* and *delivers*. No mechanism is specified for implementing the change propagation. (Aryani et al., 2010) also rely on the propagation concept to define a conceptual coupling measurement for software components. Based on this information a dependency matrix is established which allows to predict change impacts.

In (Lankhorst, 2012), a tool for impact-of-change analysis is described. The author represents enterprise architectures in XML and uses the Rule Markup Language (RML) to define transformations which represent the rules which define the impact-of-change. The RML rules are analyzed through a pattern matching of the antecedent against the input XML. If a rule matches, the variables will be bound and an output XML is generated based on the rule output.

3 A CONTEXT-SENSITIVE APPROACH TO IMPACT ANALYSIS

The foundation for the definition of any impact is the computation of reachable elements. According to (Bohner, 2002), *reachability* denotes transitive connections, whereas *dependability* refers to directly connected elements. To determine reachability relationships we employ data-flow analysis, a technique which is based on the principle of information propagation. This allows to directly implement the following recursive specification: An element is reachable if at least one predecessor element is reachable. In this context, a predecessor is defined as the source element of an incoming edge. Since there are typically no isolated areas in an EA model, this would normally result in almost all elements being classified as reachable. For a more focused analysis, we therefore need to extend the reachability computation with contextual information. For this purpose, we establish two different categorization mechanisms for relationships. For each relationship class in these categories we define a change propagation rule which specifies how a change will be propagated through the model.

In the following we will first formalize the representations of model and meta model data in a way which ensures the applicability of the approach even if an organization employs a customized version of the meta model. We will then describe a data-flow based specification of a naive reachability analysis and subsequently propose extensions which enable a context-sensitive analysis of change impacts.

3.1 Formalizing the Meta Model and the Model

The high diversity of meta models results in a major challenge when devising techniques in the context of EAM. To overcome this issue, we developed a generic meta model which is able to support any EA language based on traditional modeling paradigms. Apart from abstracting from the particular structure of an input language, this approach has the benefit of combining meta model and model data in a single representation.

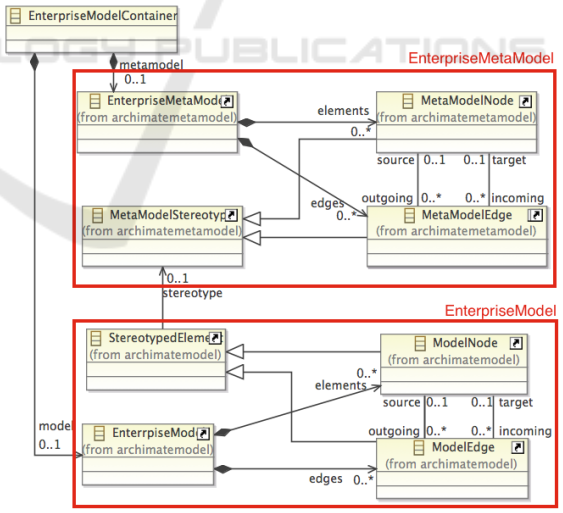

Figure 1: Generic representation for EA (meta) model data.

A condensed version of this specification is depicted in figure 1. The relevant elements can be described as follows: Each concept of the respective target EA language is translated into either a *MetaModelNode* or a *MetaModelEdge*. Connections between these elements have to be established accordingly during the transformation process. Both types

also carry additional meta information such as their stereotype, the concept's name and its properties. Instances from the target EA model are converted into *ModelNodes* and *ModelEdges* and connected to their respective meta model stereotypes.

3.2 Analyzing Reachability for EA Models

The computation of reachability information forms the basis for the subsequent impact analysis. An element is declared reachable, if there exists a path connecting the element to the starting point (*indirectly connected elements*). The reachability analysis is carried out using the Model Analysis Framework (MAF) (Saad and Bauer, 2011) which supports the specification and execution of data-flow based analyses on models.

Data-flow analysis is used by compilers to derive optimizations by examining the structural composition of program instructions. Canonical examples include the calculation of reaching definitions and variable liveness. For this purpose, the program is converted into a control-flow graph with the nodes representing the basic blocks and the edges denoting the flow of control. A set of data-flow equations is then evaluated in the context of each node. Each equation takes the results computed at the immediate predecessor nodes as input, applies a confluence operator (union or intersection) to combine these sets and finally modifies the values to reflect the effects of the local node's instructions. Effectively, this method describes an equation system which propagates information throughout the underlying graph, thus enabling a context-sensitive evaluation of each instruction. If loops are present, fixed-point evaluation semantics are employed to approximate the runtime behavior of the program.

In (Saad and Bauer, 2013) we discussed an adaption of this analysis technique to the modeling domain which we referred to as a *generic "programming language" for context-sensitive model analysis*. This approach defines a declarative specification language that allows to annotate data-flow attributes at meta model classes that can subsequently be instantiated and evaluated for arbitrary models. This technique has several significant advantages: Data-flow analysis provides inherent support for the implementation of recursive specifications which iteratively propagate information throughout a model. Also, since information is routed along model edges, each model element can be evaluated in its overall context, thus eliminating the need for static navigational expressions which are common in languages such as OCL. This is important in the EAM domain where the structure of both meta models and models is highly dynamic. Finally, the usage of fixed-point semantics allows to implement a correct handling of cyclic paths.

Using MAF, a reachability analysis for model elements can be specified in the following way:

```
1: analysis reachability_analysis {
2:   attribute is_reachable : Boolean initWith false;
3:   extend node with {
4:     occurrenceOf is_reachable calculateWith
5:       self.incoming.source.is_reachable()
6:         ->includes(true);
7:   }
8:   extend startnode with {
9:     occurrenceOf is_reachable calculateWith true;
10:  }
11: }
```

As described above, an element $e1$ is reachable from another element $e2$, if there exists a path between $e1$ and $e2$. Here, we assume that the meta model defines the classes *node* and *startnode*, the latter one being a specialization of the former one. We further classify changed elements in the model as *startnodes* for the analysis. The reachability status is computed by a data-flow attribute *is_reachable* of type *boolean* which is initialized with the value *false* (line 2). Lines 3-7 attach this attribute to all instances of the *node* class. To determine the reachability status of a node, the data-flow equation in lines 5-6 accesses the *is_reachable* values computed at the respective node's predecessors, thereby directly implementing the recursive specification. Finally, lines 8-10 overwrite this equation at *startnodes* which are, by definition, always reachable.

3.3 Context-aware Change Propagation

The execution of the reachability analysis in section 3.2 will result in an large result set, containing mostly false positives regarding change impact. By enriching the rules with context-specific declarations, the impact set can be restricted to contain only meaningful data and to additionally reflect different types of changes.

In the following, we will differentiate between the change types *extend*, *modify* and *delete* as proposed by (de Boer et al., 2005). Extensions refer to cases where new issues are added but the initial functionality or structure remains the same. Consequently, extensions do not propagate to depending elements. By contrast, a modification also affects the functionality or the structure and therefore it cannot be guaranteed that initially provided issues will still be available

or that their behavior remains unchanged. Finally, deletion indicates that an element will be removed from the enterprise architecture. The change types are prioritized as follows: *delete* overrides *modifies* overrides *extends* overrides *no change* (NO). Depending on the respective requirements, additional change types can be implemented.

Due to the lack of detailed information in enterprise architecture models, an accurate definition of the impact of a change is not possible. We therefore propose to approximate the impact using a worst case and a best case analysis similar to the practices in software analysis. For the worst case, the impact is defined as the maximal set of affected elements, whereas the best case includes only the minimal set. The real impact typically lies somewhere between both cases.

To implement the context-dependent impact analysis, we define rules which are able to differentiate between the different change and relationship types. To make the technique generically applicable, we introduce custom relationship classes to which the specific relationship types in the target EA language can be mapped. The developed specifications can be divided into two categories: In section 3.3.1 describe how the propagation of effects is influenced by different relationship classes while section 3.3.2 introduces an additional classification along different effect types.

3.3.1 Change Propagation Rules Depending on Relationship Classes

To classify the relationships of an enterprise architecture, we grouped them according to their semantics, which we identified through a literature review of existing EA frameworks and their meta models. This includes the Core Concepts Model (CC) of ArchiMate (The Open Group, 2012) and the DM2 Conceptual Data Model of DoDAF (U.S. Department of Defense, 2010).

Overall, we were able to identify five classes of relevant EA relationship types: *Locate* denotes the allocation to some location or organization unit. Any kind of provision of functionality, information and behavior is of the type *provide* while the *consume* class denotes the consumption of those elements. *Structural dependency* relationships define the structure or organization of entities in one layer. The *behavioral dependency* class on the other hand summarizes relationships which declare dependencies between the behavior of elements in a single layer which are neither of the type provide nor consume. The following table lists all classes along with corresponding examples from the ArchiMate Core Concepts and the DoDAF DM2.

Table 1: Classification of EA relationships.

class	examples
`locate`	*CC*: assignment *DM2*: is-at
`provide`	*CC*: realize, assess *DM2*: provide, performedby
`consume`	*CC*: used by, access *DM2*: consume
`structural dependency`	*CC*: aggregate, composite *DM2*: part-of
`behavioral dependency`	*CC*: trigger, flow to

Note that the mapping in table 1 is only a suggestion based on our interpretation of the concepts and has to be adapted if an organization assigns different semantics to these types. It is also important to realize that each relationship may belong to multiple categories. In the worst case analysis, the strongest rule will be chosen while the best case analysis will use the weakest one.

To formalize the change semantics of these classes, we employ the following syntax:

$$A.X \rightarrow B.Y \tag{1}$$

This statement indicates that if element A is changed in the manner X then element B has to be changed in manner Y. A and B represent the source and the target of the relationship while $X, Y \in \{modify, delete, extend\}$. It is also possible to cluster change operations on the left hand side. $A.\{X,Y\} \rightarrow B.Z$ means that if A is changed in the manner X or in the manner Y, B has to be changed in the manner Z. Optionally, it is possible to differentiate between a worst case (WC) and a best case (BC) impact on the right hand side of the rule.

We will now demonstrate this concept using the location relationship. Assuming that an application component (A) is hosted by a organization unit (B), this connection is mapped to the class *located at*. If a change to the application component has no effect on the organization unit the rule will be $A.\{del, mod, ext\} \rightarrow B.NO$. If, on the other hand, the organization unit is deleted, the application component loses its host. In the worst case it needs to be deleted as well while in the best case it will simply be assigned to another host. This is formalized as: $B.del \rightarrow WC : A.del, BC : A.ext$. Finally, if the organization unit is modified or extended, the worst case demands that the application component has to be modified too while, in the best case, it remains as is. This can be addressed with the rule: $B.\{ext, mod\} \rightarrow WC : A.mod$, $BC : A.NO$. Change rules for other relationship classes are defined in a similar manner as shown in table 2.

Table 2: Impact rules for the relationship classes.

class	rule
located at	A.{del,mod,ext} → B.NO B.del → WC: A.del BC: A.ext B.{ext,mod} → WC: A.mod BC: A.NO
provides	A.del → WC: B.del BC: B.ext A.mod → WC: B.mod BC: B.NO A.ext → WC: B.ext BC: B.NO B.{del,mod,ext} → A.NO
consumes	A.{del,mod,ext} → B.NO B.{del,mod} → WC: A.mod BC: A.ext B.ext → A.NO
structurally dependent	A.del → WC: B.del BC: B.mod A.{mod,ext} → B.NO B.{del,mod} → WC: A.mod BC: A.NO B.ext → WC: A.ext BC A.NO
behaviorally dependent	A.{del,mod,ext} → B.NO B.{del,mod,ext} → A.NO

3.3.2 Change Propagation Rules Depending on Effect Types

In addition to the classification along the lines of relationship types, a differentiation between different effect types can be useful as well. We therefore define the following three effects: *strong*, *weak* and *no effect*. The type of effect has to be specified for each direction of a relationship. The notation $X - Y$ indicates that a change in the source has a effect of type X on the target and vice versa. Overall, this leads to six effect classes: *Strong-Strong*, *Strong-Weak*, *Strong-No effect*, *Weak-Weak*, *Weak-No effect* and *No effect-No effect*.

The semantics of these effects can be defined using rules similar in nature to those presented in section 3.3.1. They are shown in table 3.

Table 3: Impact rules for the effect classes.

effect	rule
strong	A.del → WC: B.del, BC: B.ext A.mod → B.mod A.ext → B.ext
weak	A.del → WC: B.mod, BC: B.no A.mod → WC: B.mod,BC: B.ext A.ext → WC: B.ext, BC: B.NO
no effect	A.{del,mod,ext} → B.NO

If A strongly affects B, this indicates that if A is deleted, in the worst case, B has to be deleted as well and, in the best case, it only needs to be extended. A modification in A leads to a modification of B and the same applies to extensions. If, for example, an application component realizes a service, then the application component has a strong impact on the service while the service may only have a weak impact on the application component. This specific interpretation of *realize* would result in an assignment to the *Strong-Weak* class. A weak effect denotes that the deletion of A conducts no change in B in the best case and a modification in the worst case. A modification of A in the worst case requires a modification of B. In the best case it has only to be extended. Finally if A is extended, in the best case B must not be changed, in the worst case it has to be extended, too. If the relationship is mapped to *no effect*, any change of A has no effect on B.

Further examples for effect mappings of ArchiMate relationships are:

- Strong-Weak: realize
- Strong-No effect: aggregation
- Weak-No effect: use, assign
- Weak-Weak: triggers

3.3.3 Realization of the Rules

The rules defined in sections 3.3.2 and 3.3.1 can be realized as data-flow equations. First, the meta model and model data has to be converted to the generic representation presented in section 3.1. Then, the status of the changed elements whose impact should be analyzed is set to the respective value while the result for all other elements is initialized with *no change*. Afterwards, these values will be iteratively recomputed to propagate the effects of the changes. For illustration purposes, we include a Java-based implementation of the rule which calculates the best case result based on the presented effect types:

```
1:  Object node_changestatus_bestcase(Node currentNode){
2:    for (Edge edge : currentNode.getIncomingEdges()){
3:      Status sourceStatus = edge.source.getStatus()
4:      Status currentStatus = currentNode.getStatus()
5:      if (edge.effectClass == StrongEffectTarget)
6:        if (sourceStatus == (DEL||EXT))
7:          return computeStatus(currentStatus, EXT)
8:        else if (sourceStatus == MOD)
9:          return computeStatus(currentStatus, MOD)
10:     if (edge.effectClass == WeakEffectTarget)
11:       if (sourceStatus == (DEL||EXT))
12:         return computeStatus(currentStatus, NO)
13:       else if (sourceStatus == MOD)
14:         return computeStatus(currentStatus, EXT)
15:     if (edge.effectClass == NoEffectTarget)
16:       if (sourceStatus == (DEL||MOD||EXT))
17:         return computeStatus(currentStatus, NO)
18:   }
19:   for (Edge edge : currentNode.getOutgoingEdges()){
20:     ...
21:   }
22: }
```

The status of the current element (*currentNode*) depends on the status of the connected elements as well as the direction of the relationship. Therefore, to correctly determine the change status, all incoming (lines 2 - 18) and outgoing edges (lines 19 - 21) have to be processed. The status value which has been computed for a connected element is retrieved through an invocation of *getStatus()* (line 3). This call instructs the data-flow solver to recursively compute and return the requested value. Based on the type of each incoming edge, it is then decided whether it has a *strong effect* (line 5), a *weak effect* (10) or no effect (15) on its target. The concrete type of the change is determined by evaluating the status of the edge's source element (lines 6-9, 11-14, 16-17). Finally, *computeStatus()* is invoked to compute and return the status of the local element. To implement the priorization relationships between the change types, e.g. to ensure that a weak change like *no change* cannot override a stronger one like *delete*, this method takes both the current and the newly computed status as input. A similar approach is used to calculate the result for the source elements of outgoing edges (line 19-21).

3.4 Customization of the Impact Analysis

In the case where the rules proposed in section 3.3 are not sufficient to capture all requirements of the organization, it is possible to customize the analysis. For example, if a specific relationship type cannot be mapped to one of the proposed classes, a new rule can be created. In addition to evaluating relationship types and change status of connected elements, a rule may also consider the type of the connected elements or class properties. It would also be possible to extend the rule definitions with the ability to quantify a change (e.g. in terms of costs). These features can be implemented through additional data-flow attributes. For example, to compute potential savings on IT maintenance, the maintenance costs of all deleted application and infrastructure components and their corresponding services could be aggregated.

Another customization consists of a modification of the rule set to support change probabilities. Instead of a single status, we can define four separate data-flow attributes, which compute the respective probabilities for the types *delete*, *modification*, *extension* and *no change*. Additionally, the rule specification would have to be extended. For example, a rule could be defined as:

$$P(A.del) = X \rightarrow P(B.del) = 0.8 \times X \qquad (2)$$

This means that if the probability that A is deleted is X, then the probability that B has to be deleted is $0.8 \times X$ or, in other words, if A is deleted then in 80% of the cases B will be deleted as well.

4 EVALUATION

Most of the research work regarding change impact analysis has been carried out theoretically and thus has not yet been applied to real architecture models (e.g. (de Boer et al., 2005), (Kurpjuweit and Aier, 2009), (Kumar et al., 2008)). An exception exists in the work of (Tang et al., 2007) who employ predictive and diagnostic reasoning in BBN. However, one disadvantage of their approach can be found in the high effort required to annotate probability information. The technique proposed in this paper simplifies analysis specification through a generic representation of model data and through predefined and extensible categorizations of relationships and effects.

Many existing approaches do not address problems relating to cyclic dependencies or contradicting results, the latter one for example being a weakness of the tooling proposed by (Lankhorst, 2012). Furthermore, the issue of the scalability of the technique which is based on pattern matching and model transformations is not considered and the employed RML technique is highly dependent on specific usage scenarios as well as on the respectively chosen EA language. By utilizing the data-flow analysis method with its inherent support for cyclic dependencies, recursive specifications and iterative result computation, we are able to address these challenges. The scalability of DFA (and the Model Analysis Framework in particular) has been demonstrated in the context of other domains including the analysis of extensive AUTOSAR models (Kienberger et al., 2014).

For a practical evaluation, we implemented the proposed methods in the form of an addin for MID Innovator (MID GmbH, 2014) using the MIDWagen example which is shipped with the tooling. MIDWagen describes the IT landscape of a car rental organization with its actors, business services, business processes, application components and services as well as the required infrastructure components and services. Although it is not a real world example, the level of detail and the extensibility of the underlying EA language enables a thorough evaluation of the viability and the robustness of our technique.

To illustrate the application of our approach, we focus on a modification of the *Booking System*. We assume that this component, which is responsible for payment transactions, has to be modified due to secu-

rity issues. We further state that this change will only affect the *Payment* service while the *Bonus Booking* service does not have to be adapted. The modification of the *Payment* application service (AS) also causes a change to the supporting *Return* service, which in turn will affect the *Payment* business service (BS) as well as the *Renter* role. Figure 2 shows the respective excerpt from the MIDWagen model.

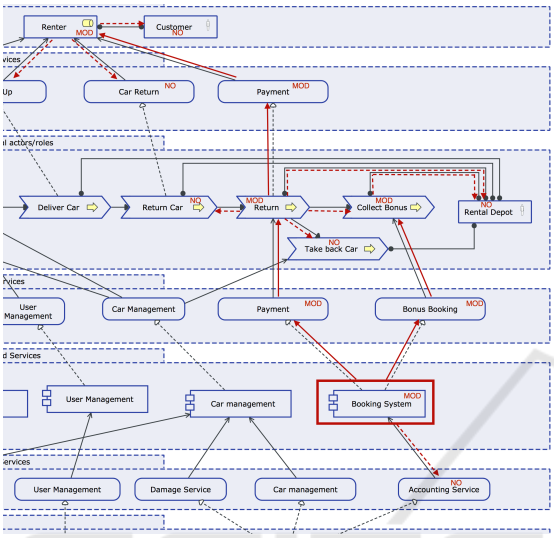

Figure 2: Excerpt of the ArchiMate model for the MIDWagen Use Case (MID GmbH, 2014) with the worst case change propagation path.

For this scenario, we employ the relationship classification described in section 3.3.1. Since the model is given in the ArchiMate language, we are able to use the mappings listed in table 1.

In figure 2, the resulting worst case change propagation path is indicated by red arcs. Solid lines represent paths along with a change is forwarded, while dashed lines stand for relationships, which are considered but do not result in a change propagation. The final impact set consists of the elements {*Payment AS* modified, *Bonus Booking* modified, *Return* modified, *Collect Bonus* modified, *Payment BS* modified, *Renter* modified}, while the final result set for the best case analysis is empty. Both results represent realistic approximations which have to be interpreted considering the severity of the modification. In the best case scenario (e.g. performance issues), the modification of the *Booking System* does not affect the provided functionality, and therefore the service does not need to be changed. In the worst case, for example a substantial change in the functionality due to security issues, the effects of the change propagate to the role in the business layer which is potentially affected by the modification. Both impact sets represent approximations, which can be of great value for estimating the real effects of a change especially in early design stages.

Carrying out the analysis using the effect classes, the result consists of {*Payment AS* modified, *Return* modified, *Payment BS* modified, *Renter* modified}. This result, which lies in between the worst and the best case of the relationship analysis, is able to provide more detailed information in the case where the required data is available.

5 CONCLUSIONS

In this paper we proposed a context-sensitive impact analysis technique for EA models. The approach relies on two underlying concepts: The problem of diverse EA languages is addressed by a generic representation of model data while the data-flow analysis method enables an intuitive specification of analyses, which depend on the iterative propagation of results.

We argued that a traditional reachability analysis which returns all direct and indirect neighbors of an element is not suitable in the EA context and therefore has to be extended with context-sensitive propagation rules. For this purpose, we defined a relationship classification which reflects the semantics of different edge types as well as the semantics of the change types *extend*, *modify* and *delete*. By applying this concept, change effect propagation depends both on the change type as well as on the meaning of the relationships which connect the respective elements. For example, if an element which is still in use is deleted, this change will affects consumers. If, on the other hand, the element is extended, leaving the existing functionality unchanged, the potential effect is not propagated.

For a more focused analysis, we proposed two different kinds of relationship classifications. Section 3.3.1 categorizes relationships according to their semantics with respect to the architecture by defining the classes *located at*, *provides*, *consumes*, *structurally dependent* and *behaviorally dependent*. In section 3.3.2, the severity of a change on the respective source and the target elements is considered by introducing the categories *strong*, *weak* and *no effect*. In both cases we defined propagation rules in the form of DFA equations for best and worst case analysis. By extending these definitions, it is possible to include support for organization-specific semantics and additional relationship types. Executing the analysis using the DFA solver of the MAF framework yields the results which can be interpreted as estimations that reflect the best and worst case of the actual impact.

The combination of the generic model represen-

tations, extensible DFA-based analysis specifications and the classification approach for relationships ensures that this technique can be applied to the various EA conventions found in different organizations. Further work has to be done to determine a suitable visualization of the results. It would also be interesting to evaluate rules for other impact scenarios such as failure impact analysis which analyzes the availability of architecture elements. Finally, it should be explored how the computation of best and worst case results could be improved through an integration of probability distributions. At the moment we only support a simple and naive way for the integration of probabilities.

ACKNOWLEDGEMENTS

This work was partially sponsered by the FuE-Programm Informations- und Kommunikationstechnik Bayern. The authors would like to thank MID GmbH for providing their demo use case, licenses for their tool as well as for their support during the implementation.

REFERENCES

Aryani, A., Peake, I., and Hamilton, M. (2010). Domain-based change propagation analysis: An enterprise system case study. In *2010 IEEE International Conference on Software Maintenance (ICSM)*, pages 1–9.

Bohner, S. (2002). Software change impacts-an evolving perspective. In *International Conference on Software Maintenance, 2002. Proceedings*, pages 263–272.

Briand, L., Labiche, Y., and O'Sullivan, L. (2003). Impact analysis and change management of UML models. In *International Conference on Software Maintenance, 2003. ICSM 2003. Proceedings*, pages 256–265.

Buckl, S., Ernst, A., Lankes, J., and Matthes, F. (2008). Enterprise Architecture Management Pattern Catalog (Version 1.0). Technical Report TB 0801, Technical University Munich, Chair for Informatics 19.

de Boer, F., Bonsangue, M., Groenewegen, L., Stam, A., Stevens, S., and van der Torre, L. (2005). Change impact analysis of enterprise architectures. In *Information Reuse and Integration, Conf, 2005. IRI -2005 IEEE International Conference on.*, pages 177 – 181.

Holschke, O., Nrman, P., Flores, W., Eriksson, E., and Schnherr, M. (2009). Using enterprise architecture models and bayesian belief networks for failure impact analysis. In *Service-Oriented ComputingICSOC 2008 Workshops*, page 339350.

Jonkers, H. and Iacob, M.-E. (2009). Performance and cost analysis of service-oriented enterprise architectures. *Global Implications of Modern Enterprise Information Systems: Technologies and Applications, IGI Global*.

Kienberger, J., Minnerup, P., Kuntz, S., and Bauer, B. (2014). Analysis and Validation of AUTOSAR Models.

Kumar, A., Raghavan, P., Ramanathan, J., and Ramnath, R. (2008). Enterprise Interaction Ontology for Change Impact Analysis of Complex Systems. In *IEEE Asia-Pacific Services Computing Conference, 2008. APSCC '08*, pages 303 –309.

Kurpjuweit, S. and Aier, S. (2009). Ein allgemeiner Ansatz zur Ableitung von Abhngigkeitsanalysen auf Unternehmensarchitekturmodellen. *Wirtschaftinformatik Proceedings 2009*.

Lankhorst, M. (2012). *Enterprise Architecture at Work*. Springer-Verlag Berlin and Heidelberg GmbH & Co. KG, Berlin.

Lehnert, S. (2011). A review of software change impact analysis. *Ilmenau University of Technology, Tech. Rep*.

Matthes, F., Monahov, I., Schneider, A., and Schulz, C. (2012). EAM KPI Catalog v 1.0. Technical report, Technical University Munich.

MID GmbH (2014). MID Innovator for Enterprise Architects. in: http://www.mid.de/produkte/innovator-enterprise-modeling.html, accessed 15/04/2014.

Närman, P., Buschle, M., and Ekstedt, M. (2012). An enterprise architecture framework for multi-attribute information systems analysis. *Software & Systems Modeling*, pages 1–32.

Niemann, K. D. (2006). *From enterprise architecture to IT governance*. Springer.

Saad, C. and Bauer, B. (2011). The Model Analysis Framework - An IDE for Static Model Analysis. In *Proceedings of the Industry Track of Software Language Engineering (ITSLE) in the context of the 4th International Conference on Software Language Engineering (SLE'11)*.

Saad, C. and Bauer, B. (2013). Data-flow based Model Analysis and its Applications. In *Proceedings of the 16th International Conference on Model Driven Engineering Languages and Systems (MoDELS'13)*.

Saat, J. (2010). Zeitbezogene Abhängigkeitsanalysen der Unternehmensarchitektur. *Multikonferenz Wirtschaftsinformatik 2010*, page 29.

Tang, A., Nicholson, A., Jin, Y., and Han, J. (2007). Using bayesian belief networks for change impact analysis in architecture design. *Journal of Systems and Software*, 80(1):127148.

The Open Group (2012). *ArchiMate 2.0 specification: Open Group Standard*. Van Haren Publishing.

U.S. Department of Defense (2010). The DoDAF Architecture Framework Version 2.02. in: http://dodcio.defense.gov/dodaf20.aspx, accessed 15/03/2015.

von Knethen, A. and Grund, M. (2003). QuaTrace: a tool environment for (semi-) automatic impact analysis based on traces. In *International Conference on Software Maintenance, 2003. ICSM 2003. Proceedings*, pages 246–255.