Using Semantic Web Technologies for Enterprise Architecture Analysis

Maximilian Osenberg[1,2,3,4], Melanie Langermeier[4](✉), and Bernhard Bauer[4]

[1] Elite Graduate Program Software Engineering, Technical University Munich, Munich, Germany
[2] Elite Graduate Program Software Engineering, Ludwigs-Maximilian-University, Munich, Germany
[3] Elite Graduate Program Software Engineering, University of Augsburg, Augsburg, Germany
[4] Software Methodologies for Distributed Systems, University of Augsburg, Augsburg, Germany
{osenberg,langermeier,bauer}@ds-lab.org

Abstract. Enterprise Architecture (EA) models are established means for decision makers in organizations. They describe the business processes, the application landscape and IT infrastructure as well as the relationships between those layers. Current research focuses merely on frameworks, modeling and documentation approaches for EA. But once these models are established, methods for their analysis are rare. In this paper we propose the use of semantic web technologies in order to represent the EA and perform analyses. We present an approach how to transform an existing EA model into an ontology. Using this knowledge base, simple questions can be answered with the query language SPARQL. The major benefits of semantic web technologies can be found, when defining and applying more complex analyses. Change impact analysis is important to estimate the effects and costs of a change to an EA model element. To show the benefits of semantic web technologies for EA, we implemented an approach to change impact analysis and executed it within a case study.

1 Introduction

Today's organizations have to deal with the complexity of large IT landscapes together with fast changing business architectures. Enterprise architecture (EA) models are used to capture the IT infrastructure elements, the used applications as well as the business processes. Especially the relationships between the elements are of major interest in order to understand the organizations' structure. The domain of Enterprise Architecture Management (EAM) captures the process of assessing the current EA of an organization as well as defining and implementing a target architecture [12]. Thus, EAM is a mean for incorporating changes throughout the whole organization as well as for driving optimizations of the architecture, especially the alignment of business and IT. The optimization of the business processes itself is dealt within the domain of Business Process Re-engineering.

F. Gandon et al. (Eds.): ESWC 2015, LNCS 9088, pp. 668–682, 2015.
DOI: 10.1007/978-3-319-18818-8_41

Current methods and tools in the EA domain provide means to calculate specific key performance indicators and visualize the results in an EA model e.g. an architecture diagram can be annotated in a way, that application systems running out of support shortly are colored red (see [18]). But combining this fact with other ones, e.g. the rate of business critical processes supported by the application, or the availability of a successor application bears challenges. A vital question in this context is also the effect of a specific change. The change in one element can cause ripple effects throughout the whole organization. Those indirect effects of a change are not always obvious, but can cause severe costs for a project. Especially since EA models are typically very large, humans cannot capture these effects easily. Change impact analysis is used to calculate the affected elements and thus provide the enterprise architect further information, whether a change should be implemented or not. Current implementations of more complex analyses and measures are highly dependent on the used meta model. Since every organization has its own, customized EA meta model, reusing existing analysis methods is not trivial.

In this paper we propose the use of semantic web technologies to represent EA models and to perform analyses on them. We reuse an existing formalization in order to transform an EA model into an ontology (Sect. 3.1). Then we show how simple measures and reports can be defined using SPARQL and reasoning. In order to implement more complex analyses in a flexible way, we propose to solely define their semantics and finally integrate them in an existing EA ontology (Sect. 3.2). The applicability of our approach is shown through the implementation of a change impact analysis (Sect. 4) and its execution in a case study (Sect. 5).

2 Foundations and Related Work

EA models are used to document the organization, its components and the relationship between those. Thus they provide a mean to capture and understand the complex dependencies between the business and the supporting IT infrastructure [11]. An EA model is documented using an organization specific set of concepts and relationships between those. Typical examples for concepts are *business processes*, *application components* and *infrastructure components* as well as *use* and *realize* relationships between those. Existing EA frameworks, like the Zachman framework [27] or TOGAF [22] propose different approaches for the documentation. There is no common standard for EA models. The actual used meta model is, in most cases, an adaption of an existing framework, tailored to the specific needs of the organization. This leads to a high variety of meta models used in organizations and is a major challenge, when defining methods and techniques to gain value from the EA model. Current research focuses on the development of EA frameworks as well as modeling and documentation approaches. EA analysis is not the main focus and thus not much work exists [16,17]. Existing approaches rely on techniques like XML [3] or a probabilistic extension of OCL [7]. In practice, reporting and measure calculation in the domain of EA is often performed using SQL databases but also with Excel sheets. In [19] SPARQL was proposed for analyzing an

EA, as it allows to perform different kinds of analyses with different complexities using the same technology. These approaches have in common that they are dependent on the underlying meta model, the adaption to a different one requires much effort. Despite for analysis purposes, semantic web technologies are also proposed for other reasons in EAM: Chen et al. present a method to integrate data from several sources into one EA repository using semantic web technologies [1]. Their goal is to automate the time-consuming documentation process through the use of semantic web technologies. In [5] a formalization of the TOGAF meta model using ontologies is presented in order to improve the quality and consistency of an EA model. The use of semantic web technologies is more common in the domain of business process analysis. E.g. [4] propose their use to integrate static and procedural domain knowledge as well as execution data in order to analyze them.

EA models can also be utilized to determine the potential impact of a change to one or more model elements. De Boer et al. describe in [2] an informal approach to change impact analysis in ArchiMate models. ArchiMate [23] is a modeling language for enterprise architectures, based on the TOGAF standard [22]. They start by considering a change to be either a *modification*, *extension* or *deletion* (or none). A *modification* is a change that modifies existing functionality, whereas an *extension* is a change that preserves existing functionality and adds new functionality (e.g. changing the signature of a method in contrast to adding a new method to an existing class); a *deletion* is a change that removes a whole component [2]. When calculating the impact of a change, starting from the first component to be changed, all components in the EA model are visited iteratively – comparable to a depth-first-search – and are annotated with how they need to be changed. The type of the relation determines how a relation between two components behaves towards a change. An example for such a propagation rule is given in the following for the relation type *access* that exists between two components A and B (e.g. application A accesses a data store B):

- In case of a change starting from A:
  - If A is **deleted**, it has no impact on B, because B doesn't depend on A.
  - If A is **extended** (or modified) this may change the way, the data stored in B is handled and thus requires an extension (or modification) of B.
- In case of a change starting from B:
  - If B is **deleted**, the object A can no longer access B. This does not mean that A needs to be changed, but that the access relation of A is lost. This has to be signaled to the user, so he can either link the access relation of A with another data object or delete A or its access relation.
  - If B is **extended**, it still provides the functionality it did before the change happened. Thus, A is not subject to change.
  - If B is **modified**, the data or the way data in B is accessed, has changed. Thus, A has to modified too.

Aier and Kurpjuweit provide an approach for change impact analysis by calculating the transitive closure of the relationships [9]. Therefore they use a relational composition operator to define the implicit relations between objects.

This composition operator must only allow compositions of relation types, which mean a dependency of objects in the context of change impact analysis. Since the authors focus on dependency analysis in general, they do not provide a differentiation between different change types. Further change impact analysis approaches are based on probability distributions [6,21]. Those methods are based on change probabilities, whom establishment requires a high workload. Thus if this information is not available, those methods are not the best approach. Change impact analysis is also supported in current EA tools. It is typically implemented as a report that determines a hard coded set of elements, e.g. all applications supporting a specific process and the related organization units (e.g. [18]). Another approach to determine the impact provided by EA tools are visualizations, e.g. using an interactive hierarchy graph (e.g. [14]) or through highlighting affected elements with colors (e.g. [18]). Thereby the tools do not differ between specific change types out of the box. Additionally the impact is rarely determined in a transitive way, thus ripple effects are not considered directly.

3 Applying Semantic Web Technologies to EA

We propose the use of semantic web technologies for the representation of an EA and to perform analysis on them. The transformation of an existing EA model into an ontology is described in Sect. 3.1. Methods for their analysis are presented in Sect. 3.2. Despite existing ones, we present a method for the execution of complex analysis in different EA models.

3.1 From Formal Description to EA Ontology

We present our method for transformation along the meta model shown in Fig. 1. The meta model is based on ArchiMate [23] and belongs to the case study used for the evaluation. Nevertheless our method can be applied to any other EA meta model as well. A *Role* is a structural concept that can "do" things (e.g. an employee or a customer of a company). A *Process* is usually executed by a *Role* if (in the model) there exists an *assign* relationship between these two concepts. A *Service* provides functionality that is *realize*d by a *Component* (e.g. a software application). A *Service* can be *use*d by either a *Role*, an *Application* or a *Process*. A *DataObject* (e.g. a database system or a part of a database) can be *access*ed manipulatively. Note that every relation type t has a corresponding t^{-1} which does not exist in Fig. 1 due to clarity.

According to Aier and Kurpjuweit [9] an EA meta model can be formalized as tuple $M = (C, T, R)$, where C is a set of *concepts*, T is a set of *relation types* with $\forall t \in T : \exists t^{-1} \in T : (t^{-1})^{-1} \in T$, and $R \subseteq C \times C \times T$ is the set of relations that can exist between two *concepts*.

An EA model is described as tuple $A = (E, T^*, Q, F)$, where E is a set of *objects*, being an instance of a *concept*, T^* is the transitive closure of the *relation types* T, $Q \subseteq E \times E \times T^*$ is a set of relations that exist between two *objects*, and F_e: $E \rightarrow C$ is a function that returns the *concept* of an *object*.

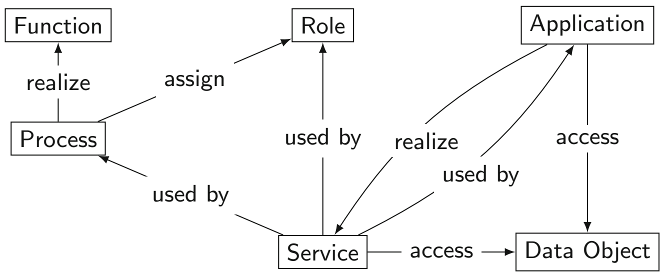


Fig. 1. EA meta model of the case study

For a *model* A to be a correct instance of a *meta model* M we will require the following condition to be fulfilled: $\forall(e_1, e_2, t) \in Q : (F_c(e_1), F_c(e_2), t) \in R$ Applying this formalization to our example, we get six different concepts and four different relations (with their inverse):

$$C = \{Role, Application, Function, DataObject, Process, Service\}$$
$$T = \{use, usedBy, access, accessedBy, realize, realizedBy, assign, assignedBy\}$$
$$R = \{(Application, Service, realize), (Service, Application, realizedBy), (Role, Service, use), (Service, Role, usedBy), ...\}$$

Due to space limitations we do not list all triples of R here. Consider Fig. 1 for the other triples. The integration of this formalism in real applications is not straightforward. Additionally, to benefit from deduction and SPARQL [26] we translated it into an OWL2 ontology [15]. Given a meta model $M = (C, T, R)$ we first create an `owl:Class` for each c $\in C$. However, all classes have to be marked disjoint to each other, as components in a EA model can only be instance of one class (see definition of F_e). For each relation type $t \in T$ of the meta model, we create an `owl:ObjectProperty`. Every object property has to be linked with its corresponding inverse through the `owl:inverseOf` annotation. Finally the set of triples R is represented using the `rdfs:domain` and `rdfs:range` information. The primary idea is to set the domain c1 and range c2 of a object property o according to the triple $(c1, c2, o) \in R$. However, in EA it is possible to use one relationship type for several pairs of classes. These are the relationships $t \in T$ with $|\{(c_1, c_2)|(c_1, c_2, t) \in R\}| > 1$. In our example *usedBy* is used for the relation between *Service* and *Process* but also between *Service* and *Application*. According to [20] we decided to solve this problem with superclasses in order to have the reasoner work correctly. For the *usedBy* relationship, this would indicate a new superclass *ServiceUser*. For the relation types *access* and *realize* the same procedure will be applied. At least we have to implement the EA Model (E, T^*, Q, F) in the ontology. For each $e \in E$ an individual $\mathcal{I}$ will be created. Using the `rdf:type` assertion the respective type F_e is assigned to $\mathcal{I}$. For each relationship $(e_1, e_2, t) \in Q$ an object property assertion with relation type $t \in T$ is defined between the corresponding individuals for e_1 and e_2.

3.2 Analyzing the Enterprise Architecture

Having transformed an EA model into an ontology, semantic web technologies can be employed to gain benefits from the modeled information. The easiest way is the deduction of implicit knowledge using the reasoning capabilities. Often relationships are only modeled in one direction. A typical example is the *usedBy* relationship. For every application, the used services are known and modeled in the EA. But from the perspective of a service, not all applications that use it are known. In this case the inverse of each relationship can be deduced and provides further knowledge about the EA. Especially the *users* of a service are getting important, when it comes to decision about potential changes.

Using semantic web technologies it is possible to realize existing methods for the deduction of implicit dependencies. For example the composition operation proposed by [9] or the relation composition proposed by [24]. Such a relationship composition can be used for dependency analysis as proposed by the former authors, but also for generating landscape maps. A landscape map is a matrix that is used to visualize dependencies between EA elements [25]. Such a matrix is a common mean for EA management.

Another widespread method for EAM is the definition of reports and the calculation of specific measures. Reports can e.g. be a list of all applications assigned to a specific organization unit or all processes that use applications hosted on a specific server. [13] proposes a catalog of key performance indicators for EA management, including e.g. the *Application criticality ranking*. This measure is calculated using the following definition: *The number of applications with criticality rating available divided by the total number of applications.* Assuming that the required information for a report or measure is modeled in the EA, the calculation of those using SPARQL [26] is straightforward. Since it is a minor effort to define such SPARQL queries, it is no problem to specify them individually for each organization.

The re-implementation effort of an analysis increases with the complexity of the calculation routine. Examples for more complex analyses are the performance and cost analysis proposed by [8] or the different analyses proposed in [16]. These analyses are dependent on a specific meta model and adapting them to an existing EA initiative requires much effort. We propose the combination of SPARQL and reasoning for the specification of more complex analyses to enable their execution in an existing EA model with slight adaption effort. Therefore, the concepts required to perform the analysis have to be defined in an own *analysis ontology*. This ontology is the foundation for the specification of the analysis, either using SPARQL or through respective assertions in the ontology (and deduction). For the execution of the analysis in an existing EA model the *analysis ontology* has to be imported. It is also possible to import both the EA and the analysis ontology in a new one. Using mapping constructs like class, property and data equivalency or subclass definitions the analysis concepts are mapped to the EA concepts. Running the reasoner infers the axioms that are required to execute the analysis in the EA model. The former defined analysis can now be executed without further adaptions.

4 Implementation of Change Impact Analysis

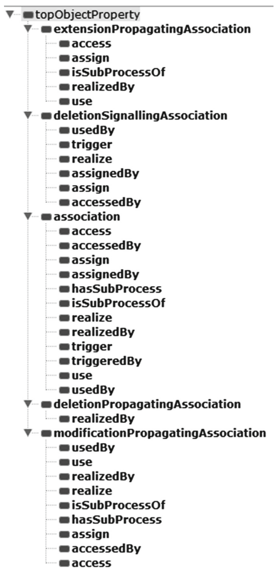
The generic approach in the previous section for the definition and adaption of complex analysis is now applied to change impact analysis. The foundation for the implementation of change impact analysis is the informal specification provided by de Boer et al. [2]. First we specify the required analysis semantics in an ontology and show how it can be integrated in an EA ontology (Sect. 4.1). Then we present an implementation approach using SPARQL in order to determine the effects of a change (Sect. 4.2). A second implementation approach uses the ability of the reasoner to deduce the change type of an element from the change semantics of the relationship (Sect. 4.3). We implemented both approaches using the ontology editor Protégé[1]. We did not implement a stand-alone application. As reasoner we used HermiT. The SPARQL queries were stored in a text file and executed in Protégé to execute the change impact analysis.

4.1 Defining and Integrating Change Semantics

The EA ontology created in Sect. 3.1 contains the EA relevant semantics. Defining the analysis based on this knowledge makes it difficult to execute it on other EA models with a different meta model. Therefore we define the required knowledge to perform change impact analysis in a separate ontology. This ontology contains information about the change semantics, i.e. concepts that indicate how changes made to a component affect the components that are in a relationship with it.

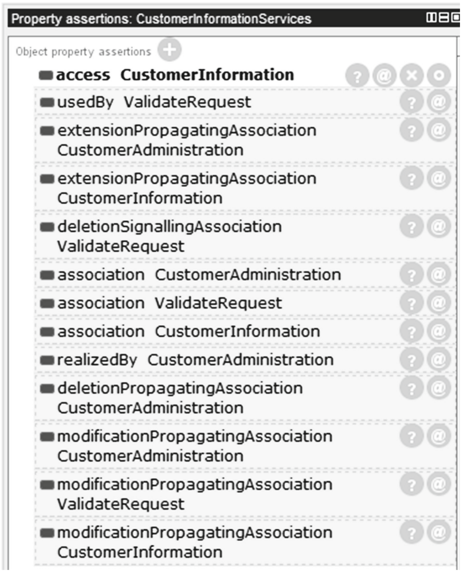
De Boer et al. define the change semantics in an EA model based on the type of change and the type of the relationship [2]. They differentiate between the change types *modification*, *extension*, *deletion* and *no change* (see Sect. 2). Depending on the type of relationship such a change will be *propagated* or *not propagated* along the relationship. It is also possible that a change is only *signaled* to the user, who has to decide about the actual propagation. The *signaling* is used, when the propagation is dependent from further aspects and not only the relationship type. For example the deletion of a service does not definitely imply the deletion of the realizing application. Nevertheless, there may be demand for action, which will be signaled to the user. According to these considerations, we know, how a change in the objects A resp. B will go through the model. If e.g. B has been changed, we have to – comparable to a depth first search – check all the relationships that link B with other objects for changes. If there is a relationship that propagates the change, for this element the same review has to be done. For each of the change types, we create a new `owl: ObjectProperty`. These are: *extensionPropagatingAssociation*, *modificationPropagatingAssociation*, *deletionPropagatingAssociation* and *deletionSignallingAssociation*. In order to use these change semantics in a specific EA ontology, the change concepts and the EA concepts have to be mapped to each other. All relation types $t \in T$ are defined as specializations of the type of changes they propagate. E.g. the relation A *realized by* B, will always propagate a respective change in B.

[1] A free, open-source ontology editor and framework. See http://protege.stanford.edu/.



- topObjectProperty
  - extensionPropagatingAssociation
    - access
    - assign
    - isSubProcessOf
    - realizedBy
    - use
  - deletionSignallingAssociation
    - usedBy
    - trigger
    - realize
    - assignedBy
    - assign
    - accessedBy
  - association
    - access
    - accessedBy
    - assign
    - assignedBy
    - hasSubProcess
    - isSubProcessOf
    - realize
    - realizedBy
    - trigger
    - triggeredBy
    - use
    - usedBy
  - deletionPropagatingAssociation
    - realizedBy
  - modificationPropagatingAssociation
    - usedBy
    - use
    - realizedBy
    - realize
    - isSubProcessOf
    - hasSubProcess
    - assign
    - accessedBy
    - access

Fig. 2. Hierarchical structure of the ObjectProperties according to their behavior towards a change.



Property assertions: CustomerInformationServices

Object property assertions

- **access CustomerInformation**
- usedBy ValidateRequest
- extensionPropagatingAssociation CustomerAdministration
- extensionPropagatingAssociation CustomerInformation
- deletionSignallingAssociation ValidateRequest
- association CustomerAdministration
- association ValidateRequest
- association CustomerInformation
- realizedBy CustomerAdministration
- deletionPropagatingAssociation CustomerAdministration
- modificationPropagatingAssociation CustomerAdministration
- modificationPropagatingAssociation ValidateRequest
- modificationPropagatingAssociation CustomerInformation

Fig. 3. ObjectProperties that have been inferred by the reasoner for the individual "CustomerInformationServices"

Following the object property *realizedBy* is a `owl:subPropertyOf` *extension-PropagatingAssociation*, *modificationPropagatingAssociation* and *deletionPropagatingAssociation*. The reasoner enables the deduction of the type of change propagation of a specific relationship, since the superior relation always applies implicitly. I.e consider an arbitrary object property p, the set of its superior object properties P_s (i.e. p is specialization of each $p_s \in P_s$) and two individuals $\mathcal{A}$ and $\mathcal{B}$:

$$\forall p_s \in P_s : \left(\mathcal{A} \xrightarrow{p} \mathcal{B}\right) \implies \left(\mathcal{A} \xrightarrow{p_s} \mathcal{B}\right) \tag{1}$$

An example structure of ObjectProperties can be found in Fig. 2. The inferred assertions by the reasoner for a specific ObjectProperty are shown in Fig. 3.

4.2 Impact Analysis with SPARQL

According to de Boer et al. [2], we want to perform a step-by-step analysis of our EA ontology. Given an individual $\mathcal{I}$ that is changing and the type of the change (extension, modification, deletion) we can calculate the impact of the change as described in pseudo code in Listing 1.1, with

- `changed_element` is the URI of the individual $\mathcal{I}$ that causes change analysis with change type c.

- `Properties(c)` is a function that maps a change type to its corresponding set of object properties.[2]
- `Visited` are the individuals of the ontology that have been visited. Initially `Visited` $= \emptyset$.
- `Result` $\subseteq I \times O$ is a set of tuples that contain an individual of the ontology and the object property of the relationship that has been found.
- `ToBeVisited` is the set of individuals that remain to be visited (initially it only contains `changed_element`).
- `p` is the individual that is currently being visited.
- `S` is the result of the SPARQL query.

Listing 1.1. Ontology-based dependency analysis in pseudo code.

```
 1 BEGIN DependencyAnalysis
 2
 3     ToBeVisited = {changed_element}.
 4     Visited = { }.
 5     Result = { }.
 6
 7     WHILE ToBeVisited IS NOT EMPTY.
 8         p = ToBeVisited.getElement().
 9
10         FOREACH property IN Properties(c)
11             S := SELECT ?object
12                  WHERE{  p  my:property  ?object  }.
13
14             FOREACH s IN S.
15                 INSERT TUPLE(s, property) INTO Result
16                     IF NOT EXISTS.
17             ENDFOREACH.
18
19             INSERT DISTINCT (S MINUS (Visited INTERSECT S))
20                 INTO ToBeVisited IF NOT EXISTS.
21
22             DELETE p FROM ToBeVisited.
23             INSERT p INTO Visited.
24         END FOREACH.
25
26        ENDWHILE.
27     RETURN Result.
28 END.
```

First we assign one of the individuals of the set `ToBeVisited` to `p` (l. 8). Then for each object property `property`, that needs to be considered for the change type c, we query the individuals that are related to `p` via `property` using SPARQL (ll. 10–12). The query result is stored into the set `S`. The individuals in `S` are added to the `Result` set (ll. 14–17), including the object property type `property`. Each element of `S`, that is not yet in the set `Visited`, is added to the set `ToBeVisited` (ll. 19, 20). Finally we remove `p` from `ToBeVisited` and store it in `Visited` (ll. 22, 23). We repeat these steps until there are no more individuals in `ToBeVisited`. All individuals that are affected by the change are now in the `Result` set, including their respective change propagation type. Therewith we can decide whether the individual is really affected, or – in case of a deletion – only needs to be signaled to the user.

[2] e.g. Change type *extension* is mapped to {extensionPropagatingAssociation} and *deletion* is mapped to {deletionPropagatingAssociation, deletionSignallingAssociation}.

According to Table 1 there are several cases, where there is only a signal for a possible impact propagated. Considering the following scenario:

$$\mathcal{I}_0 \xrightarrow{o_0} \mathcal{I}_1 \xrightarrow{o_1} \dots \xrightarrow{o_{m-1}} \mathcal{I}_m \xrightarrow{o_m} \mathcal{I}_{m+1} \xrightarrow{o_{m+1}} \dots \xrightarrow{o_{n-1}} \mathcal{I}_n \tag{2}$$

with $\forall i \in \{0, 1, ..., n\} : \nexists \mathcal{K} \in I : \mathcal{K} \xrightarrow{o} \mathcal{I}_i \wedge \mathcal{K} \neq \mathcal{I}_{i-1}$.

We assume that all relations o_i in the chain trigger only a signal about a potential change to the user and furthermore, that for all individuals with index $<m$ the user already decided that they should be changed, however for $\mathcal{I}_m$, the user decided that it is not affected by the change. Then all individuals with index $>m$ do not have to be signaled to the user (except when they could be affected by the change through other individuals). In order to realize this behavior in our algorithm Listing 1.1 we need to ask the user after line `8`, whether `p` should be changed or not – in case that `c` is signaling for all (`p`, `c`) $\in$ `Result`. If the user decides, that `p` does not need to be changed, then all tuples with `p` at their left position need to be removed from `Result` and `p` also needs to be removed from `ToBeVisited`. The algorithm can then continue with line `7`.

4.3 Impact Analysis with Defined Classes

As second alternative we propose the use of reasoning capabilities to deduce the effect of a change from the available information. Therefore we extend the analysis ontology with the classes *DeletedComponent*, *DeletionSignaledComponent*, *ExtendedComponent* and *ModifiedComponent*. In order to represent the external change that triggers the analysis, we add three data properties, *isDeleted*, *isExtended* and *isModified*. If an individual $\mathcal{I}$ of the ontology is affected by the change, the reasoner should deduce that it is instance of the respective change class. Given two individuals $\mathcal{I}_1$ and $\mathcal{I}_2$ with the relationship $\mathcal{I}_2 \xrightarrow{extensionPropagatingAssociation} \mathcal{I}_1$, the reasoner should deduce, that if $\mathcal{I}_2$ is extended, also $\mathcal{I}_1$ has to be extended (indicated by the assertion `owl:type :ExtendedComponent`). To enable this deduction, we add inverse object properties for all kinds of change propagating properties. In the example this is the *extensionReceivingAssociation*. Additionally also a *modificationReceivingAssociaten*, *deletionSignalReceveivingAssociation* and a *deletionReceivingAssociation* are added. Each of them is enriched with the assertion about the respective inverse propagating object property. For each change class we are now able to specify an equivalent class expression. Those class definitions are shown in Listing 1.2 using the Manchester OWL Syntax. For the class *ExtendedComponent* this says: An individual that has `owl:type :ModelingComponent` and that has an `:extensionReceivingAssociation` with another `:ExtendedComponent` has also the type `:ExtendedComponent`. Additionally an individual is member of this class, if it is annotated with the data propery `:isExtended true`. The expressions for the classes *DeletedComponent* and *ModifiedComponent* are defined in the same way using the corresponding receiving object property. An individual is a member of the class *DeletionSignaledComponent*, if `owl:type: ModelingComponent` can be inferred and if it has `a: deletionSignalReceivingAssociation` from a `:DeletedComponent`. If the user decides that a *DeletionSignaledComponent* needs to be deleted, he has to assign the *isDeleted* data property to it. This way, it is ensured that the *DeletionSignaledComponent*s are kept minimal, and further assertions are only made if the user decides about the deletion.

Listing 1.2. Class definitions in Manchester OWL Syntax

```
Class: cs:ModifiedComponent
    EquivalentTo:
         ((cs:modificationReceivingAssociation some cs:ModifiedComponent)
         or (cs:isModified some {true}))

Class: cs:ExtendedComponent
    EquivalentTo:
         ((cs:extensionReceivingAssociation some cs:ExtendedComponent)
         or (cs:isExtended some {true}))

Class: cs:DeletedComponent
    EquivalentTo:
         ((cs:deletionReceivingAssociation some cs:deletedComponent)
         or (cs:isDeleted some {true}))


Class: cs:DeletionSignaledComponent
    EquivalentTo:
        (cs:deletionSignalReceivingAssociation some cs:deletedComponent)
```

For the execution of this analysis in a specific EA model, the relationship types t of the EA ontology have to be mapped to the respective change propagation properties (analog to the procedure in Sect. 4.2. The change receiving properties can be deduced from this information by the reasoner. After asserting the actual change using the data properties, the reasoner can be synchronized and deduces the membership of the individuals in the four change classes. The result can be retrieved with a simple SPARQL query.

5 Case Study

We evaluated our approach using the PEIS (Personal Environmental Information System) case study from the ENVIROFI project[3]. A description of the use case can be found in [10]. This case study is a good representative to validate our approach. Since the meta model is based on ArchiMate, a popular EA modeling technique, the typical EA elements and dependencies are covered. Additionally the size of the use case is large enough to be able to test the propagation of change effects, whereas humans can still retrieve the actual effect of a change to be able to compare the results. An overview of the EA model of PEIS is shown in Fig. 4. The meta model for PEIS was already introduced in Sect. 3.1 in Fig. 1. According to the method proposed in Sect. 3.1 we manually transformed the EA model into an ontology. Using the reasoning capabilities we inferred the return directions of each relationship. As expected, reports like the number of uses of an application can be defined in a fast and easy way using SPARQL. Finally we applied our approaches proposed in Sects. 4.2 and 4.3 to the PEIS use case.

[3] Environmental Observation Web and its Service Applications within the Future Internet: www.envirofi.eu.

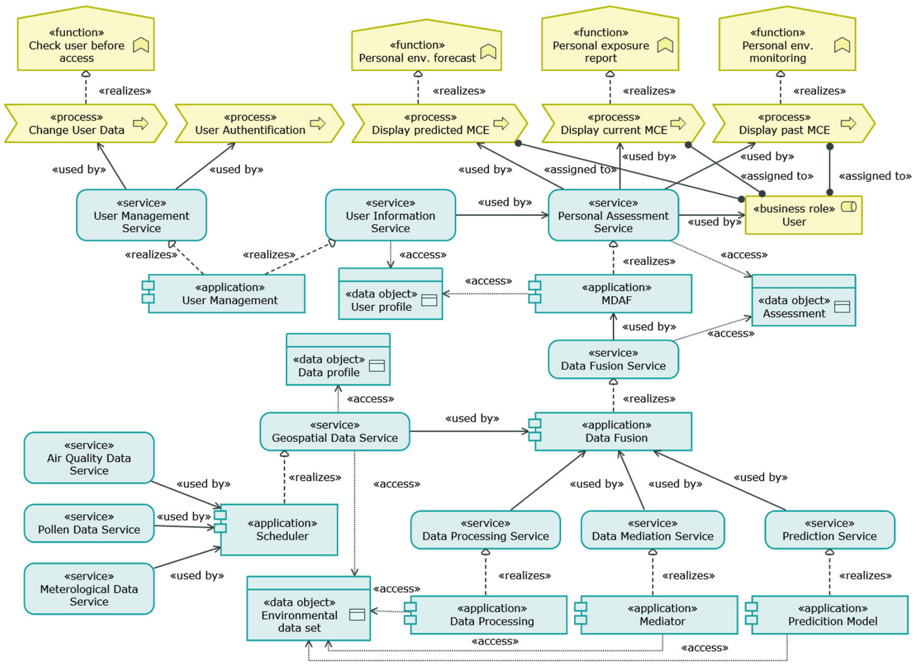

«function» Check user before access
«function» Personal env. forecast
«function» Personal exposure report
«function» Personal env. monitoring
«realizes»
«process» Change User Data
«process» User Authentification
«process» Display predicted MCE
«process» Display current MCE
«process» Display past MCE
«used by»
«assigned to»
«service» User Management Service
«service» User Information Service
«service» Personal Assessment Service
«business role» User
«application» User Management
«data object» User profile
«application» MDAF
«data object» Assessment
«access»
«data object» Data profile
«service» Data Fusion Service
«service» Geospatial Data Service
«application» Data Fusion
«service» Air Quality Data Service
«service» Pollen Data Service
«service» Meterological Data Service
«application» Scheduler
«service» Data Processing Service
«service» Data Mediation Service
«service» Prediction Service
«data object» Environmental data set
«application» Data Processing
«application» Mediator
«application» Predicition Model

Fig. 4. EA model of the PEIS use case

Table 1 contains an overview over the different relation types and the change semantics we assigned to them. The left part of the table defines the change semantic in the original direction, e.g. for A accesses B; the inverse relationship is defined in the right part of the table. For a relation $r\colon A \xrightarrow{r} B$ from component A to component B, the change semantic is defined in Table 1 with:

- **p** means that the change is **propagated** (modification is propagated as modification, deletion as deletion and extension as extension) to the associated component,
- **n** means that the change is **not propagated** and
- **s** means that the change is **signaled** to the user, who has to decide whether or not the change shall be propagated.

If there is a change propagation, we added the respective *subObjectProperty* assertion. For example, for the relation $A \xrightarrow{access} B$ only a modification and an extension is propagated. Following the assertion `:access owl:subPropertyOf :extensionPropagatingAssociation` and the assertion `:access owl:subPropertyOf :modificationPropagatingAssociation` have to be added. This is done for all cells in Table 1.

For the evaluation of the change impact analysis we defined several test cases. Every test case contains a specific change to a component as well as the components that we expect to be affected by the change. The test cases are defined in manner, that every change type and every change propagation type is covered at least once. We also considered testing the change propagation from

Table 1. Behavior of relation types towards a change $A \xrightarrow{r} B$.

Relation r	when A is changed			when B is changed		
	delete	extend	modify	delete	extend	modify
Access	n	n	n	s	n	p
Assigned to	s	p	p	s	n	n
Used by	s	n	p	s	n	n
Realizes	p	p	p	s	n	p

the business layer to the lower ones as well as from the infrastructure layer to the upper ones. The test cases are executed with both of our approaches Our evaluation was performed manually using Protégé, the reasoner HermiT, and the textual SPARQL query interface it provides. In the following we present two of our test scenarios with their respective results.

As first test case we executed the analysis for the change *Deletion of Application Mediator* and got the following result: The *Data Mediation Service* has to be deleted, too; the *Data Fusion* is probably affected by the deletion and thus signaled to the user. In this case the user decides that *Data Fusion* is to be deleted too, he has to add the data property *isDeleted* to this component. Synchronizing the reasoner again deduces the deletion of the *Data Fusion Service*. In this case the retrieved results of the analysis are meaningful to the user and a good approximation of the potential change effect.

The second test case encompasses the *Modification of Personal Assessment Service*. This change is propagated to the processes *Display predicted, current* and *past MCE* as well as their realizing functions. Also the *User* is affected by the modification. But also in the other direction the change is propagated to the realizing application *MDAF*. In this case the result set is greater than in the previous scenario. This might indicate that a change in this service can have a high impact on the business outcome, especially since many elements from the business layer are affected. However, it may also indicate that the used change impact rules in Table 1 are too weak and thus the result set is too large.

Summarizing, we observed that the results of an test case are identical for both approaches. We also realized that we had a strong match between the components that we had expected and the results returned by the approaches. In both cases it is meaningful to compare the calculated impact with the actual impact and if required, adapt the change impact rules. Nevertheless we were able to execute the change impact analysis on the PEIS EA model with only restricted amount of effort for the integration of the analysis semantics.

6 Conclusion

In this paper we proposed semantic web technologies for the enterprise architecture domain. We specified how to transform an EA model into an ontology and outlined how to add value to those data. Simple reports and measures can be directly implemented using SPARQL individually for each EA initiative. Due to the higher effort for more complex analysis, there is a need for methods to re-use the analysis definitions. We propose the separate definition of an *analysis ontology* and an *EA ontology*. The *EA ontology* contains the organization specific definition of EA concepts and their relationships as well as the concrete instance elements. The *analysis ontology* describes analysis specific concepts, which are used for the specification of the actual analysis. In order to execute an analysis on an *EA ontology*, the analysis concepts have to be mapped to the *EA ontology*.

We demonstrated the use of semantic web technologies in the EA domain using the PEIS case study. The EA, formerly not modeled in an ontology, was transformed into an OWL2 ontology. Typical EA reports and measures could be re-built using SPARQL queries very quickly. As complex analysis we redefined an existing specification of change impact analysis. We integrated the defined analysis concepts in the established EA ontology and executed the analysis. Although the definition of the analysis ontology and the execution specification was time consuming, the integration into the EA model and the final execution was slight. Through an iterative refinement of the mapping, the precision of the analysis results could be improved.

References

1. Chen, W., Hess, C., Langermeier, M., Stuelpnagel, J., Diefenthaler, P.: Semantic enterprise architecture management. In: Proceedings of the 15th International Conference on Enterprise Information Systems (2013)
2. de Boer, F., Bonsangue, M., Groenewegen, L.P.J, Stam, A.W., Stevens, S., Van Der Torre, L.: Change impact analysis of enterprise architectures. In: IEEE International Conference on Information Reuse and Integration, pp. 177–181 (2005)
3. de Boer, F.S., Bonsangue, M.M., Jacob, J., Stam, A., Van der Torre, L.: Enterprise architecture analysis with XML. In: Proceedings of the 38th Annual Hawaii International Conference on System Sciences, HICSS 2005. IEEE (2005)
4. Di Francescomarino, C., Corcoglioniti, F., Dragoni, M., Bertoli, P., Tiella, R., Ghidini, C., Nori, M., Pistore, M.: Semantic-based process analysis. In: Mika, P., Tudorache, T., Bernstein, A., Welty, C., Knoblock, C., Vrandečić, D., Groth, P., Noy, N., Janowicz, K., Goble, C. (eds.) ISWC 2014, Part II. LNCS, vol. 8797, pp. 228–243. Springer, Heidelberg (2014)
5. Gerber, A., Kotz, P., van der Merwe, A.: Towards the formalisation of the TOGAF content metamodel using ontologies. In: Proceedings of the 12th International Conference on Enterprise Information Systems (ICEIS 2010) (2010)
6. Holschke, O., Närman, P., Flores, W.R., Eriksson, E., Schönherr, M.: Using enterprise architecture models and bayesian belief networks for failure impact analysis. In: Feuerlicht, G., Lamersdorf, W. (eds.) ICSOC 2008. LNCS, vol. 5472, pp. 339–350. Springer, Heidelberg (2009)
7. Johnson, P., Ullberg, J., Buschle, M., Franke, U., Shahzad, K.: P^{2}AMF: predictive, probabilistic architecture modeling framework. In: van Sinderen, M., Oude Luttighuis, P., Folmer, E., Bosems, S. (eds.) IWEI 2013. LNBIP, vol. 144, pp. 104–117. Springer, Heidelberg (2013)
8. Jonkers, H., Iacob, M.-E.: Performance and cost analysis of service-oriented enterprise architectures. In: Gunasekaran, A. (ed.) Global Implications of Modern Enterprise Information Systems: Technologies and Applications. IGI Global, Hershey (2009)

9. Kurpjuweit, S., Aier, S.: Ein allgemeiner Ansatz zur Ableitung von Abhängigkeitsanalysen auf Unternehmensarchitekturmodellen. In: Wirtschaftsinformatik (1), pp. 129–138. Citeseer (2009)
10. Langermeier, M.: A model-driven approach for open distributed systems. Technical report 2013–03, University of Augsburg (2013)
11. Lankhorst, M.: Enterprise Architecture at Work. Springer-Verlag Berlin and Heidelberg GmbH & Co. KG, Berlin (2012)
12. Lucke, C., Krell, S., Lechner, U.: Critical issues in enterprise architecting - a literature review. In: Proceedings of the 16th Americas Conference on Information Systems (2010)
13. Matthes, F., Monahov, I., Schneider, A., Schulz, C.: EAM KPI catalog v 1.0. Technical report, Technical University Munich (2012)
14. MID GmbH. MID Innovator for Enterprise Architects (2015). http://www.mid.de/produkte/innovator-enterprise-modeling.html. Accessed 15 January 2015
15. Motik, B., Patel-Schneider, P.F., Parsia, B.: OWL 2 Web Ontology Language: Structural Specification and Functional-Style Syntax, 2nd edn. (2012). http://www.w3.org/TR/owl2-syntax. Accessed 13 January 2015
16. Närman, P., Buschle, M., Ekstedt, M.: An enterprise architecture framework for multi-attribute information systems analysis. Softw. Syst. Model. **13**(3), 1085–1116 (2014)
17. Niemann, K.: From Enterprise Architecture to IT Governance. Springer, Heidelberg (2006)
18. Software AG. Alfabet Enterprise Architecture Management (2015). http://www.softwareag.com/corporate/products/aris_alfabet/ea/overview/default.asp. Accessed 15 January 2015
19. Sunkle, S., Kulkarni, V., Roychoudhury, S.: Analyzing enterprise models using enterprise architecture-based ontology. In: Moreira, A., Schätz, B., Gray, J., Vallecillo, A., Clarke, P. (eds.) MODELS 2013. LNCS, vol. 8107, pp. 622–638. Springer, Heidelberg (2013)
20. Russ, T.: (protege-owl) Specifying domain and range in object properties (2010). https://mailman.stanford.edu/pipermail/protege-owl/2010-November/015509.html. Accessed 07 January 2015
21. Tang, A., Nicholson, A., Jin, Y., Han, J.: Using bayesian belief networks for change impact analysis in architecture design. J. Syst. Softw. **80**(1), 127–148 (2007)
22. The Open Group. TOGAF Version 9.1. Van Haren Publishing (2011)
23. The Open Group. ArchiMate 2.0 specification: Open Group Standard. Van Haren Publishing (2012)
24. van Buuren, R., Jonkers, H., Iacob, M.-E., Strating, P.: Composition of relations in enterprise architecture models. In: Ehrig, H., Engels, G., Parisi-Presicce, F., Rozenberg, G. (eds.) ICGT 2004. LNCS, vol. 3256, pp. 39–53. Springer, Heidelberg (2004)
25. van der Torre, L.W.N., Lankhorst, M.M., ter Doest, H., Campschroer, J.T.P., Arbab, F.: Landscape maps for enterprise architectures. In: Martinez, F.H., Pohl, K. (eds.) CAiSE 2006. LNCS, vol. 4001, pp. 351–366. Springer, Heidelberg (2006)
26. W3C SPARQL Working Group. SPARQL 1.1 Overview (2013). http://www.w3.org/TR/sparql11-overview. Accessed 13 January 2015
27. Zachman, J.A.: A framework for information architecture. IBM Syst. J. **26**(3), 276–295 (1987)