A Business Process-centered Approach for Modeling Enterprise Architectures

Torben Schreiter, Guido Laures

Hasso-Plattner-Institute for IT Systems Engineering,
University of Potsdam, Germany
{torben.schreiter, guido.laures}@hpi.uni-potsdam.de

Abstract: Mastering an Enterprise Architecture is crucial for the entire enterprise to operate most efficiently. However, there is a lack of appropriate means to appropriately visualize and communicate traceability from business processes to other enterprise architectural entities such as e.g. the supporting IT infrastructure. Therefore, systematic in-depth analysis of the entire Enterprise Architecture is barely feasible.

This paper aims to introduce a semi-formal and business process-centered modeling approach for capturing complex Enterprise Architectures based on the well-established modeling techniques *BPMN* (Business Process Modeling Notation) and *UML* (Unified Modeling Language). A meta model formalizing the approach is presented.

It is feasible to apply customizable views tailored to stakeholder-specific concerns providing standardized levels of abstraction to the model in order to improve communication of the big picture of an Enterprise Architecture. A case study illustrates the approach.

1 Introduction

Today's enterprises have to stay competitive with a number of competitors all over the world. Streamlined business processes are a key factor leading to improved competitiveness. However, these processes are embedded in an entire Enterprise Architecture. A proper enactment of the processes is still highly dependent on complex structures in the Enterprise Architecture. This paper presents a business process-centered modeling approach enabling the effective communication of an Enterprise Architecture to different stakeholders.

When speaking of *Enterprise Architectures* (EA), we do not mean architectures of enterprise application systems, which more precisely can be refered to as Enterprise Application Architectures. Indeed, the proper definition of an EA seems useful in order to facilitate a common understanding for the subsequent sections of this paper. The following definition is slightly adapted from [Wi06].

Definition 1. *An* Enterprise Architecture *is the current or future structuring of an organization's processes, information systems, personnel and organizational sub-units, so that they align with the organization's core goals and strategic direction.*

As an enterprise's core competence is based on its EA, the whole enterprise is sentenced to function inefficiently if the enterprise's architecture is inefficient. Of course, enterprise architectures, especially those of large companies, may become extremely complex. Mastering the EA and its complexity is a non-trivial challenge. Therefore, it is our concern to provide a means that can be utilized for the overall improvement of an EA.

The proposed concept for *Enterprise Architecture Modeling* should, consequently, cover various aspects that are important for, firstly, communication of an EA and, secondly, improvement of the EA. The conceptual basis for this is a business process-centered semi-formal modeling approach. The supposed modeling technique enables the visualization of a big picture of the respective EA. Considering the potential complexity of an EA, it seems not feasible to provide this big picture using a single diagram. Instead, a set of stakeholder-specific high level diagrams should be provided.

Modeling an EA, of course, includes IT infrastructural aspects that business tasks rely on. The proposed modeling technique additionally covers organizational aspects that might be necessary for fulfillment of a certain business task. Maintainability and extensibility of the supporting IT infrastructure as well as the identification of business processes that are particularly manpower-intensive are interesting aspects when planning or evaluating an EA. Our modeling technique enables further investigation on issues like these.

This paper is organized as follows: Section 2 will introduce the basic concepts and key objectives that underly the proposed modeling approach. The formal basis of the approach in the form of a MOF-based meta model is presented in the third section. Next, a detailed case study demonstrates a practical example using the modeling approach in the concrete business domain of an insurance company. Finally, a brief survey of related work as well as a conclusion is provided.

2 Conception of the Modeling Approach

2.1 Organizational and IT Infrastructural Aspects

Any modeling technique is based on a number of modeling elements. These elements are then utilized by modelers in order to describe certain aspects of a system of any kind.

When modeling an EA, two different kinds of aspects might be part of a model. These are either *organizational aspects* or *IT infrastructural aspects* of an EA. Thus, the set of entities that are of interest for an EA modeling approach can, in general, be assigned to one of the two kinds of aspects.

Entities such as *business processes*, *organizational (sub-)units*, *entire companies*, a company's *sites* or *roles* cover different organizational aspects of an EA.

On the other hand, e.g. *enterprise applications*, *web services*, *proprietary interfaces*, *business process engines*, *EAI systems* (Enterprise Application Integration), *adapters*, *application servers* or other kinds of servers cover IT infrastructural aspects of an EA. Note that the provided lists are *not* exhaustive.

2.2 Layering Structure

We would like to introduce a layered approach in order to describe an EA. More precisely, a view on an EA model consists of (at most) five distinct layers.

Role layer. The role layer contains organizational roles in an EA. The roles can be required for a particular business task in order to be completed. Roles of an EA may include *customer*, *head of department*, *accounting clerk* amongst others. Additionally, the roles might e.g. be assigned to a department or other (external) companies.

Presentation layer. All IT infrastructural systems related to presentation of digital content to users performing the tasks belonging to a particular role are visualized in the presentation layer. Examples for elements of this layer include different kinds of *portals*, (proprietary) *rich clients* as well as e.g. e-mail clients.

Business process layer. This layer contains elements like entire *business process models*, *process instances* or *tasks* of a process model. Business process models on this layer provide the definition for enactment by a business process engine.

Service layer. The service layer visualizes the service landscape of an EA. Additionally, systems related to the accessability of the services are part of the service layer. These systems are technically inevitable to support the provision of services. However, they do not implement any business functionality. An *application server* utilized for exposure of *web services* set up during service enablement of legacy systems can be named as an example for this.

Please note that even though the layer is named *service* layer, this does not imply that the IT infrastructure necessarily has to be based on the concept of a service-oriented architecture (SOA) as described in [Bu00]. Thus, the service layer might also contain elements like *proprietary interfaces* of legacy backend systems.

Backend layer. Software systems implementing primary business functionality as well as supporting systems are part of the backend layer. E.g. ERP systems (Enterprise Resource Planning), EAI solutions, and relational database management systems may be part of a particular EA.

The business process layer is the *central layer* since we would like to facilitate traceability to concrete business functionality at any time. This layer is modeled using elements taken from the *Business Process Modeling Notation* (BPMN). The remainder of the layers is based on elements taken from UML component diagrams. Please refer to section 3 for further details.

2.3 Dependency Structure

Model elements, and therefore the layers containing these elements, are interconnected by *dependencies*. More precisely, different entities of an EA might be dependent on other entities of the EA and, therefore, require these for fulfillment of a certain business task.

Please find below a definition, which is valid for dependencies of enterprise architectural entities.

Definition 2. *A* dependency $D = (E_1, E_2)$ *exists between two entities* E_1 *and* E_2 *of an Enterprise Architecture if and only if the functionality of* E_1 *depends on the proper functioning of* E_2.

An exemplary dependency might exist between a business task of a business process and a role, stating the fact that human interaction is necessary to fulfill this particular task.

It is important to notice, that the complex interdependent structure of an EA model provides very detailed information on how entities of the EA are interrelated. Moreover, a key property of dependencies is their *transitivity*.

Theorem 1. *Dependencies are* transitive*. This means that if there exists a dependency* $D_1 = (E_1, E_2)$ *and a dependency* $D_2 = (E_2, E_3)$*, then there is also a dependency* $D_3 = (E_1, E_3)$.

Not only due to modern multi-tier architectures of supporting enterprise IT systems, transitiveness of dependencies is, practically, a very common pattern in almost any EA. If, for instance, a business task is performed by invocation of a composite web service, there is, first of all, a dependency between the task and this particular service (the task is dependent on the service). The composite web service is dependent on a number of other web services. And these are again dependent on some backend systems providing the functionality that is exposed by the web services. The backend systems might again be dependent on other backend systems such as a database management system. Because of the transitivity of dependencies, the original business task invoking a web service is also dependent on *all* the other entities involved in the provisioning of this service.

When elaborating further on the example of web services, independence of interface description and its implementation, often, is an important aspect. Indeed, the provided functionality of the service is what matters, regardless of how it is implemented. Nevertheless, when evaluating and analyzing an EA, we want to be able to trace dependencies to as many supporting entities/systems as possible. Naturally, this is mostly impossible when using web services from *external* providers. But still, we are able to derive a dependency from the external web service to its provider (as an organization) since the provider is responsible for the proper functioning of its IT systems. Consequently, the business task relying on this external service is dependent on the external provider.

2.4 Annotation of Enterprise Architectural Entities

Entities that are part of an EA sometimes have several properties or constraints that might be advantageous to be aware of. In an EA model we would like to be able to capture these properties and constraints appropriately. We propose the annotation of model elements as a means to visualize these aspects. Therefore, annotations enable a more detailed description of the EA.

There can be different types of annotations. Very few annotations are implicit like branch-

ing conditions of sequence flows in a business process model for instance. Some other annotations might have to be defined manually such as e.g. the maximum duration of a business task. And then, it is even feasible that some annotations can, to a certain extent, be automatically derived by supporting tools.

Let us consider an exemplary monitoring framework for a business process engine, which is able to provide statistical information on the enactment of processes. If there is a branch in the business process model, normally, the alternative flows are differentiated by conditions. The conditions are now seen as annotations of the sequence flows (arcs) to the next activity. Additionally, the monitoring framework provides information on the probabilities of each branch alternative as a percentage. This would be the probability for taking this particular alternative based on the preceding enactment history of the process. Of course, this can be visualized as another annotation of the sequence flow.

It is an important characteristic of annotations, that they can be transitively passed on to other annotatable elements. Figure 1 depicts the previously introduced example for branching probabilities. In addition, there is a second, nested branch shown. The alternatives again are annotated with condition and probability. *Task 1a* is additionally annotated with a maximum duration of 120 seconds. This task is now dependent on some other entity. All previous annotations are passed on to the dependency. Consequently, the dependency is only relevant if the branch is chosen and this, again, only occurs in 1% of all cases. Please note that the probabilities have been accumulated since a basic pass-on would not be sufficient. The maximum duration has to be handled with care because the duration might be reasonable if there is a single dependency to e.g. a web service. However, if there are multiple dependencies to various types of entities, it might not be reasonable to pass this annotation to any of the dependencies at all.

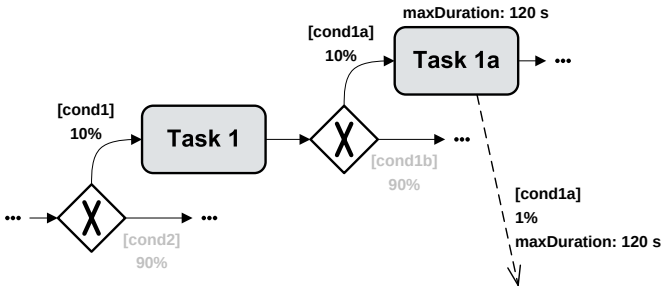

Figure 1: Example for transitivity of annotations.

Additionally, the annotation of *redundancy information* is a very practical example for annotation of dependencies. It is very common that important IT systems such as e.g. application servers are held redundant in order to compensate failure of a machine. If simply denoting a dependency from an entity to each of the redundant machines, we would, according to definition 2, semantically define that the entity is dependent on *each* of these machines. Of course, we would prefer to express that the entity is only dependent on one arbitrary of the machines instead of being dependent on all of them. This is a typical use case for a redundancy annotation.

2.5 Customizable Views

As we declared human communication of the EA to different stakeholders to be a main objective of the proposed modeling approach, it is desirable to satisfy the specific needs of these stakeholders of an EA. We suggest the facility to create customizable views providing precisely defined levels of abstraction. Each view is intended to serve a particular purpose tailored to the needs a stakeholder might have.

One stakeholder towards an EA is possibly a manager. Another stakeholder could be a system administrator. Both have very different attitudes towards the EA. The manager is primarily interested in streamlined business processes whereas the system administrator is concerned with the maintenance of the supporting IT systems. Accordingly, the visualizations should show the aspects that are of interest for the specific person.

Since a number of views can be seen as applicable to different enterprise's architectures, it is viable to provide this generic set as predefined views. Additionally, it should be possible to enable custom view definitions. See section 3 for a formal definition of views as well as section 4 for a view definition example.

A custom view, ideally, only shows the subset of enterprise architectural entities that are of interest for the particular stakeholder it is intended for. The transitivity of dependencies and annotations enables the derivation of implicit dependencies and annotations that exist even though some intermediate entities may not be shown in a particular view. If a business process is e.g. dependent on a number of services and these services are, again, dependent on some backend system, it is possible to derive a dependency from the process to the backend system, even if the whole service layer is undisplayed.

3 The Meta Model

This section aims at formalizing the proposed approach by introducing a meta model. It should be based on the *Meta Object Facility* (MOF) defined by the Object Management Group (OMG) since MOF already provides all concepts needed for definition of a meta model. Please refer to [Ob06a] for further information on the Meta Object Facility. Our modeling approach is based on the well-established modeling techniques *UML* (*Unified Modeling Language*), as specified in [Ob05a] and [Ob05b], and *BPMN* (*Business Process Modeling Notation*), which is specified in [Ob06b].

This paper does not intend to provide a complete specification of the proposed modeling approach. The meta model is rather to be seen as a basic formal definition of how to relate the diagram types UML and BPMN for possible future research on the field of EA modeling.

All layers except for the *Business process layer* are based on UML Component diagrams. As shown in figure 2, the elements of each layer are specializations of the meta class *Component* taken from [Ob05b]. This is advantageous, because we then inherit all properties from the UML meta class component, but still are able to distinguish between different

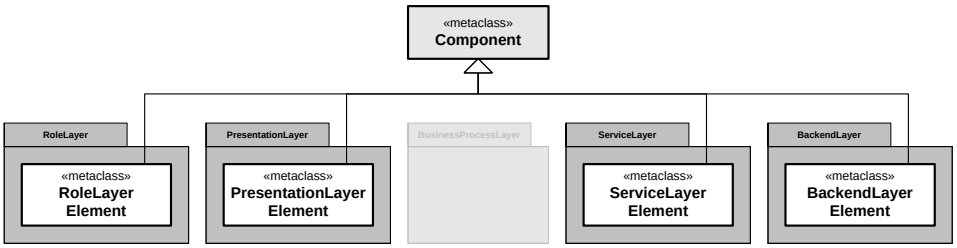

Figure 2: Layer elements based on UML Components.

layer's elements. This, again, is necessary for formal view definitions, as described in section 3.2.

When utilizing UML Components for our purposes, we do not exactly interpret them the way it is stated by the OMG. [Ob05b] describes UML Components as follows:

> [...] In particular, the [Components] package specifies a component as a modular unit with well-defined interfaces that is replaceable within its environment. [...]
>
> [...] The Components package supports the specification of both logical components (e.g., business components, process components) and physical components (e.g., EJB components, CORBA components, COM+ and .NET components, WSDL components, etc.) [...]

Our interpretation of UML Components is more abstract. We propose to model (human) roles and IT systems like application servers as specializations of Components, for instance. In our case, we are more interested in the definition and syntax of Component diagrams rather than in the methodology and intention of UML. Furthermore, we only define very basic elements for these four layers. Currently, further specializations of e.g. the meta class *BackendLayerElement* in form of other meta classes like *Adapter* or *DatabaseManagementSystem* is not intended. However, staying this generic does not pose any difficulty since both the organizational structure as well as the IT landscape of a particular enterprise might consist of a wide variety of elements. Therefore, the elements to be used should not be restricted to a limited set of specific elements but the definition should, instead, be rather abstract in order to be generic enough.

3.1 Annotations and Dependencies

Annotation of elements is accomplished as depicted in figure 3. There is a general, abstract meta class *Annotation* which is always related to at least one *NamedElement*[1]. Furthermore, each *Annotation* may be related to a number of other annotations due to the

[1] *NamedElement* is the meta class taken from UML, we will utilize as base class for all elements of the meta model. Please note that also *Component* is a *NamedElement*.

property of transitivity. This set is derived from the model.

Additionally, there are two specializations of *Annotation*: *DependencyAnnotation* and *BusinessProcessAnnotation*. As stated in section 2.3, the idea of elements is that they can be dependent on other elements. Thus, we relate a *Dependency* (defined in [Ob05b]) to an arbitrary number (including 0) of *DependencyAnnotations*.

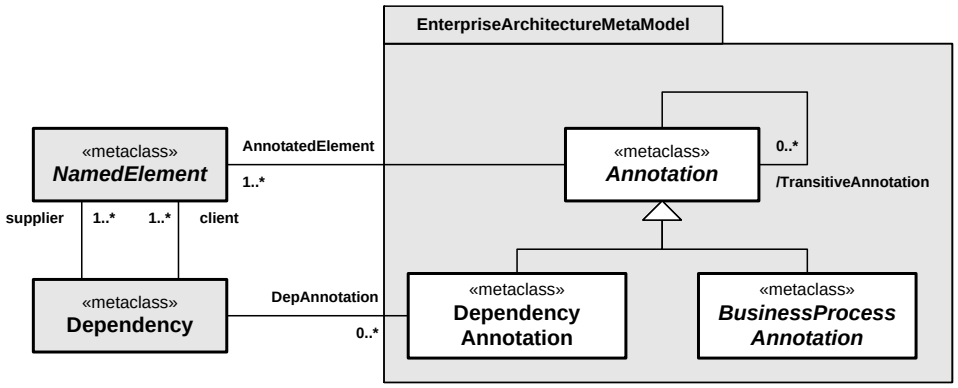

Figure 3: Definition of meta classes related to annotation of elements.

A *Dependency* is defined as an *n:m* relation between *NamedElements* in [Ob05b]. This, at first glance, seems to be not completely aligned with the provided definition in section 2.3. We defined a dependency as binary relation between enterprise architectural entities. However, the *n:m* relation for dependencies defined by [Ob05b] can be seen as an aggregation of multiple dependencies into one modeling element.

Moreover, we introduced a redundancy annotation for dependencies. This can be seen as a special case. Multiple dependencies to different elements annotated as redundant are rather to be seen as a set of dependencies of which only a subset (whose cardinality often equals 1) is necessary for proper functioning.

As we motivated before, the annotation of some elements in the *Business process layer* is desired as well. These annotatable elements are:

- *Activities* (such as tasks and sub-processes)
- *Branching sequence flows* (outgoing arcs of (X)OR gateways)
- *Business processes* (as a whole)

Elements in the Business process layer are based on BPMN. Please note that the current OMG-specification does not provide a formal meta model by itself. Only attributes and types of these attributes are defined in [Ob06b]. However, it makes sense to define at least the most important elements as interrelated meta classes and integrate these into the meta model. Original attribute specifications are, of course, considered.

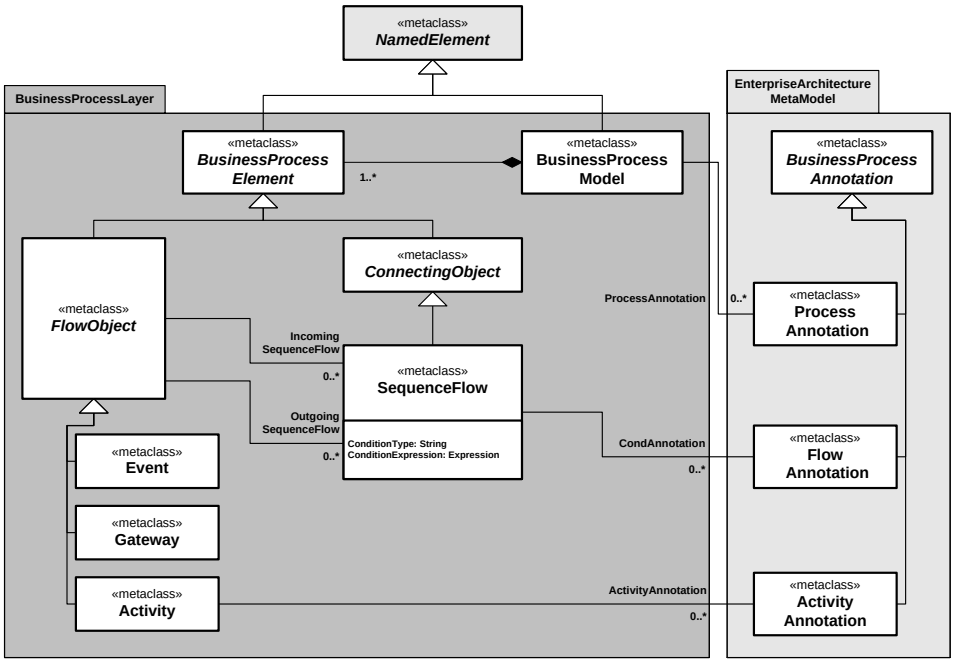

Figure 4: Meta classes for BPMN key elements and relation to annotation.

Figure 4 shows that there are basically two types of elements in the Business process layer. These are *BusinessProcessElements* and *BusinessProcessModels*. A *BusinessProcessModel*, ideally, contains a number of elements. Interdependency of the elements is facilitated due to the specialization of the meta class *NamedElement*. As defined by [Ob05b], the meta class *Dependency* relates any *NamedElements*.

The abstract meta class *BusinessProcessElement* is specialized into *FlowObjects* and *ConnectingObjects*. These meta classes are directly appendant to the corresponding concepts of BPMN. A *FlowObject* can be an *Event*, *Gateway*, or an *Activity*. A *ConnectingObject* can e.g. be a *SequenceFlow*.

Furthermore, there are three different *BusinessProcessAnnotations* and their respective associations to elements of the Business process layer shown. An *Activity* might be annotated with an *ActivityAnnotation*. This is similar for *BusinessProcessModel* and *ProcessAnnotation*. But then it is slightly different in case of the *FlowAnnotation*. [Ob06b] defines the attributes *ConditionType* and *ConditionExpression* enabling to distinguish whether a *SequenceFlow* contains a condition or not. Since simple sequence flows do not carry an additional meaning and it is, therefore, not reasonable to annotate these, we have to restrain the use of a *FlowAnnotation* to those *SequenceFlows* carrying anything but the value *"None"* for the attribute *ConditionType*.

3.2 View Definition

In section 2.5 we described the use of customizable views on the EA model. Now, the mechanisms for defining these custom views are introduced.

The definition of views is based on the facility (provided by UML) to define Profiles and apply these on diagrams [Ob05b]. A *View*, thus, is a specialization of the meta class *Profile* as depicted in figure 5.

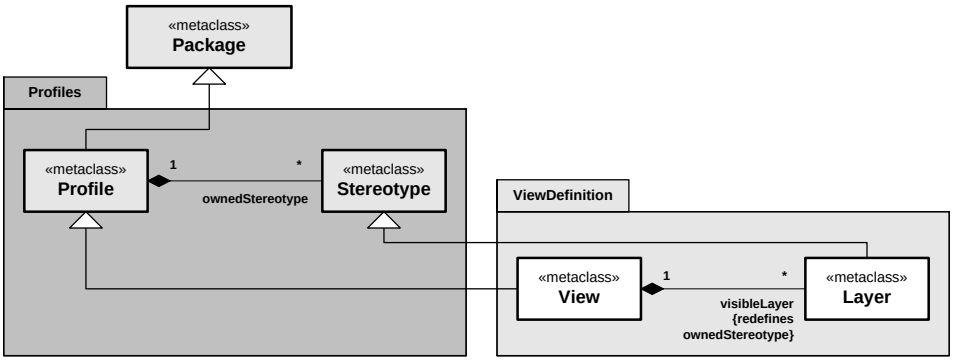

Figure 5: Definition of stakeholder-specific views.

Originally, a UML Profile contains a number of *Stereotypes*. These Stereotypes can then be applied to meta classes. Similarly, a *View* contains *Layers* in our case. Usually, there should be five or less distinct layers used for the definition of a view[2]. Each of which corresponds to the layers presented in section 2.2. A custom view then defines the elements that are to be shown by assigning a meta class to a particular layer. This is accomplished by using the UML concept *Extension* as defined in [Ob05b].

Finally, a view is applied to the EA model. As a result, only the elements based on the restricted set of meta classes is displayed. Please refer to section 4 for an example of view definition.

4 Case Study

This section introduces a case study in order to illustrate the presented EA modeling approach. Similarly to [PT06], this case study will be situated in the business domain of a fictive insurance company. The resulting EA model, thus, describes (a part of) the enterprise architecture of this company. Since our approach is business process-centered, we necessarily need information on the business processes that are part of day-to-day business of the insurance company. For simplicity reasons we assume, that the business processes

[2]Please note that we do not intend to restrain the number of layers to be defined in a view. This would, theoretically, facilitate the definition of multiple layers of the same kind or even other layers. However, we recommend to only define one layer of the same kind per view.

have already been defined and optimized. We furthermore assume that the business processes' activities are supposed to be (partially) implemented by web services. Invocation of the services should be realized using a BPEL engine.

As part of a service enablement project the company needs to define its EA model in subsequent steps to be able to identify the IT landscape's pain points as well as to track the project's progress. The first step is to define the concrete dependency structure for each of the web services to be developed. This means that for each web service, it should be described, which IT systems provide its functionality. The resulting dependency structure is the basis for the EA model. Other parts of the model can be derived from the BPEL processes, since these already define associations from the business activities to the implementing web services. Some other dependencies have to be defined manually.

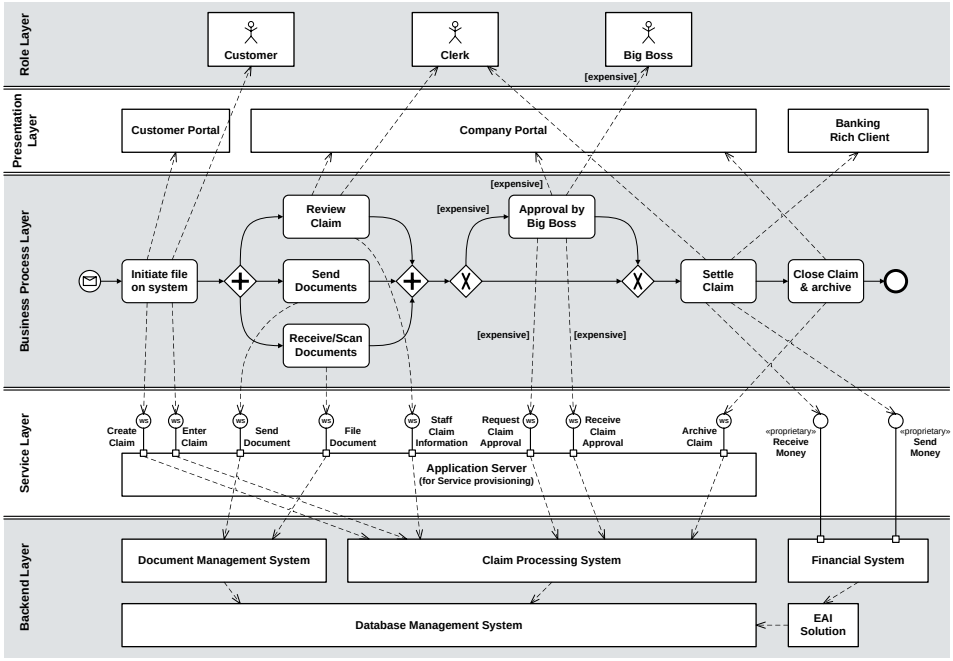

Figure 6: View on all layers of the EA model for the business process *ProcessInsuranceClaim*.

After the EA model is constructed using our modeling approach, it is possible to define views on the model. Figure 6 shows the complete dependency structure for the simple business process *ProcessInsuranceClaim*. This view includes all five layers. Starting from left to right in the business process layer, the customer first has to initiate the claims processing by informing the insurance company of the claim. In this example, the customer enters the claim details through the Internet using a customer portal. The web services *CreateClaim* and *EnterClaim* are invoked as a result of the customer's initiation of claims processing. A dedicated application server is responsible for the exposure of these web

services. It is not responsible for the services' implementation[3]. The services depend on the backend systems *Claim Processing System* and a *Database Management System*. After the initiation, a clerk reviews the claim. During the process of reviewing a claim, several documents have to be sent and others are received. After the claim was reviewed, it depends on the arising expenses whether the clerk's superior has to approve the settling of the claim or not. Consequently, the dependencies to the web services as well as to the role are annotated with the condition expression of the branching sequence flow in the business process. When settling the claim, the clerk has to use a rich client for the bank transfers, since the company portal does not support this functionality. This rich client uses proprietary interfaces of a financial backend system, which is not (yet) service enabled. Finally, the claim is closed and archived.

This example of a simple business process illustrates very well, how complex an EA can become when considering several dozens of more complex business processes and in other conceivable cases an even more heterogeneous IT landscape.

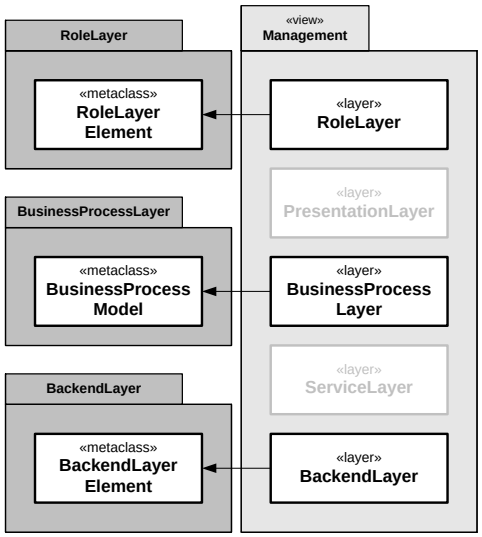

Figure 7: Definition of the Management view.

A view showing all layers as depicted in figure 6 is very detailed and contains (technical) aspects such as particular web services, which are not of primary interest for the insurance company's management. Thus, we define a custom view in order to meet the specific needs of the managers. Figure 7 shows this view definition. Presentation layer as well as the service layer should not be displayed at all. The business process layer should contain business process models instead of a single process model's details.

This results in a high level view on the EA model as depicted in figure 8. It shows two business process models in the business process layer. The first one is the same process as described above. It is obvious that the dependencies of the respective business tasks

[3]Please see section 2.2 for an explanation.

were aggregated into dependencies of the business process model. Additionally, the dependencies to backend systems were derived from the ones that exist between elements in the service layer and the backend systems.

The second business process describes the regularly performed billing of a customer, which is fully automated. Thus, the process model is not dependent on any role. On the other hand, there are dependencies to a billing system, which again is dependent on different other systems.

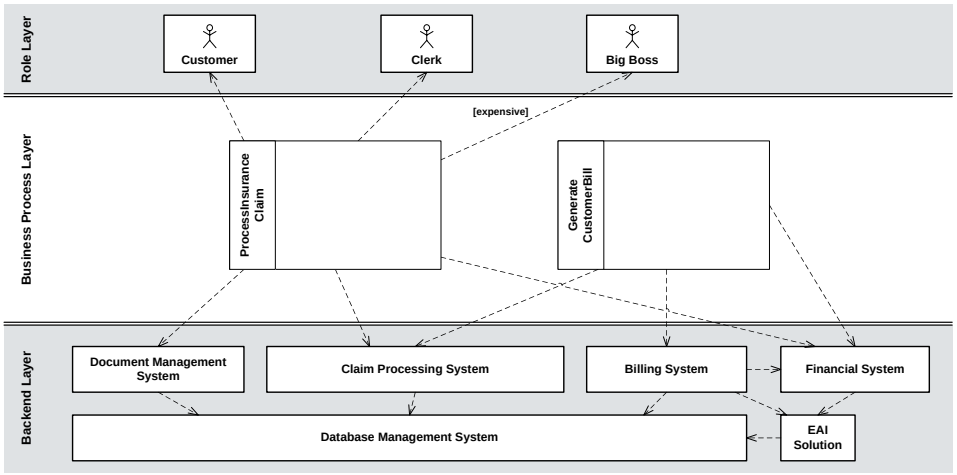

Figure 8: Management view applied on the EA model.

Diagrams like these help to communicate a big picture of the EA to different stakeholders. In this particular case, the management is able to establish an understanding of how business processes are related to organizational roles and the basic IT infrastructure. Apart from monitoring the service enabling project it is also conceivable to e.g. track how many processes rely on the recently purchased document management system. In combination with information on the status of employees (role instances) based on e.g. sick reports and vacation, an identification of bottlenecks is possible.

Similarly to the management view, it is possible to define a maintenance view concentrating the aspects that are of interest for the maintenance staff.

5 Related Work

The approach presented in this paper is the logical continuation of the IT landscape specification formalized by Breest [Br05]. It bases on well settled standards like the Unified Modeling Language [Ra97] and the Business Process Modeling Notation [Ob06b]. We combined the meta models of these techniques to form a new meta model for the entire EA. There have already been efforts spent to define comparable models.

The Common Information Model (CIM) [Dm05] provides a common definition of management information for systems, networks, applications and services. However, it is not possible to apply CIM to functional elements like business processes. It focuses more on the structural elements which are settled in the lower layers of our approach. Our means for dependencies is slightly aligned with the CIM definition.

Another approach to define a formal notation for all kinds of elements of an IT infrastructure is the Reference Model for Open Distributed Processing (RM-ODP) [Is95]. Much like the approach presented in this paper the RM-ODP defines viewpoints on a model in order to provide stakeholder-specific views. However, this model focuses much on technical issues of distributed information processing (e.g. streaming) and does not deal with enterprise entities from our higher layers (e.g. roles, business processes).

The Unified Modeling Language concentrates on the analysis and design of object-oriented applications. The coarse-grained entities of an EA are poorly reflected in the core concepts of this language. The dynamic and functional parts of the UML are rarely used to specify business processes. They are best suited for specifications of internal application flow. However, the concept of UML profiles is very useful for the definition of different views and the UML meta model provides the extension capabilities to define new concepts. Thus, we decided to use UML profiles for view definition and to derive our new modeling concepts from the UML meta model.

The ARIS tool set [Sc98] is widely spread in today's industries to define high level business processes. It uses the Event-driven Process Chain (EPC) [KNS92] notation to specify business processes of enterprises. However, it lacks a formalization for the business process reflection in the implementing IT infrastructure. Thus, it does not provide the traceability we need for our approach.

The Web Service Architecture [W304] describes a lot of concepts surrounding the entire web service area. These contain also entities like processes, tasks, agents and so on. However, it is less formal than the approach we presented. The dependencies described within this architecture aim at a basic understanding of a typical web service scenario rather than on a formal specification that could support use cases like those mentioned before.

The ArchiMate project [Jo03] elaborated an EA specification language. Our approach is very much aligned to that from this project, however, we focus more on the transitivity of dependencies between the architectural entities as well as on the concept of annotation.

Although the above related work results show that there is a complete language coverage of separate architectural concepts, there is still a lack of integration between them. We, therefore, focused on an integration and reuse of the best-of-breed of these languages.

6 Conclusion

We presented a business process-centered modeling approach aiming at the description of complex dependency structures within an Enterprise Architecture. The dependency structure is implicitly existent in any EA. Due to efficient visualization of the dependencies and

annotation of these dependencies with non-functional information, it is, for the first time, possible to *explicitly* visualize and communicate these complex structures. The approach is based on the well-established modeling techniques UML and BPMN. A MOF-based meta model was provided, formalizing the approach.

Furthermore, we provided a facility to define customized views on the EA model tailored to the specific needs of different stakeholders. This enables the effective communication of complex EAs in form of a big picture of the EA covering the aspects of interest for each particular stakeholder.

Finally, the approach was illustrated in a case study. A practical use of the approach was depicted. The added value of the approach during and after a service enablement project features improvement of communication and analysis as well as optimization of the EA.

However, there is still a number of aspects that might be subject of further research. Various (semi-)automated auditing techniques for an EA such as *impact analysis*, *availability analysis*, and *performance analysis* based on metrics and heuristics are conceivable. All auditing is enabled due to traceability of dependencies between enterprise architectural entities and their annotation. Depending on the kind of analysis, automated report generation might be desirable. In other cases, it might be interesting to interactively simulate certain scenarios in order to investigate issues that, otherwise, would only appear when actively modifying the EA. So, it is, to a certain extent, possible to investigate issues that are mostly inherent with major changes to an EA even before actually changing the EA. Concrete heuristics and metrics are to be identified in order to facilitate auditing of EAs.

Next, our approach is based on the assumption that it can be seamlessly integrated into a business process engine enacting the business processes. This integration might prove to be a complex task. Then, for reasonable and extensive annotation of elements with non-functional information it is, possibly, necessary to access different frameworks such as e.g. a monitoring framework providing statistical information on process enactment.

Generally, the approach should be applied in a real company in a productive environment in order to prove its added value and to, eventually, improve or redesign details. It might, for instance, turn out that a hierarchical structuring of organizational entities (such as companies, departments, roles) is advantageous in certain situations.

References

[Br05] Breest, M.: Specifying Service Landscapes. Seminar Reader of the Hasso-Plattner-Institute, 2005
http://bpt.hpi.uni-potsdam.de/twiki/pub/Public/SeminarPublications/ReaderBPM2.pdf

[Sc05] Schubert, H.: Entwicklung eines QoS-Frameworks für Service-orientierte Architekturen. Masters Thesis. Hasso-Plattner-Institute for Software Systems Engineering, SAP Systems Integration AG, 2005

[Bu00] Burbeck, S.: The Tao of e-business Services. Emerging Technologies, IBM Software Group, 2000

[PT06] Pulier, E., Taylor, H.: Understanding Enterprise SOA. Manning Publications Co., 2006

[TGK06] Tabeling, P., Gröne, B., Knöpfel, A.: Fundamental Modeling Concepts - Effective Communication of IT Systems. John Wiley & Sons, Ltd, 2006

[Ob05a] Object Management Group: UML Infrastructure Specification, v2.0, 2005

[Ob05b] Object Management Group: UML Superstructure Specification, v2.0, 2005

[Ob06a] Object Management Group: Meta Object Facility (MOF) Core, v2.0, 2006

[Ob06b] Object Management Group: Business Process Modeling Notation (BPMN) Specification, 2006

[Wi06] Wikipedia: Enterprise architecture, July 2006
http://en.wikipedia.org/w/index.php?title=Enterprise_architecture&oldid=64463956

[Ra97] Rational Software (editor): UML Notation Guide 1.1, Unified Modeling Language Version 1.1. Santa Clara (USA), 1997

[Dm05] DMTF: CIM: Common Information Model Schema Version 2.11, December 2005
http://www.dmtf.org/standards/cim/cim_schema_v211

[Is95] ISO/IEC: ITU-T X.901 ISO/IEC 10746-1 Open Distributed Processing Reference Model Part 1. Draft International Standard (DIS) output from the editing meeting in Helsinki (Finland), May 1995

[Sc98] August-Wilhelm Scheer: Aris-Business Process Frameworks. Berlin; New York: Springer, 1998

[KNS92] G. Keller and M. Nüttgens and A.-W. Scheer: Semantische Prozeßmodellierung auf der Grundlage Ereignisgesteuerter Prozeßketten (EPK). Veröffentlichungen des Instituts für Wirtschaftsinformatik, Heft 89, Saarbrücken, 1992

[W304] W3C: WSA: Web Services Architecture, February 2004
http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/

[Jo03] Henk Jonkers, Buuren Buuren, Farhad Arbab, Frank de Boer, Marcello Bonsangue, Hans Bosma, Hugo ter Doest, Luuk Groenewegen, Juan Guillen Scholten, Stijn Hoppenbrouwers, Maria-Eugenia Iacob, Wil Janssen, Marc Lankhorst, Diederik van Leeuwen, Erik Proper, Andries Stam, Leon van der Torre, Gert Veldhuijzen van Zanten: Towards a Language for Coherent Enterprise Architecture Descriptions. Seventh International Enterprise Distributed Object Computing Conference (EDOC), p. 28., 2003