Supporting Viewpoint-Oriented Enterprise Architecture

M.W.A. Steen¹, D.H. Akehurst², H.W.L. ter Doest¹, M.M. Lankhorst¹
¹Telematica Instituut, the Netherlands
²University of Kent at Canterbury, United Kingdom
{Maarten.Steen,Hugo.terDoest,Marc.Lankhorst}@telin.nl, D.H.Akehurst@kent.ac.uk

Abstract

Increasingly, organisations establish what is called an enterprise architecture. The enterprise architecture combines and relates all architectures describing some aspect of the organization, such as the business process architecture, the information architecture, and the application architecture. It is a blueprint of the organisation, which serves as a starting point for analysis, design and decision making.

Viewpoints define abstractions on the set of models representing the enterprise architecture, each aimed at a particular type of stakeholder and addressing a particular set of concerns. The use of viewpoints is widely advocated for managing the inherent complexity in enterprise architecture. Viewpoints can both be used to view certain aspects in isolation, and for relating two or more aspects. However, in order to make such a viewpoint-oriented approach practically feasible, architects require a tool environment, which supports the definition, generation, editing and management of architectural views. Moreover, such an environment should work in concert with existing domain-specific modelling tools. In this paper, we present the design of such a tool environment for viewpoint-oriented enterprise architecture.

1. Introduction

Increasingly, organisations establish what is called an enterprise architecture. The enterprise architecture combines and relates all architectures describing some aspect of the organization, such as the enterprise business architecture, the enterprise information architecture, the enterprise-wide technology architecture, and the enterprise application portfolio [3]. The goal of enterprise architecture (EA) is to provide insight in the organisational structures, processes and technology that make up the organisation, highlighting opportunities for efficiency improvements and improved alignment with business goals [26].

Establishing and maintaining a coherent enterprise architecture is clearly a complex task, because it involves many different people with differing backgrounds using various notations. In order to get a grip on this complexity, researchers have initially focused on the definition of architectural frameworks for classifying and positioning the various architectural descriptions with respect to each other. A prime example is the Zachman Framework for Enterprise Architecture [25]. This two-dimensional framework identifies 36 views on architecture (“cells”), based on six levels (scope, business model, system model, technology model, detailed representations and functioning enterprise) and six aspects (data, function, network, people, time, motivation). A problem with looking at EA through the lens of an architectural framework like Zachman’s is that it categorizes and divides architectural descriptions rather than providing insight into their coherence.

We envisage a more flexible approach in which architects and other stakeholders can define their own views on the enterprise architecture. In this approach views are specified by viewpoints. Viewpoints define abstractions on the set of models representing the enterprise architecture, each aimed at a particular type of stakeholder and addressing a particular set of concerns. Viewpoints can both be used to view certain aspects in isolation, and for relating two or more aspects.

However, in order to make such a viewpoint-oriented approach practically feasible, architects require a tool environment, which supports the definition, generation, editing and management of architectural views. Moreover, such an environment should work in concert with existing domain-specific modelling tools, since we cannot expect architects to start using other tools, let alone other languages, than the ones they are used to. The contribution of this paper is the design of a viewpoint-support environment, which has the following innovative features:

• In the definition of viewpoints a clear separation is made between view content (which model elements should be included in a view?) and view presentation (how should the view content be presented?).
• Model integration, which is necessary for relating models expressed in different domain-specific modelling languages, is achieved through the mapping to a common enterprise architecture modelling language, the ArchiMate Language [14].
• Viewpoints are specified and created using model transformation techniques [21].
1.1. Acknowledgments

This paper is the result of a collaboration between the ArchiMate project and the University of Kent at Canterbury, which is partially supported by the Commission of the European Communities under the sixth framework programme (INTEROP Network of Excellence, Contract No 508011, http://www.interop-noe.org). The ArchiMate project (http://archimate.telin.nl/) is a research initiative that aims to provide concepts and techniques to support enterprise architects in the visualisation, communication and analysis of integrated architectures. It is jointly funded and executed by ABN AMRO Bank, Stichting Pensioenfonds ABP, the Dutch Tax and Customs Administration, Ordina, Telematica Instituut, Centrum voor Wiskunde en Informatica, Katholieke Universiteit Nijmegen, and Leiden Institute of Advanced Computer Science.

2. Viewpoints for Enterprise Architecture

The notion of viewpoint-oriented architecture has been around for a while in requirements and software engineering. In the 1990’s, a substantial number of researchers worked on what was phrased as “the multiple perspectives problem” [8][19][17][23]. By this term they referred to the problem of how to organise and guide (software) development in a setting with many actors, using diverse representation schemes, having diverse domain knowledge and different development strategies. A general framework has been developed in order to address the diverse issues related to this problem [8][19]. In this framework, a ViewPoint combines the notion of ‘actor’, ‘role’ or ‘agent’ in the development process with the idea of a ‘perspective’ or ‘view’ which an actor maintains. More precisely, ViewPoints are defined as loosely coupled, locally managed, distributable objects; thus containing identity, state and behaviour. A ViewPoint is more than a ‘partial specification’; in addition, it contains partial knowledge of how to develop that partial specification. These early ideas on viewpoint-oriented software engineering have found their way into the IEEE-1471 standard for architectural description [12] on which we have based our definitions below.

Viewpoints are also prominently present in the ISO standardized Reference Model for Open Distributed Processing (RM-ODP) [13]. The RM-ODP identifies five viewpoints from which to specify ODP systems, each focusing on a particular area of concern, i.e., enterprise, information, computational, engineering and technology. It is claimed that the ODP viewpoints form a necessary and sufficient set to meet the needs of ODP standards.

More recently, the term ‘viewpoint’ is also used in OMG’s Model Driven Architecture (MDA) initiative to refer to the different model types, i.e., platform-independent model (PIM) and platform-specific model (PSM) [20]. Hence we conclude that the use of viewpoints and architectural views are well-established concepts in software architecture. One of the objectives of our current research is to transfer this concept to the domain of enterprise architecture.

2.1. Enterprise Architecture Frameworks

In order to define the field and determine the scope of enterprise architecture both researchers and practitioners have produced a number of, so-called, “architecture frameworks”. Such frameworks provide structure to the architectural descriptions by identifying and sometimes relating different architectural domains and the modelling techniques associated with them. They typically define a number of conceptual domains or aspects to be described. Well-known examples of architectural frameworks are:

- Zachman’s Framework for Enterprise Architecture [25].
- The Reference Model for Open Distributed Processing (RM-ODP) [13].
- The Open Group Architecture Framework (TOGAF) [22].

In our work we use a simplification of the Zachman framework to position the various architectural descriptions encountered in EA literature and practice [14]. This simplified EA framework, which is illustrated in Figure 1, mainly serves to organize our own thoughts and work; it is not intended as yet another enterprise architecture framework. This ArchiMate framework distinguishes three functionality layers and three aspects. The layers – business, application and technology – correspond to the business model, system model and technology model levels in the Zachman framework. The aspects – structure, behaviour and information – correspond to the network, function and data aspects in the Zachman framework.

![Figure 1. The ArchiMate framework](image-url)

Information aspect

Behaviour aspect

Structure aspect

Business layer

Application layer

Technology layer

Information domain

Product domain

Process domain

Organisation domain

Data domain

Application domain

Technical infrastructure domain
As is shown in the figure, different known conceptual domains can be projected onto this framework. Business process modelling, for example, is concerned with the specification of behaviour in the business layer, whereas application architectures describe both behavioural and structural aspects in the application layer. A domain-specific architectural description may be seen as a partial view on the entire enterprise architecture. However, in our work we are especially interested in views that relate two or more domains.

2.2. Definitions

View and viewpoint are central concepts in the IEEE 1471 Standard for architectural description [12]. According to IEEE 1471, a view is a representation of a whole system from the perspective of a related set of concerns. Concerns are those interests, which pertain to the system’s development, its operation or any other aspects that are critical or otherwise important to one or more stakeholders. A viewpoint is a pattern or template from which to develop individual views. It establishes the purposes and audience for a view and the techniques for its creation and analysis. In order to satisfy the IEEE 1471 standard a viewpoint should specify at least:

- A viewpoint name,
- The stakeholders the viewpoint is aimed at,
- The concerns the viewpoint addresses,
- The language, modelling techniques, or analytical methods to be used in constructing a view based upon the viewpoint.

Since we intend to design a viewpoint-support environment, we need to make the last two bullets more precise. Concerns can vary a great deal. Managers may be concerned with the mission of the enterprise and the appropriateness of the IT-infrastructure in fulfilling its mission. In contrast, software engineers may be concerned with the components, their interfaces, and the available mechanisms for interconnection. We do not intend to provide a complete taxonomy for all possible concerns related to EA, but provide a framework for classifying concerns and viewpoints in section 2.3 below. In order to support the automatic generation of views from other architectural descriptions, we also need to specify very precisely how models and views are related. In section 3.4, we define an approach for specifying these relationships, which is based on model transformation techniques.

2.3. A Framework for Classifying Viewpoints

In order to support architects and other stakeholders in selecting the appropriate viewpoints from a possibly large viewpoint library, we propose to classify viewpoints according to their purpose and content. This classification scheme is not intended to replace, but rather to complement, existing viewpoint models or frameworks, such as Kruchten’s 4+1 Views [18] or the RM-ODP viewpoints [13]. The classification we propose can be applied to any existing set of viewpoints, or it can be used to identify new viewpoints that still need to be defined. Our classification scheme can be used to categorise and understand viewpoints based on the stakeholder’s objectives and concerns they support.

The classification scheme, illustrated in Figure 2, is based on two dimensions: purpose and content. The purpose of a viewpoint is determined by what a stakeholder wants to do or achieve. For the purpose we distinguish the following categories:

- **Designing** – Design viewpoints support architects and designers in the design process from initial sketch to detailed design. Design viewpoints typically focus on a specific conceptual domain, e.g., application architecture, business process model, but may also be used to define the interdependencies between domain architectures.

- **Deciding** – These viewpoints assist managers in the process of decision making by offering insight into cross-domain architecture relations, typically through projections and intersections of underlying models, but also by means of analytical techniques. Typical examples are cross-reference tables, landscape maps, lists and reports.

- **Informing** – These viewpoints help to inform any stakeholder about the enterprise architecture, in order to achieve understanding, obtain commitment and convince adversaries. Typical examples are illustrations, animations, cartoons, and flyers.

For characterising the content of a view the following abstraction levels are defined:
• **Details** – Viewpoints at this level typically consider one layer and one aspect from the ArchiMate framework (see Figure 1). Typical examples are UML component and deployment diagrams, and business process models. This level is relevant to stakeholders concerned with design or operations, such as software engineers, responsible for the design and implementation of a component, and process owners, responsible for the effective and efficient execution of a business process.

• **Coherence** – At this level, either multiple layers are spanned or multiple aspects. Extending the view to more than one layer or aspects enables the viewpoint user to focus on architectural relationships, e.g., process-uses-system (multiple layer) or application-uses-object (multiple aspect). Typical stakeholders are operational managers responsible for a collection of IT services or business processes.

• **Overview** – Viewpoints at this level address both multiple layers and multiple aspects. Typically such overviews are addressed at enterprise architects or management.

### 2.4. Example

The following example serves to illustrate our vision on the use of viewpoints in enterprise architecture. ArchiSurance is an imaginary insurance company, which has an enterprise architecture consisting of various architectural models. The application architecture is modelled using UML, part of which is shown in Figure 3. The business processes are modelled using BPMN, the Business Process Modelling Notation [27], one of which is shown in Figure 4. In order to assess and achieve alignment between the business process architecture and the application architecture, the enterprise architect requires a viewpoint, which allows him to relate business processes to application components. A viewpoint-support environment should enable the enterprise architect both to define such a viewpoint and to generate and manage views based on that definition. A partial specification for this viewpoint is given in Table 1. The viewpoint language, i.e., the collection of concepts and relationships required to express views according to this viewpoint, is specified separately in Figure 5. It specifies that the Process-Application Alignment Viewpoint may contain three types of model elements: business process, application service and application component. These are related as follows. A component provides one or more services, which in turn may be used by one or more business processes.

![UML model of ArchiSurance application and system architectures](image)

![BPMN model of claim registration process](image)

**Table 1. Viewpoint specification**

<table>
<thead>
<tr>
<th>Viewpoint name</th>
<th>Process-Application Alignment</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Stakeholders</strong></td>
<td>Enterprise, application, process architects</td>
</tr>
<tr>
<td><strong>Concerns</strong></td>
<td>Process-application alignment</td>
</tr>
<tr>
<td><strong>Purpose</strong></td>
<td>Designing or Deciding</td>
</tr>
<tr>
<td><strong>Abstraction level</strong></td>
<td>Coherence</td>
</tr>
<tr>
<td><strong>Layers</strong></td>
<td>Business, Application</td>
</tr>
<tr>
<td><strong>Aspects</strong></td>
<td>Behaviour</td>
</tr>
<tr>
<td><strong>Source languages</strong></td>
<td>BPMN, UML</td>
</tr>
<tr>
<td><strong>Viewpoint language</strong></td>
<td>Defined in Figure 5</td>
</tr>
</tbody>
</table>
Figure 5. Concepts and relations for process-application alignment

An example of a process-application alignment view, i.e., an instance of the given viewpoint, is shown in Figure 6. It contains abstract representations of the application components and their services from the UML model and abstract representations of the business processes from the BPMN model. In addition, the view has allowed the enterprise architect to draw (used_by) relationships between the processes and the services of the application components to document dependencies.

Figure 6. A Process-Application Alignment view

3. Design of a Viewpoint Support Environment

One of the main goals of the ArchiMate project is to provide enterprise architects with instruments that support and improve the disclosure and visualization of enterprise architectures without being obstructed by the narrowness of specific domains. The clients of this project are three large financial organizations. In interaction with their architects we formulated the following high-level requirements for viewpoint support:

- A viewpoint should indicate for which types of users, i.e., stakeholders, it is intended and which concerns it covers. This is also required by the IEEE 1471 standard, as discussed in section 2.2.
- View content (the model elements making up the view) and view presentation (the way these model elements are presented to the stakeholders) should be separated. In this way, different visualisation techniques can be applied to the same view content in order to serve different stakeholders. Likewise, this separation allows us to use the same visualisation techniques on different types of concepts.
- A viewpoint should specify rules about the use of the ensuing views, related to the role of the current user. Can they be edited, for example? And does this concern the layout and other graphical aspects only, or also the view contents, or even the underlying models?
- The tool should support interactive viewpoint definition. Probably a kernel of frequently consulted viewpoints exists, but it must be possible to customise these and create new ones.

3.1. Tool Integration

Creating such a viewpoint support environment is clearly a challenging task. The enterprise architecture of large organizations, such as the ones participating in the ArchiMate project, consists of many different types of description. Moreover, different types of people generally create these architectural descriptions using different languages and different tools. Table 2 illustrates the typical situation in a large organisation.

Table 2. The enterprise architecture landscape in a typical organisation.

<table>
<thead>
<tr>
<th>Model type</th>
<th>Tool</th>
<th>Owner</th>
</tr>
</thead>
<tbody>
<tr>
<td>Organisation model</td>
<td>Visio</td>
<td>Business Analyst</td>
</tr>
<tr>
<td>Information model</td>
<td>Rochade</td>
<td>Information Architect</td>
</tr>
<tr>
<td>Business process model</td>
<td>Testbed Studio</td>
<td>Business Process Architect</td>
</tr>
<tr>
<td>Application architecture</td>
<td>Enterprise Architect</td>
<td>Chief System Architect</td>
</tr>
</tbody>
</table>
that companies will throw their existing design practices and tools overboard and replace these by an entirely new approach. Rather, enterprise architecture should focus on bringing together already existing techniques and integrating these at the appropriate level of abstraction. Nevertheless, there is a need to relate and combine the various domain-specific architectures in order to support the creation of the required views. Therefore, we need to integrate existing tools into a viewpoint-support environment.

Our first idea was to achieve this by defining tool-specific adaptors, which can access and “translate” model data, and directly invoke required tool-functionality. However, due to the sheer quantity of tools in the market, each providing a different level of access, some not providing any type of programmatic access, we soon concluded that this was not a feasible solution. Rather than interfacing directly with existing tools, we therefore decided to assume that all models are stored in a common repository. The viewpoint-support environment can then be built on top of this repository. Ideally such a repository would be compliant to OMG’s Meta-Object Facility (MOF) or another metamodelling standard, so we would have access to models, their elements and their metamodels in a uniform manner. Boosted by the success of the MDA initiative [20], vendors are increasingly including MOF support in their modelling tools. Therefore we are optimistic about the feasibility of the common repository assumption in the near future. In the worst case, the common repository is the operating system’s file system, in which case we would need to equip our viewpoint-support environment with tool-specific model parsers, but the assumption would still hold.

3.2. Model Integration

Although the use of a repository solves the technical integration problem, it still leaves us with a conceptual integration problem. Before we can create views on an architecture that comprises a number of heterogeneous models, these models must in some way be related or integrated with each other. Model integration or conceptual integration can be obtained in two ways [6]. One possibility is to define a direct mapping between each pair of modelling languages to facilitate direct relations between models expressed in arbitrary languages. The other possibility is to use an intermediate language, which would require only \( O(n) \) mappings instead of the \( O(n^2) \) mappings required with direct relations. We have opted for the second solution and defined a high-level language, the ArchiMate language, for representing and relating concepts relevant to enterprise architecture, while abstracting from the detailed representations used in domain-specific modelling languages.

A key challenge in the development of such a general language for enterprise architecture is to strike a balance between the specificity and comprehensiveness of languages for individual architectural domains, and a too abstract language in which all architectures are represented as entities and relations. Figure 7 illustrates that concepts can be described at different levels of specialisation. At the base of the triangle, we find the architectural concepts used in existing, domain-specific modelling languages and standards; UML is an example of a language in this category. At the top of the triangle we find the “most general” concepts for system architectures, merely comprising notions such as “object”, “component”, and “relation”. Some architectural description languages such as ACME [10] partly fall into this category. With the definition of the ArchiMate language we have aimed at a level somewhere between these two extremes. The concepts of this language provide a common basis and integration layer for the more specific concepts in the bottom layer. If desired, they can be further specialised or composed to form concepts tailored towards a more specific context. In this way, we also hope to be more future proof that existing modelling languages, which may not be able to deal with paradigm shifts.

![Figure 7. Modelling languages at different levels of specificity.](image-url)

The value of the ArchiMate language does not lie in the individual concepts, but in the fact that business, applications, and technology can all be represented at an appropriate level of abstraction for enterprise architecture using a single language. Existing languages focus on detailed descriptions of individual domains, but lack sufficient means for relating these to other domains. The ArchiMate language acts as a bridge between existing models for these individual domains. These models can be seen as more detailed views on specific parts of the enterprise architecture. If they are tied in to an overall ‘umbrella’ model, an integrated enterprise architecture model can be constructed and models from formerly separate domains can be related in a meaningful way. Another advantage over comprehensive languages, like UML, is that the ArchiMate language provides the minimal set of
concepts required for enterprise architecture. Therefore enterprise architects are not burdened with unnecessary details.

3.3. The ArchiMate language

The ArchiMate language covers all layers and aspects of the framework given in Figure 1. A common abstract metamodel lies behind the concrete modelling concepts at all three layers, comprising (abstract) concepts for structure, behaviour, and information. The most important concrete modelling concepts are explained below. For a more detailed description please refer to [15].

The main structural concept at the business layer is the business actor, an entity that performs behaviour such as business processes or functions. Business actors may be individual persons (e.g. customers or employees), but also groups of people and resources that have a permanent (or at least long-term) status within the organisations. To each actor business roles can be assigned, which in turn signify responsibility for one or more business processes, which may manipulate business objects. The externally visible behaviour of a business process is modelled by the concept organisational service, which represents a unit of functionality that is meaningful from the point of view of the environment. Services are grouped to form (financial or information) products, together with a contract that specifies the characteristics, rights and requirements associated with the product.

The main structural concept for the application layer is the application component. This concept is used to model any structural entity in the application layer: not just (reusable) software components that can be part of one or more applications, but also complete software applications or information systems. Data objects are used in the same way as data objects (or object types) in well-known data modelling approaches, most notably the ‘class’ concept in UML class diagrams. In the purely structural sense, an application interface is the (logical) location where the services of a component can be accessed. In a broader sense (as used in, among others, the UML definition), an application interface also has some behavioural characteristics: it defines the set of operations and events that are provided by the component, or those that are required from the environment. Behaviour in the application layer can be described in a way that is very similar to business layer behaviour. We make a distinction between the externally visible behaviour of application components in terms of application services, and the internal behaviour of these components to realise these services.

The main structural concept for the technology layer is the node. This concept is used to model structural entities in the technology layer. Nodes come in two flavours: device and system software, both inspired by UML 2.0 (the latter is called execution environment in UML). A device models a physical computational resource, on which artifacts may be deployed for execution. System software represents the software environment for specific types of components and data objects. An infrastructure interface is the (logical) location where the infrastructural services offered by a node can be accessed by other nodes or by application components from the application layer. An artifact is a physical piece of information that is used or produced in a software development process, or by deployment and operation of a system. A communication path models the relation between two or more nodes, through which these nodes can exchange information. The physical realisation of a communication path is a modelled with a network, i.e., a physical communication medium between two or more devices. In the technology layer, the central behavioural concept is the infrastructure service. We do not model the internal behaviour of infrastructure components such as routers or database servers; that would add a level of detail that is not useful at the enterprise level of abstraction.

3.4. Viewpoints are Model Transformations

A viewpoint specifies on the one hand which model elements should be selected from the source models and how they should be mapped onto ArchiMate model elements, and on the other hand how the resulting ArchiMate model should be mapped onto elements of a presentation language for visualisation (also see Figure 10). These mappings are essentially model transformations. Within the OMG-led MDA movement, model transformation is a key technology and a very hot topic [9][7]. In fact, the request for proposals (RFP) issued by OMG for a transformation specification standard [21] explicitly calls for proposals that are also suitable to query models and generate views. Therefore, we decided to look into available techniques for specifying and implementing model transformations to generate architectural views.

In principle, the mentioned mappings can also be interpreted the other way around. If an architect changes something in the visualisation of a view, it should be possible to propagate this change to the ArchiMate model representing the view, and eventually from there to the originating model(s). In order to support such back-and-forth synchronization between views and their originating models we require a transformation specification technique, which can be applied in both directions. From the current submissions to the Query / Views / Transformations (QVT) RFP only those by Compuware and Sun [5], and by Codagen Technologies [4] support such bi-directional transformations according to Gardner et
al. [9]. However, these two proposals do not support N-to-1 transformation, which we require for generating views from multiple originating models. Fortunately, the relational approach to defining and implementing transformations between metamodels by Akehurst, Kent and Patrascoiu [1] does satisfy all our requirements. In the rest of this section, we show how this approach can be used to specify viewpoints as model transformations.

3.5. Specification Principles

The principle on which this transformation specification technique is based is that of mathematical relations. A Relation (R) is a set of elements, where each element is a pairing of two objects (a and b) that map to each other. The objects are each drawn from two other sets defined as the domain and range of the relation. An additional matching condition can be specified which is a relation specific expression that must evaluate to true for all elements in the relation. In mathematics this can be written:

\[ R = \{(a, b) \in A \times B | a \in \text{domain} \land b \in \text{range} \land \text{matching condition}\} \]

In an object-oriented modelling context we can express such a relation with a pattern of classes and constraints, as shown in Figure 8.

![Figure 8. Modelling relations as classes.](image)

Given this pattern of classes we can define relations between classes from different models and specify the conditions under which the model instances should be mapped (i.e. contained in the relation). It is also necessary to define the domain and range sets; this can be ‘allInstances’ of the domain and range types (classes) or we can define a more local ‘scope’ for each relation. One useful way to do this is by nesting relations into a hierarchy, using a subRelation relationship on which the domain and range expressions are defined for the sub relation. This causes the scope of one relation to be in the context of an element from its parent relation, and facilitates reuse and recursive nesting of relations; [1] discusses this in more detail. To aid the concise specification of relations, we can use a single graphical icon to represent this pattern of classes or a single class with a `<<relation>>` stereotype. The former option has been used in the examples in section 4.

3.6. Tool Support for Viewpoint Specification

Currently, we use a generic UML class diagramming tool to draw the relations and annotate them with OCL expressions for the matching conditions. These models are imported by the KMF tools [16] to generate transformers, i.e., code for performing the transformations. These transformers need to know how to build objects from one side of the relation given objects from the other side. A constraint solver of some description could potentially do this for simple matching conditions. However there are limitations to what a constraint solver can do and they are generally pretty inefficient. Our alternative approach is to explicitly define expressions for each relation that will build an object of one side given an appropriate object from the other. The result of such an expression should give a new object which when paired with the original meets the matching condition.

3.7. Tool Architecture

The overall architecture of the viewpoint-support environment is given in Figure 9. The Viewpoint Editor has already been discussed above. The Viewpoint Manager stores, retrieves and supplies viewpoint specifications. The View Content Manager generates and maintains the view content based on viewpoint specifications. The Presentation Manager presents the view content to the user and allows them to interact with the view. In our prototype, we use Microsoft Visio as presentation manager, but in principle the viewpoint specification should determine which presentation tool to use.
4. Specifying the Example Viewpoint

In the example, introduced in section 2.4, we defined a viewpoint, which combines some elements from a UML model of the application architecture, with a business process model in BPMN. Views for this viewpoint are created in two steps (see Figure 10). First, the two source models are transformed onto the ArchiMate language. Actually, we only map those concepts that are relevant to this viewpoint, as defined in Figure 5, which is a subset of the ArchiMate language. The next step is to visualise the obtained view content. In our prototype, we use Microsoft Visio as the presentation tool. The view presentation is therefore obtained as a fairly straightforward transformation of ArchiMate concepts onto Symbols defined in a Visio stencil. In the remainder of this section, we therefore focus on the definition of the transformations between the source languages and the viewpoint language. These two transformations are specified as sets of relations defining which model elements from UML and BPMN should be mapped onto model elements from ArchiMate. In order to fully appreciate these definitions, some knowledge of the respective meta-models is required. We assume the reader is sufficiently familiar with the UML meta-model. A simplified version of the BPMN meta-model is included in the appendix.

![Figure 10. Transformations defining the viewpoint](image)

4.1. BPMN to ArchiMate

![Figure 11. Mapping between BPMN and ArchiMate](image)

Figure 11 shows the transformation relations between a BPMN diagram and the Process-Application Alignment Viewpoint. The first relation simply maps a diagram to an instance of the viewpoint. The second relation states that activities within the BPMN diagram should be mapped to BusinessProcess objects in the viewpoint. This viewpoint is only interested in the activity objects defined by the BPMN model, hence all other model objects are ignored. Also note that the transformation ignores sub activities (activities within activities) from the BPMN model, as we are only interested in the high-level overview of the business processes. The following OCL statements define the domain, range and matching conditions for the relations.

```oclnl
context DiagRelVp$Element
ActivityRelProc.domain :
  diagram.pool.lane.element->select
    (e | e.oclIsKindOf(Activity))
ActivityRelProc.range :
  vp->select
    (e | e.oclIsKindOf(BusinessProcess))
context ActivityRelProc$Element
  matchingCondition :
    activity.label.text = proc.name and
    relation.hasElement(activity.
      sequenceFlow, proc.triggers )
```

4.2. UML to ArchiMate

Figure 12 shows the relations for transforming UML Components and Interfaces into the ArchiMate application components and services. The mapping is quite straightforward defining a one-to-one relationship between UML components and their interfaces and a representation of them in the ArchiMate viewpoint language.

![Figure 12. Mapping between UML and ArchiMate](image)

```oclnl
context PkgRelVp$Element
  CompRelAppComp.domain :
    package.ownedElement->select
      (e | e.oclIsKindOf(Component))
CompRelAppComp.range :
  vp->select(e |
    e.oclIsKindOf(ApplicationComponent))
context CompRelAppComp$Element
  matchingCondition :
    component.name = appComp.name
IntfRelService.domain :
  component.interfaces
IntfRelService.range :
  appComp.applicationService
context IntfRelService$Element
  matchingCondition :
    interface.name = service.name
```
As stated before, we also define build expressions for the domain and range elements rather than making use of a constraint solver. Each expression is interpreted in the context of the relation element that maps a domain element onto a range element. Depending on the direction of transformation, domain $\rightarrow$ range or range $\rightarrow$ domain, the appropriate domain or range build expression is used to build a missing element. After evaluating the build expression the matching condition must evaluate to true. For all of the example mappings shown above these expressions simply create an object of the correct type and set the name indicator of the new object to the corresponding value from the existing (domain or range) part of the relation element. For example, using an action language built on top of OCL, the expressions for building the domain and range parts of an element of the Activity-to-BusinessProcess relation are defined as follows:

```
context ActivityRelProc$Element
buildDomainExpression: Activity {
  label = Label{text = proc.name} }
let
next = relation.lookupRange{
  proc.triggers}.domain
in Activity {
  label = Label{text = proc.name,
  sequenceFlow = next }
buildRangeExpression:
let
trig = relation.lookupDomain{
  activity.sequenceFlow}.range
in BusinessProcess {
  name = activity.label.text,
  triggers = trig }
```

These expressions use the lookup function on relations, passing either a domain or range element as parameter, the function returns an element of the relation that has the parameter equal to either the domain or range part.

5. Conclusion

In this paper, we have described the design of a tool environment for viewpoint-oriented enterprise architecture. This environment supports the definition of viewpoints and the generation and management of views based on such definitions. Before we could build this environment we had to solve the problems of tool integration, model integration and viewpoint specification. Tool integration is achieved through the use of a shared repository. Model integration is achieved by making use of an abstract intermediate language, the ArchiMate language. Finally, viewpoints are specified as transformations between the originating models and the ArchiMate language, on the one hand, and between the ArchiMate language and presentation languages on the other hand. These transformations are specified using relations that relate concepts in the respective language meta-models. This technique was illustrated using an example view in which a business process model could be related to an application architecture.

Our tool environment currently consists of two different prototypes: one for the visualization and editing of views that make use of the ArchiMate concepts, and one for the specification and execution of the transformations. In the near future, we will aim to integrate these into one viewpoint-support environment.

References

Appendix: BPMN Metamodel

Figure 13 illustrates the BPMN metamodel that we reverse-engineered from [27] in order to define the transformation in section 4.1.

![BPMN metamodel diagram](image-url)