*Software ENgineering*

# Integrating architectural models

F. Arbab, F.S. de Boer, M.M. Bonsangue,
M.M. Lankhorst, H.A. Proper, L.W.N. van der Torre

**REPORT SEN-E0515 NOVEMBER 2005**

ISSN 1386-369X

# Integrating architectural models

ABSTRACT
The diversity of architectural models in enterprise architecture is a problem for their integration. In this paper we distinguish three kinds of models from each other and their visualization, and we illustrate how the distinctions can be used for model integration within the architectural approach. Symbolic models express properties of architectures of systems, semantic models interpret the symbols of semantic models, and subjective models are purposely abstracted conceptions of a domain. Building on results obtained in the ArchiMate project, we illustrate how symbolic models can be integrated using an architectural language, how integrated models can be updated using the distinction between symbolic models and their visualization, and how semantic models can be integrated using a new kind of enterprise analysis called semantic analysis. We also suggest that subjective models can be integrated using techniques from natural language analysis.

*2000 Mathematics Subject Classification:* 68N30 Mathematical aspects of software engineering (specification, verification, metrics, requirements, etc.)
*Keywords and Phrases:* Enterprise Architecture, Model Integration

# Integrating Architectural Models: Symbolic, Semantic and Subjective Models in Enterprise Architecture

F. Arbab[a,b] F. de Boer[a,b] M. Bonsangue[b] M.M. Lankhorst[c] H.A. Proper[d] L. van der Torre[a,e]

[a]*CWI, Amsterdam, The Netherlands, EU*

[b]*LIACS, Leiden University, The Netherlands, EU*

[c]*Telematica Instituut, Enschede, The Netherlands, EU*

[d]*ICIS, Radboud University Nijmegen, Nijmegen, The Netherlands, EU*

[e]*Delft University of Technology, Delft, The Netherlands, EU*

**Abstract**

The diversity of architectural models in enterprise architecture is a problem for their integration. In this paper we distinguish three kinds of models from each other and their visualization, and we illustrate how the distinctions can be used for model integration within the architectural approach. Symbolic models express properties of architectures of systems, semantic models interpret the symbols of semantic models, and subjective models are purposely abstracted conceptions of a domain. Building on results obtained in the ArchiMate project, we illustrate how symbolic models can be integrated using an architectural language, how integrated models can be updated using the distinction between symbolic models and their visualization, and how semantic models can be integrated using a new kind of enterprise analysis called semantic analysis. We also suggest that subjective models can be integrated using techniques from natural language analysis.

*Key words:* Enterprise Architecture, Model Integration

*Email addresses:* `F.Arbab@cwi.nl` (F. Arbab), `F.de.Boer@cwi.nl` (F. de Boer), `M.Bonsengue@liacs.nl` (M. Bonsangue), `Marc.Lankhorst@telin.nl` (M.M. Lankhorst), `E.Proper@cs.ru.nl` (H.A. Proper), `L.W.N.van.der.Torre@cwi.nl` (L. van der Torre).

## 1 Introduction

In the development of information systems, software systems, and enterprise architectures, many different architectural descriptions are used, usually in the form of architectural *models*. However, whereas companies have long since recognized the need for an integrated architectural approach, and have developed their own architecture practice, they still experience a lack of support in the design and communication of architectures. For example, when designing architectures, architects do not have a common, well-defined vocabulary to avoid misunderstandings and promote clear designs, that allows for the integration of different types of architectures related to different domains, and that is shared with various stakeholders within and outside the organization, e.g., management, system designers, or outsourcing partners. Other disciplines, for example building and construction, mechanical engineering, or chemical engineering, also use abstractions such as models to describe an object being designed, but have a much more limited and standardized vocabulary and therefore do not seem to face the problems encountered in information technology.

The important distinction between information technology architecture and, for instance, building and construction is that the latter is concerned with information about concrete physical things, whereas the former gives us information about abstractions that need not have a physical counterpart. Therefore, in information technology architecture, we do not have one abstract and one concrete thing, but we have to deal with two abstract things: information systems and models of these systems. One of the main difficulties in dealing with these 'abstractions of abstractions' is that it is much harder for the various stakeholders involved in the design and use of an information system to conceptualize this system than it is for, say, a contractor or inhabitant of a house to think about its structure, functions, and other aspects. The abstract nature of both the object being designed and the descriptions of this design in the form of models leads to at least the following problems:

- Confusion exists with respect to the distinction among a model's presentation, content, and semantics: what does the model look like, what elements does it contain, and what are the relations of these elements to parts of reality (i.e., of the information system)?
- To capture the diverse and abstract nature of information systems often requires the use of multiple large, complex, and interrelated models providing insight into the system from different viewpoints. Comprehending these in their entirety may be a daunting task.
- In IT the technological building blocks, their abilities and their boundaries, are not as clear (and stable) as they are in the other disciplines.
- The architectures are not just referring to technological phenomena, but

also refer to socio-economical phenomena such as business/work processes, etc. This makes it much harder to come up with a limited set of architectural descriptions, models and associated languages.

Due to these reasons, we need in enterprise architecture a more general and flexible approach to the integration of architectural models. In this paper we go beyond the kinds of model integration studied with a long tradition in information systems, and elsewhere, by addressing the following two issues.

- We are not only interested in the static case where architectural models are related to each other and should satisfy some coherence criteria, but we are in particular interested in the dynamic case where models are updated, and as a consequence other models are updated too.
- We are interested not only in syntactic approaches relating one formalism to another one, but we also use the semantics of the models during the integration.

To address these issues, is is essential not to confuse the various uses of "model" in the literature. The colloquial use of the term *model* in enterprise architecture generally refers to a (graphical) symbolic model (viz. the IEEE standard as presented in (IEEE (2000)), the use in UML, etc). When stakeholders refer to architectures and systems, they can do so only by interpreting the symbols in the symbolic models. We call such an interpretation of a symbolic model a *semantic model.* A semantic model does not have a symbolic relation to architecture, as it does not contain symbolic references to reality. However, then stating that the semantic model associated to some given symbolic model captures the meaning of the latter model, we ignore some important issues that are at play when dealing with models in an architecting context. Finally, in some studies such as Falkenberg et al. (1998), models are defined as purposely abstracted conceptions of a domain; we call them *subjective models.*

Note that our three kinds of models are not simple instances of the IEEE standard 1471-2000 concept of model, since they have distinct relations to other concepts. On the contrary, our symbolic model is most closely related to the notion of model in the standard, and the semantic and subjective model can best be seen as new concepts.

The layout of this paper is as follows. We first discuss motivating examples of model integration. Then we introduce and discuss the concepts of symbolic, semantic and subjective model. Finally we demonstrate the usefullness of our distinction by discussing some model integrations in the ArchiMate project.

## 2 Integration of Architectural Domains

In this section we discuss the motivating problem for the distinction among symbolic, semantic and subjective models. We also discuss the relation to complexity of architectures, and compositionality. Although companies have long recognized the need for an integrated architectural approach and have developed their own architecture practice, they still experience a lack of support in the design, communication, realization and management of architectures. Several needs can be categorized as follows with respect to different phases in the architecture life cycle:

**Design** – When designing architectures, architects should use a common, well-defined vocabulary to avoid misunderstandings and promote clear designs. Such a vocabulary must not just focus on a single architecture domain, but should allow for the integration of different types of architectures related to different domains.

**Communication** – Architectures are shared with various stakeholders within and outside the organization, e.g., management, system designers, or outsourcing partners. To facilitate the communication about architectures, it should be possible to precisely represent the relevant aspects for a particular group of stakeholders.

**Realization** – To facilitate the realization of architectures and to provide feedback from this realization to the original architectures, links should be established with design activities on a more detailed level, e.g., business process design, information modeling or software development.

**Change** – An architecture often covers a large part of an organization and may be related to several other architectures. Therefore, changes to an architecture may have a profound impact. Assessing the consequences of such changes beforehand, and carefully planning the evolution of architectures are therefore very important. Until now, support for this is virtually non-existent.

In current practice, enterprise architectures often comprise many heterogeneous models and other descriptions, with ill-defined or completely lacking relations, inconsistencies, and a general lack of coherence and vision. The main driver behind most of the needs identified above is the *complexity* of architectures, their relations, and their use. Many different architectures or architectural views co-exist within an organization. These architectures need to be understood by different stakeholders, each at their own level. The connections and dependencies that exist among these different views make life even more difficult. Management and control of these connected architectures is extremely complex. Primarily, we want to create *insight* for all those that have to deal with architectures. There are many instances of this integration problem, of which we discuss two examples below. In general, some integration problems can be easily solved, for example by using an existing standard; others are intrinsic to the architectural approach and cannot be "solved" in the usual sense. These hard cases are intrinsic to the complexity of architecture, and removing the problem would also remove the notion of architecture itself. This is illustrated by Example 2.1 below.

**Example 2.1**
As a first example of an integration problem, consider Figure 1 that contains several architectures. The five architectures may be models expressed in UML, or models from cells of Zachman's architectural framework, or any kind of combination. For instance, there may be a company that has modelled its applications in UML, and its business processes in BPMN. In all these cases, it is unclear how concepts in one view are related to concepts in another view. Moreover, it is unclear whether views are compatible with each other.

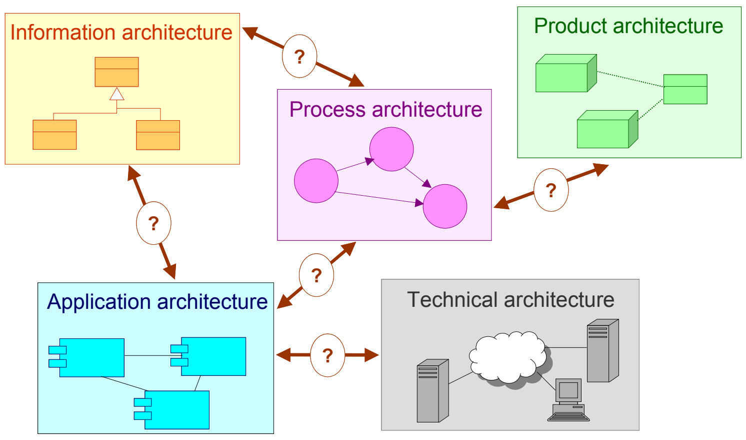


Fig. 1. Heterogeneous architectural domains

The integration of the architectures in Figure 1 is problematic due to the fact that the five architectures are developed by distinct stakeholders, with their own concerns. One might even imagine multiple stakeholders who have distinct models, for example of the process architecture. Relating architectures means relating the ideas of these stakeholders, of which most remain implicit. A consequence is that we often cannot assume to have complete one-to-one mappings, and the best we can ask for is that views are in some sense consistent with each other. This is often called a problem of *alignment*.

In the complex integration cases that involve multiple stakeholders, it is clear that integration is a bottom-up process, in the sense that first concepts and languages of individual architectural domains are defined, and only then the integration of the domains is addressed. We can summarize Example 2.1 by observing that the integration of architectures is hard due to the fact that architectures are given and used in practice, and cannot be changed. It is up to those who integrate these architectures to deal with the distinct nature of architectural domains.

When looking at everyday architectural practice, it is clear that some integration problems occur more frequently than others. A typical pattern is that some architectural models describe the structure of an architecture at some point in time, whereas other models describe how the architecture changes over time. The second example that we discuss in this paper addresses this issue.

**Example 2.2**

As a second example of an integration problem, consider the first two viewpoints discussed in the IEEE 1471-2000 standard (IEEE (2000)): the structural viewpoint and the behavioral viewpoint. How are structure and behavior related?

The second example touches on a problem that has been studied for a long time, the integration of structural and behavioral models. One instance of this problem is how structural concepts like software components are related to behavioral concepts like application functions. Another area where this issue has been studied is in formal methods and in simulation.

In these two examples, compositionality plays a central role in the architectural approach to deal with the complexity of systems. For example, the IEEE 1471 standard defines architecture as the fundamental organization of a system embodied in its components, their relationships to each other, and to the environment (together with principles guiding its design and evolution). Moreover, compositionality also plays a role when varying viewpoints on a system are defined. The latter type of decompositions are usually functional, in the sense that the functionality of an architecture is decomposed in the functionality of its parts and their relations.

## 3 Symbolic, semantic and subjective models in enterprise architecture

To discuss the examples of integrating architectural models, we need a common terminology. Just like architectural diagrams are often misinterpreted due to the fact that each stakeholders interprets the picture in its own way, also architectural concepts are often misinterpreted. This has led to the IEEE 1471 standard which had the ambition to resolve these ambiguities. However, despite the fact that there seems to be increasing consensus on the terminology used, in practice one still finds many distinct definitions of relevant architectural concepts, such as model, meta-model, and view. In this section we therefore define and discuss our terminology.

A *symbolic model* expresses properties of architectures of systems. It therefore contains symbols that refer to reality, which explains the name of this type of models. The role of symbols is crucial, as we do not talk about systems without using symbols. The reason is that systems are parts of reality, and we cannot directly talk about reality as we cannot know the system by itself. A symbolic model is the formalization of one or more aspects of the architecture of a concrete system. It comprises those parts of an architecture that can be modelled mathematically, as opposed to the more pragmatic aspects of an architecture that are concerned with characteristic notions like rationale, goals and plans.

A symbolic model is expressed using a description language, a representation of the model that is often confused with its interpretation. For example the expression $3 + 5$ may be intended to mean a particular natural number, but here is just notation for the syntactic model of the natural numbers. Strictly speaking, a description language describes both the *syntactic structure* of the model and its *notation*, i.e., the words or symbols used for the concepts in the language. We make a strict separation between structure and the notation, and we will use the term 'model' to refer to the structure.

The core of every symbolic model is its *signature*. It categorises the entities of the symbolic model according to some names that are related, linguistically or by convention, to the things they represent. These names are called *sorts*. Relations between entities of some sorts and operations on them are also declared as relation symbols in the signature. After the relations have been specified, they can be used in languages for constraining further or analyzing the nature of the symbolic model. An example is in order here, before we go any further. Figure 2 exhibits a structural description of the employees of a company.

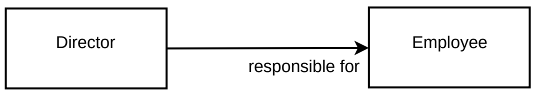


Fig. 2. Syntactic model of director-employee relationship

We need to recall that the above is a syntactic structure, that is, a description of a symbolic model with a signature whose sorts are Employee and Director, and with respective entities related by a relation named responsible_for. As yet we have assigned no meaning to it, we have only categorized the entities of the symbolic model into two categories and named a relation between the entities belonging to two sorts. The syntactic names used for the sorts and relations push our intuition some steps ahead: we know what an employee is, what a director is and what responsible for means. However, while these syntactic names help us in our understanding, they are also the main source of confusion in the communication and analysis of an architecture. We could have named the above sorts $X$ and $Y$ to better retain the meaningless quality of the syntax, and avoid confusion with semantics.

A signature thus provides a conceptual glossary in whose terms everything else in the symbolic model must be described, similar to the English dictionary for the English languages. Additionally, a signature comprises information to capture certain aspects of the ontology of an architecture. For example it may include hierarchical information between sorts in terms of a "is_a" relationship, or containment information in terms of an "includes" relationship, or dependency information in terms of a "requires" relationship. Signatures containing this additional information are more general than a glossary. They provide a conceptual schema, similar to the schema provided to biologist by the species classification.

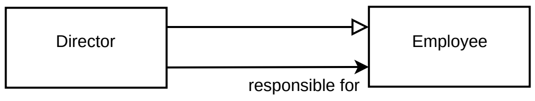


Fig. 3. Extended symbolic model

For example, Figure 3 extends the previous signature with an "is_a" relationship between the sorts Director and Employee, intuitively suggesting that every director is also an employee. Moreover, the symbolic model may also contain a set of actions, and the signature a set of action symbols, the meaning of which we discuss in the following section below.

### *3.2 Semantic Models*

To make the notion of semantics explicit, we distinguish between a symbolic model and a semantic model. When stakeholders refer to architectures and systems, they can do so only by interpreting the symbols in the symbolic models. We call such an interpretation of a symbolic model a *semantic model.* A semantic model does not have a symbolic relation to architecture, as it does not contain symbolic references to reality.

However, there is a relation between semantic model and reality, because a semantic model is an abstraction of the architecture. To understand this relation between semantic model and architectures, one should realize that an important goal of modeling is to predict reality. When a symbolic model makes a prediction, we have to interpret this prediction and test it in reality. The relevant issue in the relation between a system and semantic models of it is how we can translate results such that we can make test cases for the symbolic model. There are various ways in which we can visualize the relation between the four central concepts of enterprise, architecture, symbolic model and semantic model. We put the concept of architecture central, as is illustrated in Figure 4. In general, there can be a large number of different interpretations for the same symbolic model. This reflects the intuition that there can be many architectures that fit a specific architectural description. The signature of a symbolic model of an architecture specifies only some basic building blocks by means of which the architecture is described.

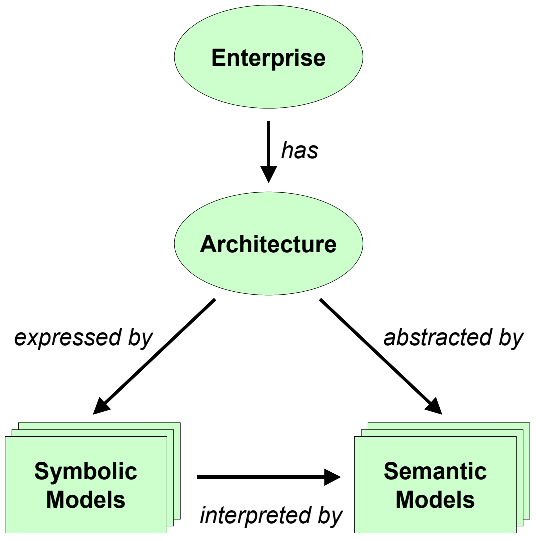


Fig. 4. The enterprise, its architecture, symbolic and semantic models

There are (at least!) two kinds of abstraction we use in creating a model of reality. The first is abstracting from (properties of) the precise entity in reality to which a concept refers. This occurs for example when we make a model of the static structure of an application in terms of its components, leaving out (i.e., abstracting from) their behavior. The second kind is abstraction from differences between entities in reality by grouping them into a single concept. This is sometimes referred to as generalization, and occurs for example when we use the concept 'employee', which groups the individuals in a company. This is related to the notion of 'sorts' discussed below.

The above four concepts and their relations are used in engineering both for informal as well as formal models. The relevant distinction we emphasize between symbolic and semantic models is the distinction between using symbols to refer to reality, and abstractions of reality that only refer to reality by interpreting the symbols of the symbolic model. Note that this is not the same distinction as that between informal and formal models: Within the class of informal models, expressed for example in natural language, both kinds exist, as well as within the class of formal models, expressed for example in first order logic. In the remainder of this section we consider formal semantics only.

The semantics of a modeling language is given by a *semantic model*, an interpretation of the symbolic model. A semantic model usually assumes the existence of some mathematical objects (sets for example), used to represent the basic elements of a symbolic model. Operations and relations of a symbolic model are mapped to usually better understood operations and relations amongst the mathematical objects. As an example, the formal semantics of a signature is provided by a collection of sets (one for each sort of the signature), and a set of relations and functions among them, one for each relation symbol and function symbol in the signature. Hierarchical information between sorts is captured by the ordinary subset inclusion, whereas containment information is denoted by the usual element-of relation.

In other words, we see the formal semantics of a symbolic model as a concrete collection of mathematical objects interpreting a system according to a specific architectural description. As such, it involves concrete components and their concrete relationships which may change in time because of the dynamic behavior of a system. Concrete situations of a system are described by means of variables typed according to the sort of the individuals they are referring to. More concretely, for a symbolic model, we will denote by $x : T$ a variable $x$ which ranges over individuals of sort $T$. For example, we could use the logical sentence:

$$\exists x : \mathsf{Director}.\forall y : \mathsf{Employee.Responsible\_for}(x, y)$$

to constraint the interpretation of the sort Director to be a non-empty set. Note that since Director is_a Employee, also the interpretation of the latter sort will be non-empty.

An important issue in the semantics of architectural models is the meaning of actions. The actions occurring in a symbolic model are interpreted as changes of the model based on interaction with the user. To define actions, we have to define the input variables of the action, and how we can retrieve these input variables from the user. In Section 4.3 we further discuss this issue. As illustrated in Section 4, these formal semantics are rich enough to capture the dynamics of a system by interpreting the symbolic (and often pictorial) information available for describing business and software processes in ArchiMate.

### *3.3 Subjective models*

Besides symbolic and semantic models, one finds in the enterprise architecture literature references to a third kind of model, in particular in linguistic, psychological or social theories. Here we refer to this kind of models as subjective models.

For example, the FRISCO Framework of Information system concepts defines a model as a purposely abstracted, clear, precise and unambiguous conception.

To understand this framework, consider the relationships between stakeholder, enterprise, architecture, and architecture description expressed in the form of a tetrahedron in Figure 5 (which is a specialization of the FRISCO tetrahedron Falkenberg et al. (1998)). Different stakeholders, have a different view of the world. Not everyone's needs can be easily accommodated by a single model. They therefore first consider what happens if some viewer observes 'the universe' around him. FRISCO assumes that any viewer that perceives the world around him first produces a conception, i.e., a mental representation, of that part he deems relevant. Such a conception cannot be communicated about directly, unless it is articulated somehow. In other words, a conception needs to be represented. They argue that the distinction between subjective model on the one hand and semantic and symbolic model on the other hand goes back to long philosophical tradition. In particular, Peirce (1969a,b,c,d) argues that both the perception and conception of a viewer are strongly influenced by his interest in the observed universe.

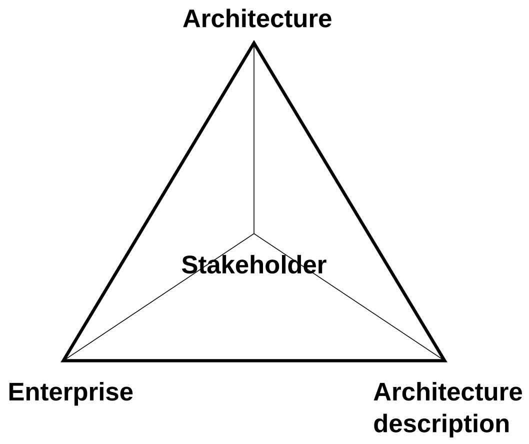


Fig. 5. Relationship between enterprise, stakeholder, architecture, and architecture description

Finally, in architecture often a distinction is made between the *architectural semantics* and the *formal semantics* of a modeling language. The enterprise under consideration is thought of in terms of architecture concepts, which exist in the minds of, e.g., the enterprise architect. These concepts can be represented in models, which are expressed in a modeling language. Architectural semantics is defined as the relationship between architectural concepts and their possible representations in a modeling language (Turner (1987)). To understand this distinction, consider Venn diagrams. They are useful structures for the visualization of the language of Boolean logic, but they are not a model themselves. Their semantic model is given by the set-theoretical explanation of their meaning. The formal semantics of a model, on the other hand, is a mathematical representation of specific formal properties of that model. The formal semantics of a computer program, for example, expresses the possible computations of that program. Different branches of formal semantics exist, such as denotational, operational, axiomatic, and action semantics.

## 4 Integration of models in ArchiMate

In this section we illustrate how the distinction among symbolic, semantic and subjective modelbetween the various kinds of models is used in the integration of architectural models, based on results from the ArchiMate project.

### *4.1 Integration of symbolic models (static case)*

The basis for model integration is an architectural description language, called the ArchiMate language. Service orientation may typically lead to a layered view of enterprise architecture models, where the service concept is one of the main linking pins between the different layers. *Service layers* with services made available to higher layers are interleaved with *implementation layers* that realize the services. Within a layer, there may also be *internal* services, e.g., services of supporting applications that are used by the end-user applications. How this leads to a stack of service layers and implementation layers is shown in Figure 6. These are linked by *used by* relations, showing how the implementation layers make use of the services of other (typically 'lower') layers, and *realization* relations, showing how services are realized in an implementation layer. In this context, we distinguish three main layers:

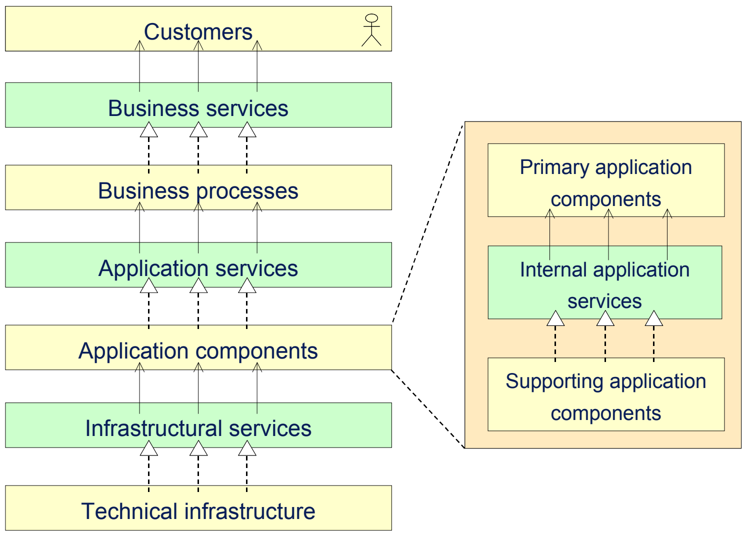


Fig. 6. Layered view

- The *business layer* offers products and services to external customers, which are realized in the organization by business processes (performed by business actors or roles).

- The *application layer* supports the business layer with application services which are realized by (software) application components.
- The *technology layer* offers infrastructural services (e.g., processing, storage, and communication services) needed to run applications, realized by computer and communication devices and system software.

A premise of the ArchiMate language is that the general structure of models within the different layers is similar. The same types of concepts and relations are used, although their exact nature and granularity differ. As a result of this uniformity, models created for the different layers can be aligned with each other quite easily. Within each layer, the language is structured according to the three dimensions: internal-external, individual-collective, and behavior-structure. Figure 7 shows the core concepts that are found in each layer along these dimensions.

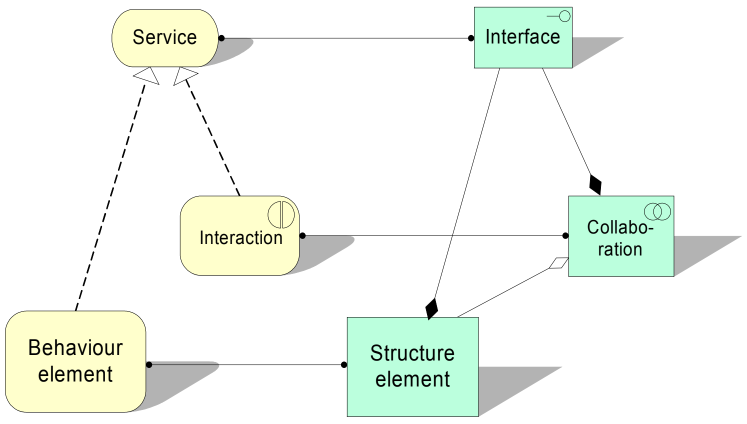


Fig. 7. The core concepts in three dimensions

As an example, Figure 8 presents two models, a diagram and a landscape map Sanden and Sturm (1997). The diagram on the left canvas visualizes five products on the left, five business functions on the right, and ten application components in the middle. The landscape map on the right canvas visualizes an easy to understand 2D 'map'. The two models refer to the same architecture. Moreover, in this particular case the landscape map has been automatically generated from the underlying model.

A more detailed exposition of the ArchiMate language and its uses can be found in Lankhorst and et al (2005). It is developed and tested in cooperation with several companies. It is a coarse grained language, which facilitates the integration of symbolic models. However, the use of a symbolic language also has its limitations, in particular when we are interested in changing models, and when the symbolic models have semantics which have to be respected. These two issues are discussed in the following two subsections.

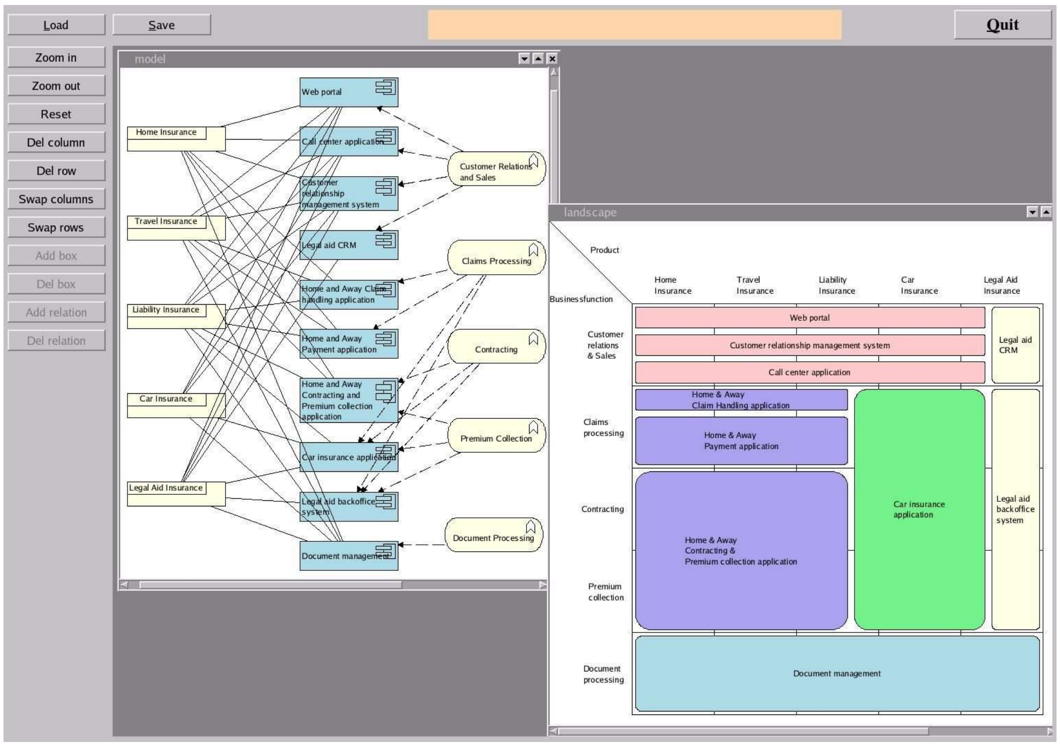


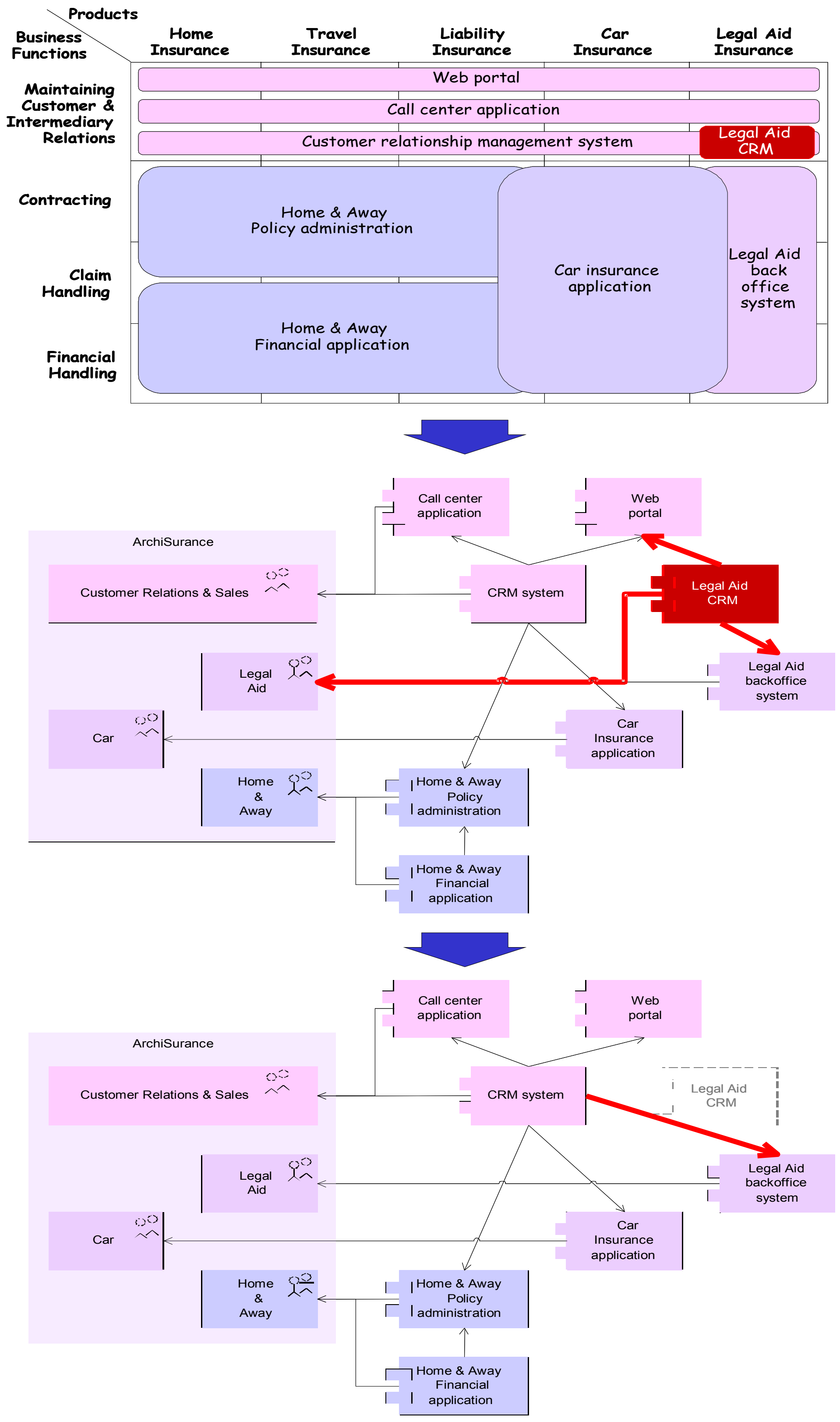
Fig. 8. Model with associated landscape map view

## *4.2 Integration of symbolic models: dynamic case*

Reconsider the model together with its landscape map in Figure 8, and assume that they are integrated in the sense that the landscape map is generated from the diagram. Now assume moreover that someone changes one of these two models. Then it may be the case that the models are no longer integrated. The problem of the dynamic case of symbolic model integration is to develop techniques to ensure that the models stay integrated.

We introduce special actions-in-models. They are defined in terms of the effects they have on elements of the underlying model. For example, consider a view on a business process model, and an action that merges two processes into a single process. Issues that are relevant for this action are the effects of the merger, for example the removal of processes, the addition of a new process, or the transfer of some relations from an old, removed process to a new process.

Mapping a seemingly simple change to the landscape map onto the necessary modifications of the model may become quite complicated. Since a landscape map abstracts from many aspects of the underlying model, such a mapping might be ambiguous: many different modifications to the model might correspond to the same change of the landscape map. Human intervention is required to solve this, but a landscape map tool might suggest where the impact of the change is located.



Products
Business Functions
Home Insurance
Travel Insurance
Liability Insurance
Car Insurance
Legal Aid Insurance
Maintaining Customer & Intermediary Relations
Contracting
Claim Handling
Financial Handling
Web portal
Call center application
Customer relationship management system
Legal Aid CRM
Home & Away Policy administration
Home & Away Financial application
Car insurance application
Legal Aid back office system
Call center application
Web portal
ArchiSurance
Customer Relations & Sales
CRM system
Legal Aid CRM
Legal Aid
Legal Aid backoffice system
Car
Car Insurance application
Home & Away
Home & Away Policy administration
Home & Away Financial application
Call center application
Web portal
ArchiSurance
Customer Relations & Sales
CRM system
Legal Aid CRM
Legal Aid
Legal Aid backoffice system
Car
Car Insurance application
Home & Away
Home & Away Policy administration
Home & Away Financial application

Fig. 9. Editing a landscape map

In the example of Figure 9, you may for instance want to remove the seemingly redundant Legal Aid CRM system by invoking a 'remove overlap' operation on this object. This operation influences both the visualization and the architectural model. Figure 9 illustrates the effects of the operation on the underlying model. First, you select the object to be removed, in this case the Legal Aid CRM system. The envisaged tool colors this object and maps it back onto the underlying object in the architecture. Next, the relations connecting this object to its environment are computed, possibly using the impact-of-change analysis techniques described in the following section (the second part of Figure 9). Here, this concerns the relations of Legal Aid CRM with the Web portal and the Legal Aid back-office system. These relations will have to be connected to one or more objects that replace the objects that are to be removed. Since we have chosen a 'remove overlap' operation, the landscape tool computes with which other objects Legal Aid CRM overlaps, in this case the CRM system. The relations formerly connecting Legal Aid CRM are then moved to the other CRM system, unless these already exist (e.g., the relation with the Web portal).

Naturally, this scenario presents an ideal situation with minimal user intervention. In reality, a tool cannot always decide how a proposed change is to be mapped back onto the model, and may only present the user with a number of options. For example, if the functionality of the Legal Aid CRM system would overlap with more than one other system, remapping its relations requires knowledge about the correspondence between these relations and the functions realized by these other systems.

## *4.3 Integration of semantic models*

We can go beyond the syntactic approach of integrating symbolic models by taking their semantics into account. In particular, we show that formal methods can be used when we introduce a few basic definitions we briefly explained before, such as signature, symbolic model and interpretation. Our approach can be contrasted with the original approach in UML, see also the related work section of this paper. In this approach, semantics was explicitly left out of the programme. People who used the models can develop semantics for them, but a general semantics was not supplied. This approach also stemmed from the origins of the UML as a combination of three existing notations that did not have formal semantics. Hence, the focus of UML was and is on notation, i.e., syntax, and not on semantics. Although some of the diagrams of the more recent versions of UML have a formal semantics, see, e.g., the token-based Petrinet-like semantics of activity diagrams in UML 2.0, there is no overall semantics for the entire language. We have taken the opposite approach. We do not put the notation of the language central, but rather focus on the meaning of the language concepts and their relations. Of course, any modeling language needs a notation and we do supply a standard way of depicting the ArchiMate concepts, but this is subordinate to the architectural semantics of the language.

For dynamics of architectures, functional analysis techniques based on formal approaches such as process algebras and data flow networks are useful. Issues like two roles acting at the same time, overwriting or destroying each other's work, can be identified and then a suitable protocol can be designed to prevent the problem. Thus, a functional behavior analysis based on formal methods is primarily a qualitative analysis that can detect logical errors, leads to a better consistency and focuses on the logic of models.

The dynamics of a concrete system with an architectural description given by its signature can be specified in different ways; we distinguish between specifications tailored towards control flow modeling and those tailored towards data flow modeling. For control flow modeling, we give a brief introduction into process algebra, while for data flow modeling, we introduce the reader into data flow networks.

To illustrate the use of these formal methods, we use the enterprise architecture of a small company, ArchiSell, modelled using the ArchiMate language. In ArchiSell, employees sell products to customers. Various suppliers deliver the products to ArchiSell. Employees of ArchiSell are responsible for ordering products and for selling them. Once products are delivered to ArchiSell, each product is assigned to an owner responsible for selling the product. More specifically, we look at the business process architecture for ordering products, visualized in Figure 10. To describe this enterprise we use the ArchiMate modeling concepts and their relationships. In particular, we use structural concepts (*product*, *role* and *object*) and structural relationships (*association*), but also behavioral concepts (process) and behavioral relationships (*triggering*). Behavioral and structural concepts are connected by means of the *assignment* and *access* relations.

In order to fulfill the business process for ordering a product, the employee has to perform the following activities:

(1) Before placing an order, an employee must register the order within the Order Registry. This Order Registry is for administration purposes. It is used to check orders upon acceptance of goods later in the process. Orders contain a list of products to be ordered.
(2) After that, the employee places the order with the supplier. Based on the order, the supplier is supposed to collect the products and to deliver them as soon as possible.
(3) As soon as the supplier delivers the products, the employee first checks if

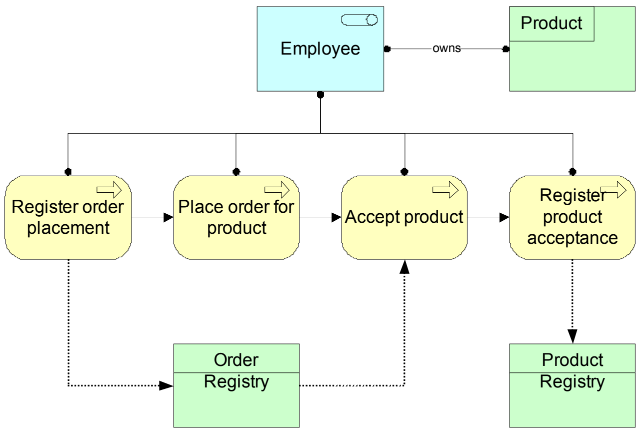


Fig. 10. An example business-process architecture

there is an order that refers to this delivery. Then, the employee accepts the products.

(4) Next, the employee registers the acceptance of the products within the Product Registry and determines which employee will be the owner of the products.

Although the example is rather trivial, it serves to illustrate how an architecture description can be formalized and how it can be subjected to functional analysis.

To obtain a formal model of a system as a semantic interpretation of the symbolic model of its architectural description, we start with an interpretation of the signature. An *interpretation* $\mathcal{I}$ of the types of a signature assigns to each primitive sort $S$ a set $\mathcal{I}(S)$ of *individuals* of sort $S$ which respects the sub-sort ordering: if $S_1$ is a sub-sort of $S_2$ then $\mathcal{I}(S_1)$ is a subset of $\mathcal{I}(S_2)$. Any primitive sort is interpreted by a subset of a universe which is given by the union of the interpretation of all primitive sorts. The subset relation expresses the hierarchy between primitive sorts. An interpretation $\mathcal{I}$ of the primitive sorts of a signature of an architecture can be inductively extended to an interpretation of more complex types. For example, an interpretation of the product type $T_1 \times T_2$ is given by the Cartesian product $\mathcal{I}(T_1) \times \mathcal{I}(T_2)$ of the sets $\mathcal{I}(T_1)$ and $\mathcal{I}(T_2)$. The function type $T_1 \rightarrow T_2$ thus denotes the set of all functions from the universe to itself such that the image of $\mathcal{I}(T_1)$ is contained in $\mathcal{I}(T_2)$. In general, there can be a large number of different interpretations for a signature. This reflects the intuition that there are many possible architectures that fit a specific architectural description.

The semantic model of a system involves its concrete components and their concrete relationships, which may change in time because of the dynamic behavior of a system. To refer to the concrete situation of a system, we have to extend its signature with names for referring to the individuals of the types and relations. For a symbolic model, we denote by $n : T$ a name $n$, which ranges over individuals of type $T$.

To formalize the behavior of a system using this semantic model, we can, for instance, use process algebra. Process algebra (Baeten and Weijland (1990); Bergstra et al. (2001)) is a formal description technique for specifying the control flow behavior of complex systems. Starting from the language syntax, each statement of the language is supplied with some kind of behavior, and a semantic equivalence says which behaviors are identical. Process algebras express such equivalences in *axioms* or *equational laws.* The axioms are to be *sound*, i.e., if two behaviors can be equated then they are semantically equivalent. The converse statement is optional, and is called *completeness*, i.e., if two behaviors are semantically equivalent then they can be equated.

The system is captured as a set of processes interacting with each other according to predefined rules. Starting from a set of *basic actions*, processes may be hierarchically composed by means of operators, e.g., sequential composition, choice, parallel composition.

We derive these basic actions from the functions of a symbolic model of an architecture. Now let us consider the process steps within the ArchiSell example. Within the process algebra interpretation, processes are specified as functions. The types of the arguments and result values are determined as follows:

- A *role*, which is assigned to a process specifies the type of both an argument and a result value of the corresponding function.
- An outgoing *access* relation from a process to a data object specifies the type of both an argument and a result value of the corresponding function.
- An incoming access relation from an object to a process only specifies the type of the corresponding argument (this captures the property of 'read-only').

This results in the following functions:

```
Register_order_placement:
  domain name = Employee
  domain name = Order_Registry
  codomain name = Employee
  codomain name = Order_Registry
```

A data flow network (Jagannathan (1995)) is another formal description technique for the behavioral specifications of complex systems. Such a network consists of some *processes*, the functions of a symbolic model that communicate by passing data over *lines.* A process is a transformation of data within the system, whereas a line is a directed FIFO channel connecting at most two processes. Data passed over a line by a process will arrive in an unspecified but finite amount of time at the destination process in the same order as they are sent.

Data flow diagrams can be used to provide a clear representation of any business function. The technique starts with an overall picture of the business and continues by analyzing each of the functional areas of interest. This analysis can be carried out to the level of detail required. The technique exploits a method called top-down expansion to conduct the analysis in a targeted way. The result is a series of diagrams that represent the business activities in a way that is precise, clear and easy to communicate.

In a data flow interpretation of the ArchiSell process, we consider each individual process step as an independent data-consuming/data-producing entity. Such an entity has *input ports* and *output ports*. Within the data flow interpretation we are interested in the data flow within the process, but not directly in the actors (or roles) that perform the process. Therefore, this interpretation is specifically suited for situations in which many details are known about the data and less about the actors. However, as we will illustrate, a data flow interpretation can help us in the assignment of actors to process steps.

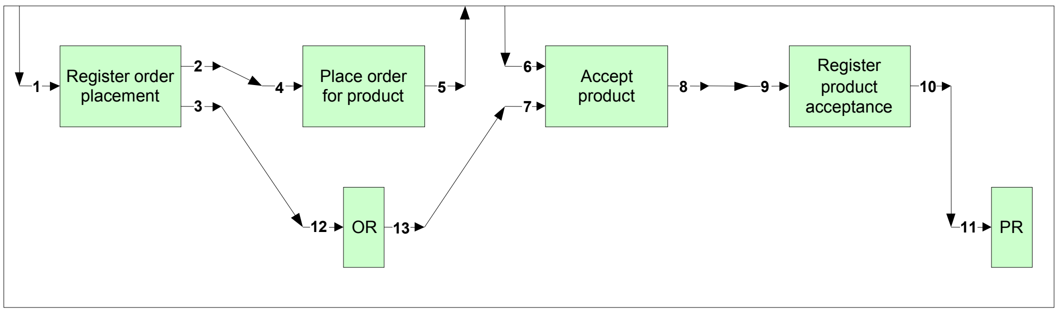


Fig. 11. A example data flow network

Figure 11 illustrates the way in which we can interpret the example as a data flow network. Note the following:

- We leave out any information about roles and individuals within the role sort. So, the data flow diagram does not contain information about which actor performs which process steps.
- We specify registries as stores, i.e., special functions, which resemble places in which information can be stored and from which the same information can be retrieved later.
- We explicitly identify which input/output ports receive/send which kind of values. A practical way is to begin with identifying the values on the input/output ports, and then to connect the output ports to other input ports.

### *4.4 Integration of subjective models*

Just like semantic models are important to enterprise architecture because they are a bridge to formal methods and theoretical computer science, subjective models are important to enterprise architecture as they are a bridge to for example linguistic, psychological and social theories. Consequently, using semantic models we argue that this distinction with symbolic models facilitates (or opens up) the use of formal methods in enterprise architecture, here using subjective models we argue that its distinction with symbolic and semantic models facilitates the use of (computational) linguistic methods in enterprise architecture.) For example, subjective models may be expressed in natural language and consequently formal machinery developed in linguistics (eg around van Benthem in the Amsterdam school of logic, but also Kamp's DRT or any other classical technique) can be used within (enterprise) architecture.

The ArchiMate project has not directly addressed these ideas, but present research at University of Nijmegen is looking at tools to aid groups of actors to disambiguate models (such as architectural models) and **also** ground their common understanding. This requires a combination between formal approaches and communicative approaches from social sciences. The distinction will be important for the following applications:

- To create a mutual understanding among stakeholders, we need to ensure that the subjective models that they harbor are as similar as possible. Because of their different backgrounds, fields of expertise, needs, and possibly even their psychological make-up, different stakeholders may need distinct symbolic models to arrive at approximately the same subjective model.
- Especially important in this respect is to bring about a successful communication on relations among different domains described by different architectures (e.g., processes vs. applications), since this will often involve multiple groups of stakeholders. Clear communication is also very important in the case of outsourcing of parts of the implementation of an architecture to external organizations. The original architect is often not available to explain the meaning of a design, so the architecture should speak for itself.

## 5 Related Work

A wide variety of organization and process modeling languages are currently in use. The conceptual domains that are covered differ from language to language. In many languages, the relations between domains are not clearly defined. Some of the most popular languages are proprietary to a specific software tool. Relevant languages in this category include the ebXML set of standards for XML-based electronic business Team (2001), developed by OASIS and UN/CEFACT, IDEF of Commerce (1993), originating from the US Ministry of Defence, ARIS (Scheer (1994)), part of the widely used ARIS Toolset, and the Testbed language for business process modeling (Eertink et al. (1999)). Recent standardization efforts in this area are carried out by the Business Process Management Initiative, with the graphical Business Process Modeling Notation BPMN (BPMI (2003)) as its main result. Support for this language from vendors of business process modeling and enterprise architecture tools is increasing. However, BPMN's scope is limited to business processes and it does not provide concepts for modeling e.g. organizational structures, data models, or the relation between business activities and supporting IT applications, making it of limited use in enterprise architecture.

The Reference Model for Open Distributed Processing (ISO (1998)) is a joint ISO/ITU-T standard for the specification open distributed systems. It defines five viewpoints on an ODP system that each has their own specification language. Important for enterprise architecture is the enterprise viewpoint, which describes purpose, scope and policies of a system, the RM-ODP Enterprise Language has been defined in which, e.g., business objectives and business processes can be modelled Union (2002).

In contrast to organization and business process modeling, where there is no single, standard modeling language, in software modeling the Unified Modeling Language (UML) (Booch et al. (1999)) has become a true world standard. UML is the mainstream modeling approach within IT, and its use is expanding into other areas, e.g., in business modeling (Eriksson and Penker (1998)). Compared to the earlier versions, the support for architectural modeling has improved in the recent UML 2.0 standard (OMG (2003)).

The UML has a so-called profile for Enterprise Distributed Object Computing (EDOC), which provides an architecture and modeling support for collaborative or Internet computing, with technologies such as web services, Enterprise Java Beans, and Corba components (OMG (2002)). This makes UML an important language not only for modeling software systems, but also for business processes and for general business architecture. The UML has either incorporated or superseded most of the older IT modeling techniques still in use. However, it is not easily accessible and understandable for managers and business specialists; therefore, special visualizations and views of UML models should be provided. Another important weakness of the UML is the large number of diagram types, with poorly defined relations between them. This is another illustration of the lack of integration discussed in the introduction of this paper. Given the importance of the UML, other modeling languages will likely provide an interface or mapping to it.

Most languages mentioned above provide concepts to model, e.g., detailed business processes, but not the relationships between different processes. They are therefore not particularly suited to model architectures (IEEE (2000)). Architecture description languages (ADLs) define high-level concepts for architecture description, such as components and connectors. A large number of ADLs have been proposed, some for specific application areas, some more generally applicable, but mostly with a focus on software architecture. Medvidovic and Taylor (2000) describe the basics of ADLs and compare the most important ADLs with each other. Most have an academic background, and their application in practice is limited. However, they have a sound formal foundation, which makes them suitable for unambiguous specifications and amenable to different types of analysis. The ADL ACME Garlan et al. (1997) is widely accepted as a standard to exchange architectural information, also between other ADLs. There are initiatives to integrate ACME in UML, both by defining translations between the languages and by a collaboration with OMG to include ACME concepts in UML 2.0 Group (2003). In this way, the concepts will be made available to a large user base and be supported by a wide range of software tools. This obviates the need for a separate ADL for modeling software systems. The Architecture Description Markup Language (ADML) was originally developed as an XML encoding of ACME.

Finally, another important trend is OMG's Model Driven Architecture (MDA) approach (Frankel (2003)). Although it strongly leans on OMG standards such as UML, the applicability of the approach is not limited to specific languages. MDA comprises three abstraction levels:

- The requirements for the system are modelled in a Computation Independent Model (CIM) describing the situation in which the system will be used. Such a model is sometimes called a subjective model or a business model. It hides much or all information about the use of automated data processing systems.
- The Platform Independent Model (PIM) describes the operation of a system while hiding the details necessary for a particular platform. A PIM shows that part of the complete specification that does not change from one platform to another.
- A Platform Specific Model (PSM) combines the specifications in the PIM with the details that specify how that system uses a particular type of platform.

UML is endorsed as the modeling language for both PIMs and PSMs. At the CIM level, which roughly corresponds with the enterprise-architectural level at which the ArchiMate ideas are targeted, things are less clear.

## 6 Conclusion

A model is an abstract and unambiguous representation of something (in the real world) that focuses on specific aspects or elements and abstracts from other elements, based on the purpose for which the model is created. Because of their formalized structure, models lend themselves to various kinds of automated processing, visualization, analysis, tests, and simulations. Furthermore, the rigour of a model-based approach also compels architects to work in a more meticulous way and helps to dispel the unfavorable reputation of architecture as just drawing some "pretty pictures."

An integrated architectural approach is indispensable to control today's complex organizations and information systems. It is widely recognized that a company needs to "do architecture"; the legacy spaghetti of the past has shown us that business and IT development without an architectural vision leads to uncontrollable systems that can only be adapted with great difficulty. However, architectures are seldom defined on a single level. Within an enterprise, many different but related issues need to be addressed. Business processes should contribute to an organization's products and services, applications should support these processes, systems and networks should be designed to handle the applications, and all of these should be in line with the overall goals of the organization. Many of these domains have their own architecture practice, and hence different aspects of the enterprise will be described in different architectures. These architectures cannot be viewed in isolation.

For example, architectural domains are related, and structural and behavioral viewpoints are related. The integration has to deal with the the fact that the various viewpoints are defined by stakeholders with their own concerns.

The core of our approach to enterprise architecture is therefore that multiple domains should be viewed in a coherent, integrated way. We provide support for architects and other stakeholders in the design and use of such integrated architectures. To this end, we have to provide adequate concepts for specifying architectures on the one hand, and on the other hand support the architect with visualization and analysis techniques that create insight in their structure and relations. In this approach, relations with existing standards and tools are to be emphasized; we aim to integrate what is already available and useful. The approach that we follow is very generic and systematically covers both the necessary architectural concepts and the supporting techniques for visualization, analysis and use of architectures.

A distinction is made between the content of a view and its visualization, and a distinction is made between a symbolic model that refers to the enterprise architecture, and a semantic model as an abstraction from the architecture and which interprets the symbolic model. The core of every symbolic model is its signature, which categorises the entities of the symbolic model.

## Acknowledgment

This paper results from the ArchiMate project (http://archimate.telin.nl/), a research initiative that has provided concepts and techniques to support enterprise architects in the visualisation, communication and analysis of integrated architectures. The ArchiMate consortium consists of ABN AMRO, Stichting Pensioenfonds ABP, the Dutch Tax and Customs Administration, Ordina, Telematica Instituut, Centrum voor Wiskunde en Informatica, Radboud Universiteit Nijmegen, and the Leiden Institute of Advanced Computer Science.

## References

Baeten, J., Weijland, W., 1990. Process Algebra. Vol. 18 of Tracts in Theoretical Computer Science. Cambridge University Press, Cambridge, United Kingdom, EU.

Bergstra, J., Ponse, A., Smolka, S. (Eds.), 2001. Handbook of Process Algebra. Elsevier Science Publishers, Amsterdam, The Netherlands, EU.

Booch, G., Rumbaugh, J., Jacobson, I., 1999. The Unified Modelling Language User Guide. Addison-Wesley, Reading, Massachusetts, USA. ISBN 0-201-57168-4

BPMI, 2003. The business process modeling notation. Working draft (1.0), Business Process Management Initiative.
URL `http://www.bpmi.org`

Eertink, H., Janssen, W., Oude Luttighuis, P., Teeuw, W., Vissers, C., 1999. A business process design language. In: Proceedings of the First World Congress on Formal Methods. Toulouse, France, EU.

Eriksson, H.-E., Penker, M., 1998. Business Modeling with UML: Business Patterns at Work. Wiley, New York, New York, USA.

Falkenberg, E., Verrijn-Stuart, A., Voss, K., Hesse, W., Lindgreen, P., Nilsson, B., Oei, J., Rolland, C., Stamper, R. a. (Eds.), 1998. A Framework of Information Systems Concepts. IFIP WG 8.1 Task Group FRISCO, IFIP, Laxenburg, Austria, EU. ISBN 3-901-88201-4

Frankel, D., 2003. Model Driven Architecture: Applying MDA to Enterprise Computing. Wiley, New York, New York, USA.

Garlan, D., Monroe, R., Wile, D., 1997. Acme: An architecture description interchange language. In: Proceedings of CASCON '97. Toronto, Canada, pp. 169–183.

Group, O. M., 2003. Unified Modeling Language: Superstructure. Final Adopted Specification ptc/03-08-02. Http://www.omg.org/docs/ptc/03-08-02.pdf.

IEEE, September 2000. Recommended Practice for Architectural Description of Software Intensive Systems. Tech. Rep. IEEE P1471-2000, The Architecture Working Group of the Software Engineering Committee, Standards Department, IEEE, Piscataway, New Jersey, USA.
URL `http://www.ieee.org` ISBN 0-738-12518-0

ISO, 1998. Information technology – Open Distributed Processing – Reference model: Overview. ISO/IEC 10746-1:1998(E).
URL `http://www.iso.org`

Jagannathan, R., 1995. Dataflow models. In: Zomaya, E. (Ed.), Parallel and Distributed Computing Handbook. McGraw-Hill, New York, New York, USA.

Lankhorst, M., et al, 2005. Enterprise Architecture at Work : Modelling, Communication and Analysis. Springer, Berlin, Germany, EU. ISBN 3-5402-4371-2

Medvidovic, N., Taylor, R., January 2000. A classification and comparison framework for software architecture description languages. IEEE Transactions on Software Engineering 26 (1), 70–93.

of Commerce, U. D., 1993. Integration Definition for Function Modeling (IDEF0) Draft. Federal Information Processing Standards Publication FIPSPUB 183. Springfield, VA, USA.

OMG, 2002. Uml profile for enterprise distributed object computing specification. Tech. rep., The Object Management Group.
URL `http://www.omg.org/docs/ptc/03-09-05.pdf`

OMG, August 2003. UML 2.0 Superstructure Specification – OMG Draft Adopted Specification. Tech. Rep. ptc/03-08-02, The Open Management Group.
URL `http://www.omg.org`

Peirce, C., 1969a. Volumes I and II – Principles of Philosophy and Elements of Logic. Collected Papers of C.S. Peirce. Harvard University Press, Boston, Massachusetts, USA. ISBN 0-674-13800-7

Peirce, C., 1969b. Volumes III and IV – Exact Logic and The Simplest Mathematics. Collected Papers of C.S. Peirce. Harvard University Press, Boston, Massachusetts, USA. ISBN 0-674-13800-5

Peirce, C., 1969c. Volumes V and VI – Pragmatism and Pragmaticism and Scientific Metaphysics. Collected Papers of C.S. Peirce. Harvard University Press, Boston, Massachusetts, USA. ISBN 0-674-13802-3

Peirce, C., 1969d. Volumes VII and VIII – Science and Philosophy and Reviews, Correspondence and Bibliography. Collected Papers of C.S. Peirce. Harvard University Press, Boston, Massachusetts, USA. ISBN 0-674-13803-1

Sanden, W. v. d., Sturm, B., 1997. Informatie-architectuur – de infrastructurele benadering. Panfox, Rosmalen, The Netherlands, in Dutch. ISBN

9080127027

Scheer, A.-W., 1994. Business Process Engineering: Reference Models for Industrial Enterprises, 2nd Edition. Springer, Berlin, Germany, EU.

Team, B. P. P., 2001. ebXML Business Process Specification Schema Version 1.01. Http://www.ebxml.org/specs/ebBPSS.pdf.

Turner, K., 1987. An architectural semantics for lotos. In: Proceedings of the 7th International Conference on Protocol Specification, Testing, and Verification. pp. 15–28.

Union, I. T., 2002. Information technology - Open Distributed Processing - Reference Model - Enterprise Language. ITU-T Recommendation X.911 ISO/IEC 15414.