Farhad Arbab, Frank de Boer, Marcello Bonsangue,
Marc Lankhorst, Erik Proper, Leendert van der Torre

Integrating Architectural Models

Symbolic, Semantic and Subjective Models in Enterprise Architecture

The diversity of architectural models in enterprise architecture poses a problem to their integration. Without such integration the effectiveness of these models in the process of architecting enterprises diminishes. In this paper we make a distinction between three classes of models. We will illustrate how the distinctions can be used for model integration within the architectural approach. Symbolic models express properties of architec-tures of systems, semantic models interpret the symbols used in symbolic models, and subjective models are purposely abstracted conceptions of a domain. Building on results obtained in the ArchiMate project, we il-lustrate 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.

1 Introduction

In the development of enterprises and information systems, many different architectural descriptions are used, usually in the form of architectural *models*. However, while 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 term architecture has been used in the field of information technology since the 1960's. In the early days it was used to refer to the principles underlying the design of computer hardware and operating systems. This led to the use of the term computer architecture. Later, when software applications became larger and larger, researchers such as Mary Shaw and David Garlan coined the term software architecture [ShGa96]. This notion of architecture deals with the key design principles underlying software artefacts. A dedicated IEEE working group [IEEE00] has defined it as follows:

> An architecture is the fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution.

In the United States of America, certain government agencies are required by law to have an IT architecture. This is laid down in the so-called Clinger-Cohen Act[1]. In this act, architecture is defined as:

> The term 'information technology architecture', with respect to an executive agency, means an integrated framework for evolving or maintaining existing information

[1] http://www.cio.gov/Documents/it_management_reform_act_Feb_1996.html

technology and acquiring new information technology to achieve the agency's strategic goals and information resources management goals.

The Open Group's architecture work group[2] provides two definitions of architecture depending on the context:

- A formal description of a system, or a detailed plan of the system at component level to guide its implementation.
- The structure of components, their interrelationships, and the principles and guidelines governing their design and evolution over time.

Architectures are usually described in terms of models [IEEE00]; architectural models. These architectural model can either have a design oriented nature (architectural blueprints, see e.g. [Boar99]) or a regulative nature (architecture principles, see e.g. [TaCa93]). What distinguishes an architectural model from other models, is the role the former models play (quoting the earlier mentioned IEEE definition) in expressing the fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution.

The work reported in this paper is the result of the ArchiMate project[3], which involved a consortium comprising ABN AMRO Bank, Stichting Pensioenfonds ABP, the Dutch Tax and Customs Administration, Ordina, Telematics Institute, Centrum voor Wiskunde en Informatica, Radboud University Nijmegen, and the Leiden Institute of Advanced Computer Science. The aim of the project was to provide concepts and techniques [JVB+03, JLB+04] to support enterprise architects in the visualization, communication and analysis of integrated architectures.

Before the ArchiMate project was initiated a survey was conducted with the industrial partners. The aim of this survey was to gain an understanding of the problems these organizations were suffering from with regards to their use of architectures. Using the outcome [Bosm02] of this initial survey the goals of the ArchiMate project were set [Lan+05]. As part of this survey, it was found that 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 information technology 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 it was concluded [Lan+05] that a more general and flexible approach to the integration of architectural models was called for. In doing so, this paper will 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 as well.
- 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, it is essential not to confuse the various uses of 'model' in 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 [IEEE00], the use in UML, etc). The interpretation of such a symbolic model in terms of a formal language (such as logic or set theory) is referred to as a *semantic model*. A semantic model does not have a symbolic relation to architecture, as it does not contain symbolic references to reality. However, stating that the

[2] http://www.togaf.org
[3] http://www.archimate.com

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. What is still missing is the (inherently subjective) nature of human interpretation of these models. In some studies such as [FVV+98], models are defined as purposely abstracted conceptions (as held by a human viewer) of a domain; we call them subjective models. It should be noted that the field of enterprise architecture requires a perspective involving both computer science, information systems and business sciences perspective.

This paper should also be understood from such a mixed perspective. In enterprise architecture, both the formal and informal worlds meet, as well as the physical, social and informational worlds. The results of the ArchiMate project, as reported in [Lan+05], are indeed being used in practice by companies within the Netherlands as well as abroad. Tool vendors such as BizzDesign, IDS Scheer and Troux also provide (certified!) support for the ArchiMate modelling language. Currently, an ArchiMate foundation (www.archimate.com) is furthering the use and evolution of ArchiMate.

2 Integration of Architectural Domains

The aim of this section is to provide a motivation for the distinction among symbolic, semantic and subjective models. We provide some examples which aim to illustrate the problem, each of which are based on the issues brought forward in the aforementioned initial survey [Bosm02] of the ArchiMate project.

As mentioned before, even though companies have long since recognized the need for an integrated architectural approach and have indeed developed their own architecture practice, they still suffer from a lack of support in the design, communication, realization and management of architectures and related models. Several categories of needs with regards to architectural models can be identified. With respect to the different phases in the architecture life cycle, we identify the following categories of needs:

- 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 architectural models related to different domains.
- Communication - Architectural models 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 & integration - 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. These links should be established between different plains of realization (conceptual, logical and physical), as well as between different aspects (information, process, services, etc).
- Change - An architecture often covers a large part of an organization and may be related to several architectural models. Therefore, changes to an architecture may have a profound impact. Analysis of impact of change is also needed to select between different design alternatives. One alternative may be able to better absorb anticipated changes than another. Assessing the consequences of such (potential) 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

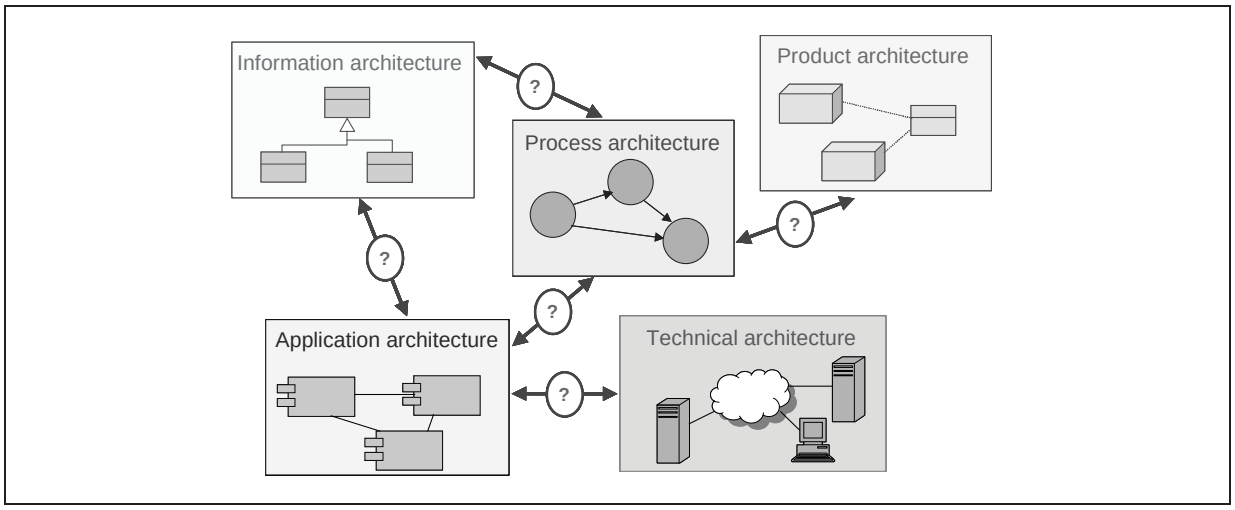

Figure 1: Heterogeneous architectural domains

the complexity of architecture, and removing the problem would also remove the notion of architecture itself. This is illustrated by the example below.

> Consider Figure 1, which contains several architectural models. The five architectures may be expressed as models in UML, or models from cells of the 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.

The integration of the architectural models in Figure 1 is likely to be problematic due to the fact that they have been developed by distinct stakeholders, with their own concerns. 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.

In complex integration cases involving 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 0.1 by observing that the integration of architectural models 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 models to deal with the distinct nature of architectural domains.

In every organization there are likely to be some (architectural) models which have not been integrated, simply because integration takes time and effort. In some cases, the integration is not worth the costs and effort. However, lack of integration means that certain questions involving multiple models cannot be answered. A particular problem in enterprise architecture is that due to a lack of model integration, stakeholders in an organization do not have access to all the same/relevant information. In extreme cases, they may even have conflicting information. The industrial partners of the ArchiMate project regarded the information mismatch due to lack of integration between architectural models as being one of the stumbling blocks on the road to better business/IT alignment [HeVa93]. At the same time, however, quantifying this jointly held belief was found to be hard, given the fact that there are many other interfering issues in business/IT alignment, of course.

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 above discussed example illustrates how compositionality also introduces integration problems. Finally, the importance of model integration, and its challenges, also comes to the fore in the move towards model-driven software development. In the context of MDA [Fran03], the modelling techniques from the original UML [BoRJ99] need to be integrated better. It is no secret, however, that the relationships between the original UML diagramming techniques is not always explicit and unambiguous.

In the more recent version [OMG03] of the UML, part of these problems has been remedied.

3 Symbolic, semantic and subjective models in enterprise architecture

To discuss the integration of architectural models, a common terminology is needed. Just like architectural diagrams are often misinterpreted due to the fact that each stakeholder interprets the picture in its own way, also architectural concepts are often misinterpreted. Despite the fact that there seems to be an increasing consensus on the terminology used, for example brought forward by the efforts of the IEEE 1471 working group, 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. More specifically, we introduce the notions of subjective, symbolic and semantic model. These three classes of models will be discussed in more detail below. The distinction between these three classes of models is essentially based on Morris' meaning triangle [Morr46], where a distinction is made between a ''sign'' (symbolic model), ''object'' (semantic model) and ''concept'' (subjective model).

3.1 Symbolic models

A *symbolic model* expresses properties of architectures of systems. As such it 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. Symbolic models are the formalization of one or more aspects of the architecture of a concrete system.

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 in this case it should just be regarded as notation being part of 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* (as used in first order logic). 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.

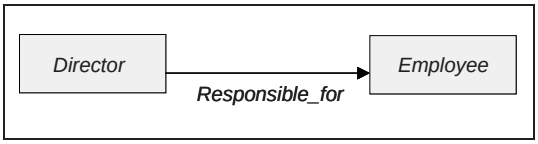

Figure 2: Syntactic model of director-employee relationship

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.

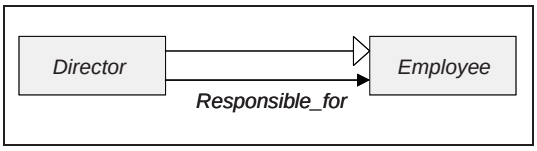

Figure 3: Extended symbolic model

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. There is, however, a relation between a 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/mimic a planned/pre-existing reality. When a symbolic model makes a prediction, we have to interpret this prediction and test it in reality.

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.

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 behaviour. 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.

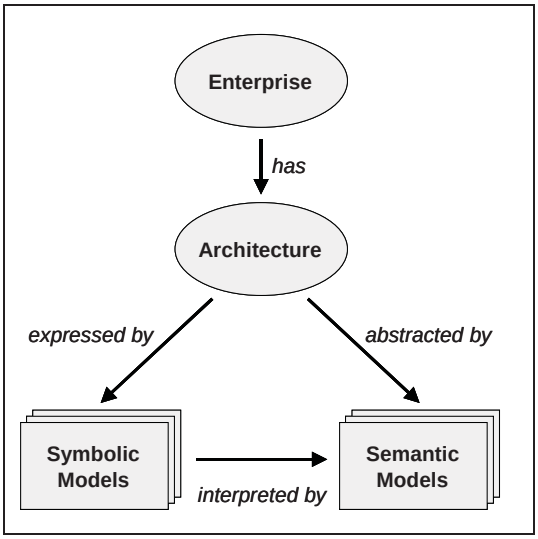

Figure 4: The enterprise, its architecture, symbolic and semantic models

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 the one 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.

The semantics of a modelling 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.

3.3 Subjective models

Besides symbolic and semantic models, one finds in the enterprise architecture 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.

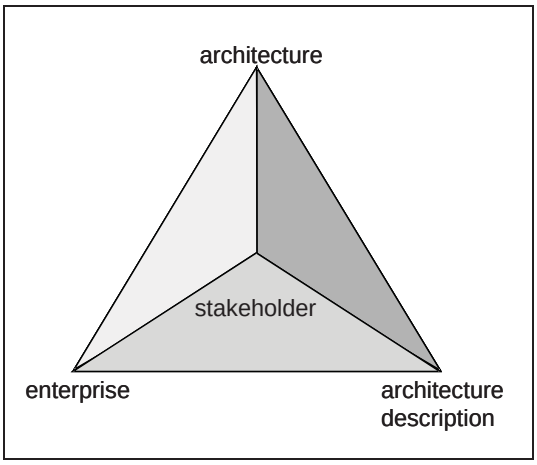

Figure 5: Relationship between enterprise, stakeholder, architecture, and architecture description

For example, the FRISCO Framework of Information system concepts defines a model as a purposely abstracted, clear, precise and unambiguous conception. This notion of model is what we refer to as a *subjective* model. To better 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 [FVV+98]). 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 [Peir69] argues that both the perception and conception of a viewer are strongly influenced by his interest in the observed universe. As mentioned before, the distinction between subjective, symbolic and semantic model can also traced back to Morris' meaning triangle [Morr46], where a distinction is made between a "sign" (*symbolic model*), "object" (*semantic model*) and "concept" (*subjective model*).

4 Integration of models in ArchiMate

In this section we illustrate how the distinction between symbolic, semantic and subjective models is used in the integration of architectural models, based on results from the ArchiMate project. In doing so, we will use the fictive ArchiSurance case, which was also used in [Lan+05]. Our work on integration of architecture models has, however, also been applied to real-life cases provided by the industrial partners of 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 [Lan+05]. 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:

- 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 behaviour-structure. Figure 7 shows the core concepts that are found in each layer along these dimensions.

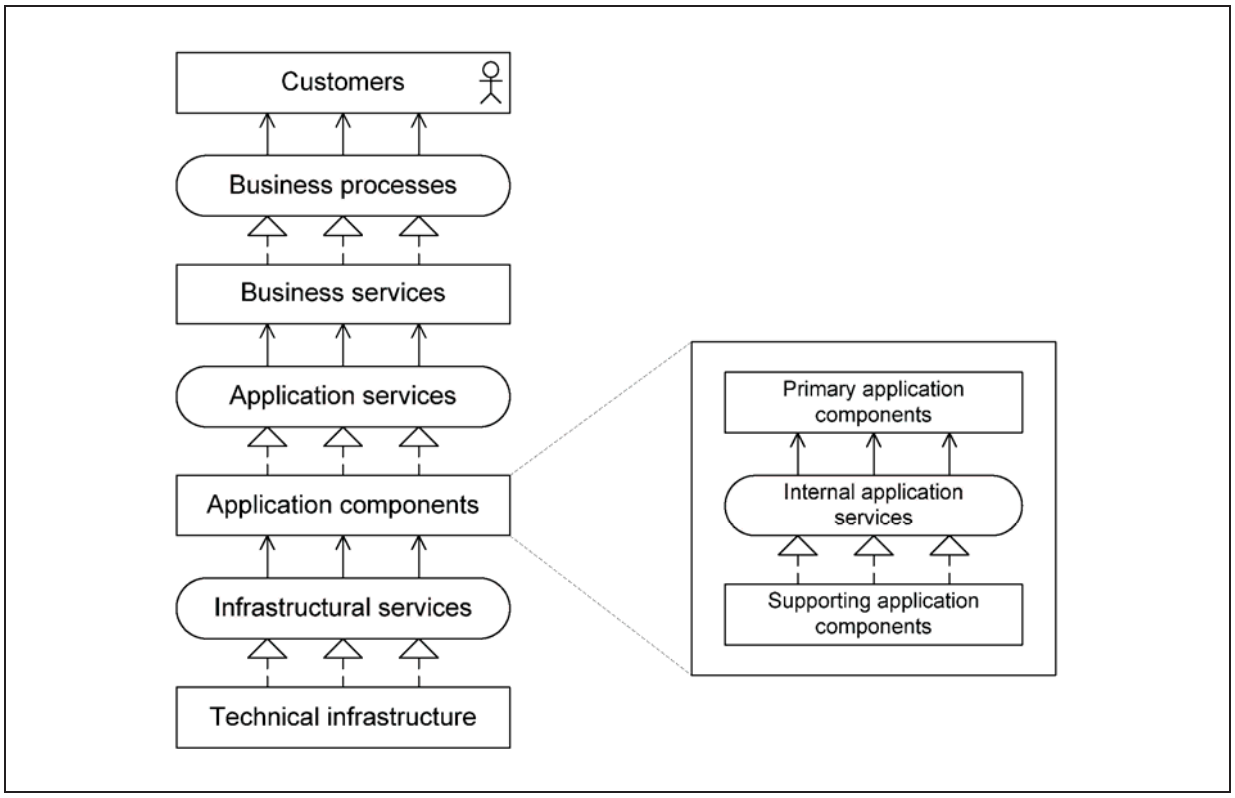

Figure 6: Layered view

As an example, Figure 8 presents two models, a diagram and a landscape map [SaSt97]. 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 [Lan+05]. The language 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.

4.2 Integration of symbolic models – Dynamic case

Reconsider the situation depicted 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 remain 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.

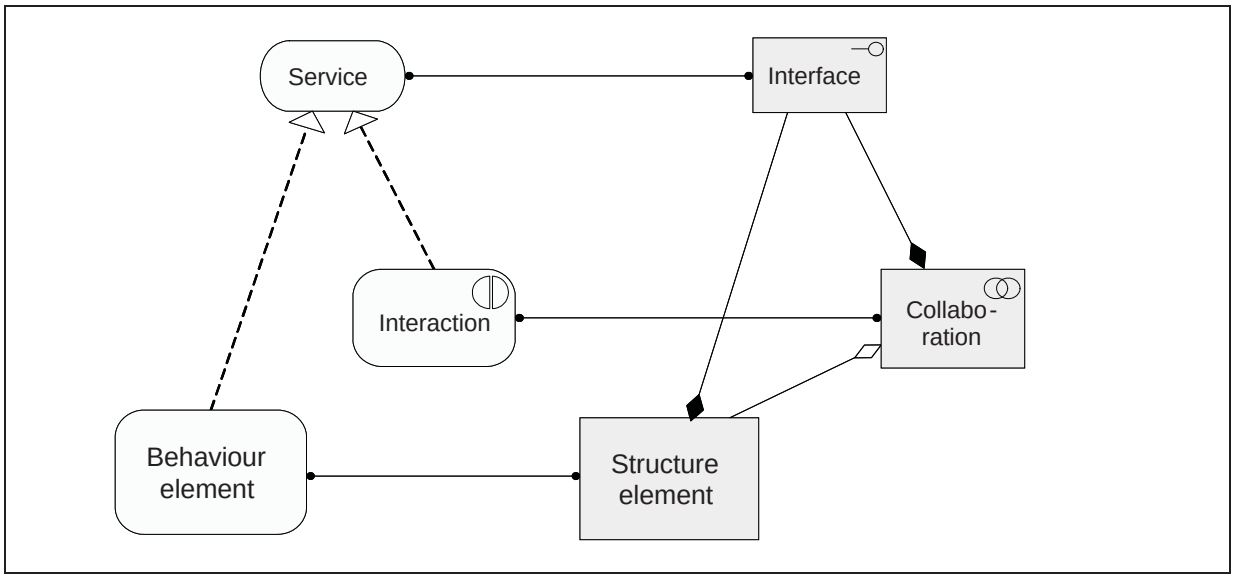

Figure 7: The core concepts in three dimensions

In the example of Figure 9, one 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, one selects 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. 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.

For architecture models dealing with dynamical aspects, functional analysis techniques based on formal approaches such as process algebras and data flow networks are useful. Issues such as 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 behaviour 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 modelling and those tailored towards data flow modelling. For control flow modelling, we give a brief introduction into process algebra, while for data flow modelling, 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, while various suppliers deliver the products to ArchiSell. Employees of ArchiSell are responsible for

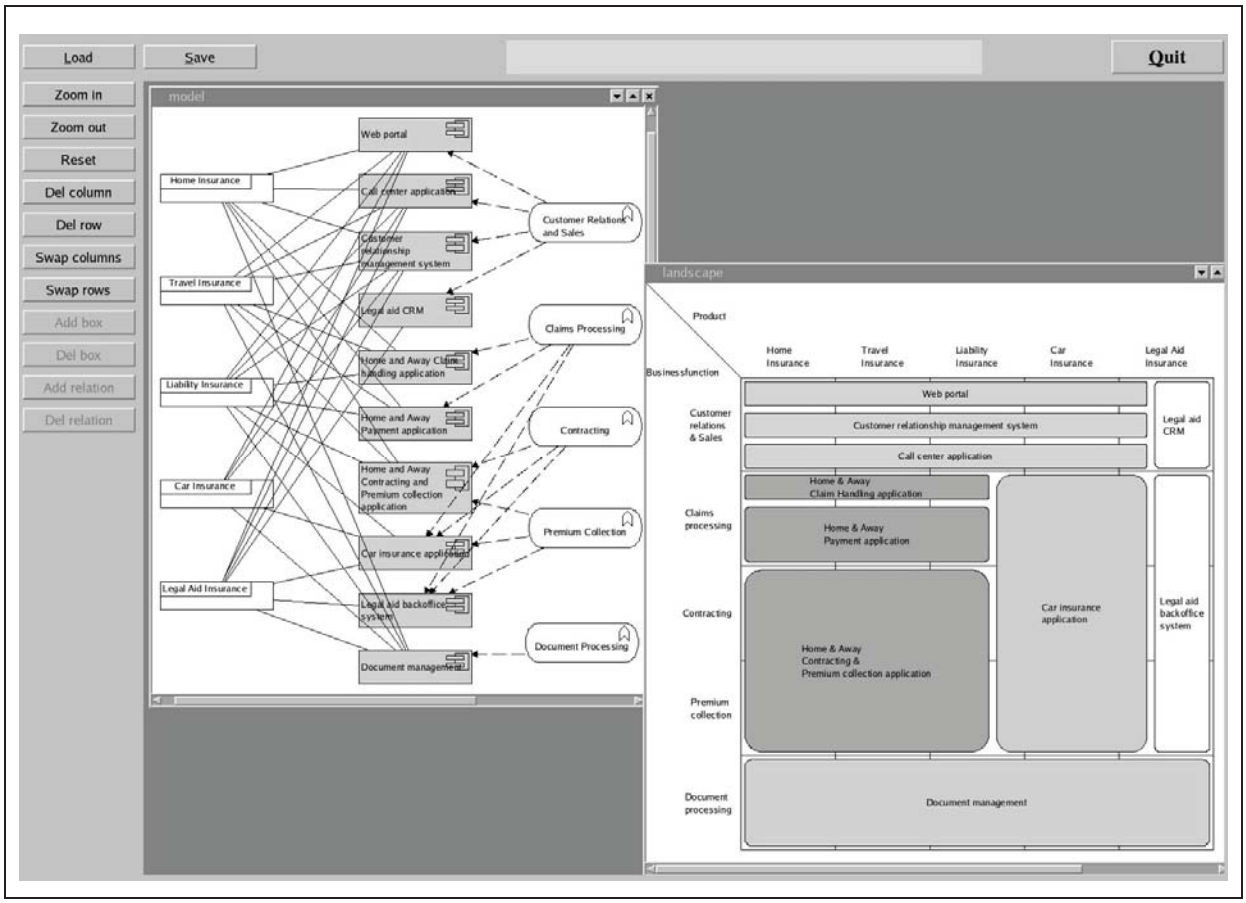

Figure 8: Model with associated landscape map view

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 modelling concepts and their relationships. In particular, we use structural concepts (*product*, *role* and *object*) and structural relationships (*association*), but also behavioural concepts (process) and behavioural relationships (*triggering*). Behavioural and structural concepts are connected by means of the *assignment* (the lines with the black dots) and *access* (the dotted lines with arrow) relations.

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

- 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.
- 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.
- As soon as the supplier delivers the products, the employee first checks if there is an order that refers to this delivery. Then, the employee accepts the products.
- 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 I* of the types of a

signature assigns to each primitive sort S a set $I(S)$ of *individuals* of sort S which respects the sub-sort ordering: if S_1 is a sub-sort of S_2 then $I(S_1)$ is a subset of $I(S_2)$. 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 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 $I(T_1) \times I(T_2)$ of the sets $I(T_1)$ and $I(T_2)$. 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 $I(T_1)$ is contained in $I(T_2)$. 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 [BaWe90, BePS01] is a formal description technique for specifying the control flow behavior of complex systems. Note, however, that process algebra is mainly intended to express the semantics of the actual flow of the process and only to a lesser extend for the formalization of resource allocations, etc. 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.

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.

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 as 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.

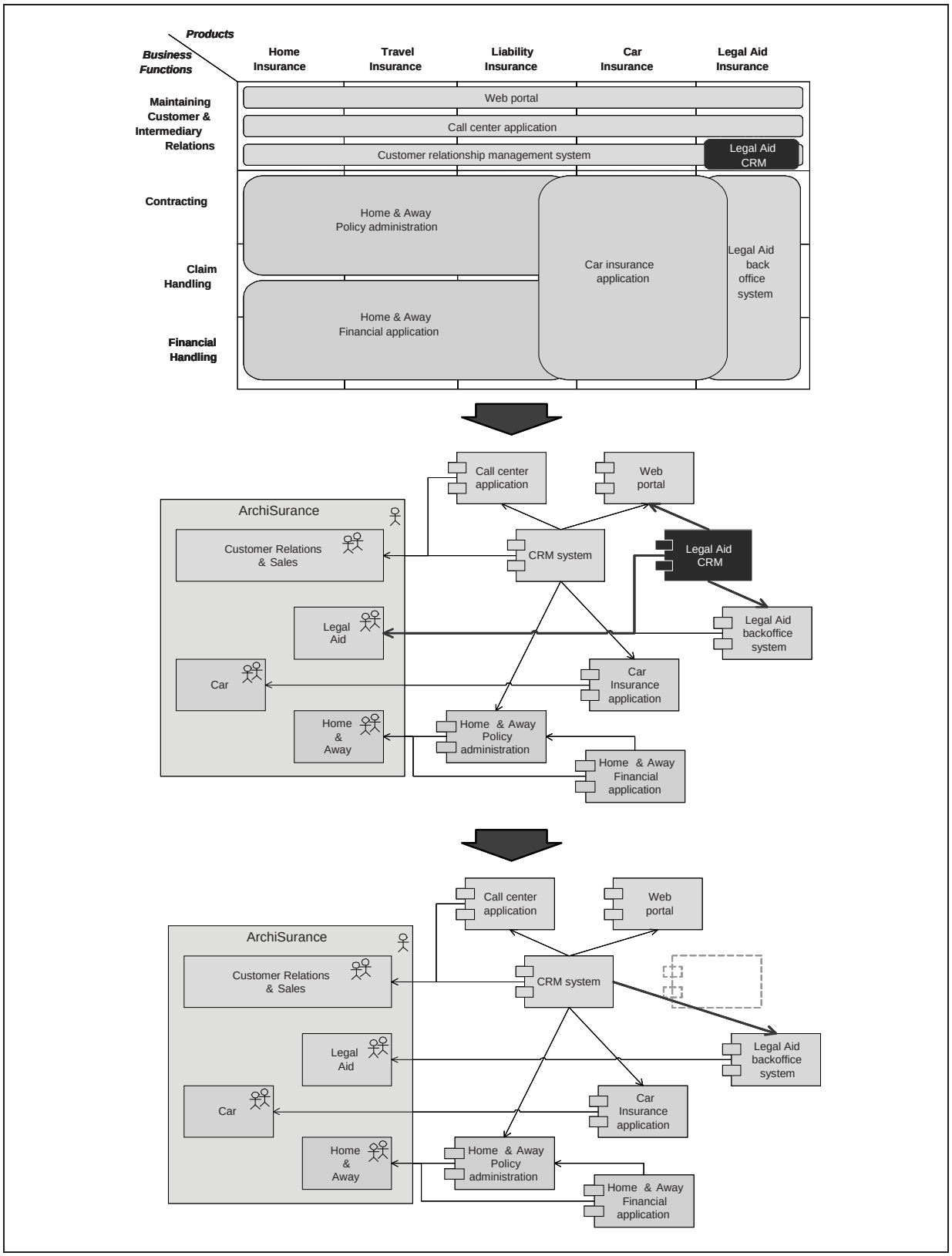

Figure 9: Editing a landscape map

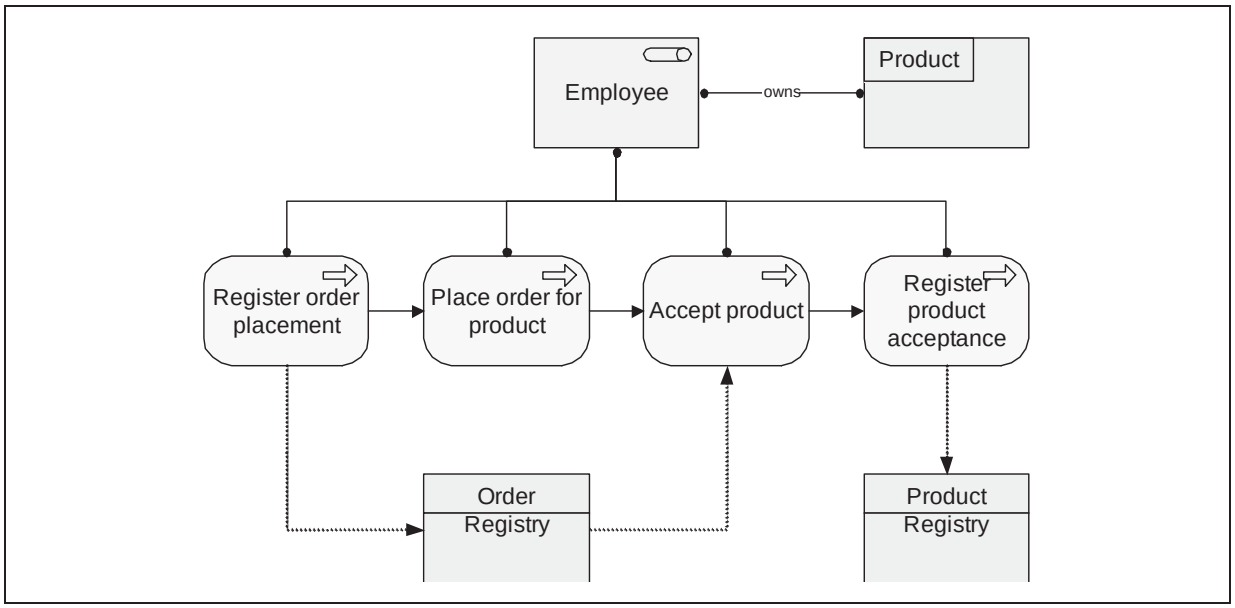

Figure 10: An example business-process architecture

The ArchiMate project has not directly addressed these ideas, but current research within our group is indeed looking at ways to aid groups of actors to disambiguate models (such as architectural models) and also ground their common understanding [HoBP05, HoPW05a, HoPW05d, HoPW05b, PrVH05, HoPR05, HoPW05c]. 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 harbour 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 modelling 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 [Busi01], developed by OASIS and UN/CEFACT, IDEF [USAD93], originating from the US Ministry of Defence, PCE [Sche94], part of the widely used ARIS Toolset, and the Testbed language for business process modelling [EJO+99]. Recent standardization efforts in this area are carried out by the Business Process Management Initiative, with the graphical Business Process Modelling Notation BPMN [BPMI03] as its main result. Support for this language from vendors of business process modelling and enterprise architecture tools is increasing. However, the scope of these languages is typically limited to business processes alone. They tend not to provide concepts for modelling e.g. organizational structures, data models, or the relation/integration between business activities and supporting IT applications, making it of limited use in enterprise architecture.

In contrast to organization and business process modelling, where there is no single, standard modelling language, in software modelling the Unified Modelling Language (UML) [BoRJ99] has become a true world standard. UML is the mainstream model-

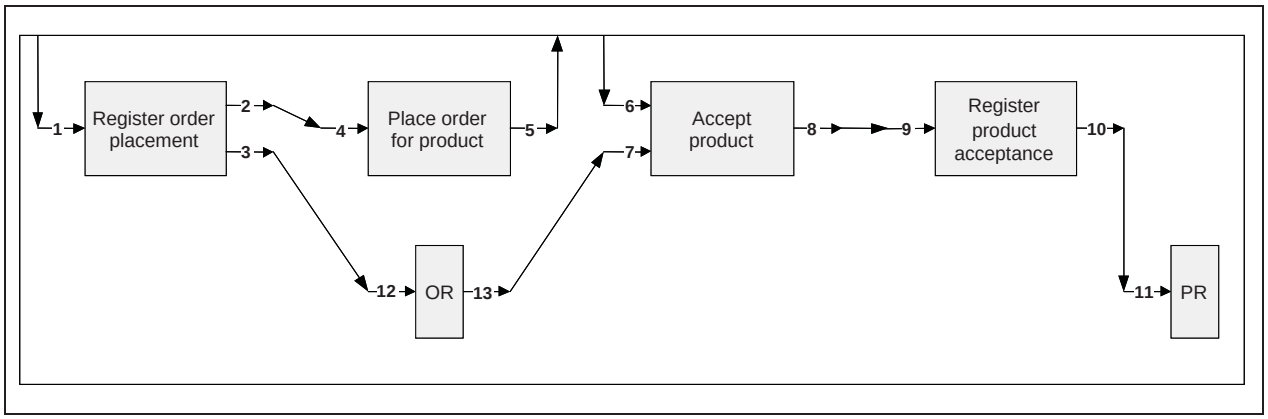

Figure 11: An example data flow network

ling approach within IT, and its use is expanding into other areas, e.g., in business modelling [ErPe98]. Compared to the earlier versions, the support for architectural modelling has improved in the recent UML 2.0 standard [OMG03]. Still, at present, the integration between the different models remains limited, although the advent of model-driven system development also requires such integration [Fran03]. Furthermore, the modelling languages used in the UML standard are of limited use to enterprise architecture. The UML has a so-called profile for Enterprise Distributed Object Computing (EDOC), which provides an architecture and modelling support for collaborative or Internet computing, with technologies such as web services, Enterprise Java Beans, and Corba components [OMG02]. This makes UML an important language not only for modelling software systems, but also for business processes and for general business architecture. The UML has either incorporated or superseded most of the older IT modelling 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 modelling 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 ([1999-IEEE-Architecture]). 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. [MeTa00] 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 [GaMW97] 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 [OMG03]. 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.

Another important trend is OMG's Model Driven Architecture (MDA) approach [Fran03]. 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 modelling 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 still less clear.

Finally, work on ontology engineering is also relevant to the issue of model integration. Ontology engineering [Guar98] starts from the same semantic interpretation of symbolic models as we discussed in section typically in a description logic [BCM+03]. Though description logics are a useful combination of expressive power and computational efficiency, other alternatives to describe relations among concepts are for example relational algebras. In this paper we have not discussed these areas (see e.g. the ArchiMate book [Lan+05] for a further discussion), but raised the question relevant in architecture about representation and reasoning about dynamics. Within the area of description logic (see e.g. the description logic handbook [BCM+03]) various extensions have been proposed to description logics. What we have shown here and in more detail in [BBJS05], based on the same signature we can also use modelling and analysis techniques from other formal methods like process algebra or dataflow networks. Summarizing, whereas the work on ontology is relevant for enterprise architecture as well as many other areas, such as for example the semantic web, we focus in this paper on issues which have been raised in our project as particular for enterprise architecture: how to deal with dynamics of models, how to interpret the actions in behaviour models, where we make the point that starting from a given signature extracted from a diagram, they can even be interpreted in completely distinct models like process algebras and dataflow networks.

6 Conclusion

A *subjective* 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. Subjective models are represented using some *symbolic model*, which has a formal semantics leading to a *semantic model*. 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 unfavourable 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.

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.

Finally, as mentioned in the introduction, it should be noted that the results of the ArchiMate project are indeed being used increasingly in industry. Both in the Netherlands and beyond, while tool vendors such as BizzDesign, IDS Scheer and Troux provide support for the ArchiMate language. Maintenance and proliferation of the language is managed and monitored by the ArchiMate foundation[4].

[4] http://www.archimate.com

7 References

[BBJS05] F.S. de Boer, M.M. Bonsangue, J. Jacob, and A. Stam. Enterprise Architecture Analysis with XML. In *Proceedings of the 38th Annual Hawaii International Conference on System Sciences (HICSS'05), Big Island, Hawaii, USA*, page 222, Piscataway, New Jersey, USA, January 2005. IEEE Computer Society Press.

[BCM+03] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, editors. *The Description Logic Handbook - Theory, Implementation and Applications*. Cambridge University Press, Cambridge, United Kingdom, EU, January 2003.

[Boar99] B.H. Boar. *Constructing Blueprints for Enterprise IT architectures*. Wiley, New York, New York, USA, 1999.

[Bosm02] H. Bosma. Requirements. Technical Report ArchiMate/D4.1, December 2002.

[BPMI03] BPMI. The Business Process Modeling Notation. Technical report, Business Process Management Initiative, 2003.

[BePS01] J.A. Bergstra, A. Ponse, and S.A. Smolka, editors. *Handbook of Process Algebra*. Elsevier Science Publishers, North Holland, 2001.

[BoRJ99] G. Booch, J. Rumbaugh, and I. Jacobson. *The Unified Modelling Language User Guide*. Addison Wesley, Reading, Massachusetts, USA, 1999.

[Busi01] Business Process Project Team. *ebXML Business Process Specification Schema Version 1.01*. 2001.

[BaWe90]J.C.M. Baeten and W.P. Weijland. *Process Algebra*. Cambridge University Press, Cambridge, United Kingdom, EU, 1990.

[EJO+99] H. Eertink, W. Janssen, P. Oude Luttighuis, W. Teeuw, and C. Vissers. A Business Process Design Language. In *Proceedings of the First World Congress on Formal Methods*, 1999.

[ErPe98] H.-E. Eriksson and M. Penker. *Business Modeling with UML: Business Patterns at Work*. Wiley, New York, New York, USA, 1998.

[Fran03] D.S. Frankel. *Model Driven Architecture: Applying MDA to Enterprise Computing*. Wiley, New York, New York, USA, 2003.

[FVV+98]E.D. Falkenberg, A.A. Verrijn-Stuart, K. Voss, W. Hesse, P. Lindgreen, B.E. Nilsson, J.L.H. Oei, C. Rolland, and R.K. and Stamper, editors. *A Framework of Information Systems Concepts*. IFIP WG 8.1 Task Group FRISCO, IFIP, Laxenburg, Austria, EU, 1998.

[GaMW97] D. Garlan, R.T. Monroe, and D. Wile. ACME: An Architecture Description Interchange Language. In *Proceedings of CASCON '97*, pages 169-183, 1997.

[Guar98] N. Guarino. Formal Ontology and Information Systems. In N. Guarino, editor, *Proceedings of FOIS'98, Trento, Italy, EU*, pages 3-15, Amsterdam, The Netherlands, EU, June 1998. IOS Press.

[HoBP05] S.J.B.A. Hoppenbrouwers, A.I. Bleeker, and H.A. (Erik) Proper. Facing the Conceptual Complexities in Business Domain Modeling. *Computing Letters*, 1(2):59-68, 2005.

[HoPR05] S.J.B.A. Hoppenbrouwers, H.A. (Erik) Proper, and V.E. van Reijswoud. Navigating the Methodology Jungle - The communicative role of modelling techniques in information system development. *Computing Letters*, 1(3), 2005.

[HoPW05a] S.J.B.A. Hoppenbrouwers, H.A. (Erik) Proper, and Th.P. van der Weide. A Fundamental View on the Process of Conceptual Modeling. In *Conceptual Modeling - ER 2005 - 24 International Conference on Conceptual Modeling*, volume 3716 of *Lecture Notes in Computer Science*, pages 128-143, June 2005.

[HoPW05b] S.J.B.A. Hoppenbrouwers, H.A. (Erik) Proper, and Th.P. van der Weide. Formal Modelling as a Grounded Conversation. In G. Goldkuhl, M. Lind, and S. Haraldson, editors, Proceedings of the 10th International Working Conference on t he Language Action Perspective on Communication Modelling (LAP'05), pages 139-155, Kiruna, Sweden, EU, June 2005. Linköpings Universitet and Hogskolan I Boras, Linköping, Sweden, EU.

[HoPW05c] S.J.B.A. Hoppenbrouwers, H.A. (Erik) Proper, and Th.P. van der Weide. Towards explicit strategies for modeling. In T.A. Halpin, K. Siau, and J. Krogstie, editors, Proceedings of the Workshop on Evaluating Modeling Methods for Systems Analysis and Design (EMMSAD'05), held in conjunctiun with the 17th Conference on Advanced Information Systems 2005 (CAiSE 2005), Porto, Portugal, EU, pages 485-493. FEUP, Porto, Portugal, EU, 2005.

[HoPW05d] S.J.B.A. Hoppenbrouwers, H.A. (Erik) Proper, and Th.P. van der Weide. Understanding the Requirements on Modelling Techniques. In O. Pastor and J. Falcao e Cunha, editors, *17th International Conference on Advanced Information Systems Engineering, CAiSE 2005, Porto, Portugal, EU*, volume 3520 of *Lecture Notes in Computer Science*, pages 262-276, Berlin, Germany, EU, June 2005. Springer-Verlag.

[HeVa93] J.C. Henderson and N. Venkatraman. Strategic alignment: Leveraging information technology for transforming organizations. *IBM Systems Journal*, 32(1):4-16, 1993.

[IEEE00] Recommended Practice for Architectural Description of Software Intensive Systems. Technical Report IEEE P1471-2000, The Architecture Working Group of the Software Engineering Committee, Standards Department, IEEE, Piscataway, New Jersey, USA, September 2000.

[Jaga95] R. Jagannathan. Dataflow Models. In E.Y. Zomaya, editor, *Parallel and Distributed Computing Handbook*. McGraw-Hill, New York, New York, USA, 1995.

[JLB+04] H. Jonkers, M.M Lankhorst, R. van Buuren, S.J.B.A. Hoppenbrouwers, M. Bonsangue, and L. van der Torre. Concepts for Modeling Enterprise Architectures. *International Journal of Cooperative Information Systems*, 13(3):257-288, 2004.

[JVB+03] H. Jonkers, G.E. Veldhuijzen van Zanten, R. van Buuren, F. Arbab, F. de Boer, M. Bonsangue, H. Bosma, H. ter Doest, L. Groenewegen, J. Guillen Scholten, S.J.B.A. Hoppenbrouwers, M.-E. Iacob, W. Janssen, M.M. Lankhorst, D. van Leeuwen, H.A. (Erik) Proper, A. Stam, and L. van der Torre. Towards a Language for Coherent Enterprise Architecture Descriptions. In M. Steen and B.R. Bryant, editors, *7th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2003), Brisbane, Queensland, Australia*, pages 28-39, Los Alamitos, California, USA, September 2003. IEEE.

[Lan+05] M.M. Lankhorst and others. *Enterprise Architecture at Work: Modelling, Communication and Analysis*. Springer, Berlin, Germany, EU, 2005.

[Morr46] C. Morris. *Signs, Language and Behaviour*. Prentice-Hall/Braziller, New York, New York, USA, 1946.

[MeTa00] N. Medvidovic and R.N. Taylor. A classification and comparison framework for software architecture description languages. *IEEE Transactions on Software Engineering*, 26(1):70-93, January 2000.

[OMG02] OMG. UML Profile for Enterprise Distributed Object Computing Specification. Technical report, The Object Management Group, 2002.

[OMG03] OMG. UML 2.0 Superstructure Specification - Final Adopted Specification. Technical Report ptc/03-08-02, August 2003.

[Peir69] C.S. Peirce. *Volumes I and II - Principles of Philosophy and Elements of Logic*. Collected Papers of C.S. Peirce. Harvard University Press, Boston, Massachusetts, USA, 1969.

[PrVH05] H.A. (Erik) Proper, A.A. Verrijn-Stuart, and S.J.B.A. Hoppenbrouwers. Towards Utility-based Selection of Architecture-Modelling Concepts. In S. Hartmann and M. Stumptner, editors, *Proceedings of the Second Asia-Pacific Conference on Conceptual Modelling (APCCM2005), Newcastle, New South Wales, Australia*, volume 42 of *Conferences in Research and Practice in Information Technology Series*, pages 25-36, Sydney, New South Wales, Australia, January 2005. Australian Computer Society.

[Sche94] A.-W. Scheer. *Business Process Engineering: Reference Models for Industrial Enterprises*. Springer, Berlin, Germany, EU, 2nd edition, 1994.

[ShGa96] M. Shaw and D. Garlan. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice-Hall, Englewood Cliffs, New Jersey, USA, 1996.

[SaSt97] W. van der Sanden and B. Sturm. *Informatie-architectuur - de infrastructurele benadering*. Panfox, Rosmalen, The Netherlands, EU, 1997. In Dutch.

[TaCa93] D. Tapscott and A. Caston. *Paradigm Shift - The New Promise of Information Technology*. McGraw-Hill, New York, New York, USA, 1993.

[Turn87] K.J. Turner. An Architectural Semantics for LOTOS. In *Proceedings of the 7th International Conference on Protocol Specification, Testing, and Verification*, pages 15-28, 1987.

[USAD93] U.S.A. Department of Commerce. *Integration Definition for Function Modeling (IDEF0) Draft*. Federal Information Processing Standards Publication. 1993.

Farhad Arbab, Frank de Boer

Centrum voor Wiskunde en Informatica
Amsterdam
The Netherlands
{farhad.arbab|f.s.de.boer}@cwi.nl

Marcello Bonsangue

Leiden Institute of Advanced Computer Science
Leiden University
The Netherlands
marcello@liacs.nl

Marc Lankhorst

Telematica Instituut
Enschede
The Netherlands
marc.lankhorst@telin.nl

Erik Proper

Instititute for Computing and Information Science
Radboud University Nijmegen
Nijmegen
The Netherlands
e.proper@cs.ru.nl

Leendert van der Torre

Computer Science and Communications
University of Luxembourg
Luxembourg
leon.vandertorre@uni.lu