Concepts for Modelling Enterprise Architectures

Henk Jonkers[1], Marc Lankhorst[1], René van Buuren[1], Stijn Hoppenbrouwers[2], Marcello Bonsangue[3], Leendert van der Torre[4]

[1]*Telematica Instituut, P.O. Box 589, 7500 AN Enschede, the Netherlands*
Phone: +31 53 4850485, fax: +31 53 4850400, e-mail: Henk.Jonkers@telin.nl

[2]*University of Nijmegen, Nijmegen, the Netherlands*

[3]*Leiden Institute for Advanced Computer Science, Leiden, the Netherlands*

[4]*CWI, Amsterdam, the Netherlands*

Abstract

A coherent description of an enterprise architecture provides insight, enables communication among stakeholders and guides complicated change processes. Unfortunately, so far no enterprise architecture description language exists that fully enables integrated enterprise modelling, because for each architectural domain, architects use their own modelling techniques and concepts, tool support, visualisation techniques, etc. In this paper we outline such an integrated language and we identify and study concepts that relate architectural domains. In our language concepts for describing the relationships between architecture descriptions at the business, application, and technology levels play a central role, related to the ubiquitous problem of business–IT alignment, whereas for each architectural domain we conform to existing languages or standards such as UML. In particular, usage of services offered by one layer to another plays an important role in relating the behaviour aspects of the layers. The structural aspects of the layers are linked through the interface concept, and the information aspects through realisation relations.

Concepts for Modelling Enterprise Architectures

Abstract

A coherent description of an enterprise architecture provides insight, enables communication among stakeholders and guides complicated change processes. Unfortunately, so far no enterprise architecture description language exists that fully enables integrated enterprise modelling, because for each architectural domain, architects use their own modelling techniques and concepts, tool support, visualisation techniques, etc. In this paper we outline such an integrated language and we identify and study concepts that relate architectural domains. In our language concepts for describing the relationships between architecture descriptions at the business, application, and technology levels play a central role, related to the ubiquitous problem of business–IT alignment, whereas for each architectural domain we conform to existing languages or standards such as UML. In particular, usage of services offered by one layer to another plays an important role in relating the behaviour aspects of the layers. The structural aspects of the layers are linked through the interface concept, and the information aspects through realisation relations.

Note for the referees: in the full version a running example will be introduced in Sections 4, 5 and 6, which illustrates the various distinctions introduced and discussed in this paper.

1. Introduction

For the state of the art in enterprise modelling, we have to consider languages for organisation and process modelling as well as languages for application and technology modelling. First, a wide variety of organisation and process modelling languages are currently in use, because there is no single standard for models in this domain. 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, developed by OASIS and UN/CEFACT, the Business Process Modeling Language BPML (Arkin, 2002) of the Business Process Management Initiative, IDEF (IDEF, 1993), originating from the US Ministry of Defense, ARIS (Scheer, 1994), part of the widely used ARIS Toolset, and the Testbed language for business process modelling (Eertink et al., 1999). Second, and in contrast to organisation and business process modelling, in modelling applications and technology the Unified Modelling Language (UML) (Booch, Rumbaugh, and Jacobson, 1999) has become a true world standard. The UML is the mainstream modelling approach within ICT, and its use is expanding into other areas, e.g., in business modelling (Eriksson and Penker, 2000).

Most languages mentioned above provide concepts to model, e.g., detailed business processes, but not the high-level relationships between different processes. They are therefore not really suitable to describe *architectures* (IEEE Computer Society, 2000). Architecture description languages (ADLs) define high-level concepts for architecture description, such as components and connectors, and are used to describe or model enterprise architectures. 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 (2002) 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, Monroe and Wile, 1997) is widely accepted as a standard to exchange architectural information, also between other ADLs. The Reference Model for Open Distributed Processing (RM-ODP) is a joint ISO/ITU-T standard for the specification open distributed systems. There is a fairly complete language coverage of the seperate architectural domains, but a weak integration between the languages for the domains. No complete set of architecture description techniques exist that fully enable and exploit integrated enterprise modelling (Jonkers *et al.*, 2003).

In this paper we discuss a language that makes this integration possible. For example, we consider the so-called "business-IT alignment", the relationship between the organisational processes and the information systems and applications that support them. This relationship has attracted some attention lately, but modelling techniques to really express this relationship hardly exist yet. This lack of integration is an important problem for enterprise modelling, because changes in a company's strategy and business goals have significant consequences within all domains such as for the organisation structure, processes, software systems, data management and technical infrastructures. Companies have to adjust processes to their environment, open up internal systems and make them transparent to both internal and external parties. The use of an enterprise architecture helps to chart the complexity of an organisation. Many organisations have recognised the value of architectures and use them during the development and evolution of their products, processes, and systems. The goals of such an integrated model are to help in creating insight, aiding communication between stakeholders, and assessing the impact of changes.

Take for example a company that needs to assess the impact of introducing a new product in its portfolio. This may require defining additional business processes, hiring extra personnel, changing the supporting applications, and augmenting the technological infrastructure to support the additional load of these applications. Perhaps this may even require a change of the organisational structure. Many stakeholders within and outside the company can be identified, ranging from top-level management to software engineers. Each stakeholder requires specific information presented in an accessible way, to deal with the impact of such wide-ranging developments. It is very difficult to obtain an overview of these changes and their impact on each other, and to provide both decision makers and engineers implementing the changes with the information they need. The development of a coherent view of an enterprise and a disciplined architectural working practice significantly contribute to the solution of complex organisational puzzles. An integrated enterprise architecture provides insight, enables communication among different stakeholders, and guides complicated change processes.

The first steps towards an integrated language are described in (Jonkers *et al*., 2003). This paper presents an extended and improved outline of this language by presenting a more detailed discussion of the concepts used in the language. In particular, in this paper we address the following questions:

1. At which level of specificity should concepts be described, and more generally, what is the relation between the integrated language and existing detailed languages? Typically, languages aimed at a single domain are very specific and result in detailed models. We want to integrate these different domains, without substituting the existing, domain-specific modelling approaches. A single language for all domains with the level of detail offered by these individual approaches, however, would probably result in an unworkable behemoth. Our aim is to provide an overview of an entire enterprise; drilling down to the individual domains should be done using the existing approaches.
2. Which domains should be identified in the language? In order to arrive at a coherent architectural description, several architecture domains and layers as well as their relations must be modelled. Depending on the type of enterprise and the maturity of its architecture practice, different architectural domains are distinguished, such as the product, business, information, and application domains.
3. For each domain, which concepts should be included in the language? Currently, we restrict ourselves to describing 'operational' concepts and relations, i.e., those that directly contribute to the realisation of the products or services of that layer. Many other types of concepts and relations are likely to be relevant in architectural descriptions, such as ownership, governance, support, responsibility, etc. Where needed, these will be added in later versions of the language.
4. How to describe the relations between the domains? The relations between the business, application, and technology layers, which play a central role in this version of the language, should contribute to solving the business-ICT alignment problem that we try to tackle.

To address these questions, we proceed as follows. First, we define concepts at an intermediate abstraction level. They are applicable to describe enterprise architectures of any information-intensive organisation and, if desired, they can be further specialised or composed to form concepts tailored towards a more specific context. Second, we base our choice for the conceptual domains on the domains commonly distinguished in architectural frameworks or methods, such as the TOGAF framework (Open Group, 2003), the Zachman framework (Sowa and Zachman, 1992), and the architectural practice within organisations participating in the ArchiMate project. Third, within the architectural domains, we reuse elements from existing languages as much as possible. Moreover, we base our model on an actor perspective. Fourth, our language identifies the service concept as linking pin.

This paper is a result from the ArchiMate project (Jonkers *et al*., 2003), a public/private cooperation between several companies and research institutes that aims to provide enterprise architects with concepts and techniques for modelling, visualising, and analysing integrated architectures. The set of architecture modelling concepts described in this paper, together with their relationships, will be referred to as the *ArchiMate metamodel*. The ArchiMate project studies the enterprise modelling language that represents the complexity of architectural domains and especially their relations within the scope of a set of architecture instruments and techniques visualized in Figure 1. Views and presentation techniques are tailored to the needs of different stakeholders, providing them with insight in their particular area of interest, and facilitating cross-domain discussion and understanding. They are centered around the notion of a viewpoint in accordance with the IEEE standard 1471 (IEEE Computer Society, 2000). Analysis techniques are used to assess the impact of developments and changes. The enterprise modelling language is evaluated by case studies, as well as its suitability to visualize views and perform analyses. The relations among the elements of the ArchiMate project has been sketched in (Jonkers *et al.*, 2003). A more detailed description of the relations as well as a discussion on views, analysis and presentation is beyond the scope of this paper, though to illustrate the integration of architectures in our ArchiMate language we discuss an example that involves a simple analysis technique.

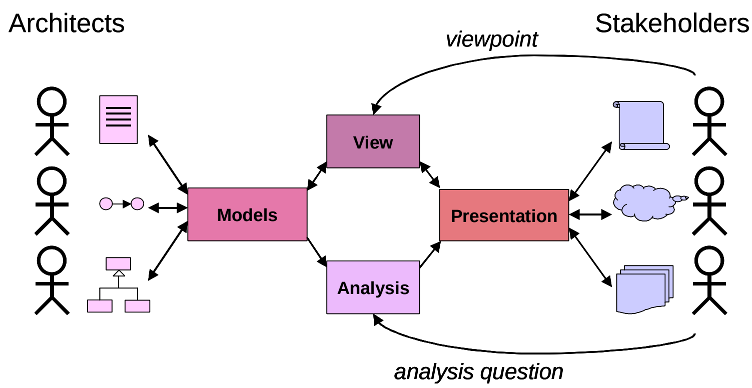

Figure 1. ArchiMate Context

The layout of this paper is as follows. In Section 2 we discuss the level of granularity of the ArchiMate language and the relationship between the ArchiMate language and other languages. In Section 3 we discuss our framework of conceptual domains. In Sections 4, 5 and 6 we discuss the concepts of the business layer, the application layer and the technology layer, respectively. Finally, in Section 7 we discuss intra- and inter-relationships. In Section 8 we present an integrated example that involves also a qualitative analysis technique.

2. Specificity of concepts for modelling enterprise architectures

A key challenge in the development of a general metamodel for enterprise architecture is to strike a balance between the specificity of languages for individual architecture domains, and a very general set of architecture concepts which reflects a view of systems as a mere set of interrelated entities. Figure 2 illustrates that concepts can be described at different levels of specialisation.

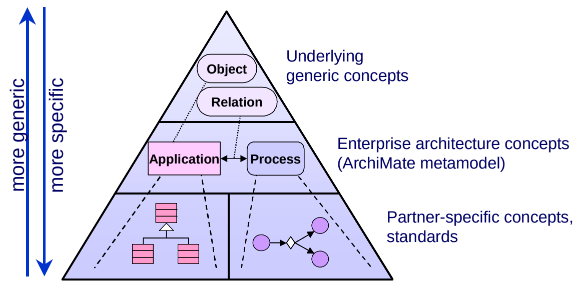

Figure 2. Metamodels at different levels of specificity

At the base of the triangle, we find the metamodels of the architecture modelling concepts used by specific organisations, as well as a variety of existing modelling languages and standards; UML is an example of a language in this category. Relevant languages include:

- The ebXML set of standards for XML-based electronic business, developed by OASIS and UN/CEFACT, specifies the Business Process Specification Schema (Business Process Project Team, 2001). It provides a standard framework by which business systems may be configured to support execution of business collaborations consisting of business transactions. It is focussed on the external behaviour of processes for the sake of automating electronic commerce transactions. It is therefore less suited for general enterprise architecture modelling.

- The Business Process Modeling Language BPML (Arkin, 2002) of the Business Process Management Initiative, is an XML-based language for modelling business processes that has roots in the workflow management world. It can be used to describe the inner workings of, e.g., ebXML business processes.
- IDEF (IDEF, 1993), originating from the US Ministry of Defense, is a collection of 16 (unrelated) diagramming techniques, three of which are widely used: IDEF0 (function modelling), IDEF1/IDEF1x (information and data modelling) and IDEF3 (process description).
- ARIS (Scheer, 1994) is part of the widely used ARIS Toolset. Although ARIS also covers other conceptual domains, there is a clear focus on business process modelling and organisation modelling.
- The Testbed language for business process modelling (Eertink et al., 1999), is used by a number of large Dutch organisations in the financial sector, was developed by the Telematica Instituut. We have gained a lot of experience with both the definition and the practical use of this language, and it has provided important inspiration for the definition of business-layer concepts.
- Concerning languages for application and technology modelling, the UML is the mainstream modelling approach within ICT, and its use is expanding into other areas, e.g., in business modelling (Eriksson and Penker, 2000). Another example is the UML 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 (Object Management Group, 2002b). 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 ICT modelling techniques still in use. However, it is not easily accessible and understandable for managers and business specialists; therefore, special visualisations 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.

At the top of the triangle we find the “most general” metamodel for system architectures, essentially a metamodel merely comprising notions such as “object”, “component”, and “relation”. Some architectural description languages such as ACME (Garlan, Monroe & Wile, 1997) partly fall into this category. These generic concepts may be suitable as a basis for the formal semantic description of the concepts and formal analysis techniques. 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 (U2 Partners, 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 modelling software systems. The *Architecture Description Markup Language* (ADML) was originally developed as an XML encoding of ACME. The Open Group promotes ADML as a standard for enterprise architectures. Moreover, the Reference Model for Open Distributed Processing (RM-ODP) 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. For example, for 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 (ITU-T, 2001).

The metamodel that we propose defines the concepts somewhere between these two extremes. It is more detailed than the abstract languages, because besides abstract concepts like component it also contains more detailed concepts like business function. Moreover, it is more abstract than for example UML, because it does not contain concepts like The concepts at the intermediate level are applicable to describe enterprise architectures of any information-intensive organisation and, if desired, they can be further specialised or composed to form concepts tailored towards a more specific context.

Figure 1 raises the question how the ArchiMate language is related to more abstract languages, as well as to more detailed languages. For example, the ADL ACME (Garlan, Monroe and Wile, 1997) is widely accepted as a standard to exchange architectural information, also between other ADLs. Can a similar role be played by the ArchiMate language? Moreover, how can the ArchiMate enterprise architecture concepts be used to integrate more specific models described in other languages?

A detailed discussion on the relation between the ArchiMate language and other languages is beyond the scope of this paper, because here we want to focus on the concepts used in the ArciMate language. However, as an example we sketch an example of how to use ArchiMate concepts to describe the high-level structure of the organisation, the business processes and the application support for these processes and relations. Tools such as Testbed Studio or ARIS for detailed business process models, and tools such as Rational Rose or Select Component Architect for detailed UML design models, can be used for more detailed descriptions. Figure 3 shows an example of this approach in which an integrated architectural view on a number of disjoint UML diagrams is constructed by means of an overall ArchiMate model.

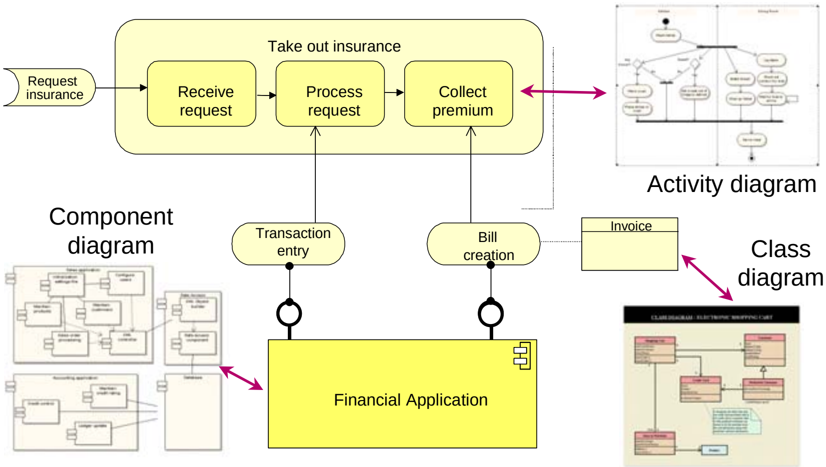

Figure 3. Example of ArchiMate as a language to link UML design models

3. The conceptual domains in the ArchiMate language

The set of architecture modelling concepts described in this paper, together with their relationships, will be referred to as the *ArchiMate metamodel*. The starting point for the development of this metamodel is a collection of so-called conceptual domains, each covering a specific business area. We base our choice of conceptual domains on the domains commonly distinguished in architectural frameworks or methods, such as the TOGAF framework (Open Group, 2003), the Zachman framework (Sowa and Zachman, 1992), and the architectural practice within organisations participating in the ArchiMate project.

The *product* domain, with the concept ‘product’ that describes the (information) products or services that an organisation offers to its customers.

The *organisation* domain, describing the business actors (employees, organisational units) and the roles they may fulfil.

The *process* domain, describing business processes or business functions consisting of business activities.

The *information* domain, representing the knowledge in an organisation and the way it is structured.

The *data* domain, in which information is represented in such a way that it is suitable for automated processing.

The *application* domain, describing software applications that support the business through application services.

The *technical infrastructure* domain, comprising concepts for, e.g., hardware platforms and communication infrastructure, needed to support applications.

In the current practice of organisations, architectural descriptions are made for different ‘layers’ of the organisation. These are layers in the sense that the lower layers provide functionality to support the higher layers. The layers that are usually recognised in this context are the *business layer*, the *application layer* and the *technology layer*. Although, to a certain extent, modelling support within each of these layers is available, well-described concepts to describe the relationships *between* the layers are almost completely missing. Such concepts are essential to tackle the business–IT alignment problem in a systematic way.

Based on the common aspects of these domains and layers, we make a first generalisation of the core concepts. In our view, a system or organisation primarily consists of a set of entities, which have an internal structure, perform behaviour, and use and exchange information. For instance, a sales organisation may consist of a number of departments, which perform business processes, using and exchanging customer data.

The aspects and layers form a framework of nine 'cells', as illustrated in Figure 4. The conceptual domains mentioned earlier are projected into this framework.

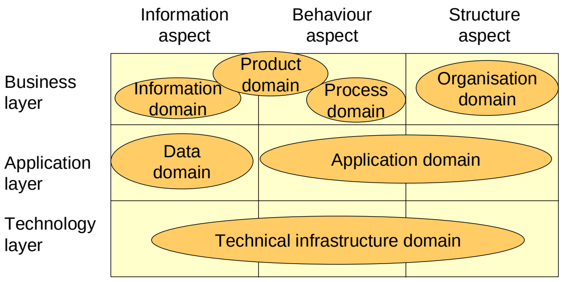

Figure 4. Architectural framework

It is important to realise that the classification of concepts based on conceptual domains, or based on aspects and layers, is only a global one. It is impossible, and undesirable, to define a strict boundary between the aspects and layers, because concepts that link the different aspects and layers play one of the most important roles in a coherent architectural description. For example, running somewhat ahead of the later conceptual discussions, services and roles serve as intermediary concepts between 'purely behavioural' concepts and 'purely structural' concepts. Also, there are concepts that cover multiple aspects and layers. An example is a 'business domain' concept, e.g. the 'mortgage domain' for a bank, which covers both the business layer and application layer, and includes elements from all of the three aspects.

4. Business layer concepts

In this section we identify the concepts for architectural descriptions that can be placed in the business layer of our framework of Section 3. We describe concepts covering each of the three aspects – structure, behaviour and information – as well as concepts linking these aspects.

Figure 5 gives an overview of the business layer concepts and their relationships. For this figure, as well as for the other metamodel representations in this document, we use a restricted version of UML class diagrams. The concepts are roughly positioned according to the three aspects of our framework. However, the boundaries between the aspects are not strict: some concepts are most naturally placed on the line that separates the aspects in Figure 4. These concepts may serve to make the link between the concepts within the different aspects. In particular, the concept 'representation' links the structural and informational aspects, and the concepts 'role', 'collaboration' and 'interface' link the structural and behavioural aspect.

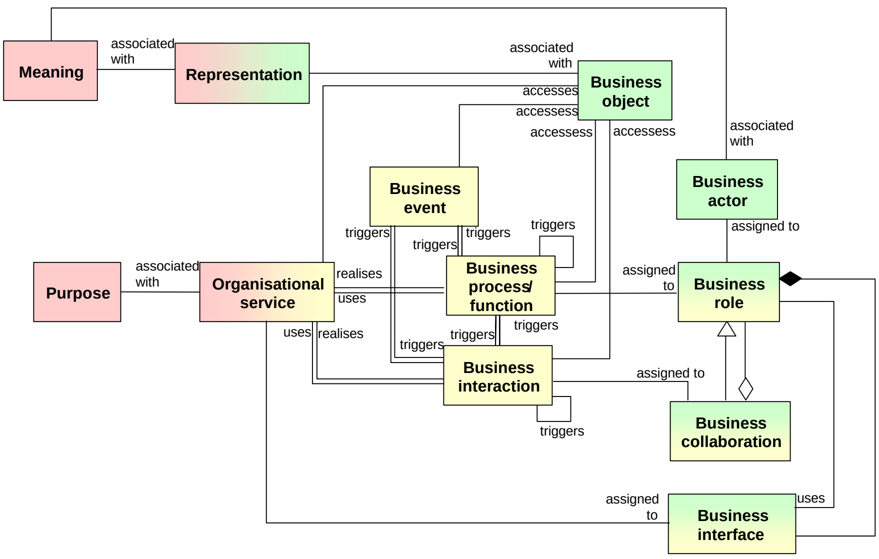

Figure 5. Business layer metamodel

Picture to be inserted for final version

Figure 6. Example of a business-layer model.

4.1 Structure

The structure aspect at the business layer refers to the static structure of an organisation, in terms of the entities that make up the organisation and their relationships. Two types of entities are distinguished:

- ***Business actors:*** the active entities (the subjects) that perform behaviour such as business processes or functions. Business actors may be individual persons (e.g. customers or employees), but also groups of people and resources that have a permanent (or at least long-term) status within the organisations. Typical examples of the latter are a department and a business unit.
- ***Business objects:*** the passive entities that are manipulated by behaviour such as business processes or functions. Business objects represent the important concepts in which the business thinks about a domain.

Although UML, which is based on the object-oriented paradigm, does not make the distinction between active and passive entities, these two concepts are very common in other enterprise modelling approaches. For example, the RM-ODP Enterprise Language (Tyndale-Biscoe, 2002) distinguishes 'actor' and 'artefact' as two specialisations of 'enterprise object'.

At the business layer, it is common to make the link between actors and behaviour more flexible by introducing the intermediary concept ***business role***. The idea is that the work that an actor performs within an organisation is always based on a certain role that the actor fulfils. There are at least two reasons. First, the set of roles in an organisation can be expected to be much more stable than the specific actors fulfilling these roles, so to describe the structure of an organization roles seem to be better candidates than actors. Second, multiple actors can fulfil the same role, and conversely, a single actor can fulfil multiple roles. Roles are typically used to distinguish responsibilities, and it can be checked whether the assignment of actors to roles satisfies desirable properties.

Architectural descriptions focus on *structure*, which means that the interrelationships of entities within an organisation play an important role. To make this explicit, the concept of ***business collaboration*** has been introduced. A collaboration is a collective of roles within an organisation which perform collaborative behaviour. Business collaborations have been inspired by collaborations as defined in the UML (U2 Partners, 2002), although the UML collaborations apply to components in the application layer. Also, our business collaboration concept has a strong resemblance to the 'community' concept as defined in the RM-ODP Enterprise Language (Tyndale-Biscoe, 2002), and to the 'interaction point' concept, defined in the AMBER language (Eertink *et al*., 1999).

As will be explained in Section 7.2, the service concept plays an important role in linking models in the different layers of our framework. In the light of this 'service-oriented' approach, it is useful to have the possibility to explicitly model the ***business interfaces***, i.e., the (logical or physical) locations where the services that a role offers to the environment can be accessed. The same service may be offered on a number of different interfaces: e.g., by mail, by telephone or through the Internet. In contrast to application modelling, it is uncommon in current business layer modelling approaches to recognise the business interface concept. However, the 'channel' concept, as defined in, among others, the NEML language (Steen *et al*., 2002), has a strong resemblance to a business interface.

4.2 Behaviour

Based on service orientation, a crucial design decision for the behavioural part of our metamodel is the distinction between "external" and "internal" behaviour of an organisation. The externally visible behaviour is modelled by the concept ***organisational service***, which represents a unit of functionality that is meaningful from the point of view of the environment. Within the organisation, these services are realised by ***business processes***, ***business functions*** or ***business interactions***. Business processes, functions and interactions, in turn, may use other services (internal to the organisation, but external to a smaller entity within the organisation).

A ***business process/function*** is a unit of internal behaviour, performed by one or more roles within the organisation. A 'business activity' could be defined as a behaviour element that has the right granularity to determine the services and applications needed to support it. We can solve this by defining a specialisation of a business process function, which has as a constraint that it cannot be further decomposed.

Although the distinction between the two is not always sharp, it is often useful to distinguish a *process view* and a *function view* on behaviour. Both concepts can be used to group more detailed business processes/functions, but based on different grouping criteria. A ***business process*** represents a 'flow' of smaller processes/functions, with one or more clear starting points and leading to some result (sometimes described as 'customer to customer', where 'customer' may also be an 'internal customer', in the case of subprocesses within an organisation). A ***business function*** offers useful functionality that may be useful for one or more business processes. It groups behaviour based on, e.g., required skills, capabilities, resources, (application) support, etc. Typically, the business processes of an organisation are defined based on the *products* and *services* that the organisation offers, while the business functions are the basis for, e.g., the assignment of resources to tasks and the application support.

A ***business interaction*** is a unit of behaviour similar to a business process or function, but it is performed in a collaboration of two or more roles within the organisation. This strongly resembles the 'interaction' concept in AMBER (Eertink *et al*., 1999). Similar to processes or functions, the result of a business interaction can be made available to the environment through an organisational service.

A ***business event*** is something that happens (externally) and may influence business processes, functions or interactions. A business event is most commonly used to model something that *triggers* behaviour, but other types of events are also conceivable: e.g., an event that interrupts a process. The business event concept is similar to the 'trigger' concept in AMBER (Eertink *et al*., 1999) and the 'initial state' and 'final state' concepts as used in, e.g., UML activity diagrams (U2 Partners, 2003).

4.3 Information

The informational concepts provide a way to link the operational side of an organisation to its business goals, the information that is processed, and to the products or services that an organisation offers to its customers.

A ***representation*** is the perceptible form of the information carried by a business object, such as a document. If relevant, representations can be classified in various ways, for example in terms of medium (e.g., electronic, paper, audio) or format (e.g., HTML, PDF, plain text, bar chart). A single business object can have a number of different representations, but a representation always belongs to one specific business object.

A ***purpose*** is the functionality of some organisational service seen from the point of view of an external user. It models the intended contribution of a service towards the achieving of a particular (business) goal, or a set of goals. A purpose concerns a high-level description of some *basic functionality* in terms of behaviour or even some condition or state supported or enabled by some organisational service, seen strictly from the point of view of some external actor (typically, a customer).

'Purpose' strongly resembles a UML *use case*. Loosely speaking, a purpose can be seen as a characterisation of a "required service", as opposed to an "offered service" from the organisation's perspective. In this way, we have a behavioural counterpart of the 'required interface'/'offered interface' duality for the structure aspect. The purpose concept provides a natural link between the operational business processes, functions and services, and the product domain that will be added to our metamodel in a later stage.

A ***meaning*** is the contribution of the representation of a business object to the knowledge or expertise of some actor, given a particular context. In other words, meaning represents the informative value of a business object for a user of such an object. It is through a certain interpretation of a representation of the object that meaning is being offered to a certain user or to a certain category of users.

5. Application layer concepts

Figure 7 gives an overview of the application layer concepts and their relationships. Many of the concepts have been inspired by the UML (including some of the UML 2.0 proposals), as this is the dominant language, and the *de facto* standard, for describing software applications. Whenever applicable, we draw inspiration from the analogy with the business layer.

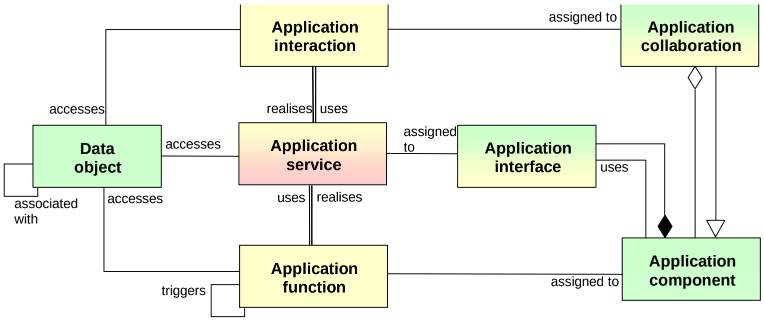

Figure 7. Application-layer metamodel

Picture to be inserted for final version

Figure 8. Example of an application-layer model.

5.1 Structure

The main structural concept for the application layer is the ***application component***. This concept is used to model any structural entity in the application layer: not just (reusable) software components that can be part of one or more applications, but also complete software applications, subapplications or information systems. This concept is very similar to the UML component.

The interrelationships of components are also an essential ingredient in application architecture. Therefore, we also introduce the concept of ***application collaboration*** here, defined as a collective of application components which perform application interactions. The concept is very similar to the collaboration as defined in the UML 2.0 proposals (U2 Partners, 2002).

In the purely structural sense, an ***application interface*** is the (logical) location where the services of a component can be accessed. In a broader sense (as used in, among others, the UML definition), an application interface also has some behavioural characteristics: it defines the set of operations and events that are provided by the component, or those that are required from the environment. Thus, it is used to describe the functionality of a component. A distinction may be made between a *provided interface* and a *required interface*. The application interface concept can be used to model both *application-to-application* interfaces, offering internal application services, and *application-to business* interfaces (or *user interfaces*), offering external application services.

Also at the application layer, we distinguish the passive counterpart of the component, which we call a ***data object***. This concept is used in the same way as data objects (or object types) in well-known data modelling approaches, most notably the ‘class’ concept in UML class diagrams.

5.2 Behaviour

Behaviour in the application layer can be described in a way that is very similar to business layer behaviour. We make a distinction between the external behaviour of application components in terms of *application services*, and the internal behaviour of these components to realise these services.

An ***application service*** is an externally visible unit of functionality, provided by one or more components, exposed through well-defined interfaces, and meaningful to the environment. The service concept provides a way to explicitly describe the functionality that components share with each other and the functionality that they make available to the environment. The concept fits well within the current developments in the area of, e.g., web services (Lankhorst, 2002). The term *business service* is sometimes used for an external application service, i.e., application functionality that is used to directly support the work performed in a business process or function, exposed by an application-to-business interface. Internal application services are exposed through an application-to-application interface.

An ***application function*** describes the internal behaviour of a component needed to realise one or more application services. In analogy with the business layer, a separate ‘application flow’ concept is conceivable as the counterpart of a business process. However, for the moment we have decided not to include this as a separate concept in our metamodel.

An ***application interaction*** is the behaviour of a collaboration of two or more application components. The UML 2.0 proposals (U2 Partners, 2002) also include the interaction concept. An application component is external behaviour from the perspective of each of the participating components, but the behaviour is internal to the collaboration as a whole.

5.3 Information

For the moment, we have not defined any purely informational concepts at the application layer, because the link to objectives and products is less apparent here than it is at the business layer. However, it is conceivable that application-layer versions of the informational concepts turn out to be useful in certain situations. Given our definition of the 'business purpose' concept, a 'use case', in the UML sense, would be a natural candidate for its application-layer counterpart. The counterpart of the 'business meaning' concept would have to be subject to automated interpretation, which suggests something in the direction of 'operational semantics'. Further study is needed to come to the most suitable set of informational concepts at the application layer, if any.

6. Technology layer concepts

In this section we identify the concepts for architectural descriptions that can be placed in the technology layer of our framework. Figure 9 gives an overview of the application layer concepts and their relationships. Many of the concepts have been inspired by the upcoming UML 2.0 standard (U2 Partners, 2003), as this is the dominant language and the de facto standard for describing software applications. Whenever applicable, we draw inspiration from the analogy with the business and application layers.

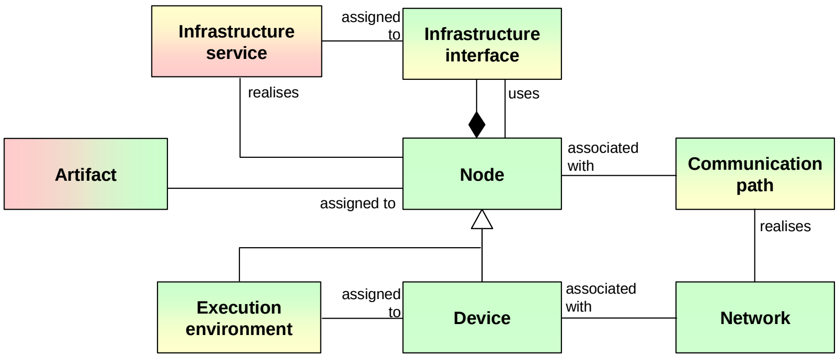

Figure 9. Technology-layer metamodel

Picture to be inserted for final version

Figure 10. Example of a technology-layer model.

6.1 Structure

The main structural concept for the application layer is the ***node***. This concept is used to model structural entities in the technology layer. It is identical to the node concept of UML 2.0. It strictly models the structural aspect of an application: its behaviour is modelled by an explicit relationship to the behavioural concepts.

An ***infrastructure interface*** is the (logical) location where the infrastructural services offered by a node can be accessed by other nodes or by application components from the application layer.

Nodes come in two flavours: ***device*** and ***execution environment***, both taken from UML 2.0. A device models a physical computational resource, upon which artifacts may be deployed for execution. An execution environment represents the software environment for specific types of components and data objects that are deployed on it in the form of artifacts. Typically, a node will consist of a number of subnodes, for example a device such as a server and an execution environment to model the operating system.

The interrelationships of components in the technology layer are mainly formed by communication infrastructure. The ***communication path*** models the relation between two or more nodes, through which these nodes can exchange information. The physical realisation of a communication path is a modelled with a ***network***, i.e., a physical communication medium between two or more devices.

6.2 Behaviour

In the technology layer, the behavioural concept that we deem relevant is the ***infrastructure service***. Modelling the internal behaviour of infrastructure components such as routers or database servers would add a level of detail that is not useful at the enterprise level of abstraction. Infrastructure services can be classified into three main types:

- Processing services;
- Data storage and access services;
- Communication services.

These services correspond to the three main types of physical infrastructure: computing devices, storage, and networks.

6.3 Information

An ***artifact*** is a physical piece of information that is used or produced in a software development process, or by deployment and operation of a system. It is the representation, in the form of e.g. a file, of a data object or an application component, and can be assigned to (i.e., deployed on) a node. The artifact concept has been taken from UML 2.0.

7. Intra- and inter-layer relationships

In the previous sections we have presented the concepts to model the business, application, and technology layers of an enterprise, respectively. However, one of the main issues in enterprise architecture is *business-IT alignment*: how can these layers be matched? Many languages exist to model business architectures on the one hand, or application and technical architectures on the other hand. However, languages that support a clear description of the relationship between these layers are missing.

7.1 Intra-layer relationships

In each of the layers presented thus far, different relationships between concepts have been used. Table 1 gives an overview of these relationships.

Table 1. Intra-layer relationships

Access	The access relationship models the access of behavioural concepts to business or data objects.
Aggregation	The aggregation relationship indicates that an object groups a number of other objects.
Assignment	The assignment relationship links units of behaviour with active elements (e.g. roles, components) that perform them, roles with actors that fulfil them, or artifacts that are deployed on nodes.
Association	Association models a relationship between objects that is not covered by another, more specific relationship.
Composition	The composition relationship indicates that an object consists of a number of other objects.
Realisation	The realisation relationship links a logical entity with a more concrete entity that realises it.
Specialisation	The specialisation relationship indicates that an object is a specialisation of another object.
Triggering	The triggering relationship describes the temporal or causal relations between processes, function, interactions and events.
Use	The use relationship models the use of services by processes, functions or interactions and the access to interfaces by roles, components or collaborations.

As we did for the concepts used to describe the different conceptual domains, as much as possible we adopt corresponding relationship concepts from existing standards. For instance, relationship concepts such as composition, association, specialization are taken from the UML while triggering is used in most business process modelling languages such as ARIS and Testbed.

7.2 Inter-layer relationships: the service concept as linking pin

Generalising from the relationships presented in the previous section, it can be observed that the architectural layers (business, application and technology) constitute some sort of hierarchy within an enterprise. A common way of looking at an enterprise is to start from the business processes and activities performed. These are carried out by some actor or role in the organisation, possibly supported by one or more business applications, or even fully automated. These activities however, can also be viewed as *services* to this business process, rendering a specific added value to the process at hand.

One may also adopt a bottom-up strategy, in which the business processes are just a mechanism for instantiating and commercially exploiting the lower-level services to the outside world. In this view, the most valuable assets are the capabilities to execute the lower-level services, and the processes are merely a means of exploitation. Applying such a service-oriented view results in a 'service hierarchy' as depicted in Figure 11. This is very similar to the layered model of e.g. the ISO-OSI model (ISO, 1984).

Each layer makes their external services available to the next higher layer. The external services of the higher layer may depend on services in the same architectural layer or one layer below. Organisational services, for example, may depend on external application services. Internal services are used within the same architectural level; for instance, an application component may use services offered by another application component. Likewise, a business process may be viewed as comprising sub-processes that offer their services to each other and to the containing process. External organisational services could also be called 'customer services', i.e., services offered to the (external) customers of the enterprise.

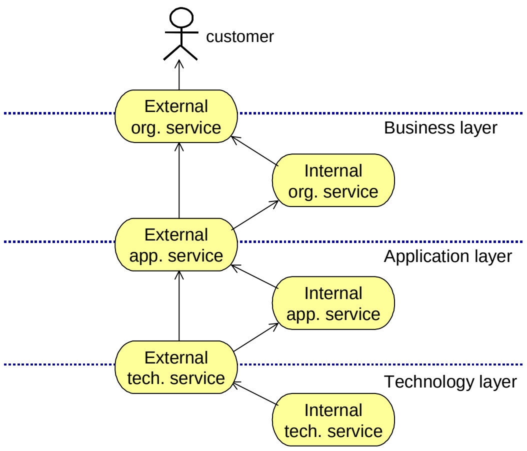

Figure 11. 'Service architecture': hierarchy of services

External organisational services could also be called 'customer services', i.e., services offered to the (external) customers of the enterprise/system. Similarly, external application services are sometimes called 'business services', i.e., services offered by applications but used by 'the business'.

Figure 12 shows the main relations between concepts in the business layer, the application layer, and the technology layer. For clarity's sake, we have omitted most intra-layer relations.

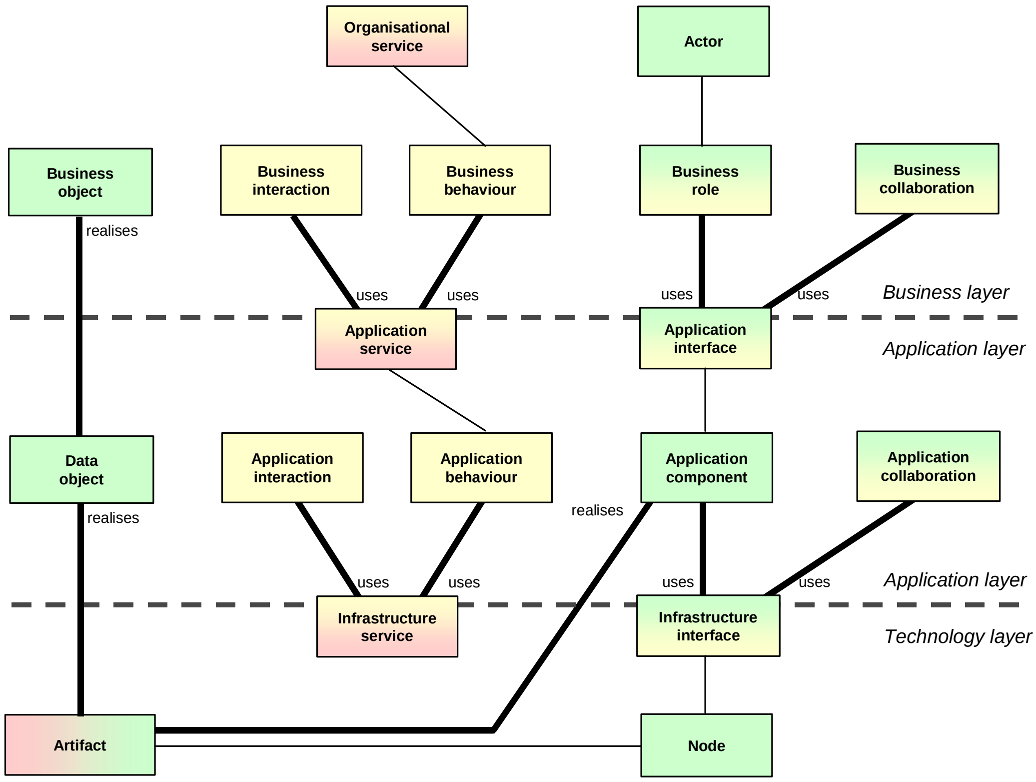

Figure 12. Relations between the layers

There is no direct *operational* link between business objects and data objects, without the intervention of behaviour (i.e., services): data objects in the application layer are only available to the business layer through services that are offered by application components. However, there is an important 'realisation' relationship: one or more data objects in the application layer realise (or represent) one or more business objects in the business layer.

We also see such relisation relations between on the one hand the data objects and application components in the application layer, and on the other hand the artifacts representing them in the technology layer.

8. Example

To illustrate our approach, we use a service-oriented enterprise architecture of an imaginary insurance company, ArchiSurance. Figure 13 gives a very simple example of a layered enterprise architecture using services to relate the infrastructure layer, the application layer, the business process layer, and the environment. The insurant and insurer roles represent the client and insurance company (ArchiSurance), respectively. Invocation of the claims registration service by the insurant starts the damage claiming process. The insurant is informed whether the claim is accepted, and, if so, receives a payment. Interaction between business processes and organisational roles is through business services. Thus, services connect the process architecture and the organisation architecture. Likewise, application services relate the business process architecture to the application architecture. The automated part of each business process is provided by an external application service. These application services are realised by application components. Finally, the technology layer consists of a number of infrastructure elements such as a mainframe and an application server, which execute application components and provide services to the application layer.

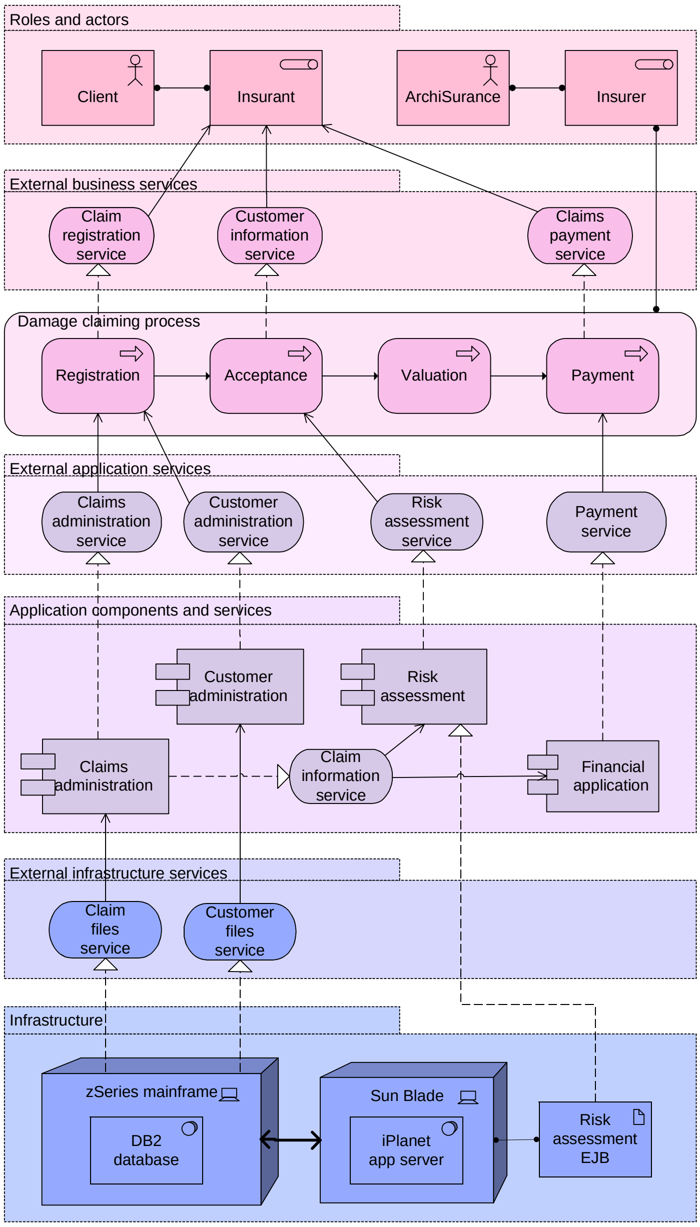

Figure 13. Example of a service-oriented enterprise architecture

For more details on the (provisional) notation used in this example, see (Van Buuren et al., 2003).

Given our integrated enterprise architecture language, as applied in the example, how does this contribute to the goals we have set in the introduction? More specifically, how does such an integrated model help in creating insight, aiding communication between stakeholders, and assessing the impact of changes?

First, as the example shows, a high-level overview of an entire enterprise can be shown in a single integrated and well-defined model. Admittedly, our example was very simple; in reality, such a model would be much larger, requiring techniques for selecting and visualising the elements that are relevant for a particular stakeholder.

Second, this model can be interpreted by, for example, both a manager requiring the 'big picture' and a software engineer that implements an application component and needs to know the context of this component. Thus, by using such a model as a means of communicating, different stakeholders can better understand each other. Within each specific domain, this high-level model may serve as a starting point for more detailed descriptions.

Third, the well-defined semantics of the concepts and their relations can be used to analyse the impact of events and changes. For instance, if the Sun Blade server in the example model fails, we can compute which applications can no longer run, which services can not be offered, which processes are impacted, and finally which services can no longer be offered to clients. This is shown by the darkened concepts in Figure 14. Thus, a manager can decide how severe the impact of the hardware failure might be, and how robust the infrastructure should be.

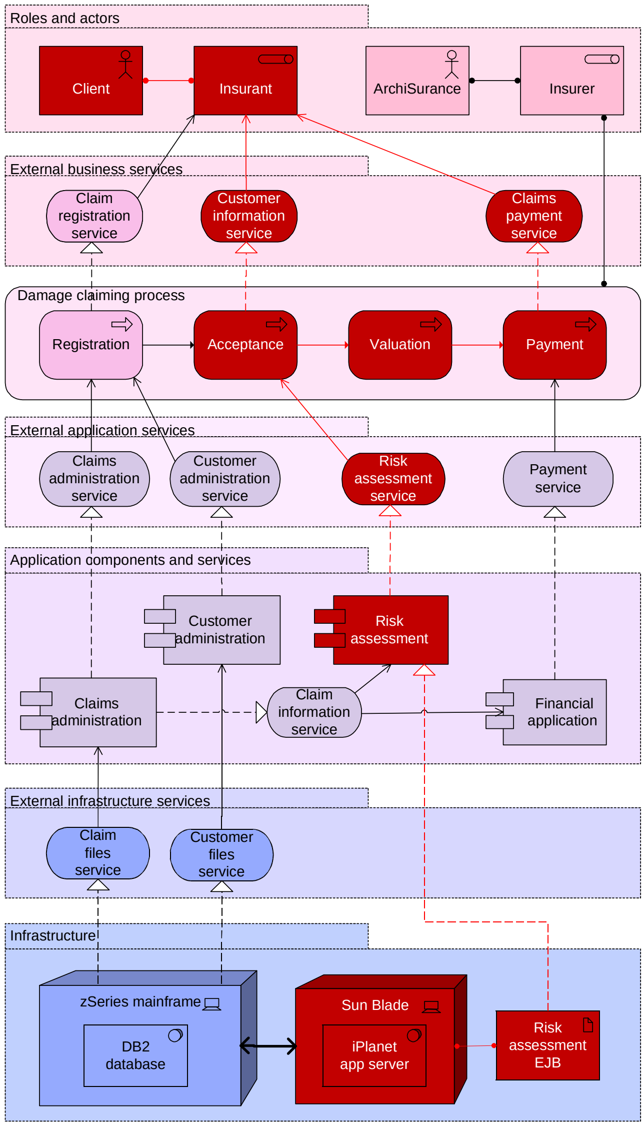

Figure 14. Example of impact analysis; darkened concepts show what is affected by failure of the Sun Blade server

9. Conclusions and future work

In this paper we have outlined a language for describing integrated enterprise architectures. This languages aims to bring the many separate architectural descriptions for specific architectural domains closer together, as at present no architectural language exists for describing the architecture of an enterprise as a whole. Since separate languages and their corresponding approaches are deeply embedded in organisations, it is not recommendable to develop an entirely new language. Therefore, our new language aims to embrace and extend successful and widely adopted languages such as the UML. In particular, in this paper we address the following questions.

First, at which level of specificity should concepts be described, and more generally, what is the relation between the integrated language and existing detailed languages? The concepts of our language for enterprise architecture description hold the middle between the detailed concepts that are used for modelling individual domains, e.g., the UML for modelling software, and very general architecture concepts that view systems merely as entities and their inter-relations. The language forms a basis for bridging the heterogeneity of existing languages. Current work in the project aim at developing a tool integration environment in which models originating from various tools can be linked. This stimulates possible reuse in a form that is still recognisable for the original designer.

Second, which domains should be identified in the language? Concepts in our language currently cover the business, application, and technology layers of an enterprise. Moreover, for each layer we distinguish the information, behaviour and structure aspects. The information, product, process, organisation, data, application and technical infrastructure domain are projected into this framework.

Third, for each domain, which concepts should be included in the language? For each layer, concepts and relations for modelling the information, behaviour, and structure aspects are defined. At the business layer we distinguish the structural concepts business actors and objects, roles and collaborations, the behavioural concepts organisational service, business process, functions and interactions and events, and the informational concepts representation, purpose and meaning. At the application layer, we distinguish the structural concepts application component, collaboration, interface, and data object, and the behavioural concepts of application service, function and interaction. At the technology layer, we distinguish the structural concepts of node, device, execution environment, infrastructure interface, communication path, and network, the behavioural concept of infrastructure service, and the informational concept of artifact.

Fourth, how to describe the relations between the domains? Usage of services offered by one layer to another plays an important role in relating the behaviour aspects of the layers. The structural aspects of the layers are linked through the interface concept, and the information aspects through realisation relations.

Looking at the metamodels for the different layers, it is apparent that they have many things in common. They use similar concepts to model the three aspects from our framework, be it at different levels of detail. It is useful to recognise this common basis of the layer-specific metamodels. This simplifies the formalisation of the metamodels, and the same or similar analysis and visualisation techniques can be developed applicable to both layers.

By means of a simple example we have demonstrated that our concepts can be used to make a coherent description, covering all aspects and layers within an enterprise. We have illustrated that such integrated models are very useful in creating insight, facilitating the communication between stakeholders, and assessing the impact of events and changes. However, even this limited example demonstrates that the complexity of the integrated models needs to be addressed. The development of views that select and visualise relevant elements from these models for specific stakeholders helps to fully exploit the models.

The work described in this paper is part of an ongoing project for the development of concepts and techniques for supporting enterprise architects. Here we focussed on the core concepts and relations of an enterprise architecture language. Further work will involve, among other things:

- Further specification of the detailed relations between concepts, aspects and layers;
- Further specification of concepts, for example, by means of attributes;
- Extension of the metamodel to the product domain;
- Formalisation of the metamodel to allow for analysis and automated visualisation;
- Identification of relevant viewpoints and related visualisations;
- Integration with other tool support environments;
- Further practical validation of the metamodel.

Acknowledgments

This paper results from the ArchiMate project (http://archimate.telin.nl), a research initiative that aims to provide 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, Katholieke Universiteit Nijmegen, and the Leiden Institute of Advanced Computer Science.

References

Arkin, A., (2002), *Business Process Modeling Language*, BPMI.org.
http://www.bpmi.org/bpmi-downloads/BPML1.0.zip

Booch, G., J. Rumbaugh and I. Jacobson (1999), *The Unified Modeling Language User Guide*, Addison-Wesley.

Business Process Project Team (2001), ebXML Business Process Specification Schema Version 1.01, UN/CEFACT and OASIS. http://www.ebxml.org/specs/ebBPSS.pdf
Eertink, H., W. Janssen, P. Oude Luttighuis, W. Teeuw and C. Vissers (1999), A Business Process Design Language, in *Proc. of the 1st World Congress on Formal Methods*, Toulouse, France, Sept. 1999.
Eriksson, H.-E. and M. Penker (2000), *Business Modeling with UML: Business Patterns at Work*, J. Wiley.
Frankel, D.S. (2003), *Model Driven Architecture: Applying MDA to Enterprise Computing*, Wiley, 2003.
Garlan, D., R.T. Monroe, and D. Wile (1997), ACME: An Architecture Description Interchange Language, in *Proceedings of CASCON '97*, Toronto, Canada, Nov, 1997, pp. 169-183.
IDEF (1993), *Integration Definition for Function Modeling (IDEF0) Draft*, Federal Information Processing Standards Publication FIPSPUB 183, U.S. Department of Commerce, Springfield, VA, USA, Dec. 1993.
IEEE Computer Society (2000). *IEEE Std 1471-2000*: *IEEE Recommended Practice for Architectural Description of Software-Intensive Systems*, Oct. 9, 2000.
ISO (1984). *Basic Reference Model for Open Systems Interconnection*, ISO 7498. International Organisation for Standardisation.
ITU-T (2002), *Information technology – Open Distributed Processing – Reference Model – Enterprise Language*, ITU–T Recommendation X.911 ISO/IEC 15414. International Telecommunication Union.
Jonkers, H. *et al*. (2003), Towards a Language for Coherent Enterprise Architecture Description, in M. Steen and B.R. Bryant (eds.), *Proceedings 7th IEEE International Enterprise Distributed Object Computing Conference* (EDOC 2003), Brisbane, Australia, Sept 2003.
Medvidovic, N. and R.N. Taylor (2000), A classification and comparison framework for software architecture description languages, *IEEE Transactions on Software Engineering*, 26 (1), Jan. 2000, pp. 70-93.
Object Management Group (2002a), *Meta Object Facility (MOF) Specification*, version 1.4, April 2002.
Object Management Group (2002b), *UML Profile for Enterprise Distributed Object Computting (EDOC)*, OMG final adopted specification ptc/02-02-05, http://www.omg.org/docs/ptc/02-02-05.pdf
Open Group (2003), *The Open Group Architectural Framework Version 7*, http://www.opengroup.org/togaf/.
Reijswoud, V.E. van, and J.L.G. Dietz (1999), *DEMO Modelling Handbook*, Volume 1, Delft University of Technology, Dept. of Information Systems, 1999.
Scheer, A.-W. (1994), *Business Process Engineering: Reference Models for Industrial Enterprises*, Springer, Berlin, 2nd ed., 1994.
Sowa, J.F. and J.A. Zachman (1992), Extending and Formalizing the Framework for Information Systems Architecture, *IBM Systems Journal*, 31 (3), 1992, pp. 590-616.
Steen, M.W.A., M.M. Lankhorst and R.G. van de Wetering (2002), Modelling Networked Enterprises, in *Proc. Sixth International Enterprise Distributed Object Computing Conference* (EDOC'02), Lausanne, Switzerland, Sept. 2002, pp. 109-119.
Tyndale-Biscoe, S., *RM-ODP Enterprise Language ISO/ITU-T 15414/X911*, 2002. http://www.itu.int/itudoc/itu-t/com17/tutorial/81999_pp7.ppt.
U2 Partners, *Unified Modeling Language: Superstructure*, version 2.0, 3rd revised submission, OMG document ad/03-04-01, April 10, 2003.