Towards a Language for Coherent Enterprise Architecture Descriptions

Henk Jonkers[1], René van Buuren[1], Farhad Arbab[3], Frank de Boer[3], Marcello Bonsangue[4],
Hans Bosma[5], Hugo ter Doest[1], Luuk Groenewegen[4], Juan Guillen Scholten[3], Stijn Hoppenbrouwers[2],
Maria-Eugenia Iacob[1], Wil Janssen[1], Marc Lankhorst[1], Diederik van Leeuwen[1], Erik Proper[2],
Andries Stam[4,5], Leon van der Torre[3], Gert Veldhuijzen van Zanten[2]

[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]*Centrum voor Wiskunde en Informatica, Amsterdam, the Netherlands*

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

[5]*Ordina Public Consulting, Rosmalen, the Netherlands*

Abstract

A coherent description of architectures provides insight, enables communication among different stakeholders and guides complicated (business and ICT) change processes. Unfortunately, so far no architecture description language exists that fully enables integrated enterprise modelling. In this paper we focus on the requirements and design of such a language. This language defines generic, organisation-independent concepts that can be specialised or composed to obtain more specific concepts to be used within a particular organisation. It is not our intention to re-invent the wheel for each architectural domain: wherever possible we conform to existing languages or standards such as UML. We complement them with missing concepts, focussing on concepts to model the relationships among architectural domains. The concepts should also make it possible to define links between models in other languages. The relationship between architecture descriptions at the business layer and at the application layer (business-IT alignment) plays a central role.

1. Introduction

Changes in a company's strategy and business goals have significant consequences 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. Architectures are a way to chart the complexity involved.

Many enterprises have recognised the value of architectures and to some extent make use of them during system evolution and development. Depending on the type of enterprise or maturity of the architecture practice, in most cases a number of separate architectural domains are distinguished such as product, business, information and application domain. For each architectural domain architects have their own concepts, modelling techniques, tool support, visualisation techniques and so on. Clearly, this way of working does not necessarily lead to a coherent view on the enterprise.

Entrepises want to have insight into complex change processes. The development of coherent views of an enterprise and a disciplined architectural working practice significantly contribute to the solution of this complex puzzle. Coherent views provide insight and overview, enable communication among different stakeholders and guide complicated change processes. Unfortunately there is a downside to this euphoria. So far no architecture description language exists that fully enables integrated enterprise modelling.

There is a need for an architecture language that enables coherent enterprise modelling. Architects need proper instruments to constructs architectures in a uniform way. Figure 1 illustrates the scope of such an integrated set of architecture instruments. Important elements of such an approach include:

- The development of a coherent enterprise modelling language.
- Development of specialised views and visualisation techniques in order to provide insight for different stakeholders.
- Development of analysis techniques that aid in understanding the complex models.
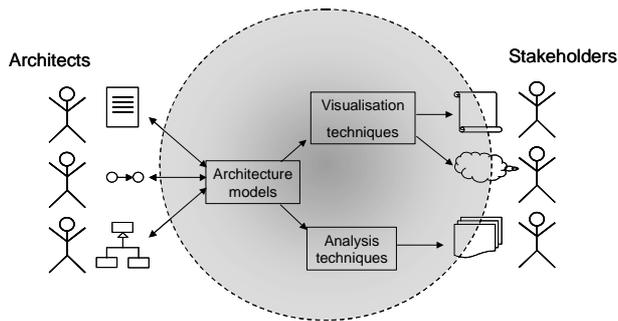
Figure 1. Scope of architecture support

By using a uniform modelling language architects can avoid a Babel-like confusion. At the same time an architectural modelling language should allow the development of specialised visualisation techniques for different stakeholders, such as end-users, project managers, system developers, etc. After all, architectures are the means by which architects communicate with the different stakeholders, and this communication works best if it is tailored towards the specific concerns and information needs that they have. Additionally, analysis techniques, for example, impact-of- change analysis, provide ways to study the properties of an integrated model in more detail. In this way architecture provides the desired insight and overview, which allows a well-organised change process. We realise that multiple languages and dialects will always exist. Striving for one unique language would be like chasing windmills. Therefore, the flexibility to use other languages is recognised, and is addressed by means of a specialisation and generalisation requirement of the language itself. In our view a well-defined enterprise architecture language forms the core of such an architecture approach. In this paper we focus on the requirements and a first design of such a language. It is not our intention to re-invent the wheel for each architecture domain. When possible we follow standards, such as UML, as closely as possible. The focus is on the identification of specific relationship concepts and the definition of cross-domain relations.

In order to arrive at a coherent architectural description, several architectural domains and layers as well as their relations must be modelled. This paper describes the first steps towards a language to support this. The relations between the business and application layer, which play a central role in this version of the language, are a first contribution to the solution of the business-ICT alignment problem that we try to tackle.

The structure of this paper is as follows. In Section 2 we give an overview of related work. Section 3 describes principles that provide requirements for our language. In Section 4 the actual metamodel is presented. Section 5 illustrates the use of the language with an example. Fi-

nally, in Section 6 we draw some conclusions and give some suggestions for future work.

2. Related work

For the state of the art in enterprise modelling, we have to consider languages for organisation and process modelling and languages for application and technology modelling. Although there is a trend towards considering the relationship between the organisational processes and the information systems and applications that support them (often referred to as "business-IT alignment), modelling techniques to really express this relationship hardly exist yet.

A wide variety of organisation and process modelling languages are currently in use: 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. Also, most languages are not really suitable to describe *architectures*: they provide concepts to model, e.g., detailed business processes, but not the high-level relationships between different processes. 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, specifies the Business Process Specification Schema [4]. 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 [1] 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 [9], originating from the US Ministry of Defence, 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 [16] 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 [5], 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.

In contrast to organisation and business process modelling, for which there is no single dominant language, in modelling applications and technology the Unified Modelling Language (UML) [3], has become a true world standard.

UML is the mainstream modelling approach within ICT, and its use is expanding into other areas, e.g., in business modelling [6]. Another example is the UML profile for 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 [15]. This makes UML an important language not only for modelling software systems, but also for business processes and for general business architecture. UML has either incorporated or superseded most of the older ICT modelling techniques still in use. However, UML 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 UML is the large number of diagram types, with poorly defined relations between them. Given the importance of UML, other modelling languages will likely provide an interface or mapping to it.

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. In [13] the basics of ADLs are described and the most important ADLs are compared 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 [8] 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 [19]. 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.

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 [11].

Although the above overview shows that there is a fairly complete language coverage of the the seperate architectural domains, the integration between the languages for the different domains is weak. In this paper, therefore, we focus on a language that makes this integration possible. Within the architectural domains, we reuse elements from existing languages as much as possible.

3. Language requirements and principles

In this section we discuss the principles underlying our approach, which provide requirements for the architecture description language.

3.1 Metamodel flexibility

A key challenge in the development of a general metamodel for enterprise architecture is to strike a balance between the specificity of the concepts used in different organisations and a very general set of architecture concepts which reflects a view of systems as a mere set of interrelated entities. This effort is illustrated in Figure 2. 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. At the top of the triangle we find the "most general" metamodel for system architectures, essentially a metamodel merely comprising the notions of "thing" and "relationship".
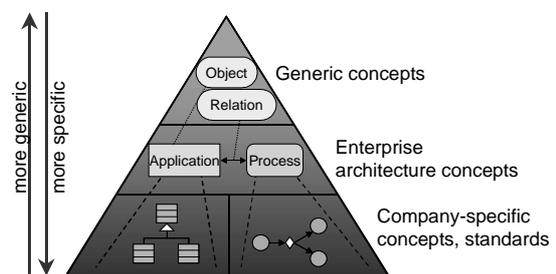
Figure 2. Concepts at three levels of specificity

The metamodel that we propose defines the concepts somewhere between these extremes, referred to as 'enter-

prise architecture concepts'. These concepts 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. Alternatively, as will be explained in the next subsection, the enterprise architecture concepts can be used to integrate more specific models described in other languages.

The enterprise architecture concepts themselves can be defined as specialisations or compositions of the generic concepts at the top of the triangle. Another way to look at this is to view the generic concepts as a general means to define the enterprise architecture concepts: they can be considered the concepts to describe the metamodel. This is a powerful tool to attain metamodel flexibility (see, e.g., [12]). This approach is very similar to OMG's Meta Object Facility (MOF) [14], which, at the highest abstraction level, defines a hardwired meta-metamodel that is used to define metamodels for different languages. It is subject to further study within the project whether the MOF meta-metamodel can be used as the basis for our most generic language.

3.2 Integration of heterogeneous models

In current practice, architectural descriptions are heterogeneous in nature: each domain has its own description techniques, textual or graphical, informal or with a precise meaning. One of the most important goals of our metamodel is to bridge the gaps between these domains, by providing a common conceptual foundation for architectural descriptions.

There are two ways in which the incorporation of different languages can be achieved:

- The concepts of other languages can be described in terms of our general concepts, for example, as specialisations or compositions of these concepts. In other words, the complete descriptions are *translated* into our model.
- Descriptions in other languages, or parts thereof, can be *associated* with objects in our model. This may be done in a 'formal' way, in which certain 'main' concepts from the original language are mapped onto our concepts. However, a simple link, for example a text document is also possible. The models in the original language remain intact. This solution is illustrated in Figure 3.

An advantage of the former solution is that analysis and visualisation techniques defined for the enterprise architecture concepts can be applied to the entire model. An advantage of the latter solution is that existing descriptions can be reused as a whole, in a form that is still rec-

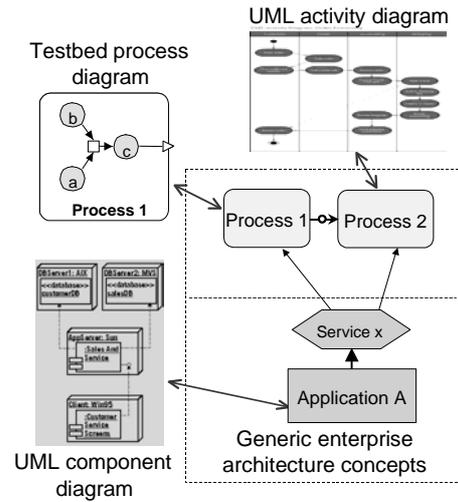ognisable by the original designer. Our metamodel should allow for both types of model integration.

Figure 3. Linking heterogeneous models

3.3 Multiple views and visualisations

In accordance with IEEE standard 1471 [10], we assume that, given an architectural description, different views on this model can be created. These views only show selected aspects of the complete description that are relevant for a certain type of stakeholder. Views are described with the same concepts (or a subset of the concepts) used for a complete architectural description.

Another important principle in our approach is that we separate the definition of the concepts and their representation. For the precise description of concepts it suffices to define the abstract syntax [7] and their semantics (depending on the application of the models, e.g., to perform certain types of analysis). The concrete syntax, i.e., the actual (graphical) notation that is used to represent the concepts and their relationships, can be chosen independently of their formal definition; this notation may depend on, for example, the selected view or the preferences of an organisation. Figure 4 illustrates the separation of the (input) model, views on this model and the representation of these views. A viewpoint definition, based on stakeholder concerns, determines the selection (or derivation) of view content and the way in which this content is presented (or visualised) to the stakeholder. In certain view presentations, it may be possible to modify the view content, which in turn may modify the original model. This is indicated by the 'update' arrows in the figure.
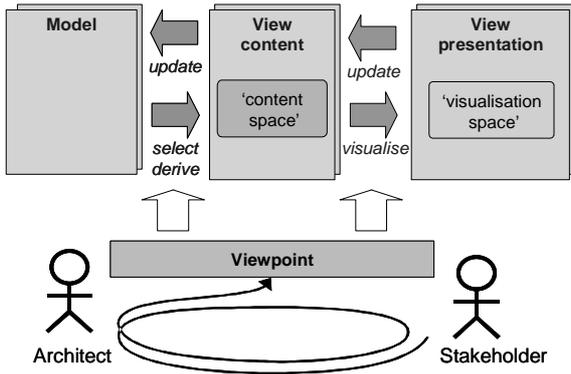
Figure 4. Separation of models, views and presentations

For practical reasons, however, it may be useful to define a 'basic representation' of the concepts, as we will do to express our example in Section 5.

4. The metamodel

In the previous sections the requirements for an architecture language were discussed. In this section we further explore the design of such a language resulting in a first version of a metamodel for coherent architecture descriptions.

4.1 Framework

When studying architecture methods like TOGAF (see http://www.togaf.org) or tools like ARIS [16], and taking into account our experience in actual organisations, it appears that roughly the following architectural domains can be distinguished:

- The *Product* domain describing the products or services that an enterprise offers tot its customers
- The *Organisation* domain describing the actors (employees, organisational units), and the roles they may fulfil, working togehter in *processes* to deliver *products*.
- The *Process* domain describing business processes or business functions that offer products or services
- The *Information* domain describing information that is relevant from a business perspective
- The *Data* domain describing information suitable for automated processing
- The *Application* domain describing software applications that support business processes or functions
- The *Technical infrastructure* domain describing hardware platforms and technical communication infrastructure needed to support applications.

As observed earlier, an important requirement for our language is to abstract from domain-specific concepts as much as possible. Revealing the similarities between the concepts used in the above domains yields a first abstraction that leads to a more generic language. In our view a 'system' in a broad sense, for example, an organisation or software system, primarily consists of a set of actors ("active things") that have at least three aspects that should be considered. Actors have *structure*, i.e. actors can be composed of other actors. In this sense, structure describes the static properties of an actor. Actors show *behaviour* (dynamics) and are likely to exchange *information*.

Next to the identification of these three aspects, we take a common layered approach distinguishing a business, application and technology layer. These aspects together with the different layers constitute a framework (see Figure 5) consisting of nine cells. The cells in this framework show resemblance to of the cells in the Zachman framework [18]. For further clarification the architectural domains mentioned earlier are projected into this framework.

The goal of this paper is not to present a new framework: the framework is mainly intended to guide the design of the metamodel. We observe that to identify relevant concepts that fill the cells in the framework, the framework does not have to be strictly applied. It is impossible and undesirable to define strict boundaries between layers or aspects. Especially considering the fact that we focus on the relation among architectural domains, it is likely that concepts are required to link the various aspects and layers. Typically, such concepts cross the boundaries indicated in the framework.
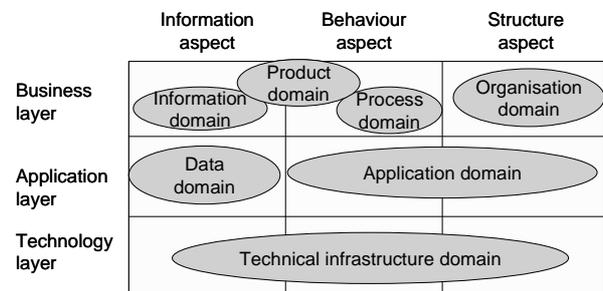
**Figure 5. Architectural framework**

We develop our description language step by step. For each next step we validate concepts and add new concepts or relations. In this, we follow a 'middle-out' approach: the focus of this paper is on the business and application layers. In a later stage, among other things the product domain and the technology layer will be added.

4.2 Relations

As observed in the introduction, this paper focuses on the business and application layers. Moreover, we focus on the concepts that are required to model the "operational"

issues in an enterprise; i.e. the issues that directly contribute to the primary processes and business goals. Our aim is to describe the relations between existing concepts or define specific relationship concepts in order to arrive at the desired coherence. Therefore, we draw inspiration from existing architecture languages or approaches such as UML, Testbed [5] and the RM-ODP Enterprise Language [11].

In addition to the concepts that are required to describe the various architectural domains, inter-domain metamodels are necessary to define the relation concepts between two or more domains. In this way, a hierarchy of domain and inter-domain metamodels can be constructed (see Figure 6).
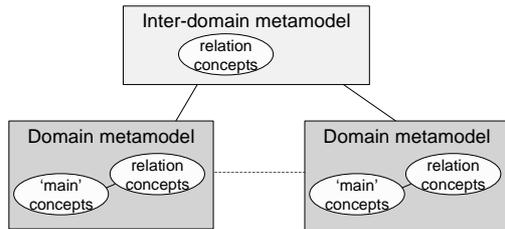
Figure 6. Domain and inter-domain concepts

The order in which the aspects are presented is arbitrary: any two aspects may be related to each other. In contrast, the layers in the framework constitute a functional or system hierarchy. We do not model all inter-layer relations explicitely. Following a common layered approach (e.g., OSI-model) layers are directly related only to layers directly above or below them.

In order to preserve the readability and clarity of models, we do not model the 'diagonal' relations between cells explicitly. In our view these relations are not required for modelling the main coherency. These relations can be derived if necessary.

4.3 Concepts and metamodel

It is our assumption that, in principle, the same generic concepts can be used to describe the structure, behaviour and information aspect of systems in all three layers of the framework in Section 4.1. In spite of the general applicability of these generic concepts, it is still very useful to also define the concepts specific to each layer. These specific concepts are more easily recognised by the relevant stakeholders. Moreover, they are needed to make the relations between the layers explicit, which is an important goal of our approach. In most cases, the layer-specific concepts are straightforward specialisations of the generic concepts.

In Table 1 we first summarise the most important generic concepts that we have identified, after wich we discuss their main relationships.

Concept	Description
Behaviour element	Unit of behaviour. Services can be offered or used by a behaviour element
Action	Atomic behaviour element performed by a single actor
Process	Grouping of causally related actions
Function	Grouping of actions according to, e.g., required expertise, skill, resources, etc.
Interaction	Atomic behaviour element performed by more than one actor
Service	Behaviour made available to the environment. A service is offered by a behaviour element and can be used by another behaviour element.
Transaction	Grouping of interactions with the environment, with a predefined result and with restrictions on the order in which the interactions may occur
Event	Something that happens and may influence behaviour (e.g., a trigger)
Actor/component	Entity that is capable of performing behaviour
Interface	The (logical) location where the behavour of an component can be accessed
Role	Representation of a collection of responsibilities that may be fulfilled by one or more actors
Collaboration/ Connector	Connects roles and interfaces with actors and components respectively
Data object	Representation of information
Message	Data objects intended to be exchanged by actors
Document	Persistent representation of data expressed by means of some medium
Medium	Physical entity or system substantiating data
Information	The interpretation of data as perceived by an actor

Table 1. Overview of concepts

Figure 7 gives an overview of the overall generic metamodel using standard UML notation. It shows the main concepts for each of the aspects, as well as the main links

between the aspects. The relations between aspects are indicated by bold lines and the relations within an aspect by normal lines.
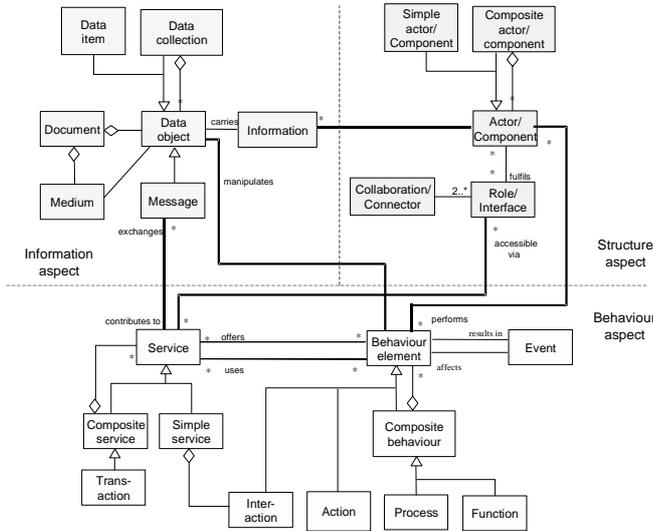

Figure 7. Summary of metamodel

A distinction is made between the externally visible behaviour of an actor (services) and the internal behaviour that is required to realise these services. Services are accessible via the role/interface of an actor, whereas the actor or component itself performs the actual behaviour. A behaviour element can manipulate or use data elements in various ways. A message is exchanged between actors via services. A link between the information and the structure aspect that we distinguish is that information may pertain to a certain actor. Manipulation of data by an actor always involves behaviour. Clearly, there are more direct relations between the structure domain and other aspects such as governance and responsibility. However, this does not fall under the "operational" view that we consider in this paper.

Up to now we considered the relations between aspects. The corresponding metamodel in Figure 7 is generic in the sense that it applies to both layers. In Table 2 we give a possible translation of the most important concepts to more specific terms for the business layer and the application layer concepts.

Figure 8 shows a condensed version of the metamodel worked out for the business and application layer, emphasising the relations between the layers.

Generic	Business layer	Application layer
Action	Business activity	Operation
Process	Business process	Flow
Function	Business function	Software function
Interaction	Business interaction	Application interaction
Service	Organisational service	Application service
Transaction	Business transaction	Application transaction
Actor/component	Business actor	Application component
Role/Interface	Role	Interface
Collaboration/ Connector	Collaboration	Connector
Data object	Business object	Data object

Table 2. Specialisations of the concepts at the business and application layer
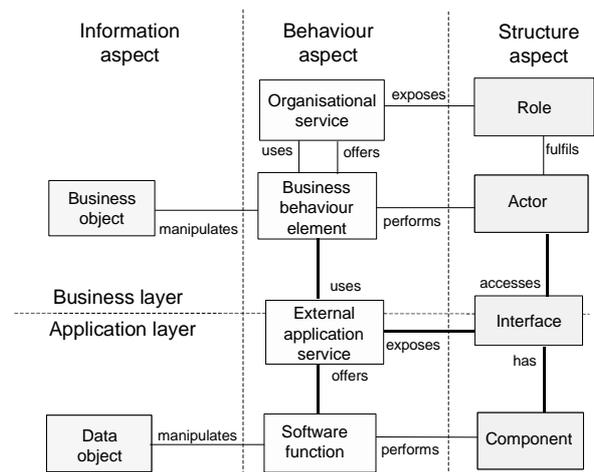

Figure 8. Core of the metamodel

In our view the strongest relation between the layers lies within the behaviour aspect. In line with the trend towards 'service orientation', both at the business level ('service organisation') and the application level (e.g., web services), we relate the layers by means of services (see Figure 9). In each layer, internal and external services are defined. Internal services are offered and used within the layers. External services, on the other hand, are offered by a layer and used by its next higher layer. Moreover, external services of a higher layer may depend on services in the same architectural layer or one layer below. Examples are external business services ("customer services")

or external application services that are used by "the business".
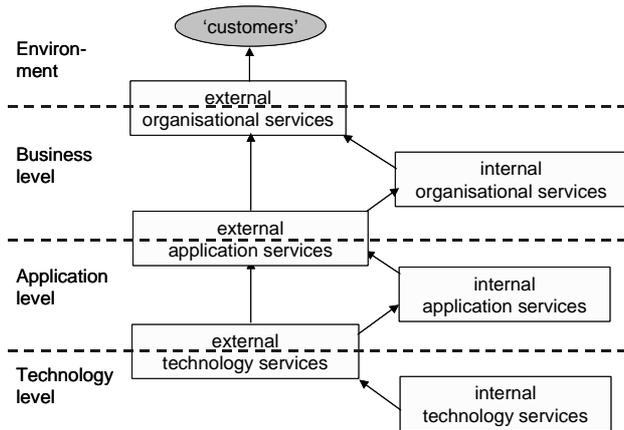

Figure 9. Hierarchy of services

For the information aspect, we do not directly link between the two layers: data objects in the application layer are available to the business layer only through services that are offered by applications. A business object is a unit of information that is relevant from a business perspective. It can be substantiated by a medium like a physical document or text on a computer screen. Without the active mediation of an application service, a business representation of application data cannot be achieved: for example, a printing service (realised by a printer and printing application) is required to transform a Word document into a hardcopy.

As for the structure aspect, an (application) interface is the location where components in the application layer interact with business actors. Therefore, 'interface' can be considered a linking concept comparable to the service concept for the behaviour aspect.

Summarising we observe that behaviour is the central aspect: structure and information are linked through behaviour. We note that the current relations concepts capture the main relations between the concepts from different layers and aspects. It is likely that other relations exist or that further refinement of the relations results in more relation types.
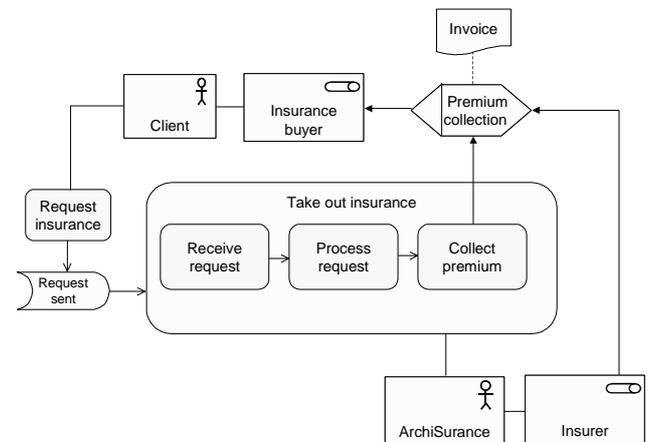
5. Example

Let us illustrate the use of our concepts by means of a simple example. For this purpose, we first propose a basic representation of the concepts.
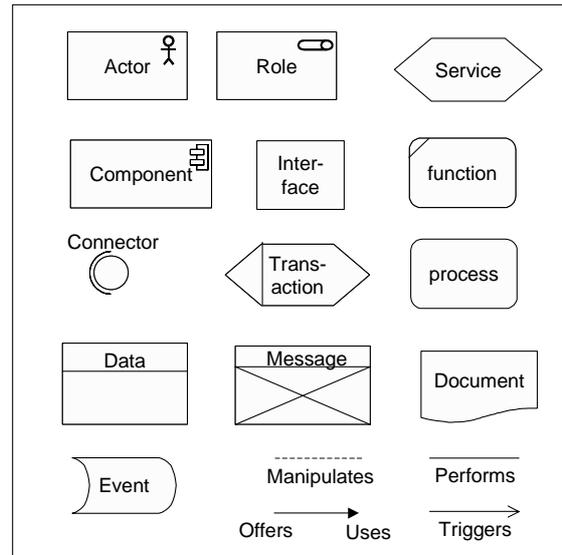
Figure 10. Representation of main concepts

Figure 11 provides an example of a model for the business layer, describing the three aspects and their relationships. It describes a situation where a client requests insurance and receives an invoice for the premium. The model is not complete but shows how business layer concepts can be used.

Figure 11. Example business layer model

The client and insurance company (ArchiSurance) are represented by the Insurance buyer and Insurer role, respectively. The request of the client results in a trigger (open arrow) for the 'Take out insurance' process, which consists of several sub-processes. Each sub-process generates a trigger for the next sub-process, indicated by the open arrow. After the request has been received and processed the sub-process 'Collect premium' offers the 'Premium collection' transaction in which the Insurance buyer

and Insurer settle the agreement. The invoice is sent to the Insurance buyer as part of the collect premium transaction.

Figure 12 provides an example of a model for the application layer. Numerous views on this model are possible that all emphasise other elements of the model. In order to emphasise the three aspects and their relationships, we create a layered view distinguishing the information, behaviour and structure aspects. It describes an application consisting of two components, linked by means of a connector. Each component realises an application function, which in turn offers an application service that can be used by the 'business' (closed arrows). The two application functions are linked by means of an internal application service, which uses a message to transfer the required transaction data. The 'Billing' function also uses 'pricing data', which is internal to this function.
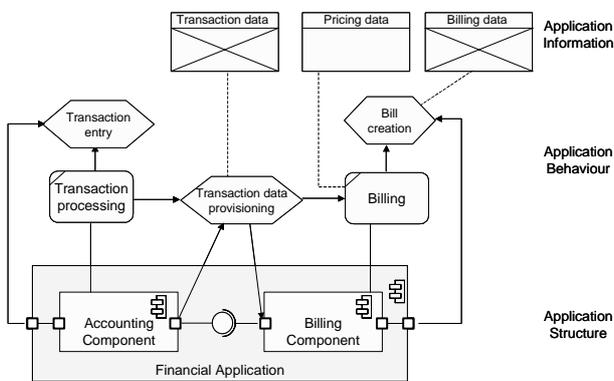

Figure 12. Example application layer model

Joining the two previous figures by relating the subprocesses "Process request" and "Collect premium" with the services "Transaction entry" and "Premium collection" yields a coherent model. In this way the business layer and application layer are linked, taking into account the three aspects and their relationships. From this model the link can be derived between, for example the request for insurance and the application components required to support this request. Adding relevant attributes and assigning appropriate values may allow for more complex types of analysis.

Note that it would be helpful to develop views (see Section 3.3) that may be used to select and visualise the relevant elements from this model (which for this small example already becomes fairly complex). For example, a simple view to show only how application services support the sub-processes at the business layer may provide

useful insight for several stakeholders in the organisation (see Figure 13).
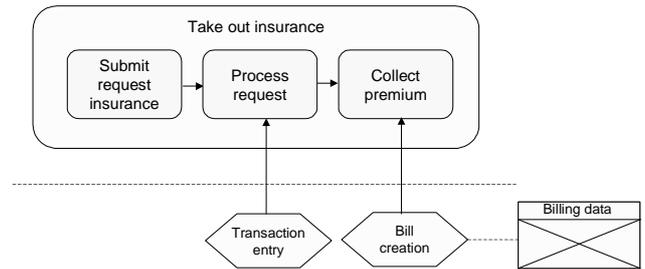

Figure 13. View of the link between the layers

6. Conclusions and future work

In this paper we identified a number of principles and requirements for a language for coherent enterprise descriptions, and we presented a first version of such a language. This languages serves to bring the many separate architectural descriptions for specific domains closer together, as at present no architectural language exists that makes it possible to describe the coherence 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 complement successful and widely adopted languages.

The concepts of our language for enterprise architecture description holds the middle between the detailed concepts used in various organisations and very general architecture concepts which view systems merely as entities and their inter-relations. Proper generalisation and specialisation mechanism to link concepts from a generic architecture language and specific modelling languages are still required for the practical application of the language.

The language forms a basis for bridging the heterogeneity of existing languages. Although the details still need to be worked out, potentially models originating from various tools can be linked. This stimulates possible reuse in a form that is still recognisable for the original designer.

In an architecture that encompasses several models, multiple views provide an essential instrument to handle the complexity. Based on the complex coherent model, relevant information can be selected depending on the stakeholder concerns. Likewise, it is possible to present this information in a way that suits the stakeholder.

Concepts in our metamodel have been inspired by international standards and cover the business and application layers. For each layer the information, behaviour and

structure aspects are described, as well as the main relations between these aspects. Moreover, the relations between the business and application layers are identified. We think that services are a suitable way to relate the layers with respect to the behaviour aspect. The relations between the layers with respect to the other aspects are weaker. Both the structure and information aspect between two layers are linked mainly through behaviour.

By means of a simple example we showed that our concepts can be used to make a coherent description covering all aspects and layers within an enterprise. Even this limited example demonstrates that the complexity of the integrated models will be a problem. The development of views that select and visualise relevant elements from these models for specific stakeholds helps to fully exploit the models.

The work described in this paper is part of an ongoing project called ArchiMate. Here we focus on the general requirements of an architecture language and the core concepts and their relations. 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 technological infrastructure layer and product domain.
- Formalisation of the metamodel to allow for analysis or automated visualisation.
- Identification of relevant viewpoints and related visualisations.
- Integration with other tool support environments.
- Further practical validation of the metamodel.

Acknowledgements

References

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

[2]   Boer, F. de, M. Bonsangue, R. van Buuren, L. Groenewegen, S. Hoppenbrouwers, H. Jonkers, M. Lankhorst, E. Proper, A. Stam and L. van der Torre, *Concepts for Architectural Description*, H. Jonkers (ed.), ArchiMate deliverable D2.2.1, version 1.0. Telematica Instituut, TI/RS/2003/007, Enschede, the Netherlands, Jan. 2003.

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

[4]   Business Process Project Team, ebXML Business Process Specification Schema Version 1.01, UN/CEFACT and OASIS, 11 May 2001.
http://www.ebxml.org/specs/ebBPSS.pdf

[5]   Eertink H., W. Janssen, P. Oude Luttighuis, W. Teeuw and C. Vissers, 'A business process design language'. In: *Proceedings of the 1st World Congress on Formal Methods*, Toulouse, France, 1999.

[6]   Eriksson, H.-E. and M. Penker, *Business Modeling with UML: Business Patterns at Work*, J. Wiley, 2000.

[7]   Erwig, M., 'Abstract syntax and semantics of visual languages', *Journal of Visual Languages and Computing*, 9 (5), 1998, pp. 461-483.

[8]   Garlan, D., R.T. Monroe and D. Wile, 'ACME: An architecture description interchange language', in *Proceedings of CASCON '97*, Toronto, Candada, Nov, 1997, pp. 169-183.

[9]   IDEF, *Integration Definition for Function Modeling (IDEF0) Draft*, Federal Information Processing Standards Publication FIPSPUB 183, U.S. Department of Commerce, Springfield, VA, USA, Dec. 1993.

[10] IEEE Computer Society. *IEEE Std 1471-2000: IEEE Recommended Practice for Architectural Description of Software-Intensive Systems*, Oct. 9, 2000.

[11] ITU–T Recommendation X.911 ISO/IEC 15414. *Information technology – Open Distributed Processing – Reference Model – Enterprise Language*. International Telecommunication Union, 2001.

[12] Karagiannis, D. and H. Kühn, 'Metamodelling platforms'. In K. Bauknecht *et al.* (eds.), *Proceedings 3rd International Conference EC-Web 2002 - DEXA 2002*, Aix-en-Provence, France, Sept. 2002, p. 182. (Springer-Verlag, LNCS 2455.)

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

[14] Object Management Group, *Meta Object Facility (MOF) Specification*, version 1.4, April 2002.

[15] Object Management Group, *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

[16] Scheer, A.-W., *Business Process Engineering: Reference Models for Industrial Enterprises*, Springer, Berlin, 2nd ed., 1994.

[17] Soley, R. and the OMG Staff Strategy Group, *Model Driven Architecture*, Object Management Group White Paper, Draft 3.2, Nov. 2000.

[18] Sowa, J.F. and J.A. Zachman, 'Extending and formalizing the framework for information systems architectures', *IBM Systems Journal*, 31(3), 1992, pp. 590-616.

[19] U2 Partners, *Unified Modeling Language: Superstructure*, version 2.0, 2nd revised submission to OMG RFP ad/00-09-02, Jan. 6, 2003.