Available online at www.sciencedirect.com
SCIENCE DIRECT
Electronic Notes in Theoretical Computer Science
ELSEVIER Electronic Notes in Theoretical Computer Science 101 (2004) 155–179
www.elsevier.com/locate/entcs

Investigating the mapping of an Enterprise Description Language into UML 2.0

M.J. Wiering[a], M.M. Bonsangue[a], R. van Buuren[b], L.P.J. Groenewegen[a], H. Jonkers[b] and M.M. Lankhorst[b]

[a] *Leiden Institute of Advanced Computer Science*
Leiden University
Leiden, The Netherlands
{*mwiering,marcello,luuk*}*@liacs.nl*

[b] *Telematica Instituut*
Enschede, The Netherlands
{*rene.vanbuuren,henk.jonkers,marc.lankhorst*}*@telin.nl*

Abstract

Business architects and process engineers, when modelling their organisation or parts of it, prefer not to use the Unified Modelling Language (UML) as they find UML too technical for their taste. Instead they use their own modelling languages, which are more intuitive for formulating business-oriented models. As these techniques are often less formalized or widely used, tool support is usually inadequate or even totally absent. Furthermore, in view of the need to relate such business models to existing or possibly future ICT systems and architecture, the non-UML business models have to be integrated with the UML models commonly used in the ICT domain.
A concrete case of this is the enterprise architecture language developed by the ArchiMate project. This paper describes a mapping of this language onto UML 2.0. On the one hand, this mapping is needed to concretize the relation between models of the business and ICT domains. On the other hand, this mapping will provide the basis for developing a UML profile for the ArchiMate language, facilitating the use of UML tools for making ArchiMate models. In this way, one is able to compensate the disadvantages of writing business models in a non-UML language.

Keywords: enterprise architecture, UML, business process

1 Introduction

Modern enterprises are active in a large number of domains, have to deal with quite many as well as diverse stakeholders and are supported by a great number of different software systems. All activities taking place in the domains,

doi:10.1016/j.entcs.2004.02.020

performed by the stakeholders or offered by the supporting software have to be sufficiently aligned. To facilitate such alignment a description of the enterprise and its activities must be present, providing an overview of what is done where, when, by whom and why. Even the question "how" should be answerable, at least globally, if only a few details are required. Such a descriptive overview is commonly referred to as enterprise architecture.

As modern supporting software systems do large parts of core business activities, enterprise architectural languages take special care to reflect the relevance of software systems in the general business. This is not easy, as the world of software and ICT mainly is rather technical and deep, whereas the world of business is much broader.

Therefore, enterprise architectural descriptions often discriminate between different levels or layers within the enterprise, like e.g. the business level, for the various business domains consisting of the business processes descriptions, the application level, describing the supporting software systems from the outside in terms of functionality provided and (business) processes monitored and guided, and the technology level, providing the physical organisation of the supporting software systems. These levels must be described on their own as well as in relation to each other.

To that aim, a wide variety of description and modelling languages languages are currently in use. No such language however has been yet recognized as a standard in the organizations and business process modelling domain. Examples are the RM-ODP Enterprise Language [9] still unsuccessfully promoted by the ITU/ISO as a standard for enterprise architectures, the Business Process Modelling Language BPML [1] an XML-based language for modelling business processes that has roots in the work-flow management world; IDEF [8], a collection of unrelated diagramming techniques for function modelling, information and data modelling and process description; ARIS [15] a language with focus on business process modelling and organization modelling; and Testbed [3] a formal language for business process modelling suitable for different types of analysis.

1.1 Role of UML

In contrast to organization and business process modelling, for which there is no single dominant language, in modelling applications and technology the Unified Modelling Language (UML) [2], has become a true world standard. UML is an important language not only for modelling software systems, but can be used for business processes using patterns [5] and for general business architecture (see for example the UML profile for Enterprise Distributed Object Computing (EDOC) [12]). However, UML is not easily accessible and

understandable for managers and business specialists [16]; therefore, special visualizations 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: UML 2.0 [13,14] consists of 13 different mostly unrelated diagrams, each with a different semantics. This means that, without some additional properties or framework, there is no hope of using the current UML 2.0 for modelling the different levels in an enterprise in a sufficiently consistent manner: relations between diagrams on these levels will certainly not remain restricted to diagrams from one diagram language only, so their consistency remains unclear.

Architecture description languages (ADLs) define high-level concepts for architecture description but mostly with a focus on software architecture. The ADL ACME [6] is widely accepted as a standard to exchange architectural information, also between other ADLs. There is a translation mapping ACME concepts into UML 2.0 [7], with goals similar to those of this paper: concepts in one domain are made available to a large user base and can be supported by a wide range of software tools.

1.2 Goal of this paper

The preceding description and modelling languages cover different architectural domains, but the integration among the domains offered by these languages is weak. The modelling language developed by the ArchiMate project [11,10] aims at describing enterprise architectures in a structured, well-defined way, covering the different levels of the enterprise in an integrated manner. Such an architectural description should serve as a basis of stability amidst organizational change: on the basis of such an architecture, interaction and alignment on one level as well as between levels should be analysable and also impact analysis of minor or major adoptions or changes in one domain or with respect to one software system should become feasible, realistic or perhaps even standard.

To that aim, ArchiMate has developed a metamodel consisting of concepts and their relations in terms of which particular architectural models can be formulated. The metamodel has four representations: one general form for the three levels "in general", and three somewhat more specialized forms, one for each level "in particular". In this way the ArchiMate metamodel defines the ArchiMate language for specifying any architectural model in the context ArchiMate is aiming at. The ArchiMate language is not meant to replace UML as a software modelling language, but rather to supplement it with business-oriented concepts at a more abstract, architectural level.

The goal of this paper is to relate the ArchiMate modelling language to the

concepts and diagrams of UML 2.0. The aim of this is twofold. On the one hand, this mapping is needed to concretise the relation between more abstract and "broad" models at the enterprise architectural level and more concrete and "narrow" software models specified in UML. On the other hand, this mapping will provide the basis for developing a UML profile for the ArchiMate language, facilitating the use of UML tools for making ArchiMate models.

The paper has the following structure. Section 2 presents the UML2.0 in terms of the diagrams (sublanguages) within the UML. The ArchiMate metamodel, containing the concepts and relations, is presented in Section 3 and the translation of this metamodel into UML 2.0 is formulated and discussed in Section 4. We proceed as follows; first we map concepts from ArchiMate to UML, preserving its intuitive meaning. In Section 5 we take a concrete model of ArchiMate and derive concrete diagrams of UML by applying our translation. In particular, we apply this to a specific business case called ArchiSurance. Section 6 contains a discussion and conclusions with respect to the translation itself, with respect to UML's suitability for modelling business processes and business architectures and with respect to the various views for different types of stakeholders.

2 Concepts and Diagrams in the UML 2.0

UML 2.0 consists of 13 languages each having its own diagram notation visualising the UML concepts to model a specific aspect of a software system. These diagrams can be grouped in three categories of UML diagrams:

- `Structure concepts and diagrams`: concepts like class and object are used to create diagrams that depict the elements of a specification that are irrespective of time. This includes class, object, package, composite structure, component and deployment diagrams.
- `Behaviour concepts and diagrams`: concepts like use case and interaction are used to create diagrams that depicts behavioural features of a system or business process. This includes activity, state machine and use case diagrams as well as the four interaction diagrams.
- `Interaction concepts and diagrams`: a subset of behaviour diagrams which emphasize object interactions. Concepts like interaction are used to create these diagrams. This includes communication, sequence, interaction overview and timing diagrams.

Many of the diagrams are well known since years already, so we will only introduce the new diagrams. The description of these diagrams will be introduced in the same order of type of diagrams (Structure, Behaviour, and

Interaction) as appeared above. The new diagram for modelling Structure is:

- `Composite structure diagram`: a subset from classes of a class diagram, moreover showing how they are physically connected. Particularly, the relationships in a composite structure diagram depict the physical connections between (logical) elements (like classes/objects, packages, use cases). There are two appearances: via ports and connectors or as collaboration (of roles: views on elements).

The new diagram for modelling Behaviour is:

- `State Machine diagram`: it describes local behaviour through state changes of a concrete object while in execution, by performing internal steps or driven by messages received from outside or by sending messages out. At each point in time the current state represents the situation reached by the execution so far. Thus a State Machine diagram models the local flow of control from state to state within an object, i.e. within a conceptually small part of the system.

The new diagrams for modelling Interaction are:

- `Interaction Overview diagram`: it combines the flow structure of activity diagrams with sequence diagrams. It does so by replacing activities by sequence diagrams; their lifelines, so to speak, correspond to the swim lanes. So all "activities" are formulated in terms of interaction scenarios. By reusing the flow structure of activity diagrams, interaction overview diagrams are "complex" sequence diagrams with many smaller sequence diagram-like fragments.
- `Timing diagram`: combines a sequence diagram and explicit time; instead of lifelines, sequences of (local) state changes specify the scenarios where time annotations make time explicit. It is typically used to show the subsequent state changes of one or more objects over time in response to events, internal as well as external.

The 13 diagrams give software engineers sufficient means to get better understanding of the problem(s) for which they want to engineer a software system and of the solution(s) they want to incorporate into the software system. Using UML the software engineer is able to model the situation which the system-to-be is supposed to solve, to model the architecture of the system, i.e. the global structure in terms of the main building blocks of the system, and of course to model the precise design of the system.

Although the original ideas behind object-orientation were geared towards modelling software system and their use, insight is growing about the applicability of object-orientation to modelling more general dynamic systems. A

strong reason for this insight is that many software systems themselves contain or even are a dynamic model of a concrete organization or of a concrete business process. So the object-oriented model of such a software system itself contains or even is a model of that particular organization or of that particular business process. In principle this indicates a certain suitability of object-orientation with respect to modelling organizations and their activities or processes.

Summarizing the above, UML offers a set of 13 diagrams for modelling different aspects of software systems. But these diagrams may be used for modelling more general dynamic systems as well.

However, as UML and its concepts originally were not geared towards modelling organizations and their activities, let alone towards business architecture, it is not a priori clear how to apply the UML concepts to modelling business architecture, see e.g., [16]. ArchiMate has developed a metamodel for modelling enterprise architecture. By investigating how the metamodel can be translated into UML, as a by-product we gain more insight into applying the UML concepts to modelling organizations, business processes and business architecture on the basis the guidance provided by the ArchiMate metamodel.

This brings us back to the topic of this paper: the *mapping* of the ArchiMate into UML. In the next section we briefly present and discuss the metamodel.

3 The ArchiMate Metamodel

The ArchiMate project has resulted in an enterprise architecture modelling language and a corresponding metamodel. This metamodel provides concepts for architecture designs in the context of ArchiMate on a most general level, covering business, application, and technology. The language is being developed by the ArchiMate project and has evolved over the course of the past one and a half years based on validation in practical (customer) cases and because of better understanding and insight of the concepts and their relations.

In ArchiMate's metamodel, experience with and knowledge of other modelling languages has been incorporated. On the one hand, the metamodel is not object-oriented, in that it separates structure from dynamics. In this, it resembles the Testbed language [3]. On the other hand, it has a UML-flavour where it connects interfaces, providing services to roles and grouping roles to form collaborations.

The general ArchiMate metamodel is shown in Figure 1. In addition, specialized forms of the general metamodel exist for the business, application, and technology levels [10]. In the remainder of this paper, we restrict ourselves

to the business and application levels, as these parts of our language are relatively stable.

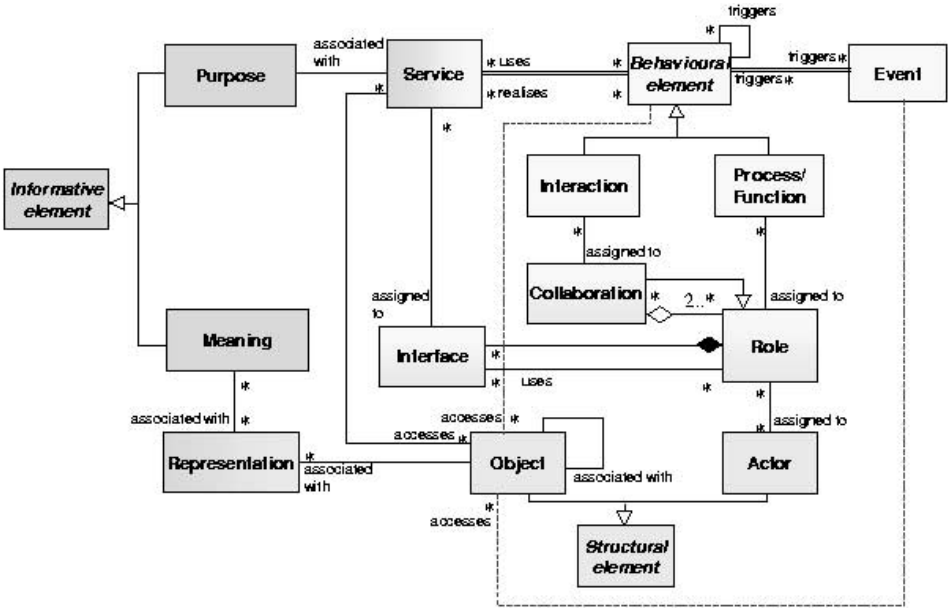

Fig. 1. Metamodel of the ArchiMate Concepts

Figure 1 shows the ArchiMate (core) concepts and their relationships. The metamodel can be grouped according to three different aspects. The three groups are: Structural concepts to model the structure of the (software) system; Behavioural concepts to model the behaviour of the structured (software) system; Information concepts to model the information that is communicated through the (software) system and the behaviour that is affected by the information.

We will refer to the three aspects as the Structural, the Behavioural and the Informative aspects. Each group consists of the following concepts.

- `Structural concepts`: Structural Element, Object, Actor, *Representation*, *Interface*, *Role*, *Collaboration*.
- `Behavioural concepts`: Behavioural Element, Interaction, Event, Process, Function, *Collaboration*, *Interface*, *Role*, *Service*.
- `Informative concepts`: Informative Element, Purpose, Meaning, *Representation*, *Service*.

The concepts in *italic* are concepts that are present in two groups, i.e. they combine two aspects: as such they function as bridges between these two aspects. The concepts are briefly discussed in the following subsection. The

translation of these concepts to UML is given immediately after it in the next section.

3.1 Definitions of the ArchiMate Concepts

For this section we reformulate the existing definitions of the ArchiMate concepts. Our reformulations of the definitions are an understanding of performed to get the ArchiMate concepts more geared toward facilitating their translation into UML. The concepts are grouped by the three aspects mentioned in the previous section, Structural, Behavioural and Informative.

The original formulations of the ArchiMate concepts can be found in [11]. Note that this metamodel is fairly general: as it covers both business domain (business level) and ICT domain (application level). So a concept covers two levels. Nonetheless we shall present separate (but rather similar) definitions for each level.

3.2 Structural concepts

- `Structural Element`: A Structural Element is used to model any structural entity in the levels. This element strictly models the structural aspects. Its behaviour is modelled by an explicit relationship to behavioural concepts.

  The Structural Element is generic. These elements are 'abstract' concepts of which no instances appear in models: only instances of their specializations are used. A Structural Element can be an Actor (active) or an Object (passive).
- `Object`: An object is a logical or physical unit of information that has relevance from a business perspective. Business objects are the passive entities that are manipulated by behaviour such as business processes or functions. Business objects represent the important items, relevant for the business domain.

  A data object is a coherent, self-contained piece of data suitable for automated processing. This concept is used in the same way as data object (or object types) in well-known data modelling approaches, most notably the class concept from UML but restricted to arbitrary attributes together with the CRUD operations (create,read,update,delete). An example of an object is a database, indeed allowing for creating, reading, updating and deleting data.
- `Actor`: Actors are the active entities that perform behaviour such as business processes or functions. In the business domain, 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 organizations.

In the application domain, the actor is an application component. It is used to model any structural entity in the application level: not just software components (part of one or more applications), but also complete software (sub)applications or information systems. Although very similar to the UML component, the ArchiMate component concept strictly models the structural aspects of an application: its behaviour is modelled by separate behavioural concepts, connected to such a component via explicit relationships.

Within the context of collaboration between actors, the dynamic part often contains one or more descriptions of roles as relevant for the collaboration.

- `Role`: The Role concept can be a business role or a role of an application component. A business role represents the work that a business actor may perform within an organization. The set of roles in an organization can be expected to be much more stable than the specific actors fulfilling these roles. Multiple actors can fulfil the same role, and conversely, a single actor can fulfil multiple roles.

  The role of an application component shows its (possible) behaviour. The behaviour of the role is modelled by an explicit relationship to the behavioural concepts.

- `Interface`: The interface concept can be a business interface or an application interface. A business interface represents the location for accessing the services offered by a business role to the environment. In a broader sense, a business interface also has some behavioural characteristics. It may e.g. list the services offered. The same service may be offered via different interfaces.

  An application interface represents the location where the services of an application component can be accessed. It defines the set of operations and events that are provided by the component, or those that are required from the environment.

- `Collaboration`: Collaborations represent two or more roles that cooperate to realise certain behaviour. Architectural descriptions focus on structure, which means that the interrelationships of entities within an organization play an important role. To make this explicit, the concept of business collaboration has been introduced. Collaboration is a collective of roles within an organization which perform collaborative behaviour.

  A business collaboration does not need to have an official status within the organization: it is specifically aimed at a specific interaction or set of interactions between roles. However, a business collaboration can be

regarded as a kind of 'virtual role', hence its designation as a specialization of role. It explicitly names the context for interaction between actors via their various relevant roles.

In application architecture, the interrelationship of components is an essential ingredient. Therefore, the ArchiMate language also introduces the concept of application collaboration here, defined as a collective of application components which perform application interactions.

3.3 Behavioural concepts

- `Behavioural Element`: A behavioural element can be a process/function (performed by one role) or an interaction (performed by a collaboration of two or more roles). It expresses what the system actually does, possibly in response to a service being requested or an event being sent. Just like the structural element, this element is a generic (or abstract) element, thus no real instance of the behavioural element present only its specializations.
- `Interaction`: The interaction concept can be a business interaction or an application interaction. In both cases it is a collective unit of behaviour performed as a collaboration of two or more roles. A business interaction represents the work performed by two or more cooperating business roles in business collaboration.

  An application interaction is the behaviour of application collaboration. An application component has external behaviour from the perspective of each of the participating components, but the behaviour is internal to the collaboration as a whole.
- `Process/Function`: The concept Process/Function can be a business process/function or an application function.

  A business process/function 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. A business function offers functionality useful for one or more business processes. It groups behaviour based on, e.g., capabilities, resources, etc. Typically, the business processes of an organization are defined based on the products and services that the organization offers, while the business functions are the basis for, e.g., the assignment of resources to tasks and the application support.

  An application function describes the internal behaviour of a component needed to realize one or more application services. It is a separate 'application flow' concept as counterpart of a business process.
- `Event`: An 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. A business event is instantaneous: it does not have duration. Events may originate from the environment of the organization, but also internal events may occur, generated by, e.g., other processes within the organization.

- `Service`: A service is an externally visible unit of functionality that is meaningful to the environment. An organizational service models the externally visible behaviour of a business process or function, as it is offered e.g. to the customers of the organization.

  An application service is an externally visible unit of functionality, provided by one or more components, exposed through well-defined interfaces. The service concept provides an explicit way of describing the functionality shared between components or offered by components to the environment.

  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.

3.4 Informative concepts

- `Informative Element`: An informative element represents some informational value which can be a meaning (static) or a purpose (dynamic). It shows the 'communication' with the surrounding environment. Like the structural and behavioural Elements, this is an abstract concept, and only instances of its specializations, purpose and meaning, are used.
- `Purpose`: the purpose concept represents the functionality of a service seen from the point of view of an external user. It models the intended contribution of a service towards achieving 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 organizational service, seen strictly from the point of view of some external actor.
- `Meaning`: the meaning concept represents the contribution of an object to the knowledge or expertise of some actor, given a particular context. Thus, 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.

- `Representation`: A representation is the perceptible form, e.g., a document, of the information carried by an object. If relevant, representations can be classified in various ways, for example in terms of medium or format. A single business object can have different representations, but a representation always belongs to one specific business object.

These are all ArchiMate metamodel concepts. On the basis of the above characteristics of these concepts, reflecting the essence of the concepts, we can select suitable UML diagrams or other UML notions that can indeed serve as translations of the various ArchiMate concepts. Moreover, the translation turns out to be sufficiently consistent. Its result can be taken as a UML profile for ArchiMate. Such a profile then can be useful for formulating and analyzing matters of alignment as well as of change.

4 UML translation of ArchiMate

A formal definition of semantics for the ArchiMate metamodel concepts is a topic of ongoing research. Similarly, quite a large part of the semantics for UML is still missing.

We therefore take the following approach in translating the ArchiMate metamodel concepts to UML. Instead of matching ArchiMate semantics with UML semantics, we match properties from the above characterizations. In some cases a match can be found for properties corresponding to different UML concepts. That means, the underlying ArchiMate concepts can be translated to these different UML concepts. Where relevant we shall discuss the different possibilities.

After discussing our translation of the various ArchiMate concepts, we shall briefly discuss a translation of the relationships between these concepts.

4.1 Translating the Concepts to UML

We start with analyzing the structural concepts, followed by the behavioural concepts and ending with the informative concepts. The structural concepts are the concepts that are easiest to understand since they are similar to well known structural notions in other formalisms. So they are relatively easy to translate. The behavioural concepts are a bit less common. Finally, the informative concepts are even more difficult to understand and therefore to translate, as they are rather uncommon.

Our translation is as follows. Arguments for our choices of UML concepts are given after the translation:

- `Structural Element` → Class. The class always is an abstract class, as it

will have no instances. It contains the common (structural) attributes for every structural element

- `Object` → Class. Behaviour is of no real interest. So the class only has the CRUD operations (Create,Read,Update,and Delete) to perform basic operations. Furthermore, it can have all kinds of attributes.
- `Actor` → Class. An actor is something (person, department, active piece of software) that performs Roles. Only the structure of such an Actor is described in the class. So the operations corresponding to its specific Role(s) do not belong to it. Instead it has special associations with these Roles (in their translated form) together with special operations playRole(..) via which it can start acting according to a certain role.
- `Role` → Class. A Role can be played by an Actor. It gives the Actor its behaviour. To that aim it offers its operations (Services) to elsewhere via its Interface. Such a Role is part of Collaboration. As soon as an instance of Collaboration exists, the Roles belonging to the Collaboration are started by calling the playRole(..) operation of the Actors playing those Roles.
- `Collaboration` → Class and Composite Structure, in particular in its visual form of a (UML) collaboration. The roles occurring in the UML collaboration are indeed the above UML classes for the Roles. The links between the roles in the UML collaboration are not (immediately) there. The collaboration can be drawn when using Collaboration and Role together.
- `Interface` → (UML) Interface, as occurring in class diagrams and composite structure diagrams. It effectively separates provided and required functionality from implementation.
- `Behavioural Element` → Interaction; Sequence or Activity. This covers the translation of both Interaction and Process/Function. So we refer to the discussion of the translation of these two concepts.
- `Interaction` → Sequence or Interaction Overview. Lifelines correspond to the Roles participating in the Collaboration the Interaction is assigned to. As long as the details of the messages exchanged are unknown, such a sequence diagram has to remain incomplete, as the translation into UML should not reflect details not occurring in the original ArchiMate model. When there is a nested hierarchy of ArchiMate Interactions, an Interaction Overview Diagram can easily reflect a similar hierarchical structure.
- `Process/Function` → Activity or Sequence. Depending on the relevance of the internal behaviour versus the communicative behaviour related to a Role, one can prefer a model reflecting the (internal) activities of a Role, or a model reflecting the interaction with a Role. If the Process/Function is assigned to a Role which itself is Collaboration, the activities taking place

can themselves be organized according to swim lanes. Such swim lanes then correspond to the (smaller, nested) Roles within that Collaboration.

- `Event` → Interaction in Sequence or Activity or Interaction Overview: a concrete exchange of a message, a trigger, a signal, a (UML) event, often between two lifelines. Usually it either triggers a whole sequence of such interactions or it closes such a sequence.
- `Service` → Interface Operation. A Service reflects functionality that is provided or required. The clear separation from any implementation of such a Service remains valid in the translation, as UML interfaces indeed hide such solutions.
- `Informative Element` → Note or Use Case. This covers the translation of both Purpose and Meaning. So we refer to the discussion of the translation of these two concepts.
- `Purpose` → Use Case(s). It expresses intended usage of the model towards the environment of the model by informing about the model's functionality. The phrasing of this usage is not so much in terms of the (internal) design solution of the model but in terms of what is relevant for the model's environment, e.g., business goals or business strategy.
- `Meaning` → Note(s). They express an explanation of structural parts of the model in terms of what is relevant for the model's environment.
- `Representation` → Note(s) or Class. Like the notes corresponding to Meaning, they also express an explanation of structural parts of the model in terms of what is relevant for the environment. But here it is more like a(n extended) view on the underlying structural parts. So the formulation of the explanation, although in terms of what is relevant for the environment, is structurally related to the (data or business) Object they are associated with. If the similarity in structure between the underlying (data or business) Object is strong enough, the UML translation of representation can make this more explicit by means of a class instead of one or more notes.

Summarizing our translation, we present the following list.

- `Structural Element` → Class.
- `Object` → Class.
- `Actor` → Class.
- `Role` → Class.
- `Collaboration` → Collaboration or Class.
- `Interface` → Interface.
- `Behavioural Element` → Sequence or Activity or Interaction Overview.

- `Interaction` → Sequence or Interaction Overview.
- `Process/Function` → Activity or Sequence.
- `Event` → Interaction in Sequence.
- `Service` → Operation of an Interface.
- `Informative Element` → Use Case(s) or Note(s).
- `Purpose` → Use Case(s).
- `Meaning` → Note(s).
- `Representation` → Note(s) or Class.

4.2 Translating the metamodel relations

The relations from the ArchiMate metamodel express how in any concrete ArchiMate model the concrete instances of the different metamodel concepts are to be connected (in a minimal sense). Most of the ArchiMate relation types (associations, specializations, and aggregations) are taken directly from UML, making the translation of an ArchiMate relation to a UML relation very straightforward.

In this way, one can infer the required ArchiMate consistencies in the corresponding UML model. This is also relevant in view of a possible UML profile for the ArchiMate approach. Then support for consistency checking and consistency management with respect to different kinds of UML diagrams, as for instance discussed in [4], can be based directly on the required ArchiMate consistencies.

5 The ArchiSurance Case

In ArchiMate we use a business case called ArchiSurance. The case consists of a relative simple, but nevertheless sufficiently rich example of a number of related business processes within an architectural setting as given by an organization and supported by some software systems. The full case covers a reorganization of an insurance company in terms of the global structure and global execution of their business processes as well of the supporting software systems.

Within the context of the translation to UML, we reuse a small part of the ArchiSurance Case: we restrict our attention to one business process: from a customer getting interested in an insurance to the signing of the insurance contract by that customer. In addition we use the small case both for testing and for explaining. It is via this example we are able to clarify, at least partially, why UML might be considered as less suited for modelling business

processes.

The ArchiSurance case is a business case concerning three parties interacting with each other: an Insurance Company, an Intermediary and a Customer. The Insurance Company cooperates with Intermediaries to get in touch with Customers wanting to get a contract for an insurance or negotiating to see which insurance is best (and hopefully wanting to have a contract after the negotiation). When a Customer has an insurance contract with the Insurance Company, the Intermediary can collect the insurance premium for the Insurance Company. The Insurance Company can handle claims from the Customer without having to interact with the Intermediary.

So far the case has resulted in a few ArchiMate models which will be briefly explained. Before discussing the models we briefly introduce the ArchiMate notation for the various metamodel concepts.

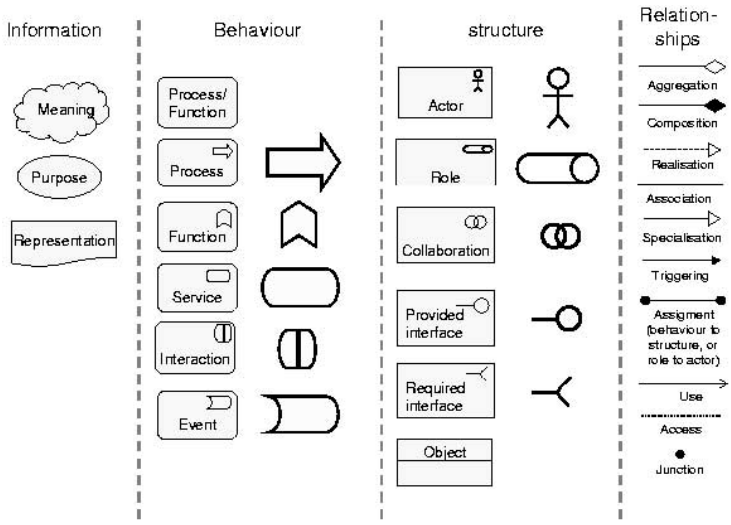

Fig. 2. ArchiSurance Case, notation of the ArchiMate concepts

All notations, categorized according to the Information, Behaviour and Structure aspect are presented in Figure 2. For example as we can conclude from the column 'structure'; the tube icon in a square represents a role and the overlapping circles icon in a square represents a collaboration. The different types of interfaces are shown and also relationships, messages and triggers. The name of the concept is inside the corresponding square, rectangle, etc.

Our selection from the ArchiMate specification of the case consists of two models.

The first model is called 'Business Structure', shown in Figure 3. The model shows three (business) roles Customer, Intermediary and Insurance

Company occurring in the ArchiSurance case. Moreover it shows in which collaboration these roles participate. There are four such collaborations: Negotiation, Contracting and Premium Collection, all with three participating roles; furthermore, claim handling with two participating roles.

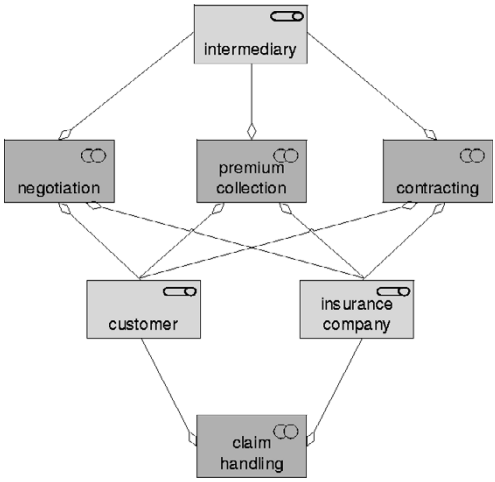

Fig. 3. ArchiSurance Case, model 1, Business Structure

The second model is called 'Business Process' shown in Figure 4. It is a detailed view of a process that shows the interactions occurring between requesting an insurance and eventually registering the signed insurance contract.

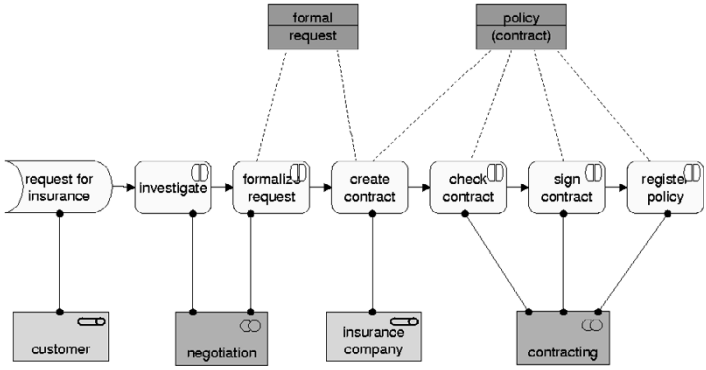

Fig. 4. ArchiSurance Case, model 2, Business Process

In this model more concepts than just role and collaboration are used. The 'request for insurance' is an event, initiating the business process. When the process is triggered by the event there are two interactions within the negotiation collaboration and when that collaboration is finished a process is

started. In this process the contract will be created by the Insurance Company. After the contract has been created three, additional interactions will lead to signing and registering the contract.

5.1 Applying our translation

The translation will be performed on the two models from the case, called 'Business Structure' and 'Business Process', introduced in the previous section.

The translation of both models is from ArchiMate to UML, such that it preserves all properties present in the ArchiMate model, but without adding any property not present in the ArchiMate model. A most important point in such a translation is the amount of detail present in both models. Whereas UML urges one to incorporate a lot more operational details, ArchiMate urges one to stay at a global level with only the relevant details. ArchiMate wants to keep models as global as possible, keeping them to-the-point, e.g., the point of view of a business architect or a manager.

Translation of both models turns out to be fairly straightforward. In both cases however, the resulting UML model lacks detail one normally expects to be present. So the UML models look somewhat incomplete. As both translations are performed step-wise, we shall discuss the effect of the step order on the resulting UML model.

5.2 Translating ArchiMate's Business Structure model

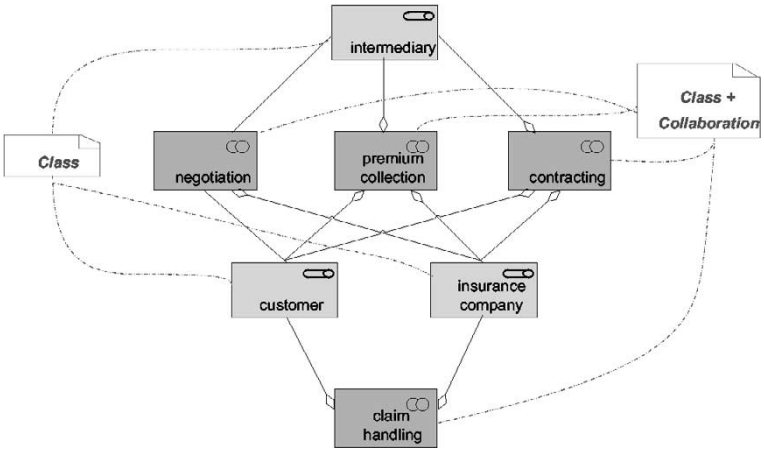

Fig. 5. Business Structure model

The Business Structure model, shown in Figure 5, consists of two concepts: Collaboration and Role. According to our ArchiMate translation, any collab-

oration will be translated into a class as well as into a UML collaboration and any role will be translated into a class.

The translation of the collaborations and roles into separate classes is grouped into one class diagram. Collaboration and role are represented by the stereotype collaboration respectively role. This results in the translation in Figure 6.

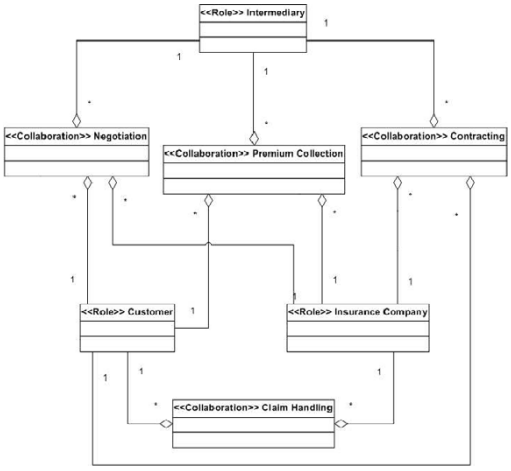

Fig. 6. Class Diagram for the Business Structure

According to the UML description of a collaboration, the classes representing the (translated) roles are to be incorporated into the collaboration. This is reflected in Figure 7. Figure 7 presents the UML translations for all four collaborations. The order of the translation steps apparently does not matter as starting form the three roles, translated to UML classes, would have resulted in the same four UML collaborations.

Normally, a UML engineer is strongly inclined to add links between the classes (roles) inside a collaboration. But by adding these links, one is introducing details totally absent in the original ArchiMate model.

While such details are normal in UML, ArchiMate models are different. Instead of adding such details, ArchiMate leaves such details out as they are considered irrelevant for the global presentation.

ArchiMate's metamodel, formulated in the UML, is deliberately kept global, whereas UML is designed for more operational and constructive details. This results into the conclusion that when translating ArchiMate to UML, the resulting UML translation is used substantially more declarative than usual.

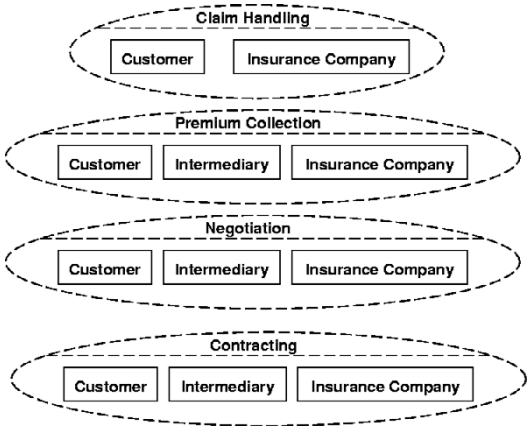

Fig. 7. Collaboration for the Business Structure Collaborations

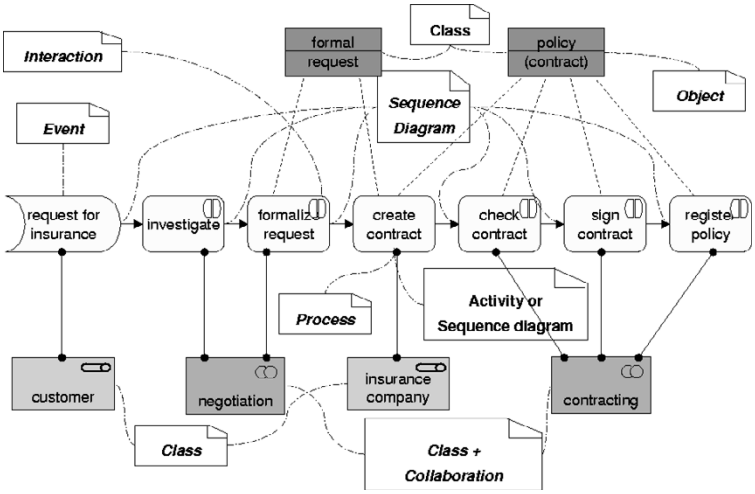

Fig. 8. Business Process model

5.3 Translating ArchiMate's Business Process model

The second ArchiMate model we shall translate to UML, is the Business Process model given in Figure 4. Apart form Collaborations and Roles, it additionally consists of an Event, five Interactions, a Process and two Objects. Based on the assigned-to-relationships the Roles contained in a Collaboration return as (instantiated) objects in a sequence diagram. These sequence diagrams are the UML translations of the Interactions assigned to that Collaboration.

Figure 8 summarizes the separate translations of the different concepts appearing in this Business Process. Figure 9 composes these translations

into one large interaction overview diagram. Our explanation and discussion mainly refers to the interaction overview diagram.

The Event is translated as a trigger starting "the first" sequence diagram corresponding to Interaction investigate. The process create contract can be translated into an activity diagram, but in order to give it its place amidst the five sequence diagrams (for the interactions) we prefer to translate it into yet another sequence diagram. This allows us to put all six sequence diagrams together in one interaction overview diagram.

The three (business) Objects formal request, policy (contract) and the (initiating) Role Customer are translated into classes. But in order to give them their place in the interaction overview diagram, they appear in the instantiated form of three objects with their own lifelines: some several times.

More or less similar to the missing links in the UML collaborations resulting from the translated ArchiMate Collaborations, the UML sequence diagrams for the ArchiMate Interactions assigned to the (same) ArchiMate Collaborations have no communications at all. The only communications present in the interaction overview diagram correspond to triggers - RequestForInsurance, and from one Interaction or Process to the next - or to access relationships between an Interaction (or a Process) and a (business) Object.

So, again, the resulting UML model is uncommonly declarative instead of operational/constructive. This indeed reflects the global point-of-view usually taken in the domains covered by ArchiMate.

6 Conclusions

We have succeeded in translating the ArchiMate concepts into UML. On the basis of the translation the advantages of two worlds can be combined. The three levels in ArchiMate induce a structuring of the model. UML provides the details about 'how' processes interact,communicate,etc. Consistency between UML diagrams, especially between different diagrams, is problematic, e.g., an object in an squence diagram doesn't have to be an instance of a class defined in a class diagram. Using the structuring provided by the use of ArchiMate can give guidelines that may help in the identifying, solving and maintaining of consistency between UML diagrams. UML diagrams that are positioned in different ArchiMate levels but describe parts of the same process can be (partially) related through ArchiMate. Further research is needed on this topic.

In this paper we have discussed examples from the business level only; examples from other levels are available but are of less interest, since the business level exhibits the largest differences from UML.

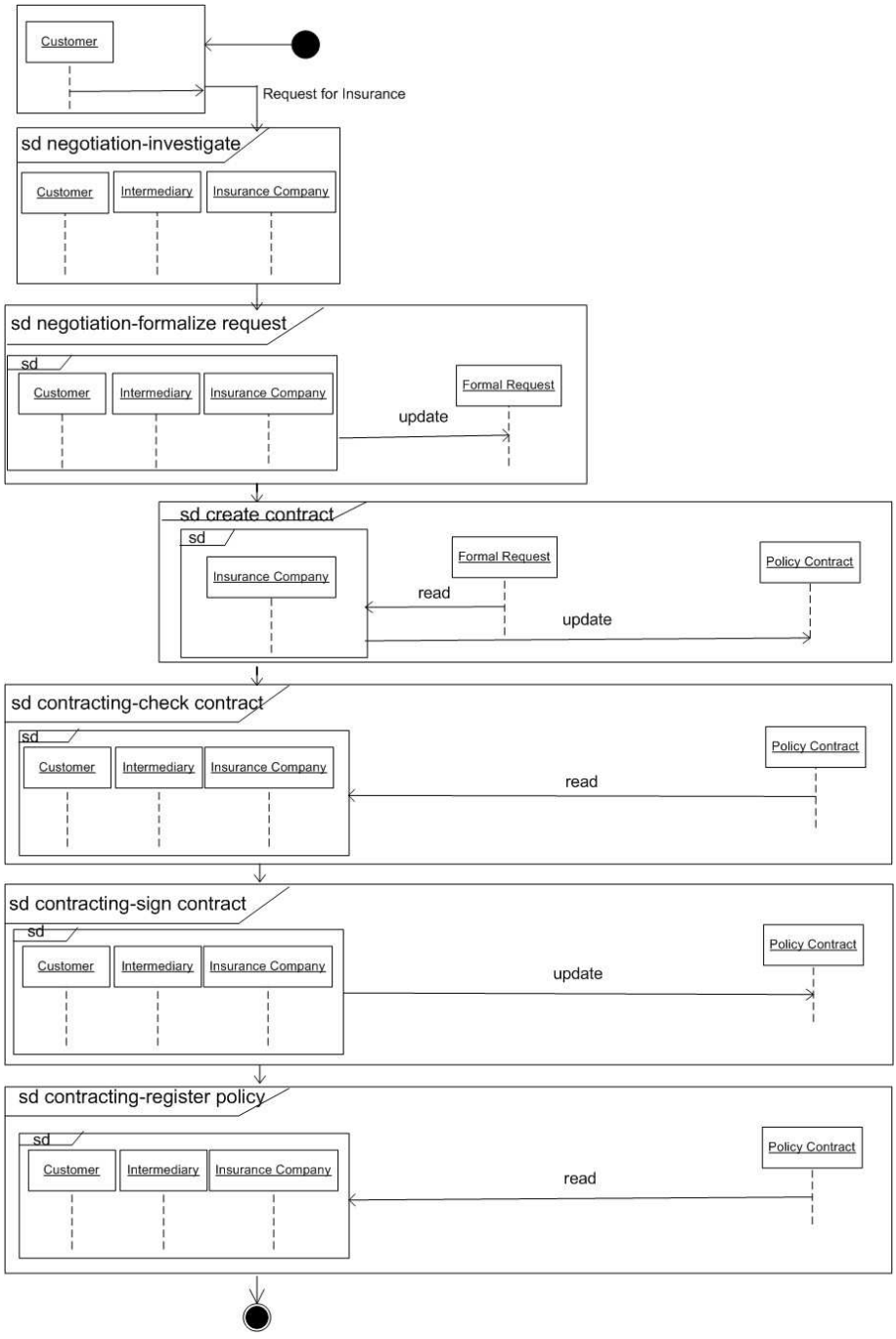

Fig. 9. Interaction Overview diagram of the Business Process model

The strong business orientation of ArchiMate meets the viewpoints of managers, business architect, CIO's (Chief Information Officers) and ICT architects, addressing alignment between business and ICT. The ArchiMate approach hides technical details that often obscure the global overview character of the model one is after. But the UML translation allows for support on the basis of UML tooling - e.g., drawing, checking, analyzing, animating, consistently refining. Moreover, via its UML translation, an ArchiMate model can be studied in the context of other UML models, thus paving the way towards integration and alignment.

As a rather unexpected by-product of our translation we have found, ArchiMate models tend to leave out technical details concerning communication within an interaction and also between interactions. In this way it is left open which participant (role) in an interaction exactly does what for whom and when. We have seen consequences thereof in the uncommonly incomplete UML collaborations, sequence diagrams and interaction overview diagrams.

When using UML directly for modelling business aspects, users are inclined to express such technical details as the language invites a user to do that explicitly. For instance, in the collaboration where the Intermediary is involved, one would probably add a link between Customer and Intermediary and between Intermediary and Insurance Company, thus precluding any direct contact between Customer and Insurance Company during the interactions (investigate, formalize request, check contract, sign contract and register policy).

Although such details certainly make sense, as they stress the mediator role of the Intermediary, these details are not present in the ArchiMate model. Managers, business architects, CIO's and ICT architects simply do not want to bother about them at their level of business (process) modelling. Perhaps it is this property of UML to entice a modeller to add such technical, constructive details, why some [16] argue that UML is not sufficiently suited for the modelling needs of business architect, CIO's and the like.

The ArchiMate approach combined with our translation facilitates the use of UML but prevents the addition of such unwanted details. Although the resulting UML models, strictly speaking, are incomplete, they efficiently reflect the declarative style needed for more business-oriented stakeholders.

By experimenting with more models and their translation, we expect to collect more material for these ideas. Our idea is to define UML profile for the ArchiMate language, which incorporates the declarative style. Within such a profile the incompleteness mentioned can remain hidden, thus preventing in a "natural manner" the addition of technical details.

Acknowledgement

This paper is a result from the ArchiMate project (http://archimate.telin.nl), a research initiative that aims to provide concepts and techniques to support architects in the visualisation, communication and analysis of integrated (enterprise) 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

[1] A. Arkin. *Business Process Modeling Language*. BMPI.org, 2002.

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

[3] H. Eertink, 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.

[4] G. Engels, J. Küster, L. Groenewegen, and R. Heckel. A methodology for specifying and analyzing consistency of object-oriented behavioral models. In V. Gruhn, editor, *Proceedings of the 8th European Software Engineering Conference (ESEC)*, pages 186–195. ACM Press, 2001.

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

[6] D. Garlan, R. Monroe, and D. Wile. ACME: An architecture description interchange language. In *Proceedings of CASCON 97, Toronto, Canada*, pages 169–193, 1997.

[7] M. Goulão and F.B. Abreu. Bridging the gap between ACME and UML 2.0 for CBD. In *Proceedings of Specification and Verification of Component-Based Systems Workshop (SAVCBS'2003) at the ESEC/FSE'2003, Helsinki, Finland*, 2003.

[8] IDEF. Integration definition for function modeling (idef0) draft. In *Federal Information Processing Standards Publication FIPSPUB 183, U.S. Department of Commerce, Springfield, VA, USA, Dec. 1993*, 1993.

[9] ITU-T. Information technology - open distributed processing: Reference model enterprise language. In *Recommendation X.911 ISO/IEC 15414*. International Telecommunication Union, 2001.

[10] H. Jonkers, M. Lankhorst, R. van Buuren, S. Hoppenbrouwers, M. Bonsangue, and L. van der Torre. Concepts for modelling enterprise architectures. *International Journal of Cooperative Information Systems*, 2004. Special issue on Architecture in IT.

[11] H. Jonkers, R. van Buuren, F. Arbab, F. de Boer, M. Bonsangue, H. Bosma, H. ter Doest, L. Groenewegen, J. Guillen-Scholten, S. Hoppenbrouwers, M. Jacob, W. Janssen, M. Lankhorst, D. van Leeuwen, E. Proper, , A. Stam, L. van der Torre, and G. V. van Zanten. Towards a language for coherent enterprise architecture description. In M. Steen and B.R.Bryant, editors, *Proceedings of the 7th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2003)*. IEEE Computer Society Press, 2003.

[12] Object Management Group. UML profile for enterprise distributed object computing (EDOC), final adopted specification ptc/02-02-05. In *6th International Enterprise Distributed Object Computing Conference (EDOC 2002), 17-20 September 2002, Lausanne, Switzerland, Proceedings*. IEEE Computer Society Press, 2002.

[13] Object Management Group. *UML 2.0 Infrastructure, final adopted specification ptc/03-09-15*. OMG document, 2003.

[14] Object Management Group. *UML 2.0 Superstructure, final adopted specification ptc/03-08-02*. OMG document, 2003.

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

[16] M. Steen, M. Lankhorst, and R. van de Wetering. Modelling networked enterprises. In A.Wegmann, editor, *6th International Enterprise Distributed Object Computing Conference (EDOC 2002), 17-20 September 2002, Lausanne, Switzerland, Proceedings*, pages 109–119. IEEE Computer Society, 2002.