Enterprise architecture modelling—the issue of integration

Marc M. Lankhorst*
René van Buuren, Diederik van Leeuwen, Henk Jonkers, Hugo ter Doest, Telematica Instituut, P.O. Box 589, 7500 AN Enschede, The Netherlands
Received 24 January 2005

Abstract

The problem of aligning and integrating business and IT is hampering many companies in their strategic and tactical development. Constructing integrated architecture models contributes to tackling this problem. Unfortunately, no enterprise architecture description language currently exists that fully enables integrated enterprise modelling. A variety of architectural domains are commonly distinguished for which architects use their own modelling techniques and concepts, tool support, visualisation techniques, etc. In this paper we outline such an integrated language and describe concepts that relate architectural domains. This language also serves as a bridge between other existing modelling languages.

Furthermore, we present the design of a workbench for enterprise architecture that serves as a modelling tool and a tool integration environment at the same time: it supports both the integration of models in existing modelling languages and the integration of existing modelling tools.

q 2005 Elsevier Ltd. All rights reserved.

Keywords: Software architecture; Enterprise architecture; Modelling tools; Tool integration

1. Introduction

In current business practice, an integrated approach to business and IT is indispensable. Take for example a company that needs to assess the impact of introducing a new product in its portfolio. This may require defining additional business processes, hiring extra personnel, changing the supporting applications, and augmenting the technological infrastructure to support the additional load of these applications. Perhaps this may even require a change of the organisational structure.

Transferring new information technology to practice requires that a company has a clear, integrated vision on the relation between its business and IT. Without such a vision, the IT infrastructure will never adequately support the business, and vice versa, the business will not optimally profit from IT developments. A vast amount of literature has been written on the topic of strategic alignment, underlining the significance of both ‘soft’ and ‘hard’ components of an organisation [12], for example, distinguish between organisational strategy and organisational infrastructure on the one hand, and IT strategy and IT infrastructure on the other hand. Achieving alignment between business and IT requires an integrated approach to all aspects of the enterprise. Organisational effectiveness is not obtained by local optimisations, but is realised by well-orchestrated interaction of organisational components [27].

Enterprise architecture is an important instrument to address this company-wide integration. It is a coherent whole of principles, methods and models that are used in the design and realisation of the enterprise’s organisational structure, business processes, information systems, and infrastructure [1]. However, in practice, these domains are not approached in an integrated way. Every domain speaks its own language, draws its own models, and uses its own techniques and tools. Communication and decision making across domains is seriously impaired. Although some commercially available tools provide the comprehensive functionality needed to develop and maintain enterprise architecture [11], in general tools provide partial support, do not integrate with other tools and cannot be sufficiently configured for the enterprise’s context [13].

To create such an integrated perspective on enterprise architecture, one needs both a description technique for architectural models and tool support to realise this in
practice. It would not be realistic to suppose that companies will throw their existing design practice and tools overboard and replace these by an entirely new approach. Rather, enterprise modelling should focus on bringing together already existing techniques and integrating these at the appropriate level of abstraction.

In this paper, we present an enterprise modelling language that captures the complexity of architectural domains and their relations, and we describe the design of an integrated enterprise architecture workbench that acts both as a modelling environment for this language and as an infrastructure for integrating with existing modelling languages and tools. These results stem from the ArchiMate project, an applied research project on enterprise architecture (http://archimate.telin.nl).

2. The ArchiMate modelling language

In many modern ICT-intensive organisations, several types of architects and architectures can be found. The technical ICT-related disciplines already have a somewhat longer architectural tradition, although the distinction between architecture and design is not always sharp. Application architects, for example, describe the relations between the many software applications used within the enterprise, as well as the global internal structure of these applications. Presently, the Unified Modelling Language (UML) is usually the language of choice for this purpose, although there are still organisations using their own proprietary notation. The architecture of the technical infrastructure, describing, among others, the layout of the computer hardware and networks hardware in the company, is generally captured in informal drawings of ‘clouds’ and ‘boxes’, if at all.

In the more business-oriented disciplines, ‘working under architecture’ is a more recent development. Since the advent of process orientation in the nineties (e.g., Business Process Redesign [48]), more and more organisations have started to document their business processes in a more or less formal way. However, these descriptions do not focus on the architectural aspects, i.e., they do not provide an overview of the global structure within processes and the relationships between them. Some organisations have a description of their product portfolio, which is generally text-based: visual modelling has not yet gained acceptance in this field.

Thus, we can say that within many of the different domains of expertise that are present in an enterprise, some sort of architectural practice exists, with varying degrees of maturity. However, due to the heterogeneity of the methods and techniques used to document the architectures, it is very difficult to determine how the different domains are interrelated. Still, it is clear that there are strong dependencies between the domains. For example: the goal of the (primary) business processes of an organisation is to realise their products; software applications support business processes, while the technical infrastructure is needed to run the applications; information is used in the business processes and processed by the applications. For optimal communication between domain architects, needed to align designs in the different domains, a clear picture of the domain interdependencies is indispensable (Fig. 1).

With these observations in mind, we conclude that a language for modelling enterprise architectures should focus on inter-domain relations. With such a language, we should be able to model:

- The global structure within each domain, showing the main elements and their dependencies, in a way that is easy to understand for non-experts of the domain.
- The relations between the domains.

Another important property of an enterprise modelling language—as for any modelling language—is a formal foundation, which ensures that models can be interpreted in an unambiguous way and that they are amenable to automated analysis. Also, it should be possible to visualise models in a different way, tailored towards specific stakeholders with specific information requirements.

None of the currently existing modelling languages completely meet these requirements. In this section, we describe the enterprise modelling language [19] that we use throughout this paper. Although, in principle, the concepts of this language are sufficiently generic and expressive to model many aspects within different domains, it is clearly not our intention to introduce a language that can replace all the domain-specific languages that exist. For specific (detailed) designs of, e.g., business processes or applications, the existing languages are likely to be more suitable. In the language that we propose, we conform as much as possible to existing standards.

2.1. Core concepts of the language

In the enterprise modelling language that we propose, the service concept plays a central role. A service is defined as a unit of functionality that some entity (e.g. a system,
organisation or department) makes available to its environment, and which has some value for certain entities in the environment. Service orientation supports current trends such as the service-based network economy and ICT integration with Web services. These examples already show that services of a very different nature and granularity can be discerned: they can be provided by organisations to their customers, by applications to business processes, or by technological facilities (e.g. communication networks) to applications.

A layered view provides a natural way to look at service-oriented models. The higher layers make use of services that are provided by the lower layers. Although, at an abstract level, the concepts that are used within each layer are similar, we define more concrete concepts that are specific for a certain layer. In this context, we distinguish three main layers:

1. The Business layer offers products and services to external customers, which are realised in the organisation by business processes performed by business actors.
2. The Application layer supports the business layer with application services which are realised by (software) applications.
3. The Technology layer offers infrastructural services (e.g. processing, storage and communication services) needed to run applications, realised by computer and communication hardware and system software.

Each of these main layers can be further divided in sub-layers. For example, in the Business layer, the primary business processes realising the products of a company may make use of a layer of secondary (supporting) business processes; in the Application layer, the end-user applications may make use of generic services offered by supporting applications. On top of the Business layer, a separate Environment layer may be added, modelling the external customers that make use of the services of the organisation (although these may also be considered part of the Business layer).

In line with service orientation, the most important relation between layers is formed by use relations, which show how the higher layers make use of the services of lower layers. However, a second type of link is formed by realisation relations: elements in lower layers may realise comparable elements in higher layers; e.g. a ‘data object’ (Application layer) may realise a ‘business object’ (Business layer); or an ‘artifact’ (Technology layer) may realise either a ‘data object’ or an ‘application component’ (Application layer) (Fig. 2).

The general structure of models within the different layers is similar. The same types of concepts and relations are used, although their exact nature and granularity differ. Fig. 3 shows the central structure that is found in each layer.

First, we distinguish the structural or static aspect (right side of the Fig. 3) and the behavioural or dynamic aspect (left side of Fig. 3). Behavioural concepts are assigned to structural concepts, to show who or what displays the behaviour. In the example, role, interface and collaboration are assigned to business process, organisational service and business interaction, respectively.

Second, we make a distinction between an external view and an internal view on systems. When looking at the behavioural aspect, these views reflect the principles of service orientation as introduced in the previous section. The service concept represents a unit of essential functionality that a system exposes to its environment. For the external users, only this external functionality, together with non-functional aspects such as the quality of service, costs etc., are relevant. If required, these can be specified in a contract or service level agreement. Services are accessible through interfaces, which constitute the external view on the structural aspect.
Although for the external users only the external view is relevant, the design of organisations or systems and their internal operations and management also requires knowledge about the internal realisation of the services and interfaces. For this realisation, we make a distinction between behaviour that is performed by an individual structural element (e.g. actor, role component, etc.), or collective behaviour (interaction) that is performed by a collaboration of multiple structural elements.

In addition to active structural elements (the business actors, application components and devices that display actual behaviour, i.e., the ‘subjects’ of activity), we also recognise passive structural elements, i.e., the objects on which behaviour is performed. In the domain of information-intensive organisations, which is the main focus of our language, these are usually information objects in the business layer and data objects in the application layer, but they may also be used to represent physical objects.

2.2. Business layer concepts

In this section we describe concepts for architectural descriptions that can be placed in the business layer of Fig. 2. An example of a business layer model is shown in Fig. 4.

In the example, Client and ArchiSurance are business actors, the active entities (the subjects) that perform behaviour such as business processes or functions. Business actors may be individual persons (e.g. customers or employees), but also groups of people and resources that have a permanent (or at least long-term) status within the organisations. To each actor a business role is assigned: Client has the role of Insurant and in this role makes use of two services offered by the insurance company. ArchiSurance plays the role of Insurer and this role it is responsible for the Damage claiming process; this is expressed by the assignment relation between the business process and the role. Note that the use of roles decouples (physical) actors from business activity and gives more flexibility in the allocation of activities to actors.

In the example a distinction has been made between ‘external’ and ‘internal’ behaviour of ArchiSurance. The externally visible behaviour is modelled by the concept organisational service, which represents a unit of functionality that is meaningful from the point of view of the environment; ArchiSurance has three such organisational services. Within ArchiSurance, these services are realised by one business process: the Damage claiming process, which consists of four subprocesses. Other concepts that can be used for modelling behaviour are business functions and business interactions. Business processes, functions and interactions, in turn, may use other services (internal to the organisation, but external to a smaller entity within the organisation).

Services are grouped to form (financial or information) products, together with a contract that specifies the characteristics, rights and requirements associated with the product. Fig. 5, for example, shows the Travel insurance product.
product. These services are often organisational services, but application services may also be part of a product. This ‘package’ is offered as a whole to (internal or external) customers. ‘Buying’ a product gives the customer the right to use the associated services. The value of a product or service is what makes some party appreciate it. Value is often expressed in terms of money, but non-monetary value is also essential to business, for example, practical or functional value (including the right to use a service), and the value of information or knowledge. In our example, the value associated by the client of the travel insurance would typically be something like ‘to be insured’ or ‘security’.

2.3. Application layer concepts

The main structural concept for the application layer is the application component. This concept is used to model any structural entity in the application layer: not just (reusable) software components that can be part of one or more applications, but also complete software applications, subapplications or information systems, such as the CRM system, the Policy administration, and the Financial application in the example of Fig. 6. This concept is very similar to the UML component concept [32]. Data objects are used in the same way as data objects (or object types) in well-known data modelling approaches, most notably the ‘class’ concept in UML class diagrams.

In the purely structural sense, an application interface is the (logical) location where the services of a component can be accessed. In a broader sense (as used in, among others, the UML definition), an application interface also has some behavioural characteristics: it defines the set of operations and events that are provided by the component, or those that are required from the environment.

Behaviour in the application layer can be described in a way that is very similar to business layer behaviour. We make a distinction between the externally visible behaviour of application components in terms of application services, and the internal behaviour of these components to realise these services. This concept fits well within the current developments in the area of, e.g. web services [49].

An application function describes the internal behaviour of a component needed to realise one or more application services. An application interaction is the collaborative behaviour of two or more application components.

2.4. Technology layer concepts

The main structural concept for the technology layer is the node. This concept is used to model structural entities in the technology layer. Nodes come in two flavours: device and system software, both inspired by UML 2.0 (the latter is called execution environment in UML). A device models a physical computational resource, on which artifacts may be deployed for execution. An example is the zSeries mainframe Fig. 7. System software represents the software environment for specific types of components and data objects, like the DB2 database in the figure. Typically, a node will consist of a number of subnodes, for example a device such as a server and an execution environment to model the operating system.
An *infrastructure service* is the (logical) location where the infrastructural services offered by a node can be accessed by other nodes or by application components from the application layer. An *artifact* is a physical piece of information that is used or produced in a software development process, or by deployment and operation of a system. It is the representation, in the form of e.g. a file, of a data object or an application component, and can be assigned to (i.e., deployed on) a node.

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

In the technology layer, the central behavioural concept is the *infrastructure service*. We do not model the internal behaviour of infrastructure components such as routers or database servers; that would add a level of detail that is not useful at the enterprise level of abstraction.

### 2.5. Relations

In the previous sections we have presented the concepts to model the business, application, and technology layers of an enterprise. In each of the layers presented thus far, different relations between concepts have been used:

- **The access** relation models the access of passive elements, e.g. business or data objects, by processes, functions or interactions.
- **The use** relation models the use of active or behavioural elements, e.g. the use of services by processes, functions or interactions, or the use of interfaces by roles, components or collaborations.
- **The composition** relation indicates that an object consists of a number of other objects, i.e., the lifecycles of the contained objects are tied to that of their container.
- **The aggregation** relation indicates that an object groups a number of other objects, but the grouped objects continue to have an independent lifecycle.
- **The assignment** relation links units of behaviour with active elements (e.g. roles, components) that perform them, roles with actors that fulfil them, or artifacts that are deployed on nodes.
- **Association** models a relation between objects that is not covered by another, more specific relation.
- **The realisation** relation links a logical entity with a more concrete entity that realises it.
- **The specialisation** relation indicates that an object is a specialisation of another object.
- **The triggering** relation describes the temporal or causal relations between processes, function, interactions and events.

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

#### 2.5.1. Relations between layers

As we observed before, the architectural layers (business, application and technology) constitute some sort of hierarchy within an enterprise. A common way of looking at an enterprise is to start from the business processes and activities performed. These are carried out by some actor or role in the organisation, possibly supported by one or more business applications, or even fully automated. These activities, however, can also be viewed as *services* to this business process.

Fig. 8 shows a small example of an integrated and service-oriented enterprise architecture model. This was constructed by connecting models from different layers, such as those shown in the previous sections, by means of services.

#### 2.6. Model integration

Companies cannot be expected to discard their existing models and tools and start from scratch using a completely new approach. An enterprise modelling language should therefore be able to act as a bridge between existing models. Model integration or conceptual integration can be obtained in two ways [3]. One possibility is to define a direct mapping between each pair of modelling languages to facilitate direct relations between models expressed in arbitrary languages. The other possibility is to use a core conceptual language as an intermediary language, which would require only O(n) mappings instead of the O(n²) mappings required with direct mappings.

To integrate existing models expressed in heterogeneous modelling languages, we take the second route, by using the ArchiMate modelling language described in the previous sections as a bridge. In the example of Fig. 9, ArchiMate concepts are used to describe the high-level structure of a business process and its application support. In this example, more detailed UML and BPMN [45] models are constructed for the individual elements of this high-level model; in other cases, some of these detailed models might be designed in e.g. a business process modelling tool such as ARIS.

For integration between languages, a detailed, bidirectional mapping between these languages is needed. Due to the potentially different abstraction levels between a specific language and the ArchiMate language, a bottom-up transformation is likely to loose details and a top-down transformation is likely to be incomplete. In extreme cases a top-down transformation may only produce a template. True round-trip modelling between different languages will
not always be possible; however, defining semantic checks that point out possible inconsistencies between models at different abstraction levels is a feasible option.

3. The ArchiMate workbench for model integration

To provide architects with concrete support in modelling and integrating enterprise architectures, a modelling language is not enough. The next step we have taken is to provide a software architecture and prototype for a workbench for the enterprise architect that acts both as a modelling environment for the ArchiMate language and as an infrastructure for integrating with existing modelling languages and tools [24]. First, we present a number of tool integration aspects. Next, we show the workbench architecture itself. Finally, we show the workbench prototype in operation.

3.1. Tool integration aspects

Our design aims for an extensible workbench to which links with different existing modelling tools can be added...
transparently. This technical integration of tools can be characterised by the following aspects [38]:

- **Data integration** addresses the issue of sharing data between tools and the storage of diagrams, models, views and viewpoints.
- **Control integration** addresses the issue of communication and coordination between tools (and the integration framework, if existent).
- **Presentation integration** concerns the user interaction with the integrated set of tools. Some frameworks completely wrap the existing interfaces whereas others keep original interfaces intact and offer integration through a repository (model integration).

This is similar to the well-known ‘model-view-controller’ pattern. Our architecture for the ArchiMate integration environment addresses all three integration aspects. Data integration is achieved by means of the ArchiMate language as an intermediary, as explained in Section 2.6, control integration is addressed by tool-specific adapters that may invoke native tools for the lower-level languages, and presentation integration is realised by the graphical user interface of the ArchiMate workbench.

To allow easy integration of new modelling tools, the workbench will adopt a **tool adapter pattern** [8] with the motivation that modelling tools should be made to integrate by means of ‘plug and play’.

The workbench prescribes the tool adapter interfaces. The workbench trusts each adapter to be capable of bottom-up and top-down transformations, between the adapter’s associated modelling language and the ArchiMate modelling language. To obtain transparency, the workbench uses the tool-specific adapter associated with a modelling construct to open that modelling construct in its associated modelling tool.

### 3.2. Workbench architecture

#### 3.2.1. The workbench architecture consists of three tiers: a workbench tier, an integration tier and a tool tier

The main component in the workbench tier is the **ArchiMate workbench**. The workbench allows the manipulation of **ArchiMate models**. Each ArchiMate model conforms to an **ArchiMate viewpoint** that defines which modelling constructs are allowed, with which symbols these constructs are presented and which connections these constructs are allowed to have (Fig. 10).

In the tool tier a **modelling tool** may be used to design **tool-specific models** according to a specific **modelling language**.
To allow ArchiMate models to elaborate upon or break down into tool-specific models, the integration tier glues modelling tools into the ArchiMate workbench. The glue used is a tool adapter specific to each modelling tool: a tool-specific adapter. This adapter can perform transformations between tool-specific models and integration content. Along with integration content, a tool-specific adapter provides the workbench with an integration schema describing the underlying modelling language in terms of possibly specialised ArchiMate constructs.

The ArchiMate workbench controls the tool-specific adapter: The workbench dictates when to transform what models or what content and tells when to open a model in its native modelling tool. In practice, the workbench architecture typically integrates a specific set of modelling tools, for example, Rational Rose, Testbed Studio and a repository (Fig. 11).

The ArchiMate workbench also contains a tool adapter to connect to itself. This may seem trivial, but is still very useful: Though transformations may resolve into identity operations, such an adapter will allow ArchiMate models to be built on top of each other, realising a chain of views as mentioned in the previous subsection.

3.3. Workbench at work

To illustrate the value of the workbench we present an example. An existing UML model and an existing Testbed model [5] are integrated in an ArchiMate model (Fig. 12).

The UML model depicts a number of application components that are used by our fictitious insurance company ArchiSurance. The components are translated to ArchiMate components in a straightforward way. The Testbed model represents a number of process blocks that realise claim handling from registration to payment. This model is translated to ArchiMate concepts as well. Now, the workbench can be used to order the objects and define relations between them. In this case a layered architecture is created with services that are realised by components and provided to business processes. This results in a view relating business processes to IT components by means of service concepts. The following operations are applied in the creation of the integrated model.

Translation The interface offered by the Claims administration component is translated to the Claim information service. UML dependency relations are translated to ArchiMate use relations.

Selection Mainly processes and components are selected. Several objects from the models on the left are not relevant in the ArchiMate model. For example, the Central administration component is left out because it is not used by the business process.

Extension Services offered by components to processes are added; concepts are grouped using ArchiMate grouping constructs.
The GUI of the workbench prototype divides the application window in three frames (Fig. 13). A content explorer, a canvas for modelling and a concept explorer.

The canvas (center) shows the currently opened ArchiMate model. Objects may be added to the model in two ways:

- Objects from the content explorer may be dragged and dropped onto the canvas. These objects are in fact references to objects in the underlying tool-specific models.
- Constructs from the concept explorer may be dragged and dropped onto the canvas. This way, newly created instances of those constructs are added to the model.

The content explorer (left) shows hierarchical representations of the tool-specific models on which the currently open ArchiMate model is based. These tool-specific models have been translated into (possibly specialised) ArchiMate concepts, as was explained in Section 2.6. The concept explorer (right) shows only those concepts from the ArchiMate language that are relevant to the current viewpoint.

4. Related work

A wide variety of organisation and process modelling languages are currently in use. The conceptual domains that are covered differ from language to language. In many languages, the relations between domains are not clearly defined. Some of the most popular languages are proprietary to a specific software tool. Relevant languages in this category include the ebXML set of standards for XML-based electronic business [47], developed by OASIS and UN/CEFACT, IDEF [14], originating from the US Ministry of Defence, ARIS [37], part of the widely used ARIS Toolset, and the Testbed language for business process modelling [5]. Recent standardisation efforts in this area are carried out by the Business Process Management Initiative (www.bpmi.org), with the graphical Business Process Modelling Notation BPMN [45] as its main result. Support for this language from vendors of business process modelling and enterprise architecture tools is increasing. However, BPMN’s scope is limited to business processes and it does not provide concepts for modelling e.g. organisational structures, data models, or the relation between business activities and supporting IT applications, making it of limited use in enterprise architecture.

The Reference Model for Open Distributed Processing (RM-ODP) is a joint ISO/ITU-T standard for the specification open distributed systems [17]. It defines five viewpoints on an ODP system that each has their own specification language. Important for enterprise architecture is the enterprise viewpoint, which describes purpose, scope and policies of a system, the RM-ODP Enterprise Language has been defined in which, e.g. business objectives and business processes can be modelled [42].

In contrast to organisation and business process modelling, where there is no single, standard modelling language,
in software modelling the Unified Modelling Language (UML) [2] 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]. Compared to the earlier versions, the support for architectural modelling has improved in the recent UML 2.0 standard [32,33].

The UML has a so-called profile for Enterprise Distributed Object Computing (EDOC), which provides an architecture and modelling support for collaborative or Internet computing, with technologies such as web services, Enterprise Java Beans, and Corba components [30]. This makes UML an important language not only for modelling software systems, but also for business processes and for general business architecture. The UML has either incorporated or superseded most of the older ICT modelling techniques still in use. However, it is not easily accessible and understandable for managers and business specialists; therefore, special visualisations and views of UML models should be provided. Another important weakness of the UML is the large number of diagram types, with poorly defined relations between them. This is another illustration of the lack of integration discussed in the introduction of this paper. Given the importance of the UML, other modelling languages will likely provide an interface or mapping to it.

Most languages mentioned above provide concepts to model, e.g. detailed business processes, but not the relationships between different processes. They are therefore not particularly suited to model architectures [15]. 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 [25], the authors describe the basics of ADLs and compare the most important ADLs with each other. Most have an academic background, and their application in practice is limited. However, they have a sound formal foundation, which makes them suitable for unambiguous specifications and amenable to different types of analysis. The ADL ACME [9] 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 collaboration with OMG to include ACME concepts in UML 2.0 [32,33]. 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.

Finally, another important trend is OMG’s Model Driven Architecture (MDA) approach [7]. Although it strongly leans on OMG standards such as UML, the applicability of the approach is not limited to specific languages. We believe that our language fits well within the MDA philosophy. A prerequisite is compliance with standards such as the Meta Object Facility (MOF) [29] and the XML Metadata Interchange (XMI) [31], which is still subject to further study.

5. Conclusions and future work

In this paper we have outlined both a language and an integration workbench for modelling enterprise architectures. The modelling language brings together the many separate architectural descriptions for specific architectural domains. 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 builds upon successful and widely adopted languages such as UML.

The concepts of our language for enterprise architecture description hold the middle between the detailed concepts that are used for modelling individual domains, e.g. the UML for modelling software, and very general architecture concepts that view systems merely as entities and their inter-relations. The language forms a basis for bridging the heterogeneity of existing languages.

The tool integration workbench we have presented is able to integrate existing modelling tools. Leaving existing modelling environments intact, the workbench allows the concurrent design of enterprise architecture domains: each domain may still be designed using its own languages, tools and techniques. More importantly, with the ability to reason across domain boundaries the workbench introduces an instrument for collaborative design of enterprise architectures.

By adopting the ArchiMate modelling language, the workbench not only allows the integration of existing modelling languages, but provides a language to communicate across domain boundaries as well. Moreover, the workbench serves as a starting point for the analysis of enterprise architectures using generic analysis techniques that rely on the ArchiMate modelling language.

A key factor in the success of the workbench architecture is the feasibility of transformations between tool-specific content and ArchiMate content. The semantic soundness of such transformations is particularly nontrivial and thus requires further exploration.

Currently, the workbench prototype is being finalized and the next step we will take is to validate its design, both in real-world trials with its intended users, and in collaboration with vendors of existing tools to see whether these ideas can be applied in their products.

Acknowledgements

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

References