The Architecture of the ArchiMate Language

M.M. Lankhorst[1], H.A. Proper[2,3] and H. Jonkers[4]

[1]Telematica Instituut, Enschede, The Netherlands
[2]Radboud University Nijmegen, Nijmegen, The Netherlands
[3]Capgemini, Utrecht, The Netherlands
[4]BiZZdesign, Enschede, The Netherlands

Abstract. In current business practice, an integrated approach to business and IT is indispensable. In many enterprises, however, such an integrated view of the entire enterprise is still far from reality. To deal with these challenges, an integrated view of the enterprise is needed, enabling impact and change analysis covering all relevant aspects. This need sparked the development of the ArchiMate language. This paper is concerned with documenting some of the key design decisions and design principles underlying the ArchiMate language.

1 Introduction

In current business practice, an integrated approach to business and IT is indispensable. In many enterprises, however, such an integrated view of the entire enterprise is still far from reality. This is a major problem, since changes in an enterprise's strategy and business goals have significant consequences within all domains of the enterprise, including organisational structures, business processes, software systems and technical infrastructure [1, 2]. To manage the complexity of any large system, be it an enterprise, an information system or a software system, an architectural approach is needed. To be able to represent the architecture of an enterprise, an architecture description language is needed allowing for the represetation of different core aspects of an enterprise, such as business processes, products, applications and infrastructures, as well as the coherence between these aspects.

As discussed in [2], enterprise architecture is a steering instrument enabling *informed governance*. Important applications of enterprise architecture are therefore the analysis of problems in the current state of an enterprise, determining the desired future state(s), and ensuring that the development projects within transformation programs are indeed on-track with regards to the desired future states. This implies that in enterprise architecture models, coherence and overview are more important than specificity and detail. This also implies the need for more coarse grained modelling concepts than the finer grained concepts which can typically be found in modelling languages used at the level of specific development projects, such as e.g. UML [3] and BPMN [4]. Therefore a new language was needed, leading to the development of the ArchiMate language [1].

The ArchiMate language was developed as part of a collaborative research project, funded partly by the Dutch government and involving several Dutch research institutes, as well as governmental and financial institutions. The results of the project in general are described in detail in [1] as well as several papers [5, 6, 7, 8]. An illustrative example of an ArchiMate model is provided in Figure 1. Meanwhile, the ArchiMate language has been transferred to the Open Group, where it is slated to become the standard for architectural description accompanying the Open Group's architecture framework TOGAF [9].

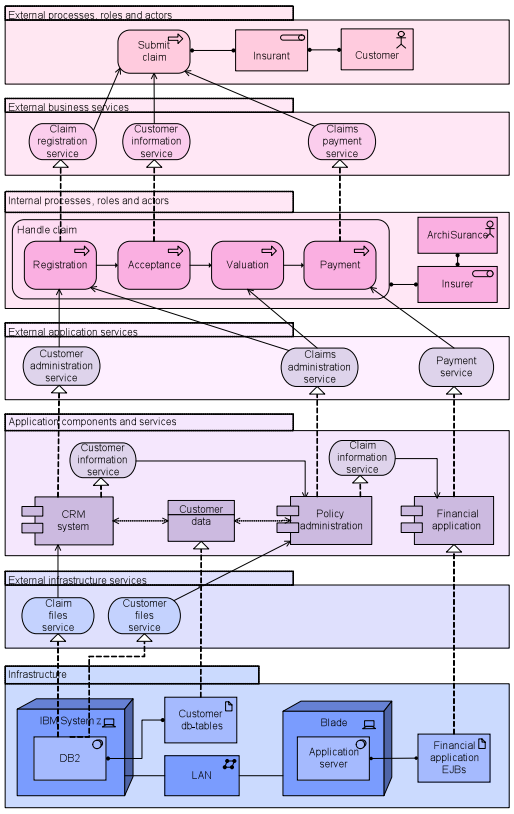

Fig. 1. An example ArchiMate model

The ArchiMate standard consists of six primary components:
A framework – A conceptual framework consisting which allows classification of architectural phenomena.
An abstract syntax – This component contains the formal definition of the language in terms of a meta-model, providing the characteristics of each language construct, and its relationships to other language constructs.
Modelling concepts – A set of modelling concepts allowing for the description of relevant aspects of enterprises at the enterprise level. This set underlies the abstract syntax, focussing on the *concepts* and their meaning, seperate from the language constructs in which they are used.

The language semantics – This component defines the meaning of each language construct and relation type.
A concrete syntax in terms of a visual notation – This syntax defines how the language constructs defined in the meta-model are represented graphically.
A viewpoint mechanism – These mechanisms correspond to the idea of diagram types in UML, though it is much more flexible as there is not a strict partitioning of constructs into views.

The focus of this paper is on documenting some the key design decisions and design principles underlying the language. This also provides a novel perspective on the design, and in particular the evolution, of a modelling language. The ability to evolve the language is of prime importance for languages which are designed as open standards. Languages used as a standard run the risk of becoming a hotchpotch of sorts. Using a clear architecture enables language evolution while still maintaining conceptual integrity of the language.

In the remainder of this paper, we start by discussing in more detail the challenges facing the design of an architecture description language, while consequently discussing the way in which the design of the ArchiMate aims to tackle these. We then continue with a discussion of the modelling concepts needed to domain models in general, which we then first refine to the modelling of dynamic systems, and finally to the modelling of enterprise architectures.

2 Challenges on an architecture modelling language

The design of the ArchiMate language was based on an extensive requirements study. In this study, both practical requirements from the client organisations[1] involved in the ArchiMate project, as well as general requirements on the soundness and other qualities [10] were taken into account [11].

From a modelling perspective, the essential requirements were the following:
Concept coverage – Several domains for grouping concepts have been identified, such as product, process, organisation, information, application and technology. The concepts in the language must at least cover the concepts in these domains.
Enterprise level concepts – At an enterprise level, it is important to be able to represent the core elements from the different domains such as product, process, et cetera, as well as the coherence between these aspects.
Concept mapping – Organisations and/or individual architects must be able to keep using their own concepts and descriptions in development projects. This requires a mapping from the coarse grained concepts in ArchiMate to the fine-grained concepts used in languages at project level.
Unambiguous definitions of concepts – The meaning and definition of the modelling concepts offered by the language must be unambiguous. Every concept must be described taking into account: informal description, specialisation, notation, properties, structuring, rules and restrictions and guidelines for use.

[1] ABN AMRO, ABP Pension Fund, and the Dutch Tax and Customs Administration

Structuring mechanisms – Composition/decomposition, generalisation/specialisation, and aggregation of concepts must be supported.
Abstraction – It must be possible to model relations at different abstraction levels. For example, relations can be formulated between concepts, groups of concepts or different architectural domains.

The ability to perform various kinds of analyses was also recognised as an important benefit of using architecture models. These benefits also contribute towards the *return on modelling effort* (RoME) with regards to the creation of architectural models. The demands following demands were therefore also taken into account in designing the modelling language:
Analysis of architectural properties – It must be possible to perform qualitative and quantitative analysis of properties of architectures.
Impact of change analysis – Impact of change analysis must be supported. In general, such an analysis describes or identifies effects that a certain change has on the architecture or on characteristics of the architecture.

3 Meeting the challenges

In this section we start with a discussion of the key design principles used in the construction of the ArchiMate language, together with their motivations as well as their actual impact on the design of the language.
Concepts should have a clear contribution – The more concepts are offered by a modelling language, the more ways in which a specific situation can be modelled. When it is clear for each of the concepts what its contribution is, the language becomes easier to use and easier to learn [12].
Underlying set of concepts should be defined incrementally – The language should be based on an incrementally defined set of modelling concepts, level by level refining and specialising the set of underlying concepts. When defining the language in this way, it becomes easier to position and discuss possible extensions of the language in relation to higher level core concepts and/or the specialisations of these at the lower levels.
The language should be as compact as possible – The most important design restriction on the language was that it was explicitly designed to be as compact as possible, while still being usable for most enterprise architecture related modelling tasks. Many other languages, such as UML, try to accommodate as much as possible all needs of all possible users. In the interest of simplicity of learning and use, ArchiMate has been limited to the concepts that suffice for modelling the proverbial 80% of practical cases.
Core concepts shouldn't dependent on specific frameworks – Many architecture frameworks are in existence. Therefore, it is not desirable for a general purpose architecture description language to be too dependent on a specific architecture framework. Doing so will also make the language more extendible in the sense that it can easily be adopted to other frameworks.
Easy mapping from/to concepts used at project level – To enable tracability from the enterprise level to the project level, a strong relationship should

exist between the modelling concepts used at project level and those used in the enterprise architecture. Therefore, the ArchiMate language needed to be set up in such a way that project level modelling concepts be expressed easily in terms of the more general concepts defined in the language (e.g., by specialisation or composition of general concepts).

Transitivity of relations – Relations between concepts should be transitive. This will not be further explained in this paper, for more details we refer to [7].

The key challenge in the development of the language meta-model was actually to strike a balance between the specific concepts used by project-level modelling languages on one extreme, and the very general modelling concepts suggested by general systems theory. The triangle in Figure 2 illustrates how concepts can be described at different levels of specialisation. The design of the ArchiMate language started from a set of relatively generic concepts (higher up in the triangle) focussing on domain modelling in general. These were then specialised towards the modelling of dynamic systems (at a course grained level), and consequently to enterprise architecture concepts. At the base of the triangle, we find the meta-models of the modelling concepts used by project-level modelling languages such as UML, BPMN, et cetera. The ArchiMate meta-model defines the concepts somewhere between these two extremes.

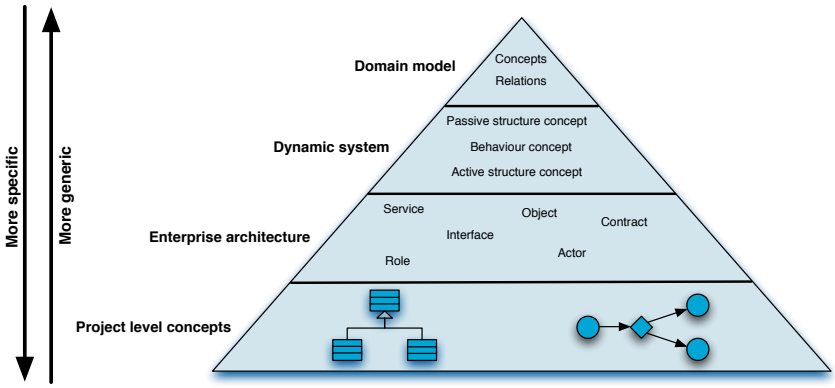

Fig. 2. Concept hierarchy

In the remainder of the paper, we discuss the stack of meta-models taking us from the top of the triangle to the level of the ArchiMate meta-model. At each level, we will present a meta-model of the additional modelling concepts provided by this level. Each level also inherits the concepts from the previous level, while also providing specialisations of the existing concepts. As an example meta-model stack, involving two levels, consider Figure 3. In this paper we have chosen to use Object-Role Modelling [13] (ORM) as a meta-modelling language, since it allows for precise modelling and elaborate verbalisations, making it well suited for the representation of meta-models. The mappings between modelling concepts at different levels are represented as: $a :: b$. What is also illustrated in Figure 3 is the fact that if a, b are both object types b is subtype of a, while if

both are fact-types b is a subset of a. More specifically, in Figure 3 A and B are a sub-type of X, while fact-type h is a sub-set of fact-type f.

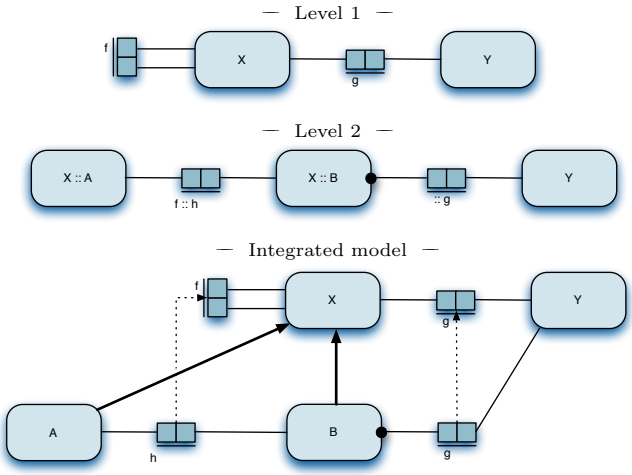

Fig. 3. Example meta-model stack

Sometimes we will want to repeat fact-types which already exist between two super-types for sub-types of these super-types. In this case we will write $::a$ as a shorthand for $a :: a$. In the example shown in Figure 3 we see how g is repeated at level 2, while the the mandatory role (the filled circle on object-type B) requires the instances of sub-type B to all play a role in fact-type g.

4 Domain modelling

In this section we are concerned with the establishment of a meta-model covering a set of modelling concepts that would allow us to model domains in general. We do so by defining three levels as depicted in Figure 4.

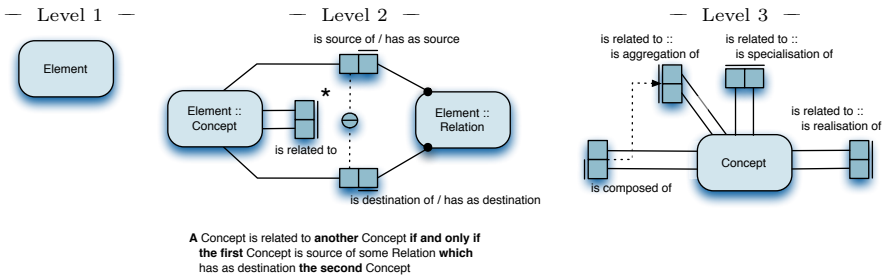

Fig. 4. Basic layers

The first level in Figure 4 shows a meta-model comprising a single modelling concept: Element. This allows us to discern several elements within a modelled

domain (end its environment). On its own, this is of course still highly impractical. We need the ability to at least identify relations between these elements. This, therefore, leads to the refinement suggested by level two. At this level, we identify two kinds of elements: Concepts and Relations. Concepts are the source of Relations as well as the destination of Relations. In other words, Concepts can be related by way of a Relation. This is abbreviated by the derived (as marked by the asterisk) fact-type is related to. The definition of this derived fact-type is provided in the style of SBVR [14].

The domains we are interested in tend to be large and complex. To harness this complexity we need special relationships between Concepts which provide us with abstraction, aggregation and specialisation mechanisms. This leads to three specialisations of the is related to fact-type: is realisation of, is specialisation of, and is aggregation of. A special class of aggregations are compositions, as signified by the is composition of fact-type.

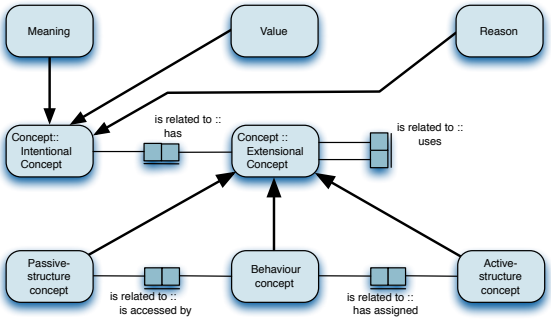

Fig. 5. Level 4

5 Modelling dynamic systems

Based on the foundation established in the previous section, we now describe general concepts for the modelling of dynamic systems. A dynamic system is any (discrete-event) system in which one or more subjects (actors or agents) display certain behaviour, using one or more objects. Examples of dynamic systems are business systems, information systems, application systems, and technical systems. In this section, we gradually extend the set of concepts, using three more or less orthogonal aspects or 'dimensions'. We distinguish: the aspects *active structure*, *behaviour* and *passive structure*, an *internal* and an *external* view, and an *individual* and a *collective* view.

5.1 Active structure, behaviour and passive structure

First, we distinguish *active structure concepts*, *behavioural concepts* and *passive structure concepts*. These three classes have been inspired by structures from natural language. When formulating sentences concerning the behaviour of a

dynamic system, concepts will play different roles in the sentences produced. In addition to the role of a *proposition* dealing with some activity in the dynamic system (selling, reporting, weighing, et cetera), two other important roles are the role of *agens* and the role of *patiens*. The *agens* role (the active structure) refers to the concept which is regarded as executing the activity, while the *patiens* role (the passive structure) refers to the concept regarded as undergoing/experiencing the activity.

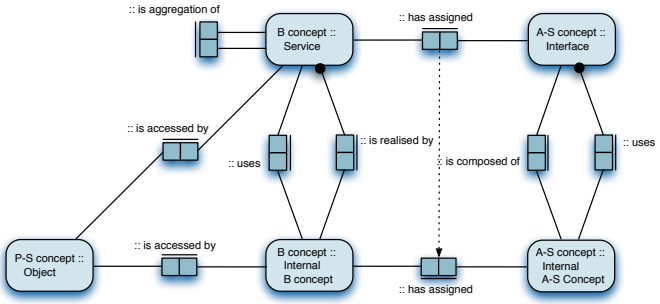

Fig. 6. Level 5

Active structure concepts are concepts concerned with the execution of behaviour; e.g., (human) actors, software applications or devices that display actual behaviour. The *behavioural concepts* represent the actual behaviour, i.e., the processes and activities that are performed. The active structure concepts can be *assigned to* behavioural concepts, to show who (or what) performs the behaviour. The *passive structure concepts* are the concepts upon which behaviour is performed. In the domain that we consider, these are usually information or data objects, but they may also be used to represent physical objects. This extension leads to the refined meta-model as shown in Figure 5.

The active structure, behaviour and passive structure concepts provide an *extensional* perspective on behaviour. In addition, one can discern an *intentional* perspective in relation to stakeholders observing the behaviour. Mirroring the *passive structure*, we identify the *meaning* concept to express the meaning attached to the passive structures. For the *behaviour* aspect, the *value* concept expresses the value exchange/addition that may be associated to the performance of the behaviour. The *active structure* is mirrored by the *reason* concept, expressing the rationale underlying the role of the *active structure* concepts.

5.2 Internal versus external

A further distinction is made between an *external* view and an *internal* view on a system. When looking at the behavioural aspect, these views reflect the principles of service orientation. The *service* concept represents a unit of essential functionality that a system exposes to its environment. This leads to the extension as depicted in Figure 6.

A service is accessible through an *interface*, which constitutes the external view on the active structural concept. An interface is a (physical or logical)

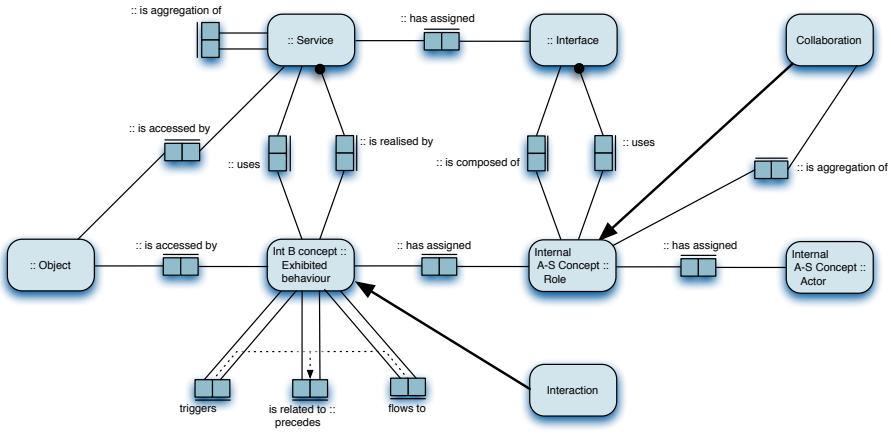

Fig. 7. Level 6

location where the functionality of a service is exposed to the environment. When a service has assigned an interface, then this assignment must be mirrored by the assignment of relevant internal active structure concepts to the internal behaviour concepts involved in the realisation of the service (the dotted arrow between the two has assigned fact-types).

5.3 Individual versus collective behaviour

Going one level deeper in the structure of the language, we distinguish between the *individual* behaviour, performed by a single active structure concept, and the *collective* behaviour performed by multiple active structure concepts in a collaboration. This leads to the refinements shown in Figure 7.

In describing individual and/or collective behaviour in more detail, the internal behaviour concept needs refinement in terms of temporal ordering of the exhibited behaviour. This leads to the precedes fact-type and its sub-sets: triggers (for activities) and flows to (for information processing). A further refinement needed is the distinction between *roles* and *actors* as *active structure* concepts. *Actors* represent the essential identities that can ultimately be regarded as executing the behaviour, e.g. an insurance company, a mainframe, a person, et cetera. The actual execution is taken to occur in the context of a *role* played by an *actor*.

A collective of co-operating *roles* is modelled by the *collaboration* concept: a (possibly temporary) aggregation of two or more active structure concepts, working together to perform some collective behaviour. A collaboration is defined as a specialisation of a *role*. The collective behaviour itself is modelled by the *interaction* concept, where interaction is defined as a specialisation of the *exhibited behaviour* concept.

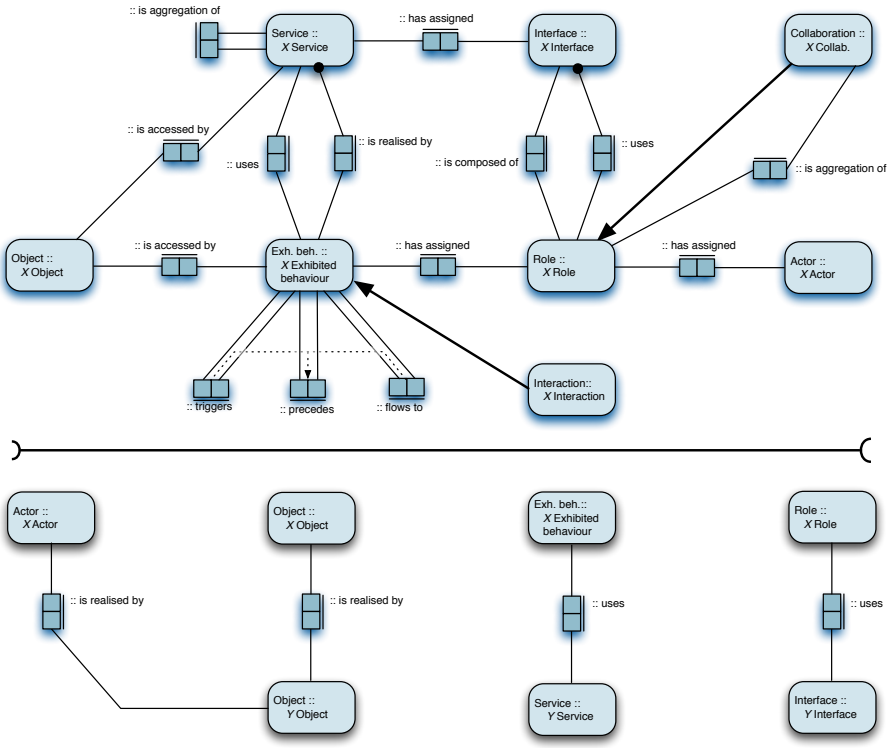

Fig. 8. Level 7 – Fragments

6 Modelling enterprise architectures

In this section we further extend the meta-model stack to arrive at the actual ArchiMate language. Two steps remain. The first step involves the introduction of an architecture framework allowing us to consider enterprises as a layered set of systems. The final step is to refine the meta-models to the specific needs of each of these layers.

As a common denominator of the architecture frameworks in use by participating client organisations, as well as a number of standard frameworks used in the industry, a framework was created involving three layers:

Business layer – Products and services offered to external customers, as well as the realisation of these within the organisation by means of business processes performed by business actors and roles.

Application layer – This layer supports the business layer with application services which are realized by (software) application components.

Technology layer – This layer offers infrastructural services (e.g., processing, storage and communication services) needed to run applications, realised by computer and communication hardware and system software.

Since each of these layers involves a dynamic system, the meta-model at level 7 comprises three copies of the fragment depicted at the top of Figure 8 for Business, Application and Technology respectively. These fragments, however, need

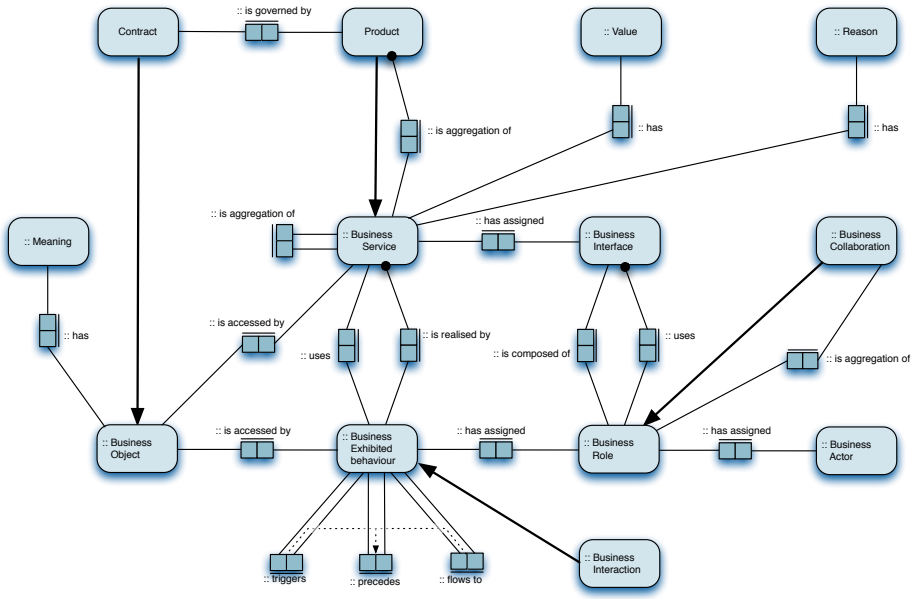

Fig. 9. Level 8 – Business layer

to be connected as well, therefore for each of the two combinations: Business, Application and Application, Technology the fragment shown at the bottom of Figure 8 should be added.

Given the focus of each of the layers, further refinements were needed to better cater for the specific needs of the respective layers. For the business layer, as shown in Figure 9, the concepts of *contract* and *product* have been introduced. At the business level, services may be grouped to form *products*, which are treated as (complex) services. A business service offers a certain value (economic or otherwise) to its (prospective) users, which provides the motivation for the service's existence. For the external users, only this external functionality and value, together with non-functional aspects such as the quality of service, costs, et cetera, are of relevance. These can be specified in a contract. This leads to the situation as depicted in Figure 9. The concepts of *meaning* and *value* have been repeated to stress the fact that they specifically play a role in the business layer.

The application layer, shown in Figure 10, does not lead to the introduction of additional concepts, and only involves the re-naming of some of the existing concepts. The renamings resulted in new names for existing concepts, which corresponded better to the names already used by the partners participating in the ArchiMate project, as well as existing standards such as the UML.

The technology layer also involves some renaming of existing concepts. In addition, some further refinements of existing concepts were needed as well, as depicted in Figure 11. The newly introduced concepts deal with the different kinds of elements that may be part of a technology infrastructure: *nodes*, *software systems*, *devices* and *networks*.

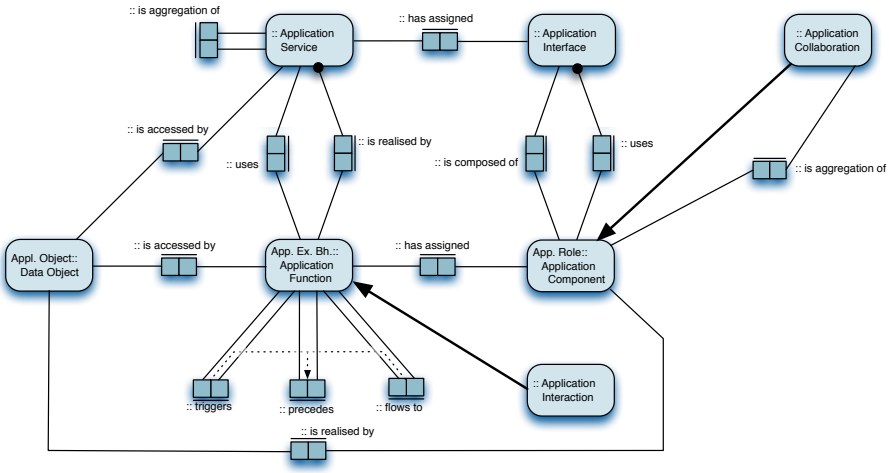

Fig. 10. Level 8 – Application layer

7 Conclusion

In this paper we discussed some of the key design decisions and design principles underlying the ArchiMate language. We have reviewed the challenges confronting an architecture description language for enterprise architecture, as well as the design principles aiming to meet these challenges. This also offered a new perspective on the design and evolution of modelling languages, which is of prime importance for languages designed as open standards. We then discussed the modelling concepts needed in the ArchiMate language, where we made a distinction between concepts needed to model domains in general, the modelling of dynamic systems, and the modelling of enterprise architectures.

Recently, the ArchiMate language has been transferred to the Open Group. It is expected that the language will evolve further to better accompany future versions of the Open Group's architecture framework (TOGAF). This can easily be accommodated by taking the meta-model at level 6 as a common denominator. At level 7 a choice has to be made for a specific architecture framework; in the case of TOGAF this corresponds to a *business architecture*, an *information systems architecture* and a *technology architecture*.

References

1. Lankhorst, M., et al.: Enterprise Architecture at Work: Modelling, Communication and Analysis. Springer, Berlin, Germany (2005).
2. Op 't Land, M., Proper, H., Waage, M., Cloo, J., Steghuis, C.: Enterprise Architecture – Creating Value by Informed Governance. Springer, Berlin, Germany (2008).
3. OMG: UML 2.0 Superstructure Specification – Final Adopted Specification. Technical Report ptc/03–08–02, OMG (2003).
4. Object Management Group: Business process modeling notation, v1.1. OMG Available Specification OMG Document Number: formal/2008-01-17, Object Management Group (2008).

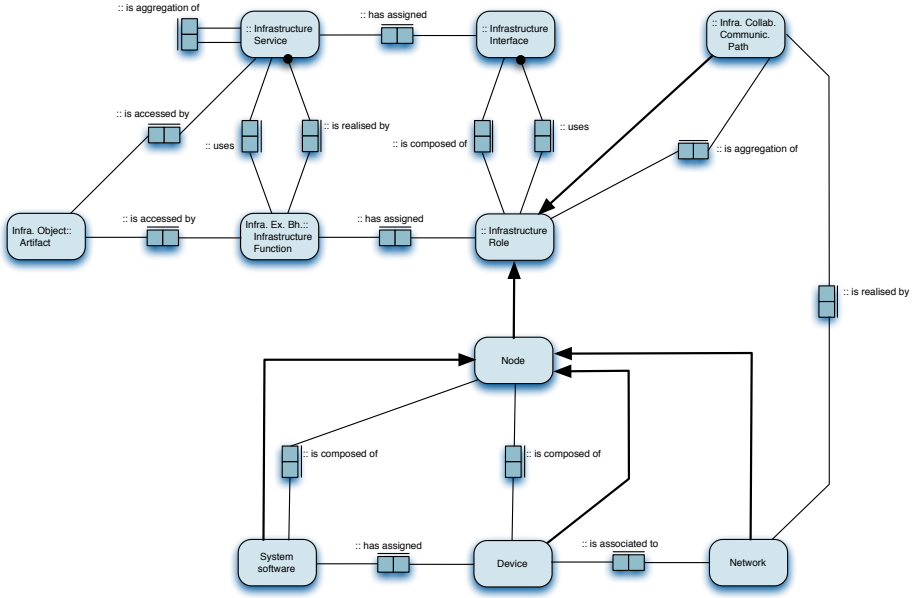

Fig. 11. Level 8 – Technology layer

5. Steen, M., Doest, H.t., Lankhorst, M., Akehurst, D.: Supporting Viewpoint–Oriented Enterprise Architecture. In: Proceedings of the 8th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2004). (2004) 20–24.
6. Jonkers, H., Lankhorst, M., Buuren, R.v., Hoppenbrouwers, S., Bonsangue, M., Torre, L.v.d.: Concepts for Modeling Enterprise Architectures. International Journal of Cooperative Information Systems **13** (2004) 257–288.
7. Buuren, R. van Jonkers, H., Iacob, M., Strating, P.: Composition of relations in enterprise architecture. In Ehrig, H., et al., eds.: Proceedings Second International Conference on Graph Transformation, Rome, Italy (2004) 39–53.
8. Arbab, F., Boer, F.d., Bonsangue, M., Lankhorst, M., Proper, H., Torre, L.v.d.: Integrating Architectural Models. Enterprise Modelling and Information Systems Architectures **2** (2007) 40–57.
9. The Open Group: The Open Group Architecture Framework (TOGAF) Version 8.1.1, Enterprise Edition. (2007).
10. Lindland, O., Sindre, G., Sølvberg, A.: Understanding quality in conceptual modeling. IEEE Software **11** (1994) 42–49.
11. Bosma, H., Doest, H.t., Vos, M.: Requirements. Technical Report ArchiMate Deliverabe D4.1, TI/RS/2002/112, Telematica Instituut (2002).
12. Proper, H., Verrijn–Stuart, A., Hoppenbrouwers, S.: Towards Utility–based Selection of Architecture–Modelling Concepts. In Hartmann, S., Stumptner, M., eds.: Proceedings of the Second Asia–Pacific Conference on Conceptual Modelling (APCCM2005). Volume 42 of Conferences in Research and Practice in Information Technology Series., Sydney, New South Wales, Australia, Australian Computer Society (2005) 25–36.
13. Halpin, T., Morgan, T.: Information Modeling and Relational Databases. 2nd edn. Data Management Systems. Morgan Kaufman (2008).
14. SBVR Team: Semantics of Business Vocabulary and Rules (SBVR). Technical Report dtc/06–03–02, Object Management Group, Needham, Massachusetts (2006).