3 Foundations

This chapter lays down the fundamental ideas and choices on which our approach is based. First, it identifies the needs of architects in the design, communication, realisation, and change of enterprise architectures. It then describes the central role of architecture models in our approach, the use of models in communication, the relationship between models and their presentation, and the formalisation of the meaning (i.e., semantics) of models.

3.1 Getting to Grips with Architectural Complexity

Companies have long recognised the need for an integrated architectural approach, and have developed their own architecture practice. Nevertheless, they still experience a lack of support in the design, communication, realisation, and management of architectures. Several needs can be categorised as follows with respect to different phases in the architecture life cycle:

- **Design**: When designing architectures, architects should use a common, well-defined vocabulary to avoid misunderstandings and promote clear designs. Such a vocabulary must not just focus on a single architecture domain, but should allow for the integration of different types of architectures related to different domains. Next to a common language, architects should be supported in their design activities by providing methodical support, general and organisation-specific guidelines, best practices, drawing standards, and other means that promote the quality of the architectures. Furthermore, to facilitate the design process, which is iterative and requires changes and updates to architectures, support for tracking architectural decisions and changes is desirable.
- **Communication**: Architectures are shared with various stakeholders within and outside the organisation, e.g., management, system designers, or outsourcing partners. To facilitate the communication about architectures, it should be possible to visualise precisely the relevant aspects for a particular group of stakeholders. Especially important in this respect is to bring about a successful communication on relations among

different domains described by different architectures (e.g., processes vs. applications), since this will often involve multiple groups of stakeholders. Clear communication is also very important in the case of outsourcing of parts of the implementation of an architecture to external organisations. The original architect is often not available to explain the meaning of a design, so the architecture should speak for itself.

- **Realisation**: To facilitate the realisation of architectures and to provide feedback from this realisation to the original architectures, links should be established with design activities on a more detailed level, e.g., business process design, information modelling, or software development. Companies use different concepts and tools for these activities, and relations with these should be defined. Furthermore, integration with existing design tools in these domains should be provided.
- **Change**: An architecture often covers a large part of an organisation and may be related to several other architectures. Therefore, changes to an architecture may have a profound impact. Assessing the consequences of such changes beforehand and carefully planning the evolution of architectures are therefore very important. Until now, support for this has been virtually non-existent.

3.1.1 Compositionality

In current practice, enterprise architectures often comprise many heterogeneous models and other descriptions, with ill-defined or completely lacking relations, inconsistencies, and a general lack of coherence and vision. The main driver behind most of the needs identified above is the *complexity* of architectures, their relations, and their use. Many different architectures or architectural views co-exist within an organisation. These architectures need to be understood by different stakeholders, each at their own level. The connections and dependencies that exist among these different views make life even more difficult. Management and control of these connected architectures is extremely complex. Primarily, we want to create *insight* for all those that have to deal with architectures.

The standard approach to dealing with the complexity of systems is to use a compositional approach, which distinguishes between parts of a system, and the relations between these parts. To understand how a car functions we first describe the parts of the car such as the engine, the wheels, the air conditioning system, and then we describe the relationship among these parts. Likewise we understand the information system of a company as a set of systems and their relations, and we understand a company as a set of business processes and their relations.

Compositionality also plays a central role in the architectural approach. For example, the IEEE 1471 standard defines architecture as the fundamental organisation of a system embodied in its components, their relationships to each other, and to the environment (together with principles guiding its design and evolution). Moreover, compositionality also plays a role when varying viewpoints on a system are defined. The latter type of decompositions are usually functional, in the sense that the functionality of an architecture is decomposed in the functionality of its parts and their relations.

3.1.2 Integration of Architectural Domains

The main goal of our approach is the integration of architectural domains, to deal with the complexity of architecture as a discipline, and to provide insight for all those that have to deal with architectures. There are many instances of this integration problem, of which we discuss two examples below. These examples also play their role in the remaining chapters of this book. In general, some integration problems can be easily solved: for example, by using an existing standard; others are intrinsic to the architectural approach and cannot be 'solved' in the usual sense. These hard cases are intrinsic to the complexity of architecture, and removing the problem would also remove the notion of architecture itself. We cannot get rid of the integration problems; we can only develop concepts and tools to make it easier to deal with these issues. This is illustrated by Example 1 below.

Example 1. As a first example of an integration problem, consider Fig. 3.1, which contains several architectures. The five architectures may be models expressed in UML, or models from cells of Zachman's architectural framework, or any kind of combination. For instance, there may be a company that has modelled its applications in UML and its business processes in BPMN. In all these cases, it is unclear how concepts in one view are related to concepts in another view. Moreover, it is unclear whether views are compatible with each other.

The integration of the architectures in Fig. 3.1 is problematic because these five architectures are developed by distinct stakeholders with their own concerns. Relating architectures means relating the ideas of these stakeholders, most of which remain implicit. A consequence is that we often cannot assume to have complete one-to-one mappings, and the best we can ask for is that views are in some sense consistent with each other. This is often called a problem of *alignment*, and the UML–BPMN example is called a business–IT alignment problem.

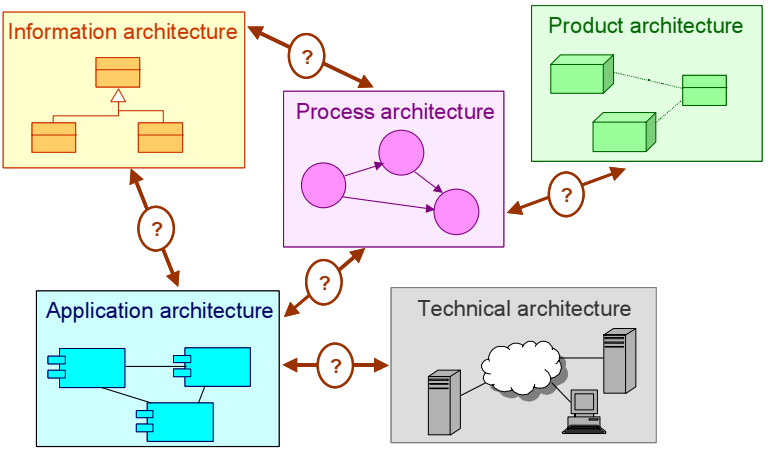

Fig. 3.1. Heterogeneous architectural domains.

In the complex integration cases that involve multiple stakeholders, it is clear that integration is a bottom-up process, in the sense that first concepts and languages of individual architectural domains are defined, and only then is the integration of the domains addressed. We can summarise Example 1 by observing that the integration of architectures is hard due to the fact that architectures are given and used in practice, and cannot be changed. It is up to those who integrate these architectures to deal with the distinct nature of architectural domains.

When we talk about the integration of architectural domains, we need a language in which we can describe these domains. For example, some sources refer to entities and relations, as in entity–relationship diagrams. Others refer to classes and objects, like in object-oriented modelling and software engineering. And yet others refer to concepts and instances; for example, in the area of conceptual modelling. These abstract concepts have been defined at a high level of abstraction, but often they also contain some implicit assumptions. For example, entities and relations are assumed to be finite, because databases are finite, which is not the case with concepts. There are many architecture languages, some of which we have discussed in Chap. 2, but here also terminology varies.

An architecture language is not only needed for the description of integrated architectures, but also a prerequisite for linking the different tools used in the various architectural domains. Furthermore, an integrated language facilitates the analysis of architectures across domains and the reuse of analysis results from specific domains on an integrated level.

It would be foolish to suggest an entirely new architecture language that is built from scratch and ignores already existing developments. In this

book we therefore take a pragmatic approach, and reuse elements from other languages, approaches, and techniques whenever possible.

When looking at everyday architectural practice, it is clear that some integration problems occur more frequently than others. A typical pattern is that some architectural models describe the structure of an architecture at some point in time, whereas other models describe how the architecture changes over time. The second example that we discuss in this chapter addresses this issue.

Example 2. As a second example of an integration problem, consider the first two viewpoints of the IEEE 1471 standard (IEEE Computer Society 2000): the structural viewpoint and the behavioural viewpoint. How are structure and behaviour related?

The second example touches on a problem that has been studied for a long time: the integration of structural and behavioural models. One instance of this problem is how structural concepts like software components are related to behavioural concepts like application functions. Another area where this issue has been studied is in formal methods and in simulation.

The enterprise modelling language described in Chap. 5 shows a strong symmetry between the behavioural and the structural aspects. A service is an 'external' reflection of the 'internal' behaviour that realises it, analogous to the way in which an interface is an 'external' reflection of the 'internal' structure behind it. For the internal behaviour, we distinguish between individual behaviour assigned to an individual structural element and collective behaviour assigned to a collaboration of structural elements.

In the next sections, we will go deeper into the foundations of our approach to modelling enterprise architectures, and in particular into the integration of architectures. However, just like architectural diagrams are often misinterpreted due to the fact that each stakeholder interprets the picture in its own way, architectural concepts also are often misinterpreted. This has led to the IEEE 1471 standard which had the ambition to resolve these ambiguities. Despite the fact that there seems to be increasing consensus on the terminology used, in practice one still finds many distinct definitions of relevant architectural concepts, such as model, meta-model, and view.

In this chapter we define the notions we need in the remainder of the book. These definitions are based on several standards, most importantly the IEEE 1471 standard, the conventions in UML, and other conventions used in daily practice. In general, we develop a language to talk about the integration of architectural domains, and we have to be precise as all concepts have been used in other areas too, and typically are already over-

loaded. In the architectural definitions we incorporate fundamental notions of architecture; for example, that an architecture never refers to reality, but only to some abstraction of it.

3.2 Describing Enterprise Architectures

To cope with the complexity of enterprise architecture, the representation of the essence of an architecture in the unambiguous form of a model can be of great value. We do not want to define the details of the individual architectural domains themselves. That would be the task of the architecture discipline within that particular field. Instead, we concentrate on what is essential for *enterprise* architecture, and therefore we limit ourselves to the core elements of these domains and focus especially on the relations and interactions between them. Precise definitions and constraints will help us to create insight into the complexity of the enterprise architecture and to evade conflicts and inconsistencies between the different domains. For this, we use *models*.

A model is an abstract and unambiguous conception of something (in the real world) that focuses on specific aspects or elements and abstracts from other elements, based on the purpose for which the model is created. In this context, models are typically represented using a formalised graphical or textual language. Because of their formalised structure, models lend themselves to various kinds of automated processing, visualisation, analysis, tests, and simulations. Furthermore, the rigour of a model-based approach also compels architects to work in a more meticulous way and helps to dispel the unfavourable reputation of architecture as just drawing some 'pretty pictures'.

Different stakeholders, however, have a different view of the world. Not everyone's needs can be easily accommodated by a single model. Let us therefore first consider what happens if some viewer observes 'the universe' around him or her.

3.2.1 Observing the Universe

We assume that any viewer that perceives the world around him or her first produces a conception, i.e., a mental representation, of that part he or she deems relevant. The viewer cannot communicate directly about such a conception, unless it is articulated it somehow. In other words, a conception needs to be represented. Peirce (1969a–d) argues that both the percep-

tion and conception of a viewer are strongly influenced by the viewer's interest in the observed universe.

In our case, the viewer is a stakeholder of (part of) the organisational, technical, or other systems that make up the enterprise, i.e., the universe that the viewer observes. The conception of this universe then is the architecture of the enterprise. The representation of this architecture is an architecture description, which may contain models of the architecture, but also, for example, textual descriptions.

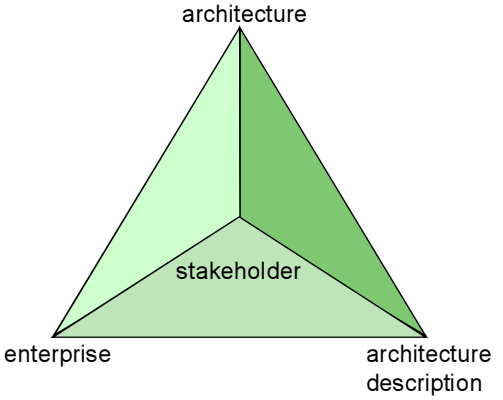

Fig. 3.2. Relationship between enterprise, stakeholder, architecture, and architecture description.

The underlying relationships between stakeholder, enterprise, architecture, and architecture description can be expressed in the form of a tetrahedron, as depicted in Fig. 3.2, which is based on the FRISCO tetrahedron (Falkenberg et al. 1998).

3.2.2 Concerns

So in conceiving a part of the enterprise, stakeholders will be influenced by their particular interest in the observed enterprise, i.e., their concerns. Note that stakeholders, as well as their concerns, may be regarded at an aggregated as well as at an individual level. For example, a single business manager conceiving an information system is a stakeholder. The collective business management, however, can also be seen as a stakeholder of the information system.

Yet concerns are not the only factors that influence a stakeholder's conception of a domain. Another important factor is the preconceptions a stakeholder may harbour as they are brought forward by his or her social,

cultural, educational, and professional background. More specifically, in the context of system development, architects will approach a domain with the aim of expressing the domain in terms of some set of concepts, such as classes, activities, constraints, etc. The concepts an architect is used to using (or trained to use) when modelling some (part of a) domain, will strongly influence the conception of that architect. As Abraham Maslow said: 'If the only tool you have is a hammer, you tend to see every problem as a nail.'

We therefore presume that when architects model a domain, they do so from a certain perspective. In general, people tend to think of the universe (the 'world around us') as consisting of related elements. In our view, however, to presume that the universe consists of a set of elements is already a subjective choice, made (consciously or not) by the viewer observing the universe. The choice being made is that 'elements' (or 'things') and 'relations' are the most basic concept for modelling the universe. In this book, we will indeed make this assumption, and presume that an architect's conception of the universe, i.e., an architecture, consists of such elements.

3.2.3 Observing Domains

Viewers may decide to zoom in on a particular part of the universe they observe, or, to state it more precisely, they may zoom in on a particular part of their conception of the universe, in our case the enterprise. This allows us to define the notion of a domain as:

> **Domain**: any subset of a conception (being a set of elements) of the universe that is conceived of as being some 'part' or 'aspect' of the universe.

In the context of (information) system development, we have a particular interest in unambiguous abstractions from domains. This is what we refer to as a model:

> **Model**: a purposely abstracted and unambiguous conception of a domain.

Note that both the domain and its model are conceptions harboured by the same viewer. We are now also in a position to define more precisely what we mean by modelling:

> **Modelling**: the act of purposely abstracting a model from (what is conceived to be) a part of the universe.

For practical reasons, we will understand the act of modelling also to include the activities involved in the representation of the model by means of some language and medium. We presume architects not only to be able to represent (parts of) their conceptions of the enterprise, but also to be able to represent (parts of) the perspectives they use in producing this conception. This requires architects to be able to reflect on their own working process. When modelling a domain in terms of, say, UML class diagrams, we presume that they are able to express the fact that they are using classes, aggregations, associations, etc., to describe the domain being modelled.

3.2.4 Views and Viewpoints

Very often, no stakeholder apart from perhaps the architect is interested in the architecture in its full scope and detail. As we observed in Sect. 3.2, different viewers have different conceptions of the universe they perceive. Their concerns dictate which parts of an enterprise architecture they deem relevant.

Stakeholders therefore require specific *views* of an architecture that focus on their concerns and leave out unnecessary information. Since we put models central in our description of architectures, this implies that we have to provide different views of these models to accommodate the stakeholders' needs.

A view is specified by means of a *viewpoint*, which prescribes how views that address particular concerns of the stakeholders are constructed, given the architecture under consideration. What should and should not be visible from a specific viewpoint is thus entirely dependent on the stakeholder's concerns.

The IEEE 1471 standard (IEEE Computer Society 2000) defines views and viewpoints as follows:

> **View**: a representation of a system from the perspective of a related set of concerns.
>
> **Viewpoint**: a specification of the conventions for constructing and using a view; a pattern or template from which to develop individual views by establishing the purposes and audience for a view and the techniques for its creation and analysis.

Simply put, a view is what you see, and a viewpoint tells from where you are looking. For example, you might define a 'financial viewpoint' that tells you how to show, say, the costs for building certain applications. Ap-

plying that viewpoint to a model of the new customer relationship management (CRM) system of your company results in a financial view of that system which shows its costs.

3.2.5 Ways of Working

Creating and using architecture models typically involves several related 'ways of working' (Wijers and Heijes 1990):

- A **way of thinking**: articulates the assumptions about the kinds of problem domains, solutions, and modellers involved.
- A **way of modelling**: identifies the core concepts of the language that may be used to denote, analyse, visualise, and/or animate architecture descriptions.
- A **way of communicating**: describes how the abstract concepts from the way of modelling are communicated to human beings, e.g., in terms of a textual or a graphical notation (syntax, style, medium).
- A **way of working**: structures (parts of) the way in which a system is developed. It defines the possible tasks, including sub-tasks, and ordering of tasks, to be performed as part of the development process. It furthermore provides guidelines and suggestions (heuristics) on how these tasks should be performed.
- A **way of supporting**: the support that is offered by (possibly automated) tools for the handling (creating, presenting, modifying, etc.) of models and views. In general, a way of supporting is supplied in the form of some computerised tool.
- A **way of using**: identifies heuristics that:
  - define situations, classes of stakeholders, and concerns for which a particular model or viewpoint is most suitable;
  - provide guidance in tuning the viewpoint to specific situations, classes of stakeholders, and their concerns.

In this book, we try to give attention to each of these 'ways', since in our view they are all essential to the effective use of architectures.

3.2.6 Enterprise Architecture Models

In an ideal situation, we would have a single model for an enterprise architecture, to ensure coherence and consistency between all its different parts. In reality, such a model will probably never exist, especially when we talk about multiple architectural domains. However, it is something we may 'think into existence' without actually constructing the model. In practice,

an architecture (and especially an enterprise architecture) will arise in a bottom-up fashion. Partial models from different domains will be constructed according to the needs in those domains. Where these touch upon each other, inconsistencies may appear, which need to be resolved eventually since the real-world system being designed must of course be consistent. In this way, we slowly move towards this Platonic underlying model, and the partial models from which it is constructed can be seen as views of the total architecture.

Having such a single underlying model makes it possible to create powerful techniques for visualisation and analysis of enterprise architectures, even if this model is incomplete and not fully consistent. Currently, if a stakeholder requires information on some aspect of an architecture that crosscuts several domains, a specialised view of the architecture will probably be patched together manually by integrating information from many different sources in these domains.[2] If we suppose that there is this single underlying model of an architecture, a view of this architecture can be expressed as a projection or subset of this model. Appropriate software tools can then automatically generate these views.

Consider the example in Sect. 3.1 on the integration of structural and behavioural views. To relate the two, we have to consider models and transitions of models. But in relating static and dynamic aspects, a new distinction appears. Are we talking about changes within a model, or changes of the modelling concepts, i.e., the conception of the universe? That is, is the change exogenous or endogenous? This distinction reveals itself only when we relate the structural and behavioural descriptions, not when we consider them in isolation.

As another example, consider the change from batch processing systems to service-oriented architectures. Someone working with batch processing systems twenty years ago could not explain to us today why they do not use service-oriented architecture, because the concept of service-oriented architecture did not yet exist. Since the concept had not been invented yet, it is not just a structural change within the model, but a change at the meta-level of the concepts underlying the model.

The importance of the set of concepts which are used to describe an architecture is acknowledged in the frequent use of 'ontology' within modelling. In our case, we refer to the set of concepts as the *signature* of the architecture. Moreover, the change of signatures and models leads to our notion of *actions in views*. This is explained in more detail in Sect. 3.3.

[2] One of the ArchiMate project partners has in the past invested more than one man-year in creating one specific view of an existing architecture…

3.3 Pictures, Models, and Semantics

In many engineering disciplines, modelling a system consists of constructing a mathematical model that describes and explains it. In the fields of enterprise and software architecture, however, there is an overwhelming tendency to see pictures and diagrams as a form of model rather than as a form of language, or, to be more precise, as a form of structure that helps in visualising and communicating system descriptions. In other words, in architecture there is a tendency to replace mathematical modelling by ad hoc visualisations.

In this book we follow the standard practice in engineering disciplines. Consequently, when we compare architectures like the ones in Fig. 3.1, we ignore irrelevant issues that have to do with arbitrary visualisation. We therefore distinguish between the *content* and the *visualisation* of a model or view, where the first refers to the concepts involved, and the second refers to the form in which these are presented.

For example, in one visualisation of an architecture a process may be visualised as a circle, and in another one by a square. Moreover, the content may express that one concept is more important than another one, which is visualised by drawing the first concept above the second one. The same relation of importance can also be visualised by the intensity of the colour which is used to visualise the concepts. The architect is motivated to make explicit whether visual information like 'above' or 'red' has a meaning in the model, or is incidental. When something is incidental the architect is motivated to remove it from the picture, as it only distracts from the message of the picture. When it is meaningful, its meaning has to be made explicit. When a new viewpoint is defined, the content and its visualisation can be defined in two separate phases.

The 'content' and 'visualisation' should be interpreted here in a loose way. For example, the visualisation may also include input devices such as menus or buttons, and the content may also include actions that change the model by for example adding or deleting concepts. Actions in models are used here to deal with interaction with the user.

Our motivation to stress the importance of modelling is that there is something about architecture independent of visualisation. Two distinct views, which are based on viewpoints from stakeholders with distinct concerns, still have something in common. This is called the *semantics* of the architecture. Semantics does not have to be explicitly given, it can also be an unspoken common understanding among the users of the architecture. It does not have to be one unified semantics, as there can also be several semantics for different purposes and uses of the architecture. But in the latter

case, these semantics again have something in common. Perhaps they just have to be consistent.

The importance of semantics has been emphasised in several other areas too, with a related motivation. In some parts of computer science, the term 'semantics' of something in a model is often used to refer to the 'effect' of that something in the model, referring to the dynamics within that model. In linguistics there is a much older distinction between syntax, semantics, and pragmatics. Another example is in the meaning of information on the Web: Web pages have traditionally been used to describe all kinds of issues, but they often refer to the same objects using distinct terminology. This led Tim Berners-Lee to the invention of the semantic Web, where ontologies play a crucial role.

3.3.1 Symbolic and Semantic Models

To make the notion of semantics explicit, we distinguish between a symbolic model and a semantic model. A *symbolic model* expresses properties of architectures of systems. It therefore contains symbols that refer to reality, which explains the name of this type of model.

> A **symbolic model** expresses properties of architectures of systems by means of symbols that refer to reality.

The role of symbols is crucial, as we do not talk about systems without using symbols. The reason is that systems are parts of reality, and we can only talk about reality by using some symbolic form of communication.

When stakeholders refer to architectures and systems, they can do so only by interpreting the symbols in the symbolic models. We call such an interpretation of a symbolic model a *semantic model*.

> A **semantic model** is an interpretation of a symbolic model, expressing the meaning of the symbols in that model.

A semantic model does not have a symbolic relation to architecture, as it does not contain symbolic references to reality.

However, there is a relation between semantic model and reality, because a semantic model is an abstraction of the architecture. To understand this relation between semantic model and architectures, one should realise that an important goal of modelling is to predict reality. When a symbolic model makes a prediction, we have to interpret this prediction and test it in reality. The relevant issue in the relation between a system and semantic

models of it is how we can translate results such that we can make test cases for the symbolic model.

There are various ways in which we can visualise the relation between the four central concepts of enterprise, architecture, symbolic model, and semantic model. We put the concept of architecture central, as is illustrated in Fig. 3.3.

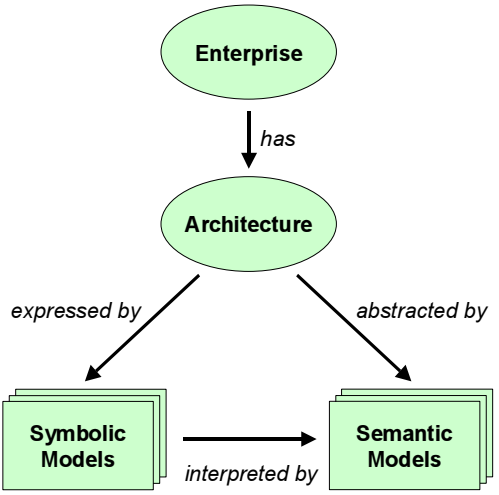

Fig. 3.3. The enterprise, its architecture, symbolic and semantic models.

There are three important observations we have to make here. First, the above four concepts and their relations are used in engineering both for informal as well as formal models. The relevant distinction we emphasise between symbolic and semantic models is the distinction between using symbols to refer to reality, and abstractions of reality that only refer to reality by interpreting the symbols of the symbolic model. Note that this is not the same distinction as that between informal and formal models: within the class of informal models, expressed for example in natural language, both kinds exist, as well as within the class of formal models, expressed for example in first-order logic.

Second, an architecture may be expressed by multiple symbolic models, and one symbolic model may in turn be interpreted by several semantic models. For example, we might define separate semantic models for performance and for cost of a system that is expressed by one symbolic model, e.g., in UML.

Third, in architecture often a distinction is made between the *architectural semantics* and the *formal semantics* of a modelling language. As explained in Sect. 3.2.1, the enterprise under consideration is thought of in

terms of architecture concepts, which exist in the minds of, for instance, the enterprise architect. These concepts can be represented in models, which are expressed in a modelling language. Architectural semantics is defined as the relationship between architectural concepts and their possible representations in a modelling language (Turner 1987). To understand this distinction, consider Venn diagrams. They are useful structures for the visualisation of the language of Boolean logic, but they are not a model themselves. Their semantic model is given by the set-theoretical explanation of their meaning. The formal semantics of a model or language, on the other hand, is a mathematical representation of specific formal properties of that model or language. The formal semantics of a computer program, for example, expresses the possible computations of that program. Different branches of formal semantics exist, such as denotational, operational, axiomatic, and action semantics. Harel and Rumpe (2004) give a clear explanation of the need for rigorously defining the semantics of modelling languages.

There are two kinds of abstraction we use in creating a model of reality. The first is abstracting from (properties of) the precise entity in reality to which a concept refers. This occurs for example when we make a model of the static structure of an application in terms of its components, leaving out (i.e., abstracting from) their behaviour. The second kind is abstraction from differences between entities in reality by grouping them into a single concept. This is sometimes referred to as generalisation, and occurs for example when we use the concept 'employee', which groups the individuals in a company. This is related to the notion of 'sorts' discussed below.

3.3.2 Symbolic Models

A symbolic model is the formalisation of one or more aspects of the architecture of a concrete system. It comprises those parts of an architecture that can be modelled mathematically, as opposed to the more pragmatic aspects of an architecture that are concerned with characteristic notions like rationale, goals, and plans.

A symbolic model is expressed using a description language, a representation of the model that is often confused with its interpretation. For example, the expression 3 + 5 may be intended to mean a particular natural number, but here is just notation for the syntactic model of the natural numbers. Strictly speaking, a description language describes both the *syntactic structure* of the model and its *notation*, i.e., the words or symbols used for the concepts in the language. As we explained in Sect. 3.3.1, we

make a strict separation between structure and the notation, and we will use the term 'model' to refer to the structure.

The core of every symbolic model is its *signature*. It categorises the entities of the symbolic model according to some names that are related, linguistically or by convention, to the things they represent. These names are called *sorts*. Relations between entities of some sorts and operations on them are also declared as relation symbols in the signature. After the relations have been specified, they can be used in languages for constraining further or analysing the nature of the symbolic model. An example is in order here, before we go any further. Fig. 3.4 exhibits a structural description of the employees of a company.

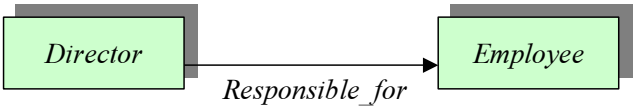

Fig. 3.4. Symbolic model of the director–employee relationship.

We need to recall that the above is a syntactic structure; that is, a description of a symbolic model with a signature whose sorts are *Employee* and *Director*, and with respective entities related by a relation named *Responsible_for*. As yet we have assigned no meaning to it; we have only categorised the entities of the symbolic model into two categories and named a relation between the entities belonging to two sorts. The syntactic names used for the sorts and relations push our intuition some steps ahead: we know what an employee is, what a director is, and what responsible for means. However, while these syntactic names help us in our understanding, they are also the main source of confusion in the communication and analysis of an architecture. We could have named the above sorts X and Y better to retain the meaningless quality of the syntax, and avoid confusion with semantics.

A signature thus provides a conceptual glossary in whose terms everything else in the symbolic model must be described, similar to an English dictionary for the English language. Additionally, a signature comprises information to capture certain aspects of the ontology of an architecture. For example, it may include hierarchical information between sorts in terms of an 'is-a' relationship, or containment information in terms of an 'includes' relationship, or dependency information in terms of a 'requires' relationship. Signatures that contain this additional information are more general than a glossary. They provide a conceptual schema, similar to the schema provided to biologists by the species classification.

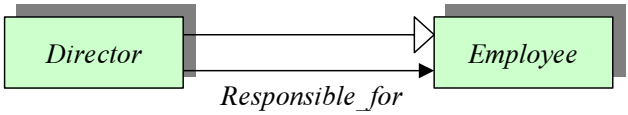

Fig. 3.5. Extended symbolic model.

For example, Fig. 3.5 extends the previous signature with an 'is-a' relationship between the sorts *Director* and *Employee* (denoted by a UML inheritance relation), intuitively suggesting that every director is also an employee.

Moreover, the symbolic model may also contain a set of actions, and the signature a set of action symbols, the meaning of which we discuss below.

3.3.3 Semantic Models

The formalised meaning of a symbolic model is given by a semantic model, an interpretation of the symbolic model. A semantic model usually assumes the existence of some mathematical objects (sets, for example), used to represent the basic elements of a symbolic model. Operations and relations of a symbolic model are mapped to usually better understood operations and relations amongst the mathematical objects.

As an example, the formal semantics of a signature is provided by a collection of sets (one for each sort of the signature), and a set of relations and functions among them, one for each relation symbol and function symbol in the signature. Hierarchical information between sorts is captured by the ordinary subset inclusion, whereas containment information is denoted by the usual element-of relation.

It is clear that,in general, there can be a large number of different interpretations for the same symbolic model. This reflects the intuition that there can be many architectures that fit a specific architecture description. In fact, the signature of a symbolic model of an architecture specifies only some basic building blocks by means of which the architecture is described.

In other words, we see the formal semantics of a symbolic model as a concrete collection of mathematical objects interpreting a system according to a specific architecture description. As such, it involves concrete components and their concrete relationships which may change in time because of the dynamic behaviour of a system. Concrete situations of a system are described by means of variables typed according to the sort of the individuals they are referring to. More concretely, for a symbolic model, we will denote by $x{:}T$ a variable x which ranges over individuals of sort T. For example, we could use the logical sentence

$$\exists\, x : Director.\ \forall y : Employee.\ Responsible\_for(x, y)$$

to constraint the interpretation of the sort *Director* to be a non-empty set. Note that since *Director is_a Employee*, also the interpretation of the latter sort will be non-empty.

The actions occurring in a symbolic model are interpreted as changes of the model based on interaction with the user. To define actions, we have to define the input variables of the action, and how we can retrieve these input variables from the user. In Chap. 7 we discuss the use of actions in models in viewpoints and visualisation, and in Chap. 10 we describe some technical aspects of implementing these actions in models.

Finally, in our approach described more explicitly in Chap. 8, the formal semantics is rich enough to capture the dynamics of a system by interpreting the symbolic (and often pictorial) information available for describing business and software processes in the ArchiMate language discussed in Chap. 5.

In the remainder of the book, whenever we use the unqualified terms 'model' or 'semantics' of an architecture, we refer to its *symbolic* model and *formal* semantics, which is the common interpretation of these terms in the architecture discipline.

3.3.4 UML vs. ArchiMate

The ArchiMate approach can be contrasted with the original approach in UML, which we described in Chap. 2. In this approach, semantics was explicitly left out of the program. People who used the models could develop semantics for them, but a general semantics was not supplied. This approach also stemmed from the origins of UML as a combination of three existing notations that did not have formal semantics. Hence, the focus of UML was and is on notation, i.e., syntax, and not on semantics. Although some of the diagrams of the more recent versions of UML have a formal semantics (see, e.g., the token-based Petrinet-like semantics of activity diagrams in UML 2.0), there is no overall semantics for the entire language.

We have taken the opposite approach. We do not put the notation of the ArchiMate language central, but rather focus on the meaning of the language concepts and their relations. Of course, any modelling language needs a notation and we do supply a standard way of depicting the ArchiMate concepts, but this is subordinate to the architectural semantics of the language.

3.4 Summary

An integrated architectural approach is indispensable to control today's complex organisations and information systems. It is widely recognised that a company needs to 'do architecture'; the legacy spaghetti of the past has shown us that business and ICT development without an architectural vision leads to uncontrollable systems that can only be adapted with great difficulty. However, architectures are seldom defined on a single level. Within an enterprise, many different but related issues need to be addressed. Business processes should contribute to an organisation's products and services, applications should support these processes, systems and networks should be designed to handle the applications, and all of these should be in line with the overall goals of the organisation. Many of these domains have their own architecture practice, and hence different aspects of the enterprise will be described in different architectures. These architectures cannot be viewed in isolation.

For example, architectural domains are related, and structural and behavioural viewpoints are related. The integration has to deal with the fact that the various viewpoints are defined by stakeholders with their own concerns.

The core of our approach to enterprise architecture is therefore that multiple domains should be viewed in a coherent, integrated way. We provide support for architects and other stakeholders in the design and use of such integrated architectures. To this end, we have to provide adequate concepts for specifying architectures on the one hand, and on the other hand support the architect with visualisation and analysis techniques that create insight into their structure and relations. In this approach, relations with existing standards and tools are to be emphasised; we aim to integrate what is already available and useful. The approach that we follow is very generic and systematically covers both the necessary architectural concepts and the supporting techniques for visualisation, analysis, and use of architectures.

We adopt a framework around a stakeholder, enterprise, architecture, and architecture description as a viewer with universe, conception, and representation. The view and viewpoint of the stakeholder are the result of modelling, an act of purposely abstracting a model from reality, i.e., from a domain that is conceived to be a part of the universe. These views consist of a set of enterprise architecture models.

Within this framework, a distinction is made between the content of a view and its visualisation, and a distinction is also made between a symbolic model, which refers to the enterprise architecture, and a semantic model as an abstraction from the architecture and which interprets the

symbolic model. The core of every symbolic model is its signature, which categorises the entities of the symbolic model.