As we have argued in previous chapters, organisational effectiveness cannot be achieved through local optimisations, but is realised by well-orchestrated interaction of organisational components (Nadler et al. 1992). To create such an integrated perspective of enterprise architecture, we need both a description technique for architectural models and model-based analysis techniques to realise this global optimisation in practice.

In Chap. 5, we have presented a description language that not only captures the complexity of architectural domains and their relations, but also enables the integration at the appropriate level of abstraction of already existing partial architecture models. However, the value of architecture models increases significantly if they can also be used to support the decision-making process. In this chapter we argue that whenever a change in the enterprise architecture is needed, model-based analysis plays a central role. Therefore, we present a number of techniques that help architects and stakeholders to compare alternative designs and, hence, take well-informed design decisions when making trade-offs between aspects like cost, quality, and performance and to be able to study the impact of a change to the design.

This chapter also explains what the added value of enterprise architecture analysis techniques is in addition to existing, more detailed, and domain-specific ones for business processes or software, for example. Analogous to the idea of using the ArchiMate enterprise modelling language to integrate detailed design models, the chapter demonstrates that analysis, when considered at a global architectural level, can play a role in the integration of existing detailed techniques or of their results.

### 8.1 Analysis Techniques

One of the central motivations for enterprise architecture in general is getting to grips with change. Architects and stakeholders want to take well-informed design decisions. To that end, they need to compare alternative designs, make trade-offs between aspects like cost, quality, and performance, and know the impact of a
change across all aspects of an architecture. Given the size and complexity of enterprise architectures, this is something that can no longer be done by hand and requires sophisticated analysis techniques. These analysis techniques do more than simply ‘walk through a picture’, but require well-defined semantic underpinnings and advanced analysis algorithms.

We can classify architecture analysis techniques according to different aspects (Fig. 8.1). First, we make a distinction based on the types of analysis inputs and results: functional (e.g., structural and dynamic properties) and quantitative (e.g., performance and costs).

Functional analysis is performed to gain insight into the functional aspects of an architecture. Among others, it is used to understand how a system that conforms to an architecture works, to find the impact of a change on an architecture, or to validate the correctness of an architecture.

Functional analysis techniques do not answer quantitative questions, like ‘how quick’ or ‘how cheap’. These are typically questions addressed by the quantitative analysis techniques. Usually, architectural models do not provide sufficient information to perform detailed quantitative studies. In our view, an approach for quantitative analysis of enterprise architectures should make it possible to structure and relate quantitative results obtained with existing detailed analysis methods (e.g., queuing analysis or simulation).

Second, for both functional and quantitative analysis, we distinguish two main types of techniques: analytical techniques and simulation.

Basically, simulation can be seen as the ‘execution’ of a model. Functional simulation and animation are useful to illustrate the dynamic behaviour of a system. The aim of functional simulation is to gain insight into the properties and behaviour of an architecture. Architects can ‘play’ with the architecture and see how it works, feels, looks, can be adapted to certain changes, etc. Moreover, functional simulation can also play an important role in the communication between stakeholders, by giving them a better common understanding of the architecture. Interpretation problems, often stemming from the high level of abstraction of architectures, may come to light when using functional simulation. Quantitative simulation is used to make statistical statements about the quantitative measures of a system based on multiple simulation runs. It can be seen as performing ‘measurements’ in a model. Thus, quantitative simulation allows for a thorough examination of the performance measures in a specific situation.

Fig. 8.1 Analysis dimensions
In this chapter, we mainly consider formal and analytical analysis techniques. In contrast to simulation, these are not of a statistical nature, but provide a unique, reproducible result. Analytical techniques for quantitative analysis are typically more efficient than quantitative simulation, and therefore more suitable for providing the architect with a first indication of performance measures and bottlenecks in an architecture model. They are also useful when a comparison of a (large) number of alternatives is needed in so-called ‘what if’ analysis.

Another issue to be addressed when using analysis techniques for enterprise architectures is whether to apply existing techniques, or to develop new ones. Buy or build? In the first case, two other questions are to be answered: which technique to choose from the available ones, and how to apply it? In the second case, the questions are for what problem a technique is developed, and how the development itself can be carried out.

This chapter illustrates both of the above-mentioned options. For quantitative analysis, described in Sect. 8.2, we have chosen to propose a new top-down bottom-up approach. Nevertheless, this approach also facilitates the integration with existing, domain-specific analysis techniques. For functional analysis, explained in Sect. 8.3, we have chosen the first approach, i.e., we show how existing techniques from formal methods can be used in analysing the functional properties of architectures.

8.2 Quantitative Analysis

As noted earlier, enterprise architecture is concerned with a description of how all the relevant elements that make up an enterprise interrelate. It covers aspects ranging from business processes and products, through software applications, to the technical infrastructure. The relations between these ‘layers’ play a central role. Also, from a quantitative perspective, these aspects are interrelated in several ways. For example, the business processes impose a workload on the software applications and infrastructure, while the performance characteristics of the lower layers directly influence the performance of the higher layers.

There is a common misconception that quantitative analysis is ‘too detailed’ to be performed at the architecture level. However, performance engineering practitioners argue that next to functional aspects, non-functional (quantitative) aspects of systems should also be taken into account at all stages of the design of a system, and not just as an afterthought (Smith 1990). This implies that these aspects are also relevant for enterprise architectures. In this case, however – as for enterprise architecture modelling – the quantitative aspects are considered at a relatively global level. Also, the emphasis is on structure: enterprise architectures can provide a useful instrument to structure quantitative properties of organisations and systems.

Quantitative analysis can serve several purposes. In the first place it is often used for the optimisation of, for example, processes or systems, by quantifying the effect
of alternative design choices. Similarly, it can be used to obtain measures to support impact-of-change analysis: what is the quantitative effect of changes in a design? This shows that the distinction between functional and quantitative analysis is not always sharp. A third application of quantitative analysis is capacity planning, e.g., how many people should fulfil a certain role to finish the processes on time, or how should the infrastructure be dimensioned (processing, storage, and network capacity) given an expected workload?

Models of organisations and systems can be quantified in several ways. Measures of interest may include:

- **Performance** measures, i.e., time-related measures such as completion and response times, throughputs, resource utilisations.
- **Reliability** measures such as availability and dependability.
- **Cost** measures.

The techniques and examples presented in this section focus on performance measures.

### 8.2.1 Performance Views

As explained earlier, the different ways to structure an enterprise architecture model provide different views of the same model. These views are aimed at different stakeholders and their concerns. Also in the context of the performance of a system, a number of views can be discerned, each having their own performance measures, explained below:

- **User/customer view** (stakeholders: customer; user of an application or system): The response time is the time between issuing a request and receiving the result, e.g., the time between the moment that a customer arrives at a counter and the moment of completion of the service, or the time between sending a letter and receiving an answer. Also in the supporting IT applications the response time plays an important role; a well-known example is the (mean) time between a database query and the presentation of its results. Examples of ArchiMate concepts for which the calculation of the response time is suited are actors, roles, and services.

- **Process view** (stakeholders: process owner; operational manager): Completion time is the time required to complete one instance of a process (possibly involving multiple customers, orders, products, etc., as opposed to the response time, which is defined as the time to complete one request). In batch processing by means of an information system the completion time can be defined as the time required to finish a batch.

- **Product view** (stakeholders: product manager; operational manager): Processing time is the amount of time that actual work is performed on the realisation of a certain product or result, i.e., the response time without waiting
times. The processing time can be orders of magnitude lower than the response time. In a computer system, an example of the processing time is the actual time that the CPU is busy.

- **System view** (stakeholders: system owner/manager): *Throughput* is the number of transactions or requests that a system completes per time unit (e.g., the average number of customers served per hour). Related to this is the maximum attainable throughput (also called the *processing capacity*, or in a more technically oriented context such as communication networks, the *bandwidth*), which depends on the number of available resources and their capacity.

- **Resource view** (stakeholders: resource manager; capacity planner): *Utilisation* is the percentage of the operational time that a resource is busy. On the one hand, the utilisation is a measure of the effectiveness with which a resource is used. On the other hand, a high utilisation can be an indication of the fact that the resource is a potential *bottleneck*, and that increasing that resource’s capacity (or adding an extra resource) can lead to a relatively high performance improvement. In the case of humans, the utilisation can be used as a more or less objective measure for work stress. In information systems architectures, a typical example of the utilisation is the network load. Examples of ArchiMate concepts for which the calculation of the utilisation is suited are the infrastructure concepts and the actor.

The different performance views are summarised in Fig. 8.2. Performance measures belonging to the different views are interrelated, and may be in conflict when trying to optimise the performance of a system. For example, a higher throughput leads to a higher resource utilisation, which may be favourable from a resource manager’s point of view; however, this generally leads to an increase in
the response times, which is unfavourable from a user’s point of view. Therefore, when aiming to optimise the performance of a system, it is important to have a clear picture of which performance measures should be optimised.

8.2.2 Performance Analysis Techniques for Architectures

Although several software tools exist to model enterprise architectures, hardly any attention has been paid to the analysis of their quantitative aspects. For detailed design models of (distributed) systems, such as computing and telecommunication systems, and manufacturing systems, a broad range of performance analysis techniques have been proposed. There are very efficient static techniques that offer relatively inaccurate first estimates or bounds for the performance. Analytical solutions of queuing models are more accurate but also more computation intensive, while they still impose certain restrictions on the models. With detailed quantitative simulations, any model can be analysed with arbitrary accuracy, although this presumes that accurate input parameters are available.

As mentioned above, enterprise architecture covers a broad range of aspects, from the technical infrastructure layer (e.g., computer hardware and networks), through software applications running on top of the infrastructure, to business processes supported by these applications. Within each of these layers, quantitative analysis techniques can be applied, which often require detailed models as input. In this subsection, we will only be able to give a global impression of analysis approaches for each of these layers.

We also noted earlier that enterprise architecture is specifically concerned with the relations between the layers. Also from a quantitative perspective the layers are interrelated: higher layers impose a workload on lower layers, while the performance characteristics of the lower layers directly influence the performance of the higher layers. The service concept that is central to the ArchiMate language plays an important role in connecting these layers, also in a quantitative sense (Jonkers and Iacob 2009).

8.2.2.1 Infrastructure Layer

Traditionally, approaches to performance evaluation of computer systems and communication systems (see Harrison and Patel 1992) have a strong focus on the infrastructure domain. Queuing models, for example, describe the characteristics of the (hardware) resources in a system, while an abstract stochastic arrival process captures the workload imposed by the applications. Also, a lot of literature exists on performance studies of specific hardware configurations, sometimes extended to the system software and middleware levels. Most of these approaches commonly are based on detailed models and require detailed input data.
8.2.2.2 Application Layer

Performance engineering of software applications (see Smith 1990) is a much newer discipline compared to the traditional techniques described above. A number of papers consider the performance of software architectures at a global level. Bosch and Grahn (1998) present some first observations about the performance characteristics of a number of often occurring architectural styles. Performance issues in the context of the SAAM method (see Kazman et al. 1994) for scenario-based analysis are considered in Lung et al. (1998).

Another direction of research addresses the approaches that have been proposed to derive queuing models from a software architecture described in an architecture description language (ADL). The method described by Spitznagel and Garlan (1998) is restricted to a number of popular architectural styles (e.g., the distributed message passing style but not the pipe and filter style). Other similar approaches are described in Aquilani et al. (2001) and Williams and Smith (1998). In Di Marco and Inverardi (2004) queuing models are derived from UML 2 specifications, which in most cases, however, do not have an analytical solution.

As we noted in Sect. 3.1.1, compositionality is an important issue in architecture. In the context of performance analysis, compositionality of analysis results may also be a useful property. This means that the performance of a system as a whole can be expressed in terms of the performance of its components. Stochastic extensions of process algebras (see Hermanns et al. 2002) are often advocated as a tool for compositional performance analysis. However, process-algebra-based approaches to performance analysis are still fairly computation intensive, because they still suffer from a state space explosion. Moreover, while they allow for a compositional specification of performance models, this does not necessarily mean that the analysis results are also compositional.

8.2.2.3 Business Layer

Several business process modelling tools provide some support for quantitative analysis through discrete-event simulation. Also, general-purpose simulation tool such as Arena\(^1\) or ExSpect\(^2\) (based on high-level Petri nets) are often used for this purpose. A drawback of simulation is that it requires detailed input data, and for inexperienced users it may be difficult to use and to interpret the results correctly. BiZZdesigner\(^3\) offers, in addition to simulation, a number of analytical methods. They include completion time and critical path analysis of business processes (see Jonkers et al. 1999) and queuing model analysis (see Jonkers and Swelm 1999).

\(^1\) http://www.arenasimulation.com
\(^2\) http://www.exspect.com
\(^3\) http://www.bizzdesign.com
Petri nets (and several of its variations) are fairly popular in business process modelling, either to model processes directly or as an underlying formalism for other languages (e.g., see Schomig and Rau 1995). They offer possibilities for performance analysis based on simulation, as described above, but they also allow for analytical solutions (which are, however, fairly computation intensive).

### 8.2.3 Quantitative Modelling

In this section we present our approach for the quantitative modelling of service-oriented enterprise architectures expressed in the ArchiMate language. First we show that ArchiMate models follow a certain structure that is explained by means of an ‘analysis meta-model’. Our technique focuses on a subset of the ArchiMate language, namely the modelling constructs encompassed by this simple meta-model. Then we clarify what the necessary quantitative input is for our analysis technique. We also introduce an example that shows how quantitative information can be attached to model elements and their relations and that will later also illustrate the application of the algorithms.

#### 8.2.3.1 Model Structure

As shown in Sect. 5.2, many architecture models can be viewed as a hierarchy of layers. We use this layered view for performance analysis as well, because it makes the explanation of our approach easier. Furthermore, layering will help the modeller to formulate and describe clearly the problem being analysed.

For each meta-model layer we can distinguish one or more model layers of two types: service layers and realisation layers. A service layer exposes functionality that can be ‘used by’ the next higher layer, while a realisation layer models shows how the consecutive service layer is ‘realised’. The number of these layers is not fixed, but a natural layering of an ArchiMate model will contain the succession of layers depicted in Fig. 8.4.

Looking at the horizontal structure of the meta-model, we can see that realisation layers typically contain three types of elements. They might model some pieces of internal behaviour (expressed as processes or functions). Further, each behaviour element can access one or more objects and it is assigned to exactly one resource (e.g., actors, devices, application components, etc.).

Thus we can summarise our findings in terms of the ‘analysis meta-model’ depicted in Fig. 8.3, where

- ‘object’ can be a business object, a data object, or an artifact;
- ‘resource’ can be a business role, a business actor, an application component, a system software component, a node, or a device;
– ‘internal behaviour’ can be a business process, a business function, an application function, or an infrastructure function;
– ‘service’ can be a business service, an application service, or an infrastructure service.

### 8.2.3.2 Approach

Before we can analyse an ArchiMate model, we have to define clearly the quantities that can be assigned to the different ArchiMate concepts and relations. In Sect. 8.2.1 we have identified the relevant performance measures, independent of any modelling language. However, we have to make these measures specific for ArchiMate models. One may notice that not all the ArchiMate language elements are included in the model structure given in Fig. 8.3. Indeed, we consider some of them irrelevant for the current approach (e.g., the meaning concept, the aggregation and association relations, etc.) and, therefore, they will be ignored.

Iacob and Jonkers (2005) explore possible ways in which the concepts and relations that have been defined in the ArchiMate language can be quantified. An important observation made is that the richest ArchiMate relations in terms of quantification prospects are the ‘triggering’, ‘access’, ‘realisation’, and ‘used by’ relations. This is a good indication that any quantitative analysis method that might be used in the context of ArchiMate (sub)models must focus on this type of relation. The fact that ‘triggering’ relations are easily quantifiable does not come as a surprise. In fact, triggering relations are essential in revealing the behaviour of dynamic systems. Thus, we can draw the conclusion that any quantitative method that works for (business) process-oriented modelling formalisms can be applied (possibly after slight adaptations) as well for ArchiMate models. However, these
types of methods have certain limitations from the ArchiMate point of view for at least two reasons. First, such methods are usually applied locally to partial architectural models that expose a mapping between a piece of behaviour and some resources (see Jonkers and Swelm 1999). Second, because only two types of elements, namely behaviour elements (e.g., processes, events, etc.), and resources (e.g., actors, devices, etc.) are concerned, such methods do not traverse all the architecture domains. They typically work within at most two layers of the architecture model (e.g., the process and the infrastructure layer, or the process and the organisational/actor–roles layer). We will refer to such analysis methods as being horizontal methods. We believe that apart from the classical horizontal methods we must expose vertical methods that work across multiple domains. We anticipate that such methods must focus on the ‘used by’ and ‘realisation’ relations bridging the different architectural domains. Nevertheless, the distinction between horizontal and vertical methods must not be considered restrictive at all, since (as it will also result from the example we are giving) the two types of methods can be combined, such that the output (i.e., calculated performance measures) of one type of method will provide the input quantities for another ‘follow-up’ analysis method.

Analysis across an architecture model is possible through the propagation of quantities through layers. A natural option for this is to consider workload measures (e.g., arrival frequencies) that are imposed as a ‘demand’ on the model elements by the users (located in the higher layers, e.g., customers). These quantities propagate towards the lower layers, eventually being translated in demands on each model element. Once workloads have been determined, we look at the effort these workloads require from the resources (modelled by structural elements) and from the behaviour elements (modelled by services, processes, and functions). This effort can be expressed in terms of performance measures (e.g., utilisations for resources, response and processing times for behaviour elements) and/or costs; it starts in the infrastructure and propagates back to the higher layers. In summary, our approach consists of the following two phases (see Fig. 8.4): a top-down calculation of the workloads imposed by the users; this provides input for a bottom-up calculation of performance measures.

8.2.3.3 Quantitative Input Data

One of the most difficult tasks related to quantitative analysis is to obtain reliable input data. There are several possible sources for this data. For existing systems or organisations, measurement can be one of the most reliable methods, although it is not easy to do this in a correct way: among others, it should be clearly defined what exactly is to be measured, the number of measurements must be sufficient, and the measurements must be taken under various circumstances that can occur in practice.

In case the system or organisation is still to be developed, measurement is no option. Possible alternatives are then the use of documentation of components to be used, or to use estimates (e.g., based on comparable architectures). However, one
should keep in mind that it is often very difficult to interpret correctly the available numerical data, and to evaluate the reliability of the available data.

We assume that the following quantitative input is provided for analysis (see Fig. 8.3):

- For any ‘used by’ and ‘access’ relation \( e \), a weight \( n_e \), representing the average number of uses and accesses.
- For any behaviour element \( a \), a service time \( S_a \), representing the time spent internally for the realisation of a service (excluding the time spent waiting for supporting services). We assume that a service inherits the service time value of the element realising it.
- For any resource \( r \), a capacity \( C_r \).
- For any node \( a \), an arrival frequency \( f_a \). Typically, arrival frequencies are specified in the top layer of a model, although we do allow for the specification of arrival frequencies for any node in the model.

These quantitative attributes are attached to the corresponding model elements using the ‘profile’ mechanism described in Sect. 5.11.1.

8.2.3.4 Example

To show the practical use of this analysis technique, we illustrate our approach with the following simple example.

Suppose we want to analyse an insurance company that uses a document management system for the storage and retrieval of damage reports. We assume that the document management system is a centralised system, used by multiple offices throughout the country, which means that it is quite heavily used. A model
of this system is depicted in Fig. 8.5. This model covers the whole stack from business processes and actors, through applications, to the technical infrastructure.

There are three applications offering services that are used directly by the business actors. The Administrator can search in the metadata database, resulting in short descriptions of the reports that meet the query and view reports that are returned by a search. The report scanning application is used to scan, digitise, and store damage reports (in PDF format). In addition to the two applications that are used directly by the end user, there are two supporting application components: a database access component, providing access to the metadata database, and a
8.2 Quantitative Analysis

document management component, providing access to the document base. Finally, the model shows the physical devices of which the database access and document management components make use. They use file access services provided by these devices.

In the model we also specify the input quantities for the analysis. On the ‘used by’ relations, we specify workload values, in terms of the average number of uses $n$ of the corresponding service by the layer above. For the business processes, an arrival frequency $f$ is specified. In this example we assume that all resources have the default capacity 1. Finally, for service elements we may specify a service time $S$.

8.2.4 Quantitative Analysis Technique

The goal of our approach is to determine the following performance measures (see Fig. 8.3):

- the workload (arrival rate) $\lambda_a$ for each node $a$ (note that, provided that no resources are overloaded, the throughput for each node is equal to its arrival rate);
- the processing time $T_a$ and the response time $R_a$, for each behaviour element or service;
- the utilisation $U_r$, for each resource $r$.

To derive the above-mentioned performance measures, given the input values, we proceed in three steps:

1. We will first ‘normalise’ any input model, using model transformations, in order to generate a model that is compliant with the structure presented in Fig. 8.3.
2. Top-down calculation of workloads (arrival rates) $\lambda$.

8.2.4.1 Step 1: Model Normalisation

Typical ArchiMate models often do not fully conform to the ArchiMate metamodel. This is due to the fact that during the modelling process, abstraction rules are used to create simplified views of the architecture. These abstractions have, however, a formal basis in an operator that has been derived for the composition of relations. The derivation of this operator has been described in great detail in Buuren et al. (2004). It allows, for instance, the composition of a realisation relation with any consecutive used by relation, resulting in a new used by relation that short-circuits, in this case, a service.

Therefore, the first step in our approach addresses a model transformation procedure, which will derive from any input model a ‘normalised’ one, i.e., a model, which is fully compliant with the structure described in Fig. 8.3. Since
some of the concepts and relations are not relevant for our approach, the normalisation procedure starts by eliminating them from the original model. The resulting model will then be subjected to a series of model transformations. One example of such a transformation rule is given in Fig. 8.6. The set of all possible transformation rules is finite, which makes the development of a normalisation algorithm based on these rules rather straightforward.

The application (if needed), following such an algorithm, of the proper rule for each edge in the input model will eventually lead to a normalised model.

Fig. 8.7 shows the normalised version of the example model given in Fig. 8.5. The input parameters for the workload on the ‘used by’ relations are the same as in the original model. The service times are now transferred also to the inserted internal behaviour elements.

However, since model normalisation is not the primary focus of this approach we will not provide a formal description of the normalisation algorithm, although such an algorithm was implemented in the quantitative analysis prototype described in Sect. 10.5.

### 8.2.4.2 Step 2: Top-Down Workload Calculation

Given a normalised model, we can now calculate the workload (i.e., arrival rate) for any node $a$. The following recursive expression applies:

$$\lambda_a = f_a + \sum_{i=1}^{d_a^+} n_{a,k_i} \lambda_{k_i},$$

where $d_a^+$ denotes the out-degree of node $a$ and $k_i$ is a child node of $a$. In other words, the arrival rate for a node is determined by adding the requests from higher layers to the local arrival frequency $f_a$. 

![Fig. 8.6 Example of a normalisation step](image-url)
The results of this step in the ‘document management system’ example are given in Table 8.1, which shows the workload for the services $s$ in the model, in terms of the arrival rates $\lambda_s$. The arrival rates depend on the frequencies of the customer input requests and the cardinalities $n$ of the ‘used by’ relations. The table also shows the scaled arrival rates expressed in arrivals/second (assuming that systems are operational eight hours per day).

![Diagram of the normalised model](image)

**Fig. 8.7** Normalised model

8.2.4.3 Step 3: Bottom-Up Performance Calculation

Once the workloads imposed on the various model components are calculated, we can proceed with the last analysis phase, the bottom-up calculation of the aforementioned set of performance measures. The approach we take is somewhat similar...
to the top-down one. In this step we focus on the bottom-up propagation of values corresponding to different time-related performance measures. The actual calculation can be done using the following recursive expressions:

– The utilisation of any resource \( r \) is

\[
U_r = \frac{\sum_{i=1}^{d_r} \lambda_{k_i} T_{k_i}}{C_r},
\]

where \( d_r \) is the number of internal behaviour elements \( k_i \) to which the resource is assigned.

– The processing time and response time of any service \( a \) coincide with the processing time and response time of the internal behaviour element realising it, i.e., \( T_a = T_k \) and \( R_a = R_k \), where \((k, a)\) is the only realisation relation having \( a \) as end point.

– The processing time and response time of any internal behaviour element \( a \) can be computed using the following recursive equations:

\[
T_a = S_a + \sum_{i=1}^{d^-_a} n_{k_i, a} R_{k_i} \quad \text{and} \quad R_a = F(a, r_a),
\]

where \( d^-_a \) denotes the in-degree of node \( a \), \( k_i \) is a parent of \( a \), \( r_a \) is the resource assigned to \( a \), and \( F \) is the response time expressed as a function of attributes of \( a \) and \( r_a \).

For example, if we assume that the node can be modelled as an M/M/1 queue (Harrison and Patel 1992), this function is

\[
F(a, r_a) = \frac{T_a}{1 - U_{r_a}}, \quad (8.1)
\]

<table>
<thead>
<tr>
<th>Table 8.1 Workloads and performance results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resource (( r ))</td>
</tr>
<tr>
<td>Doc. srv.</td>
</tr>
<tr>
<td>DB srv.</td>
</tr>
<tr>
<td>Doc.mgt.sys.</td>
</tr>
<tr>
<td>Doc.mgt.sys.</td>
</tr>
<tr>
<td>DB sys.</td>
</tr>
<tr>
<td>DB sys.</td>
</tr>
<tr>
<td>Search comp.</td>
</tr>
<tr>
<td>View comp.</td>
</tr>
<tr>
<td>Rep. scanning</td>
</tr>
</tbody>
</table>
We can replace this by another equation in case other assumptions apply, e.g., the Pollaczek–Khinchine formula for an M/G/1 if $T_a$ has a non-exponential distribution, or the solution for an M/M/n queue based on the Erlang C formula for a structural element with a capacity greater than 1 (Iacob and Jonkers 2005). We might also consider more global solutions, e.g., operational performance bounds (Eager and Sevcik 1986). In case more precise results are required, instead of simple queuing formulae, more detailed techniques such as simulation can be applied in combination with our approach.

Table 8.1 also shows the performance results for the example model after the execution of step 3. We have calculated the processing and response times for the services and the utilisations for the resources at the application and infrastructure layers (in this example, the business layer is only relevant because it provides the input for the workloads). However, the performance results can easily be extended to the business layer as well.

For simplicity, we assume Poisson arrivals and exponentially distributed service times in this example, so that every structural element $a$ can be modelled as an M/M/1 queue (Harrison and Patel 1992). Hence, the response time function is given by (1).

The results show that queuing times from the lower layers accumulate in the higher layers, which results in response times that are orders of magnitude greater than the local service times. For example, the ‘view’ component of the ‘claim handling support’ application has a utilisation of over 84%, which results in a response time of the ‘view damage report’ application service of almost 3 minutes.

Using our approach, it is easy to study the effect of input parameter changes on the performance. For example, Fig. 8.8 shows how the response time of the View component depends on the arrival frequency associated with the Administrator (assuming a fixed arrival frequency for the Damage expert). The maximum arrival frequency, which results in a utilisation of the View component of 100%, is 651 arrivals per day. In the design stage these results may help us to decide, for example, if an extra View component is needed.
8.3 Portfolio Analysis

Analysis of enterprise architecture models can provide important input for portfolio management, as is described by (Quartel et al. 2010). A desired organisational or technical change requires the investigation of the stakeholders that are involved and their concerns regarding the change. New goals and requirements are identified, or existing ones are changed, to address these concerns. Analysis of these goals and requirements is needed to guarantee consistency and completeness, and to propose one or more alternative architecture designs that realise the goals and requirements. Validation of these alternative designs aims at assessing their suitability and selecting the best alternative.

As we have outlined in Sect. 5.9, the ArchiMate 2.0 Motivation extension provides concepts for modelling stakeholders, their goals and drivers, and the resulting requirements. Since the various elements of the enterprise architecture can be related to these motivational elements, we are now able to assess in more detail how architectural decisions contribute to the organisation’s goals. Quantifying such contributions helps in evaluating your project or application portfolio and in making the right investment decisions.

A contribution can be divided into two elements: its importance to a business goal and the quality or effectiveness in supporting that goal. The value of an organisation’s service portfolio thus depends on the contribution that its constituent elements provide to the business. An interesting and useful way of computing a service portfolio’s value based on these business contributions is Bedell’s method (Schuurman et al. 2008). This method answers three questions:

1. Should the organisation invest in information systems/services?
2. On which business processes should the investment focus?
3. Which information systems should be developed or improved?

The underlying idea of the method is that a balance is needed between the level of effectiveness of the information systems and their level of strategic importance (the diagonal in Fig. 8.9). Investments are more crucial if the ratio between the effectiveness of an information system and its importance is worse. The example of Fig. 8.9 shows that application a is a candidate for aggressive investment, since its effectiveness is lower than its importance warrants; conversely, b can do with a lower investment level.

To calculate these values, Bedell’s method uses:

– the importance of each business process to the organisation (IBO);
– the importance of each business activity to the business processes (IAB);
– the effectiveness of an information system (software application) in supporting business activities (ESA).
From these inputs, various other values can be calculated, starting with the effectiveness of an information for a business process (ESB), computed as 

\[ \text{ESB} = \frac{\text{ESA}}{\text{C}} \times \text{IAB} \]

the effectiveness of all information systems for the business process EIB = \[ \sum_{S} \text{ESB} / \sum_{A} \text{IAB} \], and so on, until we know aggregate measures of the effectiveness and importance of each information system for the organisation. These can then provide input for investment decisions, as outlined above.

This type of calculations can easily be transferred to our architecture models, as Fig. 8.10 shows. However, Bedell’s method was not designed for use in combination with architecture models, and it has two issues that we needed to address. First, it has a fixed level structure consisting of organisation, business processes, activities and information systems, whereas our models may have many more levels. Second, it assumes a one-to-one relationship between activities and information systems. In both areas, we have extended and generalized the method to fit with our models; for more detail, see (Quartel et al. 2010).

Measuring importance requires insight in the ways in which an IT system, service, or business process contributes to the business goals. This value can lie in many different aspects, such as the timely and accurate information that the system delivers as input for business decisions, customer satisfaction, and return business created through its user-friendly interface, or the value of future opportunities opened up by IT.

Methods such as the Application Life Cycle Duration Measurement Method (ALMM) from the Application Services Library (ASL) methodology (Pols and Backer 2006) can provide relevant input for these calculations. The ALMM measures the life cycle of applications by determining their current Business Value (BV) and the Technical Value (TV) and then by estimating the development of the BV and TV in the future, assuming a continuation of the current IT policy. Business Value is very close to Bedell’s definition of strategic importance, and Technical Value is close to Bedell’s notion of effectiveness.

ALMM’s main limitation is that it addresses only a single level of abstraction, that of applications. Complementing it with the calculations as outlined above
provides us with a better foundation for assessing the contributions of architecture elements to business goals at different levels of our enterprise architecture.

8.4 Functional Analysis

In this section we illustrate how functional analysis techniques can be based on existing techniques in formal methods. Though these formal methods have been developed for systems and problems which have been defined in a mathematically precise way, and architecture descriptions in most cases have an informal character, we show that these formal methods can be used when we introduce a few basic definitions we briefly explained in Chap. 3, such as signature, symbolic model, and interpretation.

In functional analysis of architectures, we distinguish between static or structural and dynamic or behavioural aspects. For analysing the static structure of an architecture, its \textit{signature} (see Sect. 3.3) forms the basis. This focuses on the symbolic representation of the structural elements of an architecture and their relationships, abstracting from other architectural aspects like rationale, pragmatics, and visualisation. It emphasises a separation of concerns which helps in mastering the complexity of the architecture. Notably, the signature of an architecture can be expressed in XML for storage and communication purposes, and can be integrated as an independent module with other tools including graphics for visualisation.
For the logical analysis of the dynamics of an architecture, the formal semantics (see also Sect. 3.3) of a symbolic model of that architecture provides a formal basis. A signature of an architecture only specifies the basic concepts with which the architecture is described, but an interpretation contains much more detail. In general, there can be a large number of different interpretations for a signature. This reflects the intuition that there are many possible architectures that fit a specific architecture description.

By applying the techniques for static and dynamic analysis discussed in the next subsections, we gain a better understanding of how enterprise architectures are to be interpreted and what is meant by the individual concepts and relationships. In other words, these techniques allow enterprise architects to validate the correctness of their architectures, to reduce the possibility of misinterpretations, and even to enrich their architecture descriptions with relevant information in a smooth and controllable way.

We do not detail the formal methods themselves, which would require at least a textbook for each method (and many good textbooks for these methods exist). Instead, we use small example architectures to illustrate the use of these methods for architectural analysis. More technical details can be found in De Boer et al. (2004, 2005) and in Stam et al. (2004).

The structure of this section is as follows. In Sect. 8.4.1 we give an introduction to static analysis, in particular of impact-of-change analysis on an architecture, and we show how this can be applied to the ArchiSure case described earlier. In Sect. 8.4.2, we go deeper into dynamic analysis. Using another example architecture, we show how an ArchiMate description of an architecture can be translated into a signature, illustrate how this signature can be extended to a symbolic model, and how this symbolic model can be interpreted by a semantic model. We briefly describe two relevant formal methods, namely process algebras and data flow models. Finally, we show how we can interpret the example architecture as a process algebra and as a data flow network, respectively.

### 8.4.1 Static Analysis

For structural analysis of architectures, description logics are useful formalisms. Description logics are knowledge representation languages tailored to express knowledge about concepts and concept hierarchies. They are considered an important formalism unifying and giving a logical basis to the well-known traditions of frame-based systems, semantic networks, and KL-ONE-like languages (Woods and Schmolze 1992), object-oriented representations, semantic data models, and type systems. Description logic systems have been used for building a variety of applications including conceptual modelling, information integration, query mechanisms, view maintenance, software management systems, planning systems, configuration systems, and natural language understanding. In the case of enterprise architecture, the main application of description logics is in determining the impact
of a change to an architecture: what elements of the model will be ‘touched’ by this change?

As an example of static analysis, we again consider our fictitious ArchiSurance company, which offers insurance to customers. ArchiSurance sells its products by means of intermediaries. Intermediaries investigate the concerns of customers, negotiate a policy contract, and take care of the administrative work and the communication with ArchiSurance (see also Fig. 7.10, the Actor Cooperation view of ArchiSurance).

The role of the intermediary is purely commercial: he or she only sells products to customers and makes sure all paperwork is done correctly until the customer has signed a contract. After this, the intermediary is only involved in case of problems with the collection of premiums.

ArchiSurance architects want to investigate quickly if it would be possible to eliminate the entire idea of intermediaries. What would be the consequences of such a drastic change of the business model on the enterprise architecture of ArchiSurance?

The starting point of this analysis is the relationship between the various views and a logical theory. As we explained in Chap. 3, underlying these views is a single architecture model, which corresponds to a signature, which is used in the logical analysis.

In this signature there are sorts for roles, collaborations etc., and there are domain-dependent sorts, such as ‘insurance company’ and ‘customer’. Performing such a structural analysis implies ‘traversing’ the architecture and taking into account each relation and its meaning to determine whether the proposed change might ‘propagate’ through this relation. If, for example, a service provided by an application changes, every user of that service may be affected.

To express this, every relation in the architecture model is translated into a relation in the logic. In the translation there are also some constraints between the sorts and the relations to make the correspondence precise. Examples of such constraints, expressed in first-order logic, are the following:

\[ \forall x: \text{Customer}(x) \rightarrow \text{Role}(x) \]
\[ \forall x,y: \text{Participate}(x,y) \rightarrow \text{Role}(x) \land \text{Collaboration}(y) \]

The first rule states that every Customer is also a Role; the second states that only Roles can participate in Collaborations, and, vice versa, that participants of Collaborations are Roles. Of course, much more complex rules are used to express the impact of a change of a model element on related elements. Such logic rules can be processed by a ‘rules engine’ that automates the impact analysis. A prototype of such an analysis tool is described in Sect. 10.6.

In our example, if ArchiSurance wants to change the role of the intermediary, this will have an impact on all collaborations in which this intermediary participates (Fig. 8.11). Several business processes will be involved in this through interactions performed by these collaborations; one of these is the ‘Close Contract’ process, shown in Fig. 8.12. This uses a number of applications, some of which may also be influenced by the change (Fig. 8.13). Naturally, these examples show only a small part of the actual impact of the proposed change, but they serve to illustrate the general idea.
In these examples, we have not shown how an architecture description in the ArchiMate language can be translated into an underlying formalism that forms the basis of these analysis techniques. In the next subsection, on dynamic analysis, we will go deeper into this issue, and show how the signature of an architecture can be defined, how this signature can be interpreted semantically, and how formal analysis techniques can be built upon that.

### 8.4.2 Dynamic Analysis

For dynamic analysis of architectures, functional analysis techniques based on formal approaches such as process algebras and data flow networks are useful. Issues like two roles acting at the same time, overwriting or destroying each other’s work, can be identified and then a suitable protocol can be designed to prevent the
problem. Thus, a functional behaviour analysis based on formal methods is primarily a qualitative analysis that can detect logical errors, leads to a better consistency, and focuses on the logic of models.

The dynamics of a concrete system with an architecture description given by its signature can be specified in different ways; we distinguish between specifications tailored towards control flow modelling and those tailored towards data flow modelling. For control flow modelling, we give a brief introduction to process algebra, while for data flow modelling, we introduce the reader to data flow networks.

To illustrate the use of these formal methods, we use the enterprise architecture of a small company, ArchiSell, modelled using the ArchiMate language. In ArchiSell, employees sell products to customers. Various suppliers deliver the products to ArchiSell. Employees of ArchiSell are responsible for ordering products and for selling them. Once products are delivered to ArchiSell, each product is assigned to an owner responsible for selling the product. More specifically, we look at the business process architecture for ordering products, visualised in Fig. 8.14. To describe this enterprise we use the ArchiMate modelling concepts and their relationships. In particular, we use structural concepts (product, role, and object) and structural relationships (association), but also behavioural concepts (process) and behavioural relationships (triggering). Behavioural and structural concepts are connected by means of the assignment and access relations.

In order to fulfil the business process for ordering a product, the employee has to perform the following activities:

- Before placing an order, an employee must register the order within the Order Registry. This Order Registry is for administration purposes. It is used to check orders upon acceptance of goods later in the process. Orders contain a list of products to be ordered.
After that, the employee places the order with the supplier. Based on the order, the supplier is supposed to collect the products and to deliver them as soon as possible.

As soon as the supplier delivers the products, the employee first checks if there is an order that refers to this delivery. Then, the employee accepts the products.

Next, the employee registers the acceptance of the products within the Product Registry and determines which employee will be the owner of the products.

Although the example is rather trivial, it serves to illustrate how an architecture description can be formalised and how it can be subjected to functional analysis.

### 8.4.2.1 Signature

We first define the signature of the business process architecture described in Fig. 8.14. The sorts of the example are simply enumerated as follows:

- **Role**
- **Object**
- **Employee**
- **Product**
- **product**
- **Order_Registry**
- **Product_Registry**

Note that we did not include processes as a sort, because processes are modelled below as functions.

Further information about the architecture is expressed symbolically in terms of suitable extensions of one of its signatures. Usually, a signature is extended with
operations for constructing complex types from the primitive sorts. Examples are the standard type operations like the product type $T_1 \times T_2$ of the types $T_1$ and $T_2$, and the function type $T_1 \rightarrow T_2$ of all functions which require an argument of type $T_1$ and provide a result of type $T_2$. Given functional types, the name space of a signature can be extended with functions $F(T_1) : T_2$, where $F$ specifies the name of a function of type $T_1 \rightarrow T_2$. Functions can be used to specify the attributes of a sort. For example, given the primitive sorts Employee and N, the function Age(Employee) : N is intended for specifying the age of each person. Examples of the subsort relation are the following:

- Employee is_a Role
- Product is_a product
- Order_Registry is_a Object
- Product_Registry is_a Object
- Owns is_a association

Note that we have encoded meta-model information of an architecture description as part of the signature of the architecture itself. The relation between the meta-model sorts and relations and architecture sorts and relations is expressed by the respective partial orders between sorts and relations of the signature. For example, the sort Product in Fig. 8.14 is modelled as a subsort of the ArchiMate concept product. The ‘owns’ relation itself is specified by:

Employee owns Product

Also note that the triggering relation is not included in our concept of a signature. In our view such a relation specifies a temporal ordering between the processes, which are described in Sects. 10.1.3 and 10.1.4.

### 8.4.2.2 Interpretation

To obtain a formal model of a system as a semantic interpretation of the symbolic model of its architecture description, we start with an interpretation of the signature. An interpretation $I$ of the types of a signature assigns to each primitive sort $S$ a set $I(S)$ of individuals of sort $S$ which respects the subsort ordering: if $S_1$ is a subsort of $S_2$ then $I(S_1)$ is a subset of $I(S_2)$. Any primitive sort is interpreted by a subset of a universe which is given by the union of the interpretation of all primitive sorts. The subset relation expresses the hierarchy between primitive sorts. An interpretation $I$ of the primitive sorts of a signature of an architecture can be inductively extended to an interpretation of more complex types. For example, an interpretation of the product type $T_1 \times T_2$ is given by the Cartesian product $I(T_1) \times I(T_2)$ of the sets $I(T_1)$ and $I(T_2)$. The function type $T_1 \rightarrow T_2$ thus denotes the set of all functions from the universe to itself such that the image of $I(T_1)$ is contained in $I(T_2)$. In general, there can be a large number of different interpretations for a signature. This reflects the intuition that there are many possible architectures that fit a specific architecture description.
The semantic model of a system involves its concrete components and their concrete relationships, which may change in time because of the dynamic behaviour of a system. To refer to the concrete situation of a system, we have to extend its signature with names for referring to the individuals of the types and relations. For a symbolic model, we denote by \( n:T \) a name \( n \), which ranges over individuals of type \( T \).

As an example, we introduce the following semantic model. We define only two products: \( p_1 \) and \( p_2 \). In order to model the processing of orders and products, individuals of the sort \( \text{Employee} \) have a \textit{product} attribute and an \textit{order} attribute. These attributes indicate the order and product the employee is managing. In our model, individuals of the sort \( \text{Employee} \) are fully characterised by these attributes. Therefore in our model the sort \( \text{Employee} \) contains four elements, namely:

\[
\begin{align*}
e_1 \text{order} &= p_1 \text{product} = p_1 \\
e_2 \text{order} &= p_1 \text{product} = p_2 \\
e_3 \text{order} &= p_2 \text{product} = p_1 \\
e_4 \text{order} &= p_2 \text{product} = p_2
\end{align*}
\]

Furthermore, we define the order and product registries as possibly infinite lists of products.

Finally, in order to refer to the elements of the different sorts we introduce individual names \( \text{emp}: \text{Employee} \), \( \text{order-reg}: \text{Order Registry} \), and \( \text{product-reg}: \text{Product Registry} \). A semantic model assigns individuals to these names. For example:

\[
\begin{align*}
\text{emp} &= e_1 \text{order} = p_1 \text{product} = p_1 \\
\text{order-reg} &= \{ p_1 \} \\
\text{product-reg} &= \{ p_2 \}
\end{align*}
\]

Note that this assignment describes an employee, who manages an order of product \( p_1 \) and a delivery of product \( p_1 \), an Order Registry, which registers an order of product \( p_1 \), and a Product registry, which registers the acceptance of a product \( p_2 \).

### 8.4.2.3 Process Algebras

Process algebra (Baeten and Weijland 1990; Bergstra et al. 2001) is a formal description technique for specifying the control flow behaviour of complex systems. Starting from the language syntax, each statement of the language is supplied with some kind of behaviour, and a semantic equivalence says which behaviours are identical. Process algebras express such equivalences in axioms or equational laws. The axioms are to be sound, i.e., if two behaviours can be equated then they are semantically equivalent. The converse statement is optional, and is called completeness, i.e., if two behaviours are semantically equivalent then they can be equated.
The system is captured as a set of processes interacting with each other according to predefined rules. Starting from a set of basic actions, processes may be hierarchically composed by means of operators, e.g., sequential composition, choice, parallel composition.

We derive these basic actions from the functions of a symbolic model of an architecture. To this end, we define the action of a function \( F(S) : T \) by an assignment of the form \( n := F(m) \) where \( n : T \) and \( m : S \) are names ranging over the types \( T \) and \( S \), respectively. The execution of such an action in a semantic model \( \Sigma \) assigns to the name \( n \) the return value of \( \Sigma(F)(\Sigma(m)) \) which denotes the result of applying the function \( \Sigma(F) \in I(S \rightarrow T) \) to the element \( \Sigma(m) \in I(S) \). Note that actions transform semantic models (i.e., the state of a system) but not the interpretation of a signature (i.e., the structural information of a system).

Given this concept of an action as a transformation of semantic models, we can define more complex processes by combining actions; that is, we can define operations on actions determining the order of their execution. For example, we can define the sequential composition \( n := F(m); n' := G(m') \) of two actions \( n := F(m) \) and \( n' := G(m') \) as the composition of their transformation of semantic models.

Process algebras can be applied to model any business function and to prove its correctness. They enable properties of the business of an enterprise to be expressed in an abstract way and to deduce whether a specific process satisfies these properties.

Now let us consider the process steps within the ArchiSell example. Within the process algebra interpretation, processes are specified as functions. The types of the arguments and result values are determined as follows:

- A role, which is assigned to a process, specifies the type of both an argument and a result value of the corresponding function.
- An outgoing access relation from a process to a data object specifies the type of both an argument and a result value of the corresponding function.
- An incoming access relation from an object to a process only specifies the type of the corresponding argument (this captures the property of ‘read-only’).

This results in the following functions:

- **Register_order_placement**
  - domain name=Employee
  - domain name=Order_Registry
  - codomain name=Employee
  - codomain name=Order_Registry

- **Place_order_for_product**
  - domain name=Employee
  - codomain name=Employee

- **Accept_product**
  - domain name=Employee
8.4 Functional Analysis

- domain name=$\text{Order\_Registry}$
- codomain name=$\text{Employee}$

- $\text{Register\_product\_acceptance}$
  - domain name=$\text{Employee}$
  - domain name=$\text{Product\_Registry}$
  - codomain name=$\text{Employee}$
  - codomain name=$\text{Product\_Registry}$

The interpretation of the processes can be specified in a pseudo-language. For more simple functions, matrices of input/output pairs can be given. For example, the interpretation of the $\text{Register\_order\_placement}$ function can be as follows: add to the $\text{Order\_Registry}$ (which is a list, as defined in the signature) the product of the product attribute of the $\text{Employee}$. Other processes are formally described in a similar manner.

Within a process algebra, we now can concatenate the individual functions in order to model the transformation of an initial state of a concrete system to an eventual state. In this way, we can reason about the correctness of the transformation.

8.4.2.4 Data Flow Networks

A data flow network (Jagannathan 1995) is another formal description technique for the behavioural specifications of complex systems. Such a network consists of some processes, the functions of a symbolic model that communicate by passing data over lines. A process is a transformation of data within the system, whereas a line is a directed FIFO channel connecting at most two processes. Data passed over a line by a process will arrive in an unspecified but finite amount of time at the destination process in the same order as it was sent.

Data flow diagrams can be used to provide a clear representation of any business function. The technique starts with an overall picture of the business and continues by analysing each of the functional areas of interest. This analysis can be carried out to the level of detail required. The technique exploits a method called top-down expansion to conduct the analysis in a targeted way. The result is a series of diagrams that represent the business activities in a way that is precise, clear, and easy to communicate.

In a data flow interpretation of the ArchiSell process, we consider each individual process step as an independent data-consuming/data-producing entity. Such an entity has input ports and output ports. Within the data flow interpretation we are interested in the data flow within the process, but not directly in the actors (or roles) that perform the process. Therefore, this interpretation is specifically suited for situations in which many details are known about the data and less about the actors. However, as we will illustrate, a data flow interpretation can help us in the assignment of actors to process steps.
The way in which we can interpret the example as a data flow network is shown in Fig. 8.15. Note the following:

- We leave out any information about roles and individuals within the role sort. So, the data flow diagram does not contain information about which actor performs which process steps.
- We specify registries as stores, i.e., special functions, which resemble places in which information can be stored and from which the same information can be retrieved later.
- We explicitly identify which input/output ports receive/send which kind of values. A practical way is to begin with identifying the values on the input/output ports, and then to connect the output ports to other input ports.

The following values are communicated:

1. list of products that have to be ordered;
2. list of products that have to be ordered;
3. order registry record;
4. list of products that have to be ordered;
5. supplier order;
6. list of products received;
7. order registry record;
8. list of products accepted;
9. list of products accepted;
10. product registry record;
11. product registry record;
12. order registry record;
13. order registry record.

With such a data flow diagram, we can define data flow for each individual process step. The functions transform certain inputs into a certain output. Such functions can be defined in, for example, a pseudo-language, but it is also possible to derive a working simulation of the process architecture in this way.

The data flow diagram also enables us to reason about the assignment of process steps to actors. For example, the process diagram, correct as it is, does not reveal if the step ‘register order placement’ should be fulfilled by the same actor as the step
'accept product'. The data flow diagram reveals what is needed in order to assign actors correctly to process steps.

An example of this is the following. Suppose that we would like to have the first two process steps to be performed by an actor different from the one that performs the last two process steps. The data flow diagram reveals that this is no problem, since no values are communicated directly between those two sets of process steps. In other words, the data flow diagram shows that, given this interpretation of the process architecture, it facilitates separation between order placement and product acceptance.

8.5 Summary

In this chapter we have considered the relation between enterprise architecture models and architecture analysis. We addressed two main classes of methods, quantitative analysis and functional analysis.

Although the importance of enterprise architecture modelling has been recognised, hardly any attention has been paid to the analysis of its quantitative properties. Most existing approaches to performance evaluation focus on detailed models within a specific domain. We demonstrated the applicability of quantitative modelling and analysis techniques for the effective evaluation of design choices at the enterprise architectures level, in the context of ArchiMate models. We discerned a number of architecture viewpoints with corresponding performance measures, which can be used as criteria for the optimisation or comparison of such designs. We introduced a new approach for the propagation of workload and performance measures through a layered enterprise architecture model. This can be used as an analysis framework where existing methods for detailed performance analysis, based on, for example, queuing models, Petri nets or simulation, can be plugged in. The presented example illustrates the use of our top-down and bottom-up technique to evaluate the performance of a document management system for the storage and retrieval of damage reports. Using a simple queuing formula for the response times, we showed that queuing times from the lower layers of the architecture accumulate in the higher layers, which may result in response times that are orders of magnitude greater than the local service times. In order further to illustrate and validate our approach, we have developed a prototype, which is outlined in Chap. 10. The practical use of these techniques is illustrated in a case study of the Dutch Tax and Customs Administration, which is described in Chap. 11.

By applying functional analysis techniques, we aim for a better understanding of how architectures are to be interpreted. These techniques allow enterprise architects to validate the correctness of their architectures, to reduce the possibility of misinterpretations, and to enrich their architecture descriptions with relevant information in a smooth and controllable way.

In functional analysis, we distinguished between static or structural and dynamic or behavioural aspects. Furthermore, our approach is based on the distinction
between symbolic and semantic models of architectures. The core of a symbolic model consists of its signature that specifies symbolically its structural elements and their relationships. A semantic model is defined as a formal interpretation of the symbolic model. Semantic models are at the centre of our logical perspective of enterprise architectures, which integrates both static and dynamic aspects. This leads to more precise characterisation of the architecture concepts and provides a formal basis for functional analysis. The framework we have developed allows the integration of various techniques, ranging from static analysis to process algebras and data-flow networks. One important application of these techniques is impact-of-change analysis, a prototype of which will be described in Chap. 10.

As we have seen, both quantitative and functional analysis techniques help the architect in creating a better insight into the complexities of an enterprise architecture. For further integration into the architecture design process, combining quantitative and functional analysis (e.g., impact-of-change analysis based on quantitative results) could be fruitful.