Quantitative Analysis of Enterprise Architectures

Chapter · July 2006

DOI: 10.1007/1-84628-152-0_22

2 authors:

Maria-Eugenia Iacob
University of Twente
133 PUBLICATIONS **2,259** CITATIONS
SEE PROFILE

Henk Jonkers
BiZZdesign
70 PUBLICATIONS **2,306** CITATIONS
SEE PROFILE

Quantitative Analysis of Enterprise Architectures

Maria-Eugenia Iacob and Henk Jonkers

Telematica Instituut, P.O. Box 589, 7500 AN Enschede, the Netherlands
E-mail: {MariaEugenia.Iacob, Henk.Jonkers}@telin.nl

Abstract. Enterprise architecture is concerned with a description of all the relevant elements that make up an enterprise and how these elements inter-relate. It covers aspects ranging from the technical infrastructure, through software applications, to business processes and products. The relations between these layers play a central role. Also from a quantitative analysis perspective, the layers are interrelated: the higher layers impose a workload on the lower layers, while the performance characteristics of the lower layers directly influence the performance of the higher layers. This paper presents an approach for quantitative analysis of layered, service-based enterprise architecture models, which consists of two phases: a 'top-down' propagation of workload parameters, and a 'bottom-up' propagation of performance or cost measures. By means of an example we demonstrate the application of the approach, and show that a seamless integration with other performance analysis methods (e.g., queueing analysis) can be achieved.

1 Introduction

An enterprise can be viewed as a complex 'system' consisting of multiple domains that influence each other. Architectures are used to describe components, their relations and underlying design principles of a system [10]. Constructing architectures for an enterprise may help to, among others, increase the insight and overview required to successfully align the business and ICT. Although the value of architecture has been recognised by many organisations, mostly architectures for various organisational domains, such as business processes, applications, information and technical infrastructure, are developed in isolation. The relations between these architectures often remain unspecified.

Enterprise architecture is a discipline that focuses on making these relations explicit. The term refers to a description of all the relevant elements that make up an enterprise and how those elements interrelate. Models play an important role in all approaches to enterprise architecture. Models are well suited to express the interrelations among the different elements of an enterprise and, especially when visualised in different ways, they can help to reduce the language barriers between domains. However, currently they strongly focus on functional aspects.

In contrast to detailed design models within domains, the quantitative aspects of such enterprise architecture models have hardly received any attention in literature. Nevertheless, quantitative properties are also important at the enterprise architecture level. For example, “the business” imposes performance requirements on the applications and technical infrastructure, while the performance characteristics of systems influence the quantitative behaviour of business processes. The availability of global performance and cost estimates in the early architectural design stages can provide invaluable support for system design decisions, and prevent the need for expensive redesigns at later stages.

In this paper we present an approach for quantification and performance analysis of enterprise architectures. This approach is based on the propagation of quantitative input parameters and of calculated performance measures through a service-oriented architectural model. It complements existing detailed performance analysis techniques (e.g., queueing analysis), which can be plugged in to provide the performance results for the model elements.

2 State of the art in architecture performance analysis

As mentioned, enterprise architecture covers a wide 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 section, 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 how the different layers *interoperate*. Also from a quantitative perspective there is need for interoperability across layers: higher layers impose a workload on lower layers, while the performance characteristics of the lower layers directly influence the performance of the higher layers. However, techniques that cover quantitative analysis throughout this whole ‘stack’ hardly exist.

2.1 Infrastructure layer

Traditionally, approaches to performance evaluation of computer systems and communication systems [6] have a strong focus on the infrastructure domain. Queueing models, for example, describe the characteristics of the (hardware) resources in a system, while the workload imposed by the applications is captured by an abstract stochastic arrival process. Also, a lot of literature exists about performance studies of specific hardware configurations, sometimes extended to the system software and middleware level. Most of these approaches have in common that they are based on detailed models and require detailed input data.

2.2 Application layer

Performance engineering of software applications [18] is a much newer discipline compared to the traditional techniques described above. A number of papers consider performance of software *architectures* at a global level. Bosch and Grahn [1] present some observations about the performance characteristics of a number of often-occurring architectural styles. Performance issues in the context of the SAAM method [14] for scenario-based analysis are considered in [15].

Another direction of research address 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 [19] is restricted to a number of popular architectural styles (e.g., the distributed message passing style but not the pipe and filter style). In [3] queueing models are derived from UML 2.0 specifications which, however, in most cases do not have an analytical solution.

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 [7] are often advocated as a tool for compositional performance analysis. However, these approaches tend to be fairly computation-intensive, because they still suffer from a state space explosion.

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 or ExSpect (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 correctly interpret the results. Testbed Studio (http://www.bizzdesign.com/) offers, in addition to simulation, a number of analytical methods. They include completion time and critical path analysis of business processes [11] and queueing model analysis [13]. Petri nets (and several of its variations) are fairly popular in business process modelling, either to directly model processes or as a semantic foundation. 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). Business process analysis with stochastic Petri nets is the subject of, among others, [16].

3 The ArchiMate enterprise modelling language

Enterprise architecture refers to a consistent whole of principles, methods and models that are used in the design and realisation of organisational structure, business processes, information systems, and infrastructure. However, these domains are not approached in an integrated way, which makes it difficult to judge the effects of proposed changes. Every domain speaks its own language, draws its own models, and uses its own techniques and tools. Communication and decision making across domains is therefore seriously impaired.

In contrast to languages for models within a domain (e.g., the Unified Modelling Language, UML [17] for modelling applications and the technical infrastructure, or the Business Process Modelling Notation BPMN [2] for modelling business processes), a language for enterprise architecture should describe the elements of an enterprise at a relatively *high level of abstraction*, and should pay particular attention to the *relations* between these elements.

This line of thinking, towards integrated models, can also be recognised in the Model Driven Architecture (MDA) approach to software development [5]. MDA is a collection of standards of the Object Management Group (OMG) that raise the level of abstraction at which software solutions are specified. Recently, OMG has extended its focus to more business-oriented concepts and languages, to be developed within the MDA framework. These developments make MDA just as relevant for enterprise architecture as it is now for software development.

Following the principle of cross-domain integration, the ArchiMate project (http://archimate.telin.nl) concentrates on modelling, visualisation and analysis of architectures, providing architects with concepts, techniques and tools for architectural design. The development of an architecture language that picture various architectural domains and their relations forms the core of the project. Since the analysis technique we present in this paper was designed for ArchiMate models, we will first briefly introduce this modelling language.

As the basis for our analysis approach, we use a simplified version of the ArchiMate enterprise modelling language. Figure 1 shows an informal representation of the metamodel of this language with the main concepts and the relations that they may have. They cover the business layer of an enterprise (e.g., the organisational structure and business processes), the application layer (e.g., application components) and the technical infrastructure layer (e.g., devices and networks), as well as relation betweens the layers. The language considers the structural, behavioural and informational aspects within each layer. For a description of the full language, we refer to [12].

In order to allow for the quantitative analysis of models expressed in this language, *attributes* are added to quantify some of the concepts and relations. There can be attributes for both input parameters and analysis results, although the distinction may not always be sharp: the result of one analysis phase may be the input of a later analysis phase. In our approach we identify the specific quantitative attributes that we use for ArchiMate models.

4 Viewpoints on architecture performance

Architectures can be described from different viewpoints, which result in different views on architectural models [10]. These views are aimed at different *stakeholders* that have an interest in the modelled system. Also for the performance aspects of a system, a number of viewpoints can be discerned, resulting in different (but related) performance measures:

– **User/customer view** (stakeholders: customer; user of an application/system): *response time*, the time between issuing a request and receiving

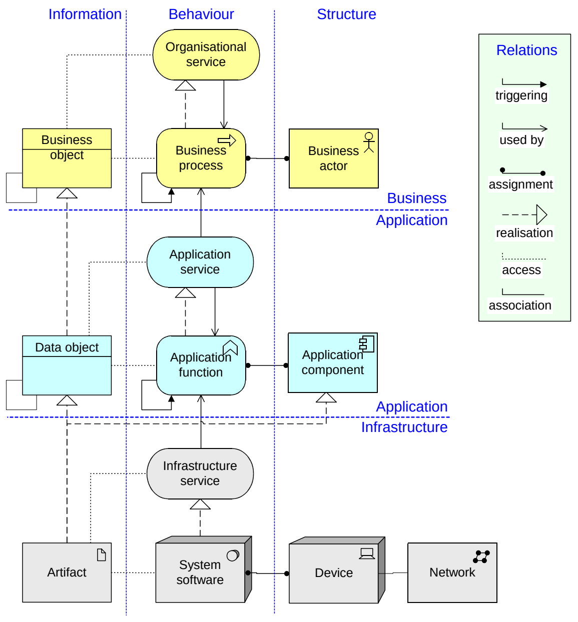

Fig. 1. The ArchiMate metamodel

the result; the response time is the sum of the processing time and waiting times (synchronisation losses).

– **Process view** (stakeholders: process owner; operational manager): *completion time*, 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).
– **Product view** (stakeholders: product manager; operational manager): *processing time*, the amount of time that actual work is performed on the realisation of a certain product or result: the response time without waiting times. This can be orders of magnitude lower than the response time.
– **System view** (stakeholders: system owner; system manager): *throughput*, the number of transactions/requests that are completed per time unit.
– **Resource view** (stakeholder: resource manager; capacity planner): *utilisation*, the percentage of the operational time that a resource is busy. On the one hand, the utilisation is a measure for 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.

This is a refinement of the views mentioned in, e.g.,, [8], which only discerns a user view and a system view. Figure 2 summarises the views on performance.

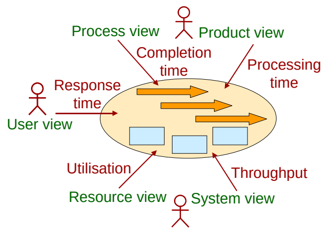

Fig. 2. Performance viewpoints

4.1 Integration of models and analysis results

As indicated in [12], one of the aims of the ArchiMate language is to provide a way to integrate detailed design models expressed in languages such as BPMN [2] for the business layer or UML [17] for the application and infrastructure layers. The ArchiMate model shows the global structure within each domain and the relations between the domains, and contains references to the design models for the details. Language and model integration also forms the basis of tool integration, as described in [21]. Since some modelling tools aimed at a specific domain also offer quantitative analysis capabilities, we ultimately also aim for the integration of quantitative results: in that case, the detailed analysis results are the input for quantitative analysis at the enterprise architecture level. A prerequisite for this is the compositionality of analysis results.

5 Analysis of enterprise architecture models

In this section we present our approach for the quantitative analysis of service-oriented models expressed in the ArchiMate language.

5.1 Model structure

The metamodel presented in Figure 1 shows that an architecture model displays some regular structure. Such a model may be viewed as a hierarchy of “layers”. Within the global metamodel layers we can distinguish layers of two types: *service layers* and *realisation layers*. A service layer exposes external functionality that can be used by other layers, while a realisation layer models the realisation of services in a service layer. Thus, we can separate the externally observable behaviour (expressed as services) from the complex internal organisation (contained within the realisation layers). Figure 6 shows an example of a layered view of an ArchiMate model. Looking at the horizontal structure of the metamodel, we notice that realisation layers basically contain three types of elements. They might model some pieces of *internal behaviour* (expressed as processes, functions or system software). Further, each behaviour element can access one ore more *objects* and it is assigned to exactly one *resource* (see Figure 3).

In a layered view, analysis across layers is possible by propagating quantities through the layers. A natural option for this is to first consider workload measures (such as arrival frequencies), that are imposed as a "demand" to the model elements from the layers that contain the users of the system (e.g., customers). These quantities propagate towards the deeper layers of the architecture, yielding the *demands* of each of the model elements. Once the workloads have been determined, we can determine the effort these workloads require from the resources (structural elements) and the behaviour elements. This effort can be expressed in terms of, e.g., performance measures (e.g., utilisations for resources, processing and response times for behaviour elements) or costs. From the 'deepest' layers of the models, these measures are propagated to the higher layers.

In summary, our analysis approach consists of the following two phases (see Figure 4): a *top-down* calculation and propagation of the workloads imposed by the top layer; this provides input for a *bottom-up* calculation and propagation of performance measures. In the rest of this section we will show how these phases of analysis can be realised in a systematic manner.

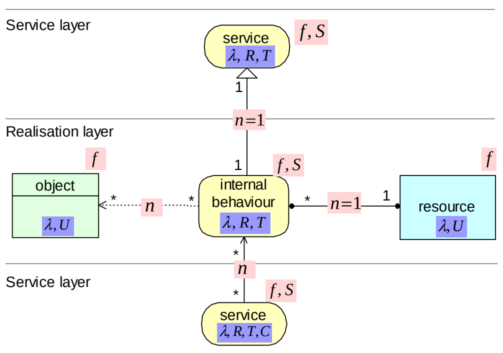

Fig. 3. Structural properties of ArchiMate models

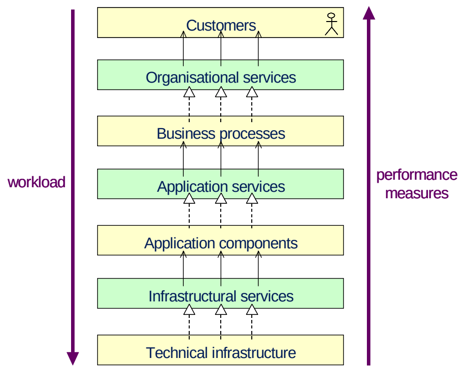

Fig. 4. Layers of ArchiMate models

5.2 Quantitative input

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: e.g., 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 may occur in practice. If 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, it often is very difficult to correctly interpret available numerical data, and to evaluate the reliability of the available data.

We assume that the following input is provided for analysis (see Figure 3):

- For any 'used by' and 'access' relation e a weight n_e, representing the average number of uses/accesses. For any 'realisation' and 'assignment' relation, we set $n_e = 1$: they represent a 1-to-1 mappings of a behaviour element to a service and to a resource, respectively.
- 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). Since a service represents the externally observable behaviour of a behaviour element a, we may assume that it inherits the service time from the behaviour element that realises it. Therefore, we leave the choice of specifying this input value for either one of these nodes.
- For any resource r a capacity C_r. By default $C_r = 1$.
- 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.

5.3 Quantitative results:

The goal of our approach is to determine the following performance measures (see Figure 3):

- the workload (arrival rate) λ_a for each node a. (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 a
- the utilisation U_r, for each resource r.

5.4 Analysis

To derive performance measures, given the inputs, we proceed in three steps:

1. We first "normalise" the model, using model transformations, in order to generate a model that conforms to the structure presented in Figure 3.
2. Top-down calculation of workloads (arrival rates) λ.
3. Bottom-up computation of performance measures T, U and R.

Step 1: Model normalisation. Typical ArchiMate models do often not fully conform to the ArchiMate metamodel. This is due to the fact abstraction rules may be used to create simplified views on the architecture. These abstractions have a formal basis in an operator that has been derived for the composition of relations (see [20] for the details). For instance, a 'realisation' relation with a consecutive used by' relation may be replaced by a new 'used by' relation that short-circuits a service.

The first step in our approach is a model transformation, deriving a normalised version of the input model. The normalised model conforms to the structure described in Figure 3. Since some concepts and relations are irrelevant for our approach, normalisation starts with eliminating them from the model. The remaining model will then be subjected to a series of transformations steps, an example of which is given in Figure 5. There is a limited set of transformation rules, which has been determined in a rather straightforward manner. The application of these rules eventually leads to the normalised model. Because model normalisation is not the primary focus of this paper, we omit a formal description of the normalisation algorithm.

The following two steps will be applied to the resulting normalised model.

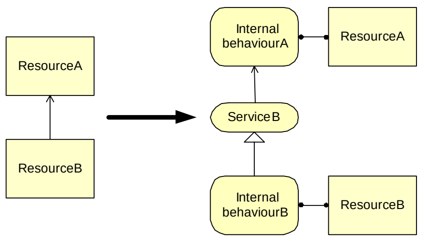

Fig. 5. Example of a normalisation step

Step 2: Top-down workload calculation. For a normalised model, we can calculate the arrival rate for any node a with the following recursive expression:

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

where d_a^+ denotes the out-degree of node a and k_i is a child 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.

Step 3: Bottom-up performance calculation. Once the workloads on the various model components have been calculated, we can proceed with the bottom-up calculation of the performance measures. The approach is similar to the top-down approach. We focus here on the bottom-up propagation of performance measures. The following recursive expressions apply:

– The utilisation of any resource r is $U_r = \frac{1}{C_r} \sum_{i=1}^{d_r} \lambda_{k_i} T_{k_i}$, 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 a service a coincide with these measures for the internal behaviour element realising it, i.e.: $T_a = T_k$ and $R_a = R_k$, where (k, a) is the *realisation* relation with a as end point. (The service merely exposes the functionality of the internal behaviour element to the environment: there is no additional time consumption).
– The processing time and response time of an internal behaviour element a is computed using the following recursive formulas:

$$T_a = S_a + \sum_{i=1}^{d_a^-} n_{k_i,a} R_{k_i} \qquad\qquad R_a = F(a, r_a) \tag{2}$$

where d_a^- denotes the in-degree of node a, k_i is a parent of a and 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 [6], this function is

$$F(a, r_a) = \frac{T_a}{1 - U_{r_a}} \tag{3}$$

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:

$$R_a = T_a + \frac{T_a}{1 - U_{r_a}} \cdot \frac{(C_{r_a} U_{r_a})^{C_{r_a}}}{(C_{r_a} U_{r_a})^{C_{r_a}} + C_{r_a}!(1 - U_{r_a}) \sum_{j=0}^{C_{r_a}-1} \frac{(C_{r_a} U_{r_a})^j}{j!}} \tag{4}$$

In most cases, this will lead to approximate results because the queueing networks are not separable [6]. At the enterprise architecture level, where we are generally interested in global performance estimates, we expect such approximations to be good enough. We might even consider more global solutions, e.g. operational performance bounds [4]. In case more precise results would be required, instead of simple queueing formulas, more detailed techniques such as simulation can be applied in combination with our approach.

6 Example

In this section we give an example to illustrate the analysis approach described in the previous section. Consider 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. We show how performance measures of this system, including resource utilisations and response times, can be derived using a model of the architecture of the system (see Figure 6). This model covers the whole stack from business processes and actors, through applications, to the technical infrastructure.

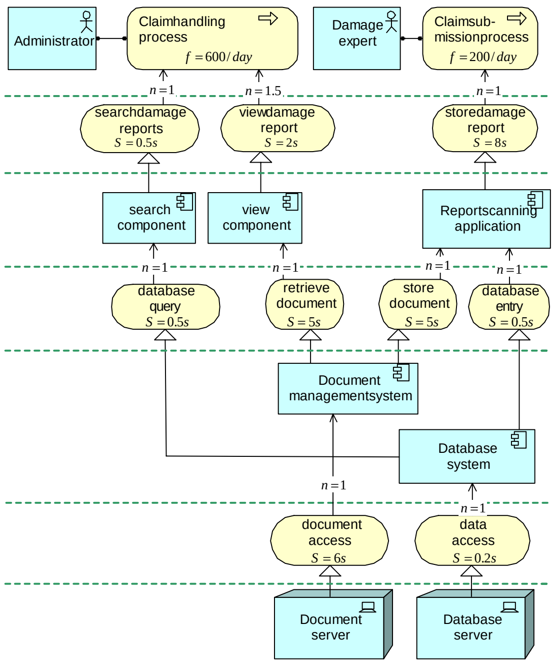

Fig. 6. Document management example

There are three applications offering services that are used directly by the business actors. The Administrator can *search* in the metadata database, resulting in a 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 *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 analysis inputs. On the 'used by' relations, we specify workloads, in terms of the average number of uses n of the corresponding service. For the business processes, an arrival frequency f is specified. Finally, for services we may specify a service time S. We now proceed to analyse this model, using the three steps described in the previous section.

Step 1: Model normalisation. We first derive a normalised model that is compliant with the modelling rules described in Figure 3. Figure 7 shows the normalised version of the model in Figure 6. 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.

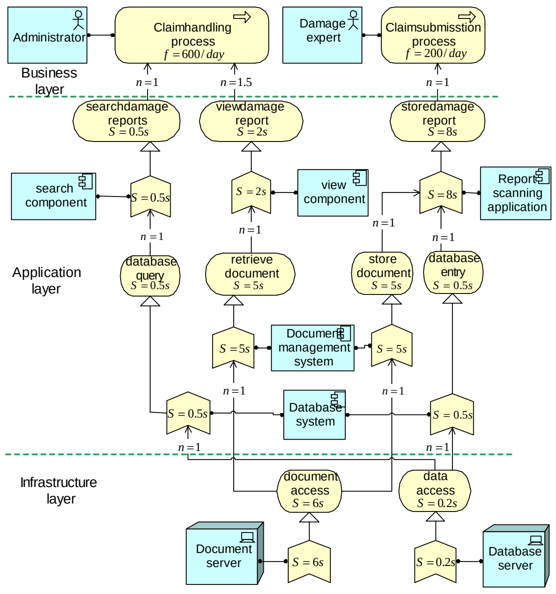

Fig. 7. Normalised model

Step 2: Top-down workload analysis. Table 1 shows the workload for the services s in the model, in terms of the arrival rates λ_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).

Step 3: Bottom-up performance analysis. Table 1 shows the performance results for the example model, i.e., the processing and response times for the services and the utilisations for the resources at the application and infrastructure layer (we assume that 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).

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 [6]. The response time function is then given in equation (3).

Resource (r)	Service (s)	λ_s (sec^{-1})	T_s (sec)	R_s (sec)	U_a
Doc. srv.	doc. acc.	0.0382	6.0	7.8	0.229
DB srv.	data acc.	0.0278	0.2	0.2	0.006
Doc.mgt. sys.	retr. doc.	0.0313	12.8	25.0	0.488
Doc.mgt. sys.	store doc.	0.0069	12.8	25.0	0.488
DB syst.	DB query	0.0278	0.7	0.7	0.019
DB syst.	DB entry	0.0069	0.7	0.7	0.019
Search comp.	search rep.	0.0278	1.2	1.2	0.025
View comp.	view rep.	0.0313	27.0	174.0	0.843
Rep. scanning	store rep.	0.0069	33.7	44.0	0.234

Table 1. Workloads and performance results

The results show that queueing times from the lower layers accumulate in the higher layers, resulting in response times that are orders of magnitude greater than the local service times. E.g., 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, Figure 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, e.g., if an extra View component is needed.

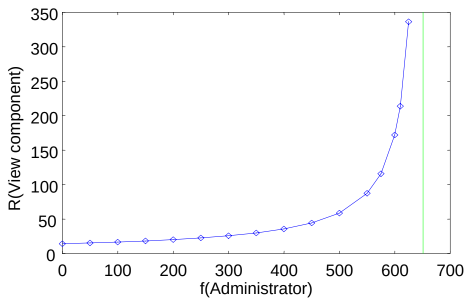

Fig. 8. Arrival rate vs. response time

7 Conclusions and future work

Although the importance of enterprise architecture modelling has been recognised, hardly any attention has been paid to the analysis of their quantitative properties. Most existing approaches to performance evaluation focus on detailed models within a specific domain. In this paper we demonstrated the applicability of quantitative modelling and analysis techniques for the effective evaluation of design choices at the enterprise architectures level. 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 an enterprise architecture model. This can be used as an analysis framework where existing methods for detailed performance analysis, based on, e.g., queueing 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 queueing formula for the response times, we showed that queueing times from the lower layers of the architecture accumulate in the higher layers, which results in response times that are orders of magnitude greater than the local service times. A prototype has been developed for further illustration and validation of the approach.

Several improvements and extensions to the approach are conceivable. For example, in [9] we show that our "vertical" approach to calculate and propagate workloads and performance measures could also be combined with "horizontal" analysis techniques to evaluate, e.g., critical paths and completion times in business processes [11]. Finally, for a further integration of architecture design process, combining quantitative analysis with functional analysis approaches (e.g., "impact of change" analysis based on quantitative results) or visualisation techniques could be fruitful.

Acknowledgement

This paper results from the ArchiMate project (http://archimate.telin.nl). The ArchiMate consortium consists of ABN AMRO, Stichting Pensioenfonds ABP, the Dutch Tax and Customs Administration, Ordina, Telematica Instituut, Centrum voor Wiskunde en Informatica, Radboud Universiteit Nijmegen, and the Leiden Institute of Advanced Computer Science.We would like to thank René van Buuren, Timber Haaker and Hugo ter Doest for their useful remarks that helped to improve our approach.

References

1. J. Bosch and H. Grahn. Characterising the performance of three architectural styles. In *Proceedings First International Workshop on Software and Performance*, Santa Fe, NM, Oct. 1998.
2. Business Process Management Initiative. Business process modeling notation. working draft (1.0), Aug. 2003.
3. A. Di Marco and P. Inverardi. Compositional generation of software architecture performance QN models. In J. Magee, C. Szyperski, and J. Bosch, editors, *Proceedings Fourth Working IEEE/IFIP Conference on Software Architecture*, pages 37–46, Oslo, Norway, June 2004.

4. D. Eager and K. Sevcik. Bound hierarchies for multiple-class queueing networks. *Journal of the ACM*, 33:179–206, Jan. 1986.
5. D. Frankel. *Model Driven Architecture : Applying MDA to Enterprise Computing.* John Wiley & Sons, 2003.
6. P. Harrison and N. Patel. *Performance Modelling of Communication Networks and Computer Architectures.* Addison-Wesley, 1992.
7. H. Hermanns, U. Herzog, and J.-P. Katoen. Process algebra for performance evaluation. *Theoretical Computer Science*, 274(1–2):43–87, Mar. 2002.
8. U. Herzog. Formal methods for performance evaluation. In *Lectures on Formal Methods and Performance Analysis: First EEF/Euro Summer School on Trends in Computer Science (LNCS 2090)*, pages 1–37. Springe Verlag, 2001.
9. M.-E. Iacob and H. Jonkers. Quantitative analysis of enterprise architectures. Technical Report ArchiMate D3.5b, Telematica Instituut, Enschede, the Netherlands, Mar. 2004.
10. IEEE Computer Society. IEEE standard 1471-2000: Recommended practice for architectural description of software-intensive systems, 2000.
11. H. Jonkers, P. Boekhoudt, M. Rougoor, and E. Wierstra. Completion time and critical path analysis for the optimisation of business process models. In M. Obaidat, A. Nisanci, and B. Sadoun, editors, *Proceedings of the 1999 Summer Computer Simulation Conference*, pages 222–229, Chicago, IL, July 1999.
12. H. Jonkers, M. Lankhorst, R. van Buuren, S. Hoppenbrouwers, M. Bonsangue, and L. van der Torre. Concepts for modelling enterprise architectures. *International Journal of Cooperative Information Systems*, 13(3), Sept. 2004.
13. H. Jonkers and M. van Swelm. Queueing analysis to support distributed system design. In *Proceedings of the 1999 Symposium on Performance Evaluation of Computer and Telecommunication Systems*, pages 300–307, Chicago, IL, July 1999. M.S. Obaidat and M. Ajmone Marsan.
14. R. Kazman, L. Bass, G. Abowd, and M. Webb. SAAM: A method for analyzing the properties of software architectures. In *Proceedings 16th International Conference on Software Engineering*, pages 81–90, Sorento, Italy, 1994.
15. C.-H. Lung, A. Jalnapurkar, and A. El-Rayess. Performance-oriented software architecture analysis: An experience report. In *Proceedings First International Workshop on Software and Performance*, Santa Fe, NM, Oct. 1998.
16. A. Schömig and H. Rau. A petri net approach for the performance analysis of business processes. Technical Report 116, Lehrstuhl für Informatik III, Universität Würzburg, 1995.
17. K. Scott. *Fast Track UML 2.0.* Apress, 2004.
18. C. Smith. *Performance Engineering of Software Systems.* Addison-Wesley, 1990.
19. B. Spitznagel and D. Garlan. Architecture-based performance analysis. In *Proceedings 1998 Conference on Software Engineering and Knowledge Engineering*, San Francisco Bay, June 1998.
20. R. van Buuren, H. Jonkers, M.-E. Iacob, and P. Strating. Composition of relations in enterprise architcture models. In H. Ehrig, G. Engels, F. Parisi-Presicce, and G. Rozenberg, editors, *Graph Transformations – Proceedings of the Second International Conference (LNCS 3256)*, pages 39–53, Rome, Italy, Sept. 2004.
21. D. van Leeuwen, H. ter Doest, and M. Lankhorst. A tool integration workbench for enterprise architecture. In *Proceedings 6th International Conference on Enterprise Information Systems*, Porto, Portugal, Apr. 2004.