Analysis of *Enterprise Architecture* Models Using Performance Quantitative Analysis Technique

Murnawan[a], Ros Sumiati[b]

[a]*Faculty of Engineering, Widyatama University, Indonesia*
E-mail : murnawan@widyatama.ac.id, ros.sumiati@gmail.com

ABSTRACT

To create such an integrated perspective of enterprise architecture, we need both a description technique for architectural models and model-based analysis techniques to realize this global optimization in practice. However, the value of architecture models increases significantly if they can also be used to support the decision making process. In this paper we argue that whenever a change in the enterprise architecture is needed, model-based analysis plays a central role.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.

Keywords: *enterprise architecture, quantitative analysis, workload parameter, performance.*

1. INTRODUCTION

The primary reason for developing enterprise architecture is to support the business by providing the fundamental technology and process structure for an information technology (IT) strategy. Further, it details the structure and relationships of the enterprise, its business models, the way an organization will work, and how and in what way information, information systems, and technology will support the organization's business objectives and goals. This makes IT a responsive asset for a successful modern business strategy. Constructing architectures for an enterprise may help to, among others, increase the insight and overview required to successfully align the business and IT.

To detailed design models within domains of architecture, 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. 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, which can be plugged in to provide the performance results for the model elements.

2. 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:

a. **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.
b. **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.

c. **Product view** (stakeholders: product manager; operational manager):
Processing time is the amount of time that actual work is performed on the realization 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
d. **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.
e. **Resource view** (stakeholders: resource manager; capacity planner):
Utilization is the percentage of the operational time that a resource is busy. On the one hand, the utilization is a measure of the effectiveness with which a resource is used. On the other hand, a high utilization 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 utilization can be used as a more or less objective measure for work stress. In information systems architectures, a typical example of the utilization is the network load.

Performance measures belonging to the different views are interrelated, and may be in conflict when trying to optimize the performance of a system. For example, a higher throughput leads to a higher resource utilization, which may be favorable from a resource manager's point of view; however, this generally leads to an increase in the response times, which is unfavorable from a user's point of view. Therefore, when aiming to optimize the performance of a system, it is important to have a clear picture of which performance measures should be optimized.

3. 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 analyzed 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, as well as in a quantitative sense [1].

3.1 Infrastructure Layer

Traditionally, approaches to performance evaluation of computer systems and communication systems 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 [2]. 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.

3.2 Application Layer

Performance engineering of software applications is a much newer discipline compared to the traditional techniques described above [3]. A number of papers consider the performance of software architectures at a global level. Bosch

and Grahn present some first observations about the performance characteristics of a number of often occurring architectural styles [4].

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 is restricted to a number of popular architectural styles (e.g., the distributed message passing style but not the pipe and filter style) [5]. In Di Marco and Inverardi queuing models are derived from UML 2 specifications, which in most cases, however, do not have an analytical solution [6].

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 are often advocated as a tool for compositional performance analysis [7]. 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.

3.3 Business Layer

Several business process modeling tools provide support for quantitative analysis through discrete-event simulation. Also, general-purpose simulation tool, e.g., 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 [8] offers, in addition to simulation, a number of analytical methods. They include completion time and critical path analysis of business processes [9] and queuing model analysis [10]. Petri nets (and several of its variations) are fairly popular in business process modeling, either to directly model processes or as a semantic foundation. They offer possibilities for performance analysis based on simulation, 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 [11].

4. QUANTITATIVE MODELING

In this section we present our approach for the quantitative modeling 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 modeling 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.

4.1 Model Structure

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 modeler to formulate and describe clearly the problem being analyzed.

For each meta-model layer we can distinguish one or more model layers of two types: service layers and realization layers. A service layer exposes functionality that can be ‘used by’ the next higher layer, while a realization layer models shows how the consecutive service layer is ‘realized’. The number of these layers is not fixed, but a natural layering of an ArchiMate model will contain the succession of layers depicted in Figure 2.

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

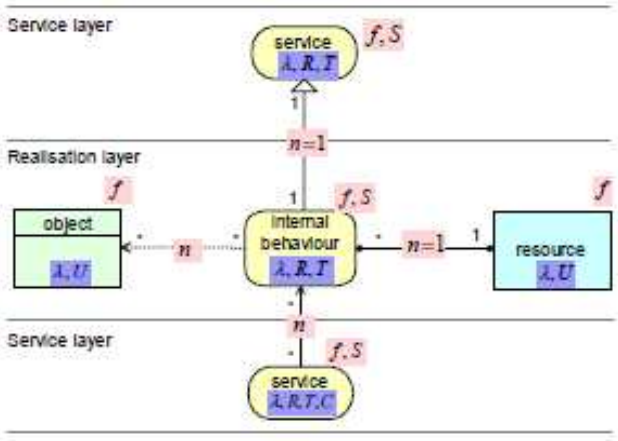

Figure 1: Structural properties of ArchiMate models

Thus we can summaries our findings in terms of the 'analysis meta-model' depicted in Figure 2, where

a. 'object' can be a business object, a data object, or an artifact;
b. 'resource' can be a business role, a business actor, an application component, a system software component, a node, or a device;
c. 'internal behavior' can be a business process, a business function, an application function, or an infrastructure function;
d. 'service' can be a business service, an application service, or an infrastructure service.

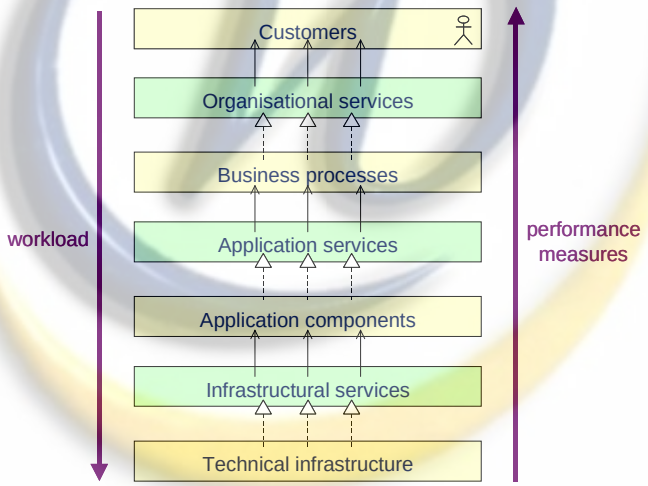

Figure 2: Layers of ArchiMate models

4.2 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 organizations, 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 organization 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 Figure 1.):

a. For any 'used by' and 'access' relation e, a weight n_e , representing the average number of uses and accesses.

b. For any behavior element *a*, a service time S_a , representing the time spent internally for the realization of a service (excluding the time spent waiting for supporting services). We assume that a service inherits the service time value of the element realizing it.
c. For any resource *r*, a capacity C_r .
d. 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.

4.2.1 Example

To show the practical use of this analysis technique, we illustrate our approach with the following simple example. Suppose we want to analyze an online store that process about sales of an online store. A customer can place one or more orders from an online store. We assume that a customer's order can have multiple items with varying quantities. A model of this system is depicted in Figure 3. This model covers the whole stack from business processes and actors, through applications, to the technical infrastructure.

The online operator can search in the metadata database, resulting in short descriptions of the order that meet the query and view orders that are returned by a search. In addition to the applications that are used directly by the end user, there are one supporting application components: a database access component, providing access to the metadata database. Finally, the model shows the physical devices of which the database access components make use.

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*.

4.3 Quantitative Results

The goal of our approach is to determine the following performance measures (see Figure 1.):
a. The workload (arrival rate) λ_a for each node *a* (note that, provided that no resources are overloaded, the throughput for each node is equal to its arrival rate);
b. The processing time T_a and the response time R_a , for each behavior element or service;
c. The utilization U_r , for each resource *r*.

5. QUANTITATIVE ANALYSIS TECHNIQUE

There are three steps to derive performance measures with given the inputs:
a. We will first 'normalize' any input model, using model transformations, in order to generate a model that is compliant with the structure presented in Figure 1.
b. Top-down calculation of workloads (arrival rates) .
c. Bottom-up computation of performance measures *T*, *U*, and *R*.

5.1 Model Normalization.

Typical ArchiMate models do often not display the regular structure ArchiMate meta-model. 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. For instance, a 'realization' 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 normalized version of the input model which conforms to the structure described in Figure 1. Since some concepts and relations are irrelevant for our approach, normalization starts with eliminating them from the model. Then the model will be subjected to a series of transformations steps, an example of which is given in Figure 4. There is a limited set of transformation rules, eventually resulting in the normalized model. Because model normalization is not the primary focus of this paper, we omit a formal description of the normalization algorithm.

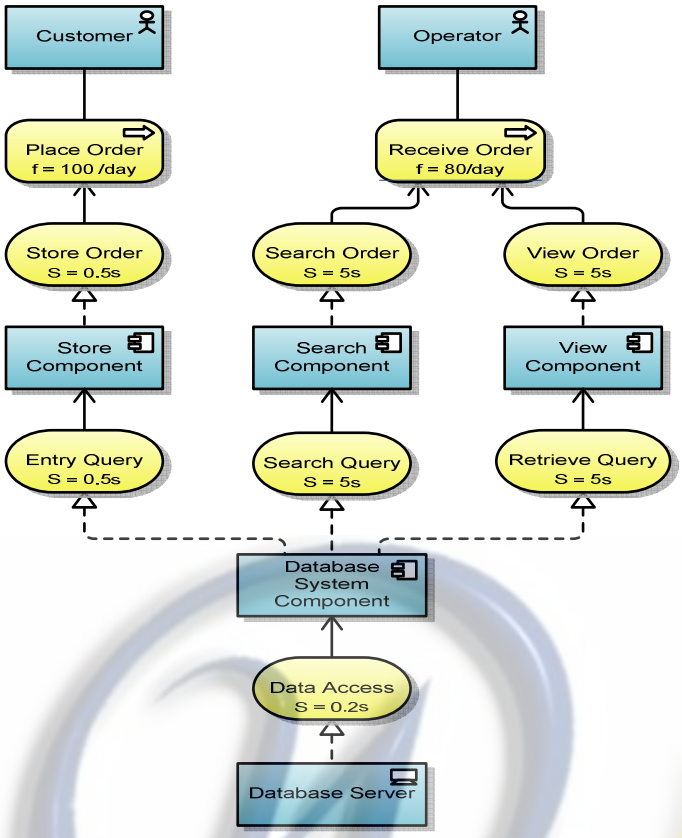

Figure 3: Online Store Order Example

Figure 5 shows the normalized version of the example model given in Figure 3. 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 behavior elements.

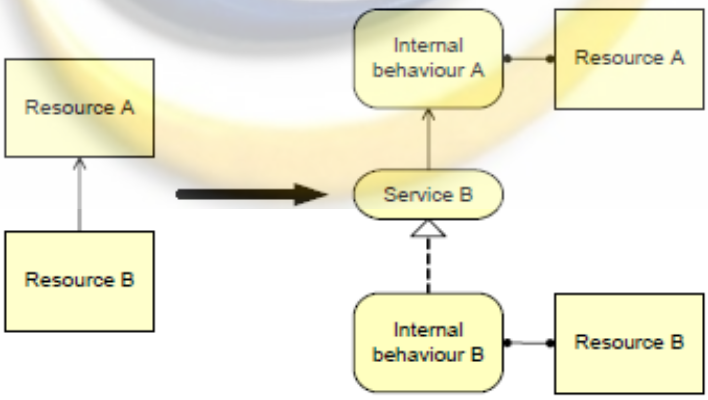

Figure 4: Example of a normalization step

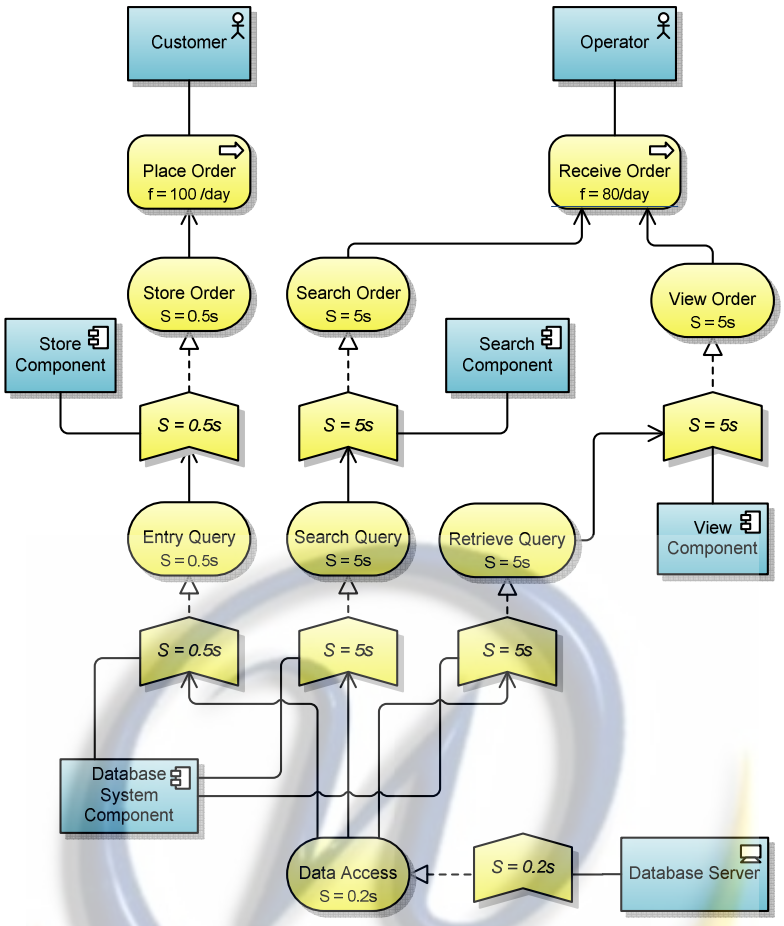

Figure 5: Normalized model.

5.2 Top-Down Workload Calculation

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

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

where d_a^+ denotes the out-degree of node *a* and *ki* 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.
Table 1 shows the workload for the services *s* in the model, in terms of the arrival rates λs. 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).

5.3 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:

a. The utilization of any resource *r* is

$$U_r = \frac{\sum_{i=1}^{d_r} \lambda_{k_i} T_{k_i}}{C_r}, \qquad (2)$$

where d_r is the number of internal behavior 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 behavior element realizing it, i.e., $T_a = T_k$ and $R_a = R_k$, where (k, a) is the only realization relation having *a* as end point.

b. The processing time and response time of any internal behavior element *a* can be computed using the following recursive equations:

$$T_a = S_a + \sum_{i=1}^{d_a^-} n_{k_i,a} R_{k_i} \text{ and } R_a = F(a, r_a), \quad (3)$$

where d_a^- denotes the in-degree of node *a*, *ki* is a parent of *a*, *a r* is the resource assigned to *a*, and *F* is the response time expressed as a function of attributes of *a* and *a r* .
For example, if we assume that the node can be modeled as an M/M/1 queue (Harrison and Patel 1992), this function is

$$F(a, r_a) = \frac{T_a}{1 - U_{r_a}} \quad (4)$$

We can replace this by another equation in case other assumptions apply, e.g., the Pollaczek–Khinchine formula for an M/G/1 if *Ta* 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 [1]. We might also consider more global solutions, e.g., operational performance bounds. 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 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 utilizations 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.

Table 1: Performance Result

Resource (*r*)	Service (*s*)	Workload (λ_s)	Proc. Time (T_s)	Response Time (R_s)	Utilization (U_r)
Database Server	Data Access	0.0045	0.2	0.2	0.09
DB Sys. Comp.	Entry Query	0.0015	0.7	0.8	0.105
DB Sys. Comp.	Search Query	0.01	5.2	1.2	5.2
DB Sys. Comp.	Retrieve Query	0.01	5.2	1.2	5.2
Store Component	Store Order	0.003	1.2	1.9	0.36
Search Component	Search Order	0.002	10.2	9.8	2.04
View Component	View Order	0.002	10.2	9.8	2.04

6. CONCLUSION

In this paper we have considered the relation between enterprise architecture models and architecture analysis. Although the importance of enterprise architecture modeling has been recognized, 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 modeling 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 optimization 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., 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 online store system. 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 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.

REFERENCES

[1] Jonkers H, Iacob M-E, Performance and Cost Analysis of Service-Oriented Enterprise Architectures. In A. Gunasekaran (ed.), Global Implications of Modern Enterprise Information Systems: Technologies and Applications, IGI Global. 2009
[2] Harrison P, Patel N, Performance Modelling of Communication Networks and Computer Architectures. Addison-Wesley, Reading, Massachusetts, 1992.
[3] Smith C, Performance Engineering of Software Systems. Addison-Wesley, Reading, Massachusetts, 1990.
[4] Bosch J, Grahn H , Characterising the Performance of Three Architectural Styles. Proc. First International Workshop on Software and Performance, Santa Fe, New Mexico, 1998.
[5] Spitznagel B, Garlan D, Architecture-based Performance Analysis. Proc. 1998 Conference on Software Engineering and Knowledge Engineering, San Francisco Bay, 1998.
[6] Di Marco A, Inverardi P, Compositional Generation of Software Architecture Performance QN Models. In Magee J, Szyperski C, Bosch J (eds.), Proc. Fourth Working IEEE/IFIP Conference on Software Architecture Oslo, Norway, 2004, pp. 37–46.
[7] Hermanns H, Herzog U, Katoen J-P (2002), Process Algebra for Performance Evaluation, Theoretical Computer Science, 2002, pp. 43–87.
[8] BiZZdesign B.V. http://www.bizzdesign.nl.
[9] 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, Chicago, IL, July 1999, pp. 222–229,
[10] 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, Chicago, IL, July 1999, pp. 300–307.
[11] A. Schomig and H. Rau. A petri net approach for the performance analysis of business processes. Technical Report 116, Lehrstuhl f¨ur Informatik III, Universit ¨at W¨urzburg, 1995.