Performance Prediction of Service-Oriented Systems with Layered Queueing Networks

Mirco Tribastone, Philip Mayer, and Martin Wirsing

Institut für Informatik
Ludwig-Maximilians-Universität München, Germany
{tribastone,mayer,wirsing}@pst.ifi.lmu.de

Abstract. We present a method for the prediction of the performance of a service-oriented architecture during its early stage of development. The system under scrutiny is modelled with the UML and two profiles: UML4SOA for specifying the functional behaviour, and MARTE for the non-functional performance-related characterisation. By means of a case study, we show how such a model can be interpreted as a layered queueing network. This target technique has the advantage to employ as constituent blocks entities, such as threads and processors, which arise very frequently in real deployment scenarios. Furthermore, the analytical methods for the solution of the performance model scale very well with increasing problem sizes, making it possible to efficiently evaluate the behaviour of large-scale systems.

1 Introduction

Service-oriented architectures (SOAs) pose challenging problems regarding the evaluation of their performance. Approaches based on field measurements are problematic when systems distributed on a large scale are to be assessed. In many cases, parts of the system are not directly accessible to the engineer, perhaps because they employ third-party services. Even if the whole system was accessible, profiling may turn out to be an unduly costly exercise. However, during early stages of the development process, the engineer may content herself with some, perhaps approximate and less expensive, prediction of the performance of the system. A predictive model is often expressed as a mathematical problem. This has the advantage that one can easily tune its parameters so as to carry out analyses such as *capacity planning*, i.e., optimising the amount of processing capacity to satisfy some assigned quality-of-service guarantees.

This is the topic addressed in this paper. Specifically, we are concerned with situations which employ model-driven development techniques for the specification of the functional behaviour of SOAs. We consider a system modelled with UML4SOA [14], a UML profile which allows behavioural specifications of services and service orchestrations by using activities and composite structure descriptions, modelling the interconnections between services. The model is augmented with elements that specify the (intended or predicted) performance characteristics of the system. To this end, we adopt MARTE [16], another UML profile

T. Margaria and B. Steffen (Eds.): ISoLA 2010, Part II, LNCS 6416, pp. 51–65, 2010.
© Springer-Verlag Berlin Heidelberg 2010

which extends behavioural UML specifications by adding timing properties. Furthermore, the model is accompanied by a deployment specification which emphasises the processing capacity of the computing platform on which the SOA is run. In this manner, a UML4SOA model becomes amenable to translation into a performance model as a layered queueing network (LQN) [7].

The motivation for the choice of LQNs is twofold. First, the LQN model features basic elements which have semantics close to corresponding elements of UML activities and behaviours in general. Indeed, research pursued in this direction has led to an abstract model of UML behaviours which can be used to automate the process of extracting an underlying LQN performance model [18]. Second, the analytical methods available for LQNs are very scalable with respect to increasing problem sizes. This makes this approach particularly convenient when the modeller wishes to predict the performance of large-scale SOAs, whose analysis would be otherwise computationally difficult when using approaches such as simulation or discrete-state models with explicit state-space enumeration.

The approach taken for the extraction of the LQN model is discussed by means of a case study and a numerical evaluation gives examples of the kinds of indices of performance that can obtained and how those can be interpreted in terms of the elements of the UML4SOA model.

Related Work. The general line of research followed by the present work is that on early-stage prediction of performance characteristics of software systems (see [1] for an excellent survey), with focus on designs of SOAs within the context of the EU Sensoria project [12]. In particular, it is most closely related to [8], where UML4SOA was translated into the stochastic process algebra PEPA [10]. In that paper, the profile for MARTE was also used for performance annotation although the translation did not take into account deployment information. In effect, the resulting model was based on an *infinite-server* assumption, i.e., it was assumed that the system had as much processing capacity as it required. In this context, delays were only due to the clients contenting for a limited number of software threads. Conversely, the translation proposed here does model processing capacity explicitly — in fact its crucial role in the overall performance of the system will be exemplified in the numerical evaluation of the case study. LQNs were also considered in [11] as the target performance description of the Palladio Component Model [4].

Paper Organisation. Section 2 gives a brief overview of UML4SOA and Section 3 discusses the case study. The main concepts of the LQN model are overviewed in Section 4. Section 5 illustrates the translation of UML4SOA into LQN and Section 6 provides an example of a numerical evaluation of the case study. Finally, Secion 7 concludes the paper with pointers for future research.

2 Modelling Services in UML4SOA

The Unified Modelling Language (UML) [15] is a well-known and mature language for modelling software systems with support ranging from requirement

modelling to structural overviews of a system down to behavioural specifications of individual components. However, the UML has been designed with object-oriented systems in mind, thus native support and top-level constructs for service-oriented computing such as participants in a SOA, modelling service communication, and compensation support are missing. As a consequence, modelling SOA systems with plain UML requires the introduction of technical helper constructs, which degrade usability and readability of the models.

Adding service functionality to UML is currently under investigation in both academia and industry. Static aspects of service architectures are addressed in SoaML [17], an upcoming standard of the OMG for the specification of service-oriented architectures. For describing the behaviour of services, we have introduced the UML4SOA profile [14] which allows the description of service behaviour in the form of specialised UML activities.

UML4SOA is built on top of the Meta Object Facility (MOF) metamodel and is defined as a conservative extension of the UML2 metamodel. For the new elements of this metamodel, a UML profile is created using the extension mechanisms provided by the UML. The principle followed is that of minimal extension, i.e. to use UML constructs wherever possible and only define new model elements for specific service-oriented features and patterns, thus increasing readability and conciseness of the resulting models and diagrams.

The core of the UML4SOA profile considered in this paper is based on the the following two concepts:

Communication Primitives. UML4SOA extends the classic UML communication actions with four specialised actions for service communication: «send», «receive», «send&receive», and «reply». As the names suggest, these actions are used to model sending a call to a remote service, receiving a call from a remote service, performing both in one step, and replying to a previous call from a remote service, respectively. Specialised pins may be added to each action. Most importantly, the link (lnk) pin indicates the remote service (or more specifically, the port where the service is attached) the current action is targeted at. The send (snd) and receive (rcv) pins indicate from which variables or variable contents to retrieve, or in which variable to place data sent or received.

Compensation. Services are often used to implemented long-running transactions, such as (parts of) business processes. Compensation is a mechanism for undoing *successfully completed work* of a service if, in the course of later transactions, an error occurs and the complete process must be rolled back. UML4SOA lifts the specification and invocation of compensation handlers up to a first-level entity of UML activities. Instead of using standard activities and structured activity nodes, UML4SOA introduces the concept of a «serviceActivity» to which handlers can be attached using edges; in the case of compensation, of a «compensationEdge». A compensation handler can be invoked with the new actions «compensate» and «compensateAll».

In this paper, the UML4SOA extensions to the UML are used to model the mobile payment case study in the next section. In order to keep the example

small and simple, only the first of the above-mentioned three core features is used (communication primitives). For more information on UML4SOA, the interested reader is referred to the UML4SOA specification [13].

3 Mobile Payment Case Study

The case study in this paper is taken from the domain of financial transactions. It models a mobile payment gateway which allows customers to pay with their mobile communication device, such as a phone. The system is implemented using a collection of services which interact with one another to complete a payment request from a customer. The architecture of the system is shown in Figure 1.

The main service is the MobilePaymentGateway shown in black. A client (MobileDevice) can use this service to perform a payment operation. The gateway first authenticates the customer using the AuthenticationService and, if the authentication is successful, performs the payment using the PaymentService. The last two services use additional services to fulfill their tasks.

Three of the services present in the system are actually orchestrations of services. These services have been modelled using UML4SOA as shown in Figure 2; from left to right, the modelled services are the MobilePaymentGateway, the AuthenticationService, and the PaymentService.

Mobile Payment Gateway. As the name suggests, the gateway is the main entrance point for the customer to the payment service. An instance of this service is started when a paymentRequest call from the customer is received; the link pin indicates that the call must come through the the device port. In the receive pin, the call payload target is specified. In this case, the data attached to the call is to be placed in the variable payment.

The gateway proceeds to authenticating the customer by using the AuthenticationService which is attached to the authService port. If authentication fails, an error is returned to the customer. Otherwise, the payment is delegated to

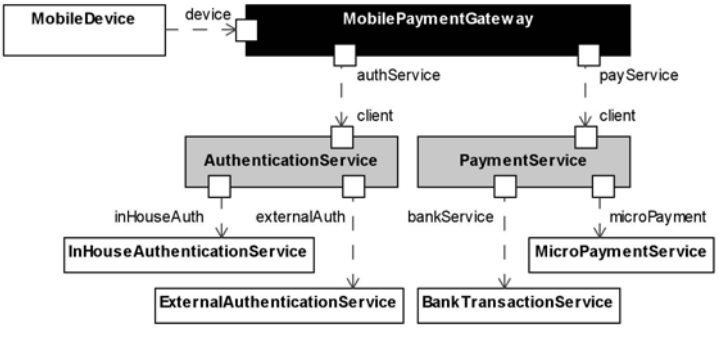


Fig. 1. Architecture of the Case Study

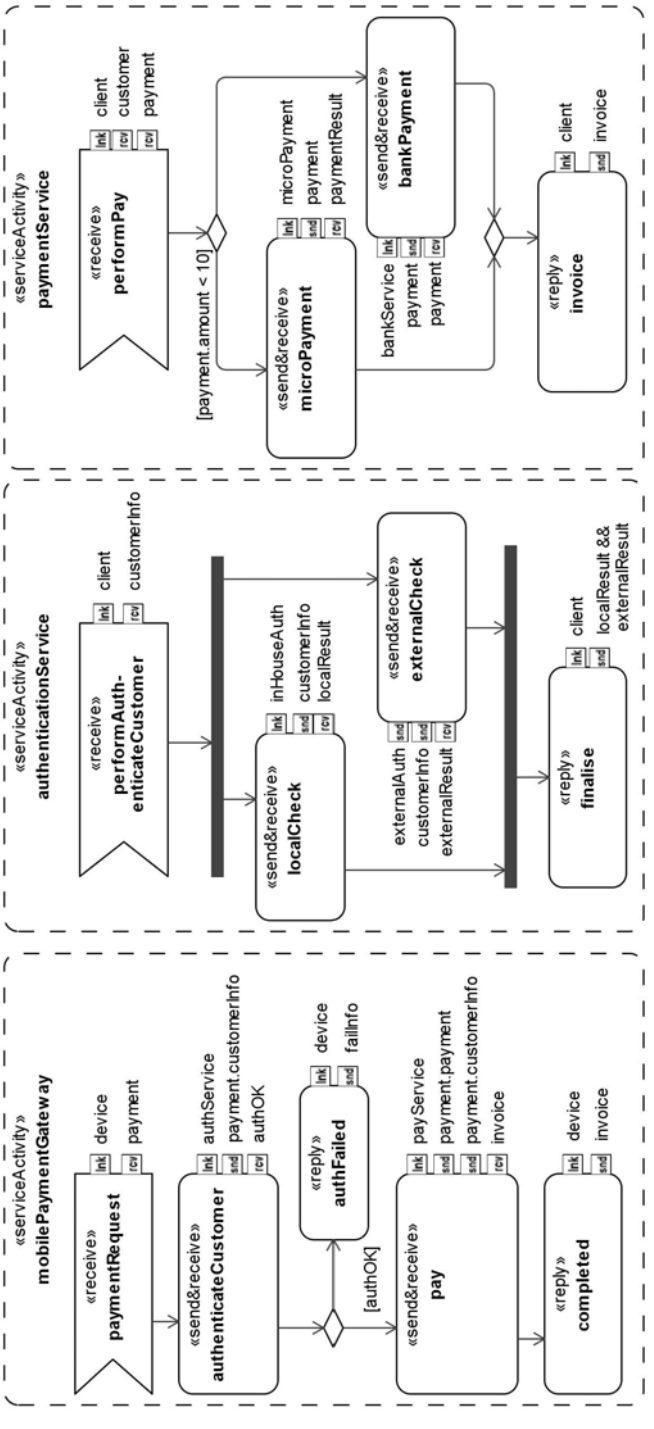


Fig. 2. Behaviour of the three orchestrations in the Payment Case Study

the PaymentService attached to the payService port: This service requires both the customer and the payment information. Finally, the result of the payment operation is returned to the client.

Authentication Service. The authentication service attempts to authenticate the customer with the system to ensure that the customer is allowed to use the service. For maximum security, both an in-house and an external authentication service are contacted in parallel: The result is only positive if both services agree. Note that the client port used in the link pins of the first and last method refers to the mobile gateway service on the left.

Payment Service. The payment service is invoked last, and tries to withdraw the appropriate amount of money from the customer's account. Depending on the amount, two different services are used. If the amount is less than 10, a micro-payment service is used, which aggregates several small payments until the customer is billed. If the amount is larger than 10, the amount is directly drawn from the customer's account via the bank. The micro-payment service has the advantage of performing faster, but cannot be used for large payments.

Summarising, the mobile payment gateway enables customers to perform a payment operation. The gateway and its invoked services have to consider a series of constraints to carry out this task. Overall, seven services and one client take part in this SOA.

4 The Layered Queueing Model

The Layered Queueing Network (LQN) model is an extension of classical queueing networks (e.g., [9,3]). Its primitives are entities that re commonly present in distributed computing systems, such as multi-threaded servers, multi-processor hardware, and inter-process communication via synchronous and asynchronous messages. The model can be solved via stochastic simulation, or more efficiently through approximate techniques which are known to be accurate (i.e., usually within 5%) and scalable with increasing problem sizes (e.g., [5,2]). The analytical methods will be preferred over stochastic simulation in the numerical evaluation conducted in Section 6. In general, they seem more appropriate when evaluating large-scale models such as SOAs. The remainder of this section gives an informal and brief overview of the LQN model, with particular emphasis on the notions that will be employed to analyse the case study presented in this paper. This description makes references to the graphical notation for LQNs. The reader may wish to consult Figure 5 for an example of such a representation. For more details on the LQN model, in addition to [7], the interested reader is referred to [6] which provides a tutorial and documents the functionality implemented in the *Layered Queueing Network Solver* tool.

The LQN model comprises the following elements.

Task. A task usually represents a software component that is capable of serving different kinds of requests. It is represented graphically as a stacked parallelogram and is associated with a multiplicity which models the number of concurrent instances of the task available at runtime. For instance, if the task models a multi-threaded application, then its multiplicity is the size of the thread pool.

Processor. A task is deployed onto a processor element, which is denoted graphically by a circle connected to the task. Its multiplicity represents the number of parallel processors available. Different tasks may be deployed on the same processor.

Entry. An entry is a kind of service exposed by a task. It is depicted as a small parallelogram in the top area inside the task. In the remainder of this paper we shall be concerned with single-entry tasks only.

Activity. An activity may be regarded as the basic unit of computation of the LQN model. A directed graph whose nodes are activities (drawn as rectangles) expresses the dynamic behaviour of an entry. This graph — called the *execution graph* — is shown inside the task's parallelogram where the entry resides. An activity denotes some computation time required on the processor on which its task is deployed. This time is assumed to be exponentially distributed with expectation specified within square brackets in the activity's rectangle. Activities may be connected through nodes to model the following behaviour:

- Sequential behaviour is specified by two activities being connected through an edge.
- Probabilistic choice (denoted by a small + circle) performs one of the activities connected by the outgoing edges according to some probability mass function, specified through labels on the outgoing edges.
- Fork/Join (denoted by a small & circle): a fork executes in parallel all the activities reached by its outgoing edges whereas a join waits until all activities connected through its incoming edges terminate.

Inter-process Communication. Activities within one task may invoke entries of another task. Service invocation is represented graphically by an solid arrow which connects the activity with the invoked entry. The arrow is labelled with a number within parentheses which denotes the number of invocations performed per execution of one activity. The semantics for *synchronous* invocation is that the calling activity suspends until the execution graph of the callee terminates. Termination of an execution graph is denoted by a dotted arrow pointing back to the graph's entry. For an *asynchronous* invocation, the invoked activity is executed concurrently with the execution graph of the calling activity. Synchronous invocations are depicted with a solid arrowhead whereas asynchronous ones are depicted with an open arrowhead.

Reference Task. A task with no incoming service invocation is called a reference task and models the system's workload.

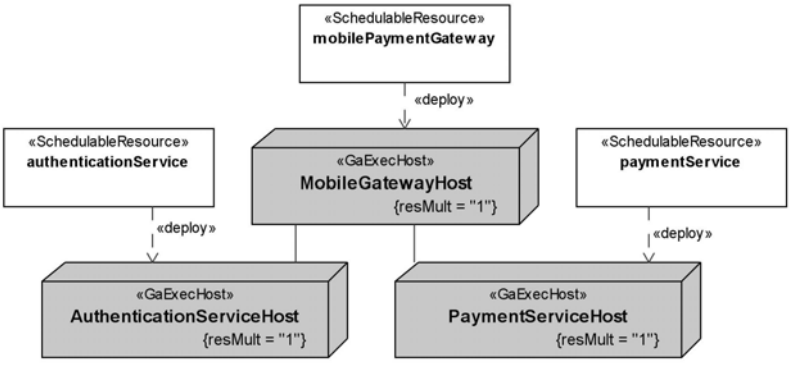


Fig. 3. Annotated deployment diagram for the case study

5 LQN Models for UML4SOA

5.1 Performance Annotations with MARTE

As discussed in Section 1, the LQN model requires further quantitative information on the required execution times of the activities, which is not available in the UML nor in the UML4SOA profile. To this end, we adopt the same approach taken in [8] where the profile for MARTE (cfr. [16]) is employed to augment the model the timing specifications.

Deployment Information. In the following, we assume that the UML4SOA model contains a deployment specification in which each of the services is associated with a node. Each node must be stereotyped with «GaExecHost», which indicates a host that is capable of carrying out computation. In particular, the property `resMult` will be used to extract the processor multiplicity in the LQN performance model. The artifacts deployed on these nodes are stereotyped with «SchedulableResource». The name of the artifact is referenced by the service activity which implements its behaviour. An example of a suitable deployment diagram is shown in Figure 3. In this scenario each service is run on a separate single-processor machine. Therefore, if the services are executed as multithreaded applications, all threads will contend for the CPU time of the same processor. This is one potential source of delays (i.e., queueing effects), as will be discussed in more detail in Section 6.

Stereotyping of Service Activities. We now discuss how each service activity is to be annotated with MARTE stereotypes. Figure 4 shows an excerpt of the complete model regarding the annotations on the MobilePaymentGateway service. The other two services are annotated similarly and are not shown due to space constraints. Each UML4SOA service activity is stereotyped with «PaRunTInstance», indicating that the activity is an instance of a runtime object. The

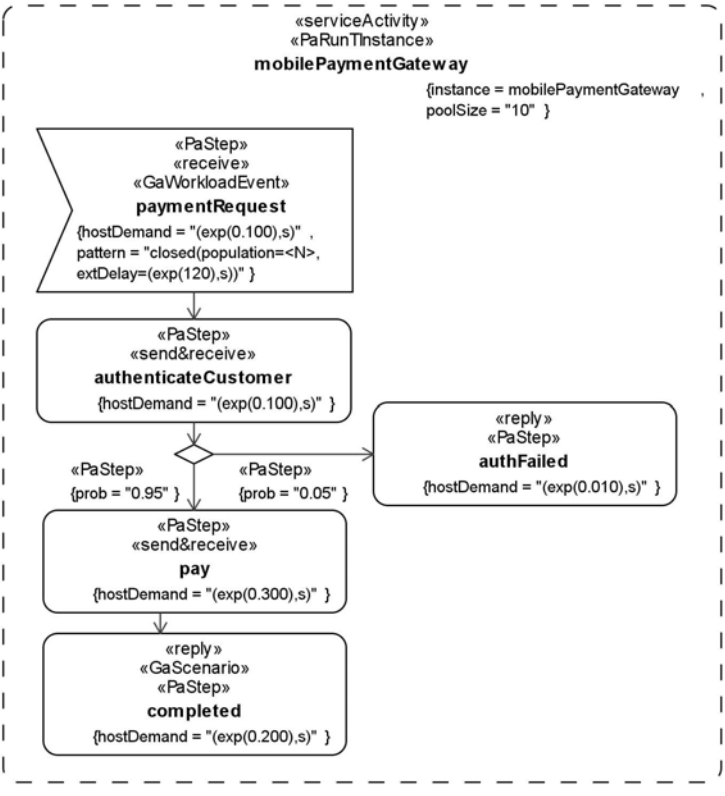


Fig. 4. The service activity for the mobile payment gateway annotated with the profile for MARTE

crucial property of this stereotype is `poolSize` which holds an integer that indicates the number of available threads at runtime. The property `instance` is set to the corresponding name in the deployment specification. In the example, MobilePaymentGateway is executed as a ten-thread application which runs on MobileGatewayHost. Each node of a UML4SOA service activity is stereotyped with MARTE's «PaStep» with the property `hostDemand` set to `(exp(<time>), s)`, indicating the exponential distribution associated with that action. Because of its probabilistic interpretation, as discussed later in this section, the outgoing edges of a decision node must be stereotyped with «PaStep» with the property `prob` set such that the sum across all edges equals one.

Workload specification. The «receive» action node which triggers the whole system must be stereotyped with «GaWorkloadEvent», which describes the dynamics of the arrival of the requests by clients. Thus, paymentRequest of MobilePaymentGateway is stereotyped with «GaWorkloadEvent», whereas authenticateCustomer and pay are not because these activities are not triggered

directly by users. The property `pattern` of this «GaWorkloadEvent» is set to `closed(population=<N>, extDelay=(exp(1/<think>),s))` to model a population of `N` users of the service-oriented architecture which make payment requests cyclically, interposing a think time of `think` seconds between successive requests.

The uses of the profile for MARTE specified above are sufficient to derive the LQN performance model from the UML4SOA specification, as discussed next.

5.2 Extracting the LQN Model

We will focus on the extraction of the LQN model from the specific case study presented in the paper. In doing so, we will describe some patterns of translation that can find a wider applicability. A more formal and general specification of the meta-model transformation of UML4SOA (and the MARTE profile) to LQN is the subject of ongoing work. For the sake of clarity, we find it more convenient to present first the overall LQN model of the case study (in Figure 5) and then guide the reader through the steps taken to obtain it.

Each «serviceActivity» is modelled as an LQN task, conveniently denoted by the same name. The task multiplicity is inferred from the application of the stereotype of «PaRunTInstance», as discussed above. The task is associated with an LQN processor which is named after the node in the deployment specification on which the «serviceActivity» is deployed. Each task has a single entry, named after the «receive» action node that triggers the service. The execution graph of an entry resembles closely the service activity in the UML model. Any action node — except for the triggering «receive» node — is translated into an LQN activity with execution demand taken from the «PaStep» stereotype application. For instance, the «send&reply» node authenticateCustomer is modelled as a basic LQN activity with execution demand equal to 0.100. (For ease of reference, basic activities are named after their corresponding action nodes as well.) Decision nodes are interpreted as probabilistic choices, with probabilities taken from the «PaStep» application to their outgoing edges. UML's forks and join are simply translated into their LQN analogues.

The exchange of messages between services is modelled as an invocation of external entries in the LQN model. In this paper we focus our attention on synchronous messages which return a reply to the caller. In the UML4SOA model, this corresponds to having action nodes stereotyped with «send&receive» in the invoking service and a «reply» node which (in this case study) terminates the service that is invoked. Therefore, the LQN activities corresponding to the «reply» nodes present dotted arrows directed to the parent entry (e.g., see paymentCompleted and authenticateCustomer).

The necessary information to translate the pattern of synchronous communication is gathered from the names of the lnk pins in the service activities and from the architectural specification. Given a «send&receive» node, the name of the lnk pin is used to find the corresponding edge which connects the two communicating services in the composite structure specification. For example, the lnk pin of the authenticateCustomer node is authService, which connects

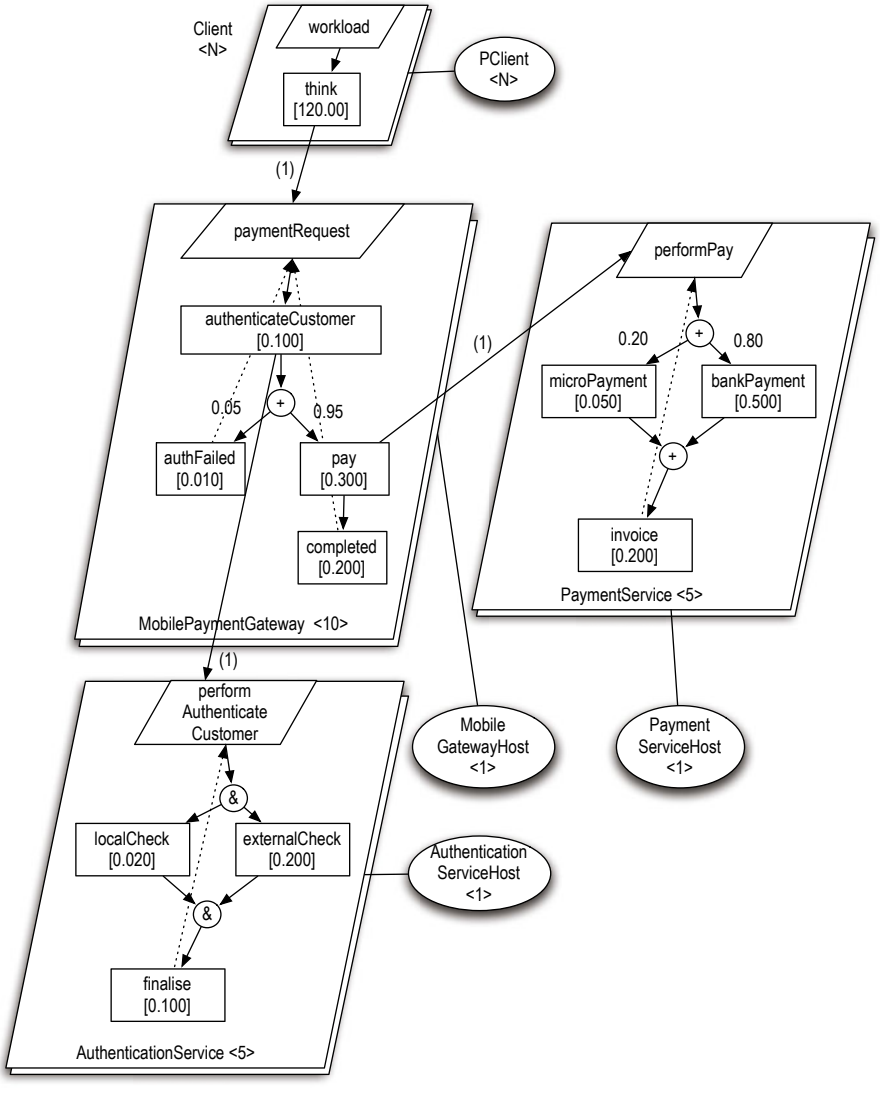


Fig. 5. Layered Queueing Network model of the Payment case study

MobilePaymentGateway to AuthenticationService. If the connected component has an explicit behavioural specification in terms of «serviceActivity» — as is the case for AuthenticationService — then the LQN model will feature a synchronous call (with multiplicity one) from the activity that models the «send&receive» node to the entry of the invoked activity. Using the same example, the LQN model has a synchronous invocation of the entry performAuthenticateCustomer from the activity node authenticateCustomer.

There may be in the UML4SOA model invocations of external services which are not given a behavioural specification — e.g., the externalCheck «send&receive» node in AuthenticationService invokes some ExternalAuthenticationService of unspecified behaviour. Such nodes will be translated as basic LQN activities which do not make calls to other entries. In effect, this external invocation is abstracted away in the LQN model and its impact on the performance of the system in encompassed in the execution demand, as specified in the «send&receive» node.

The specification of the system workload through the «GaWorkloadEvent» is translated into an LQN reference task as follows. A task named *Client* is created with one entry called *workload*. Its execution graph consists of a single activity named *think* with execution demand equal to `<think>`. The task has multiplicity `N` and is deployed on a processor with multiplicity `N` named *PClient*. A synchronous message (with multiplicity one) connects *think* with the entry corresponding to the action node with «GaWorkloadEvent», i.e., paymentRequest in the case study.

5.3 Indices of Performance

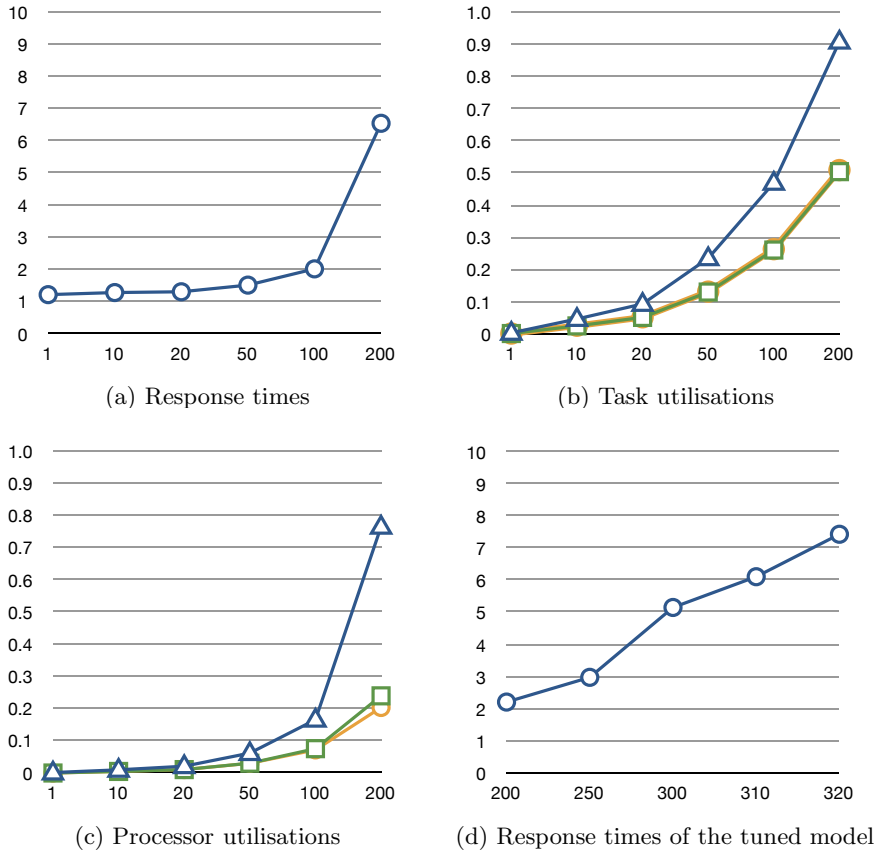
The analysis techniques available for LQNs provide the modeller with a wide range of quantitative estimates on the *long-run* (or *steady-state*) behaviour of the system, i.e., the performance attained after a sufficiently long period of time that the system was started. This appears to be an appropriate characterisation of the performance of real-life service-oriented architectures, which are usually on-line continuously. In this paper we put emphasis on two such indices:

- The *response time* for a client, measured as the average time it takes for the system to process a payment request. The response time does not include the think time by the client, but it does include all the execution times of the basic activities that are involved during the processing of a request and the time (if any) spent while waiting for the system resources (e.g., threads and processors) to be available.
- The *processor utilisation*, which measures the average number of processors in a multi-processor component that are busy. Alternatively, this value, if normalised with respect to the total multiplicity of the processor, can be interpreted as the percentage of time that a processor is busy. Analogously, the *task utilisation* measures the average number of threads that are busy.

In the following section, these two indices will be used in a numerical evaluation of the performance of our case study. Another notable performance metric — not discussed further in this paper due to space constraints — is *throughput*, i.e., the rate at which an entry (or an activity) is executed.

6 Numerical Example

The performance study addressed in this section is a typical *dimensioning* problem, in which the modeller wishes to find a suitable configuration of the system



(a) Response times (b) Task utilisations

(c) Processor utilisations (d) Response times of the tuned model

Fig. 6. Numerical results. x-axis: population sizes. The y-axis for response times are in time units, whereas utilisation is a dimensionless metric between 0 and 1. The markers in (b) and (c) are related with the UML4SOA components as follows. Triangle: MobileGatewayHost; Square: AuthenticationServiceHost; Circle: PaymentServiceHost.

in order to satisfy some quality-of-service criteria. In the following, we assume for the sake of simplicity that the execution demands are given and fixed, as shown in Figure 5. Thus, the parameters that may be changed are the multiplicities of the tasks and of the processors in the model. Perhaps the most intuitive index of performance is the average response time experienced by a client; this index is shown in Figure 6a for increasing system workload, represented by the property of `population = N` in the UML model.

The baseline $N = 1$ is of interest because it gives the minimum response time attainable, since there is no contention for resources in the system. The curve shows a typical profile, characterised by increasing response times as function of N, with sharp deterioration after some critical point. In this example, the response time at $N = 200$ is about six times as much as the baseline value.

The normalised utilisation profiles for the tasks and the processors, shown in Figures 6b and 6c, respectively, offer more insight into where the degradation of performance arises from. Clearly, the utilisations increase with increasing workload, however those related with AuthenticationServiceHost and PaymentServiceHost do not appear to be problematic since they are at most about 50% in the worst case $N = 200$. Instead, the processor utilisation of MobileGatewayHost is about 91%, indicating a heavy utilisation of this resource.

Taken together, these results suggest that an effective route toward performance improvement is to add more processing capacity to MobileGatewayHost. Figure 6d shows the response times when the node is deployed on a two-processor machine (instead of the original single-processor one). The response time at $N = 200$ is now about one third of the original model, and the system can sustain up to 310 clients with an average response time that would be delivered with only 200 clients in the original configuration.

7 Conclusion

We discussed a methodology for extracting a layered queueing network performance model from a service-oriented architecture description in UML4SOA. The level of abstraction of LQNs appears convenient for the prediction of the quantitative behaviour of a system under scrutiny. The services are modelled as multithreaded applications communicating with each other. The explicit modelling of the deployment scenario puts constraints on the level of threading and on the processing power of the hardware on which the application is run. A numerical evaluation of the case study has shown how marginal changes to the deployment can have a significant impact on the predicted performance. It is our opinion that the possibility of effortless experimentation with different configurations and a generally deeper insight into the system's dynamics outweigh the additional modelling effort required to augment the model with performance-related annotations.

In order to widen the applicability of this methodology and make it available to practitioners in SOAs, further research needs to be carried out. It is our plan to provide a precise, formal characterisation of the meta-model transformation presented in this paper, which would also support other UML4SOA constructs, such as compensations and event handling which were not considered here. On a more practical level, we would like this transformation to be implemented as a module in leading UML modelling tools to be able to experiment with larger, real-world service-oriented applications.

Acknowledgement. This work was supported by the EU-funded project SENSORIA, IST-2005-016004.

References

1. Balsamo, S., Di Marco, A., Inverardi, P., Simeoni, M.: Model-Based Performance Prediction in Software Development: A Survey. IEEE Trans. Software Eng. 30(5), 295–310 (2004)

2. Bard, Y.: Some extensions to multiclass queueing network analysis. In: Third International Symposium on Modelling and Performance Evaluation of Computer Systems, pp. 51–62. North-Holland, Amsterdam (1979)
3. Baskett, F., Mani Chandy, K., Muntz, R.R., Palacios, F.G.: Open, closed, and mixed networks of queues with different classes of customers. J. ACM 22(2), 248–260 (1975)
4. Becker, S., Koziolek, H., Reussner, R.: Model-based performance prediction with the palladio component model. In: Proceedings of the 6th international workshop on Software and performance, vol. 65. ACM, New York (2007)
5. Mani Chandy, K., Neuse, D.: Linearizer: A heuristic algorithm for queueing network models of computing systems. Commun. ACM 25(2), 126–134 (1982)
6. Franks, G., Maly, P., Woodside, M., Petriu, D., Hubbard, A.: Layered Queueing Network Solver and Simulator User Manual (2005), http://www.sce.carleton.ca/rads/lqns
7. Franks, G., Omari, T., Murray Woodside, C., Das, O., Derisavi, S.: Enhanced modeling and solution of layered queueing networks. IEEE Trans. Software Eng. 35(2), 148–161 (2009)
8. Gilmore, S., Gönczy, L., Koch, N., Mayer, P., Tribastone, M., Varró, D.: Nonfunctional properties in the model-driven development of service-oriented systems. Software and System Modeling (2010)
9. Gordon, W.J., Newell, G.F.: Closed queuing systems with exponential servers. Operations Research 15(2), 254–265 (1967)
10. Hillston, J.: A Compositional Approach to Performance Modelling. Cambridge University Press, Cambridge (1996)
11. Koziolek, H., Reussner, R.: A model transformation from the palladio component model to layered queueing networks. In: Kounev, S., Gorton, I., Sachs, K. (eds.) SIPEW 2008. LNCS, vol. 5119, pp. 58–78. Springer, Heidelberg (2008)
12. Wirsing, M., et al.: Sensoria: Engineering for Service-Oriented Overlay Computers. MIT Press, Cambridge (2009)
13. Mayer, P., Koch, N., Schroeder, A., Knapp, A.: The UML4SOA Specification. Specification, LMU Munich (2009), http://www.uml4soa.eu/wp-content/uploads/uml4soa_spec.pdf.
14. Mayer, P., Schroeder, A., Koch, N.: MDD4SOA: Model-Driven Service Orchestration. In: EDOC, pp. 203–212. IEEE Computer Society, Los Alamitos (2008)
15. Object Management Group (OMG): UML Superstructure Specification 2.1.2. Technical report, OMG (2007), http://www.omg.org/spec/UML/2.1.2/Superstructure/PDF/ (last accessed on May 5, 2009)
16. Object Management Group (OMG). A UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded systems, Beta 2. Technical report, Object Management Group (2008)
17. Object Management Group (OMG). Service oriented architecture Modeling Language(SoaML), Beta 1. Technical report, Object Management Group (2009)
18. Murray Woodside, C., Petriu, D.C., Petriu, D.B., Shen, H., Israr, T., Meseguer, J.: Performance by unified model analysis (PUMA). In: WOSP, pp. 1–12 (2005)