Model Driven Performability Analysis of Service Configurations with Reliable Messaging *

László Gönczy, Zsolt Déri, and Dániel Varró

Budapest University of Technology and Economics
Department of Measurement and Information Systems
H-1117 Budapest, Magyar tudósok körútja 2.
{gonczy,varro}@mit.bme.hu, zsolt.deri@gmail.com

Abstract. Due to the rapid increase in the number of available web services, more and more emphasis is put on their reliability, availability, security, etc. These non-functional requirements are frequently captured in service-level agreements between service requesters and providers. In order to meet such non-functional requirements, a service needs to be designed for reliability by making design decisions on a high, architectural level. In the paper, we present a model-driven approach for the precise analysis of service configurations with reliable messaging. Starting from high-level UML models of service configurations captured by a UML profile dedicated to service design, performability models are derived by automated model transformations for the PEPA toolkit in order to assess the cost of fault tolerance techniques in terms of performance.
Keywords: Model-driven Analysis, Service-Oriented Architecture, Performability Analysis, Service Configuration

1 Introduction

Service-Oriented Architectures (SOA) provide a flexible and dynamic platform for implementing business services. Due to the rapid increase in the number of available services, more emphasis is put on their reliability, availability, security, etc. In order to meet such non-functional requirements, a service needs to be *designed for reliability* by making design decisions on an architectural level.

Recently, various non-functional parameters of services have been identified by various XML-based web service standards such as WS-Reliability, WS-RM, WS-Security, etc. While these properties are attached to business-level web services, they, in fact, specify the configuration and behavior of the service infrastructure, i.e. services that are not part of a specific application, but play a dedicated role in the underlying service middleware. A focal issue in the service infrastructure is to provide reliable messaging between services where the delivery of a message can be transparently guaranteed by the underlying platform.

Unfortunately, service configurations are typically set up in a rather ad hoc way. While non-functional requirements are precisely captured in service-level

* This work was partially supported by the SENSORIA European project (IST-3-016004). The third author was also supported by the János Bolyai Scholarship.

agreements, there is no guarantee that the service configurations will actually meet these requirements. One of the reasons for this is that "design for reliability" is a complex task as performance and reliability requirements are contradicting: an inappropriate setup of reliability attributes may cause significant decrease in performance. As a consequence, *performability analysis* is necessitated to assess the cost of using fault-tolerant techniques in terms of performance.

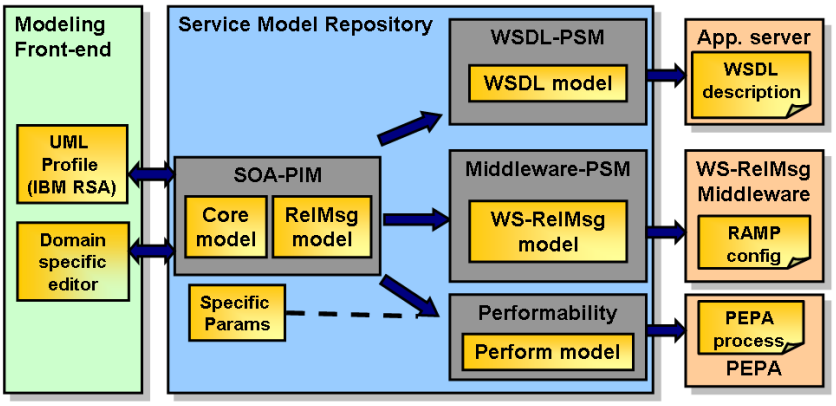


Fig. 1. Model-driven analysis of service configurations with reliable messaging

To tackle these problems, we propose a model-driven approach (illustrated in Fig. 1) to efficiently design, analyze and deploy standards-compliant service configurations with reliable messaging. Our approach is strictly in line with the model-driven service engineering framework being developed within the SENSORIA EU FP6 research project [17].

Modeling of service configurations. In order to raise the level of abstraction for service engineers when designing the configuration of the service infrastructure, we rely on high-level UML models conforming to the UML4SOA profile (developed within SENSORIA), which uses the standard extensibility mechanism of UML for modeling service-oriented applications. This profile is closely related to the core SOA metamodel presented in [2], but extended with various (e.g. non-functional) aspects of service design. In this paper (Sec. 2), we specialize this general-purpose profile in order to capture service configurations with reliable messaging based upon a metamodel developed in [9].

Model-based performability analysis. From such service configuration models, automated model transformations generate *formal process models* for the PEPA framework (Performance Evaluation Process Algebra, [5]) to provide an early performability evaluation and prediction for service configurations with reliable messaging (Sec. 3). We identify the abstract behavior of core service configuration elements, which incorporates reliable messaging semantics, but it is independent of the business functionality of services. Then model transforma-

tions assemble the formal performability model from these elementary building blocks based upon the actual service configuration model. These transformations were implemented in the VIATRA2 framework [19, 20] by following an MDA approach with separated phases for PIM-to-PSM mappings and the generation of the textual target descriptions. A brief insight to the actual transformations are provided in Sec. 3.

Analysis models serve also as the basis of deployment code generation to reliable messaging middleware (see Fig. 1). The basic method for this has already been described in [7]. Currently, we support the (semi-)automated deployment to IBM RAMP and Apache Axis2 platforms (in the latter case we also generate security configurations).

2 Modeling SOA with Reliable Messaging

In the current section, we present how service configurations can be modeled using a high-level UML model dedicated to service design by a corresponding UML profile, which was designed as part of the SENSORIA project. This profile is conceptual follow up of [2] where a semi-formal platform-independent and a SOA-specific metamodel (ontology) was developed to capture service architectures on various levels of abstraction in a model-driven development process for business-level services. The UML4SOA profile includes means to capture non-functional aspects of services on a high-level of abstraction (i.e. independently of specific non-functional parameters such as availability or performance). In this section, we briefly overview the core ideas behind this modeling language. Moreover, we specialize this general-purpose non-functional profile to capture service configurations with reliable messaging based upon a metamodel developed in [9].

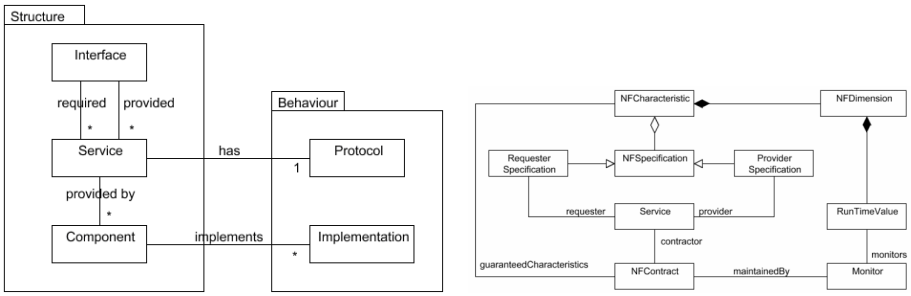
2.1 Running example

In this paper we will use the "*On Road Assistance*" scenario developed in scope of the Automotive Case Study [12] within the SENSORIA [17] project, which describes a car-to-infrastructure application scenario. In this scenario:

1. The built-in diagnostic system of a car reports a severe failure of the engine.
2. This triggers the in-vehicle diagnostic system to perform an analysis of the sensor values.
3. If the car is no longer drivable the system sends a message with the diagnostic data and the GPS data of the vehicle to the car manufacturer or service center.
4. Based on availability and the driver's preferences, the service discovery system identifies and selects the appropriate services in the area: repair shop (garage), tow truck and rental car.
5. The selection of services takes into account personalized policies and preferences of the driver.
6. Upon confirmation, the owner of the car has to deposit a security payment before being able to order services.

This scenario raises several non-functional requirements against the system, as collected in [6]. In this paper, we concentrate on the *accountability*, which means on the service architecture level that the effect of communication faults have to be eliminated by the underlying middleware based upon appropriate service configurations to guarantee the message delivery between components.

2.2 A core SOA metamodel and non-functional extensions



(a) Main concepts of UML4SOA

(b) Metamodel of non-functional properties of services

Fig. 2. Metamodels for service modeling

The UML4SOA profile [13] was developed in the SENSORIA project to capture the abstract structural, behavioral and non-functional aspects of service-oriented applications. The core concepts in the UML profile are describe in a corresponding metamodel depicted in Fig. 2(a) (for core service concepts) and Fig. 2(b) (for non-functional extensions). The profile is built in a modular way, and thus here, we mainly focus on non-functional aspects, which are most relevant for the current paper. These non-functional aspects were inspired by standard UML extensions (such as [14]).

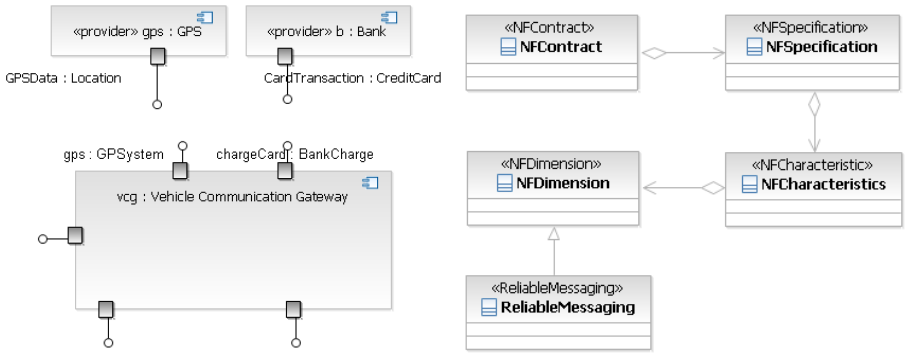
On a very abstract level, we define Services which are provided by Components. Each service defines two Interfaces, a provided interface and a required interface. Each service defines a Protocol while each component has an Implementation.

Non-functional aspects are included in the UML4SOA profile by generalizing Service Level Agreements. Attributes of a service are described by NFDimensions which are collected to NFCharacteristics, which represent logical groups of properties, such as security, performance or reliable communication. These characteristics are contained within an NFSpecification.

During the operation lifecycle of services, provided and requested properties of services are negotiated (which process is out of the scope of the current paper). After the negotiation, a contract with an agreement of the agreed specification is created. Fulfillment of the contract is monitored by a dedicated component.

Case study. An extract of the components of the "*On Road Assistance*" scenario [12] is shown in Fig. 3(a). In the current paper, we focus on the the Vehicle Communication Gateway component, which is responsible for the car-to-infrastructure communication, i.e. it manages communication between external service providers, like the Bank and the global positioning system (GPS). This way, the Vehicle Communication Gateway acts as a communication mediator between a central service Orchestrator component and the actual external services. For our initial investigations, we disregard from this Orchestrator, and focus only on the other three components.

2.3 Reliable messaging standards for web services



(a) Core components of the OnRoadAssistance scenario

(b) Extension of non-functionalconcepts for reliable messaging

Fig. 3. Core components and modeling extensions

There are various industrial standards reflecting the emerging need for reliable Web services middleware from which we focus on reliable messaging standards (e.g., WS-Reliability and WS-ReliableMessaging) in this paper.

The main importance of these reliable messaging standards lies in the fact that they are expected to replace the current mainstream messaging middleware (such as Message Queuing servers or JMS) which are now used together with SOAP to provide a reliable asynchronous communication service.

Reliable messaging in the fields traditional distributed systems is closely related to the guaranteed semantics of message delivery. Usual delivery classes are the following:

1. *At least once delivery.* In the case of normal operation, every message is transferred at least once, with the possibility of sending multiple instances of the same message. This can only be allowed in systems where this does not have an undesired side-effect.

2. *At most once delivery* guarantees that no message will be sent multiple times to the receiver, but their successful transmission is not ensured.
3. Exactly once delivery is the strongest delivery semantics, guaranteeing both the successful message delivery (usually acknowledgements are required for each message) and the filtering of duplicate messages.

The following attributes are required for the configuration of reliable messaging (besides **messagingSemantics**, which selects the messaging mode as described earlier):

- **inactivityTimeout**: (integer, seconds), after this period of time if no acknowledgment message has arrived, the connection is closed;
- **exponentialBackoff**: (boolean), if it is set to true, time amounts between retransmissions are following an exponential distribution;
- **acknowledgementInterval**: (integer, seconds), amount of time elapsed before sending acknowledgement message;
- **retransmissionInterval**: (integer, seconds), after this time a request is resent by client if no acknowledgement arrived.

As reliable messaging PIM, we use the metamodel of reliable messaging in service configurations was created in [9] to incorporate reliable messaging attributes of these standards.

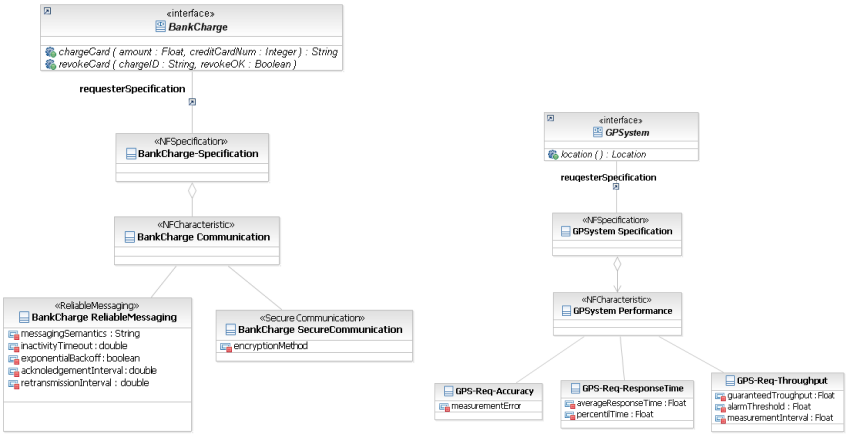
We incorporate these attributes to the UML4SOA profile by prescribing that the NFCharacteristic of an NFSpecification should contain an NFDimension specific to reliable messaging ReliableMessaging (when reliable messaging is required by a contract). The relationship between concepts is illustrated in Fig. 3(b).

Case study. We now demonstrate how the non-functional extensions of the UML4SOA profile can be used to capture reliable messaging specifications in Fig. 4(a) (for charging the bank account of the driver) and Fig. 4(b) (for the Global Positioning System).

For instance, communicating with the bank requires both reliable and secure message communication, of which here we concentrate on the first. Note that here we are at the class level, a concrete instance of this system is shown later in Fig. 8.

NFSpecifications are defined to describe the relevant properties of communication between GPSSystem and Vehicle Communication Gateway etc. These specification can contain different characteristics like availability, performance or reliable messaging. Certain parameters can be defined for non-functional attributes (averageResponseTime, messageSemantics etc.) in course of modeling which can be used for example for generation of configuration files as can be seen later.

In the current paper, we illustrate our approach by using the *at-least-once* reliable messaging semantics as a communication model. However, using other reliable messaging semantics would not cause significant complications for the presented approach.



(a) Non-functional specification of the bank service

(b) Non-functional specification of the GPS service

Fig. 4. Example models

3 Model-based Performability Analysis of Services

As the main contribution, we present a model-driven technique for the performability analysis of service configurations with reliable messaging. *Performability* refers to the behavior of the system in the presence of faults, in other words, the cost of fault-handling (or fault-tolerant) techniques in terms of response time. For our investigations, we use the PEPA (Performance Evaluation Process Algebra) toolkit [5], which offers a formal language for capturing and powerful stochastic analysis techniques for the evaluation of performance models.

Essentially, automatic model transformations derive PEPA processes from the UML models of service configurations extended with reliable messaging attributes. This transformation takes various inputs:

- *Service configuration models*, which only contain the architectural design, i.e. the dependencies between services being relevant for performability analysis. For performability analysis, we identify the main roles of service providers and requesters potentially chained to incorporate third party services.
- *Predefined library of component behavior*, which captures the core, performability related behavior of reliable messaging for each party (e.g. service provider, requester). This library should include technology-related behavior, which is typically observable but not controllable (in case of a reliable messaging middleware, the overhead of access to stored message content before generating new message instances).
- *Reliable messaging parameters*, which affect both the structure and the dynamic behavior of performability models. This includes quantitative characteristics of faults of message transmission (encoded implicitly into rates of transitions) to estimate the effect of unreliable communication layer.

3.1 The performability model

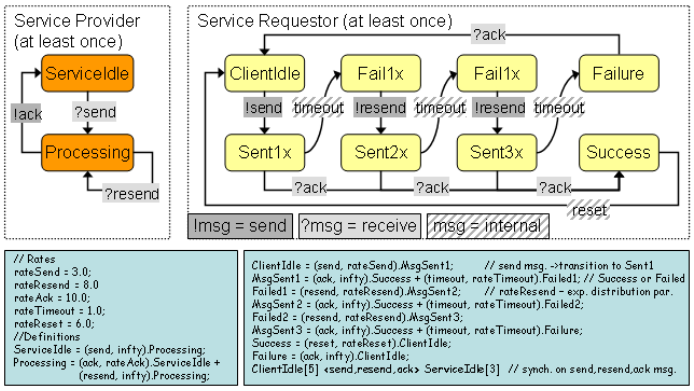


Fig. 5. PEPA process model of a service configuration with at-least-once messaging

For capturing the performability model of the basic components (in our case, the client and the server), we use a visualized version of the process algebra notation of PEPA. Each process is visualized as an automaton. Rectangles represent states, while transitions between states correspond to communication actions. ! stands for sending a message and ? means receiving a message. Sending and receiving messages is carried out by synchronization between the two processes. Internal actions without communication between processes are also distinguished (e.g., timeout trigger events). The firing frequency of transition in PEPA are considered to follow an exponential distribution.

Fig. 5 shows the stochastic performability model created as a combination of some core processes. The model represents the behavior of a service provider (Bank) and a service requester Vehicle Communication Gateway when reliable messaging is required between them.

The *service provider* (shortly, server) component is either processing a request or waiting for a new one, with the action of sending an acknowledgement to the client once the message was successfully received.

The *service requester* (or shortly, client) is assumed to behave according to the *at-east-once* semantics (with an upper limit of three on the number of messages). The automaton of the service requester represents that after sending a message, it waits for an acknowledgement until a timeout occurs. Then it resends the request until the maximum number of retransmission is reached. This is represented by the non-trivial multiplication of a basic automaton as many times as the maximum number of allowed retransmissions. If an acknowledgement arrives, the message transmission is considered successful.

It is worth pointing out that the handling of *exactly-once delivery* semantics prescribing the filtering of duplicate messages is quite similar from a performability perspective. Furthermore, note that this process model is an abstraction

of the graph transformation system presented in [9] as formal semantics of the middleware behavior.

3.2 Performability parameters for reliable messaging

Reliable messaging parameters have different impact on the structure of this model. The number of service providers and requesters can be altered by changing the system equation (e.g. ClientIdle[3]). We used RAMP parameters of services to derive send, resend, acknowledgement rates. These parameters has been set between the two parties after negotiation process resulting in a NFContract.

Values of parameters **inactivityTimeout**, **exponentialBackoff**, **acknowledgementInterval** and **retransmissionInterval** serve as a basis for deriving the actual firing rates of the transitions in Fig. 5. Note that firing times follow an exponential distribution specified by these rates, i.e. a larger rate means that the corresponding operation will be executed faster. Below we describe how the rates in the PEPA model were derived from the reliable messaging parameters in Sec. 2.3.

- **acknowledgement rate** (rateAck): it depends on the acknowledgement interval:
$$ack = 1/acknowledgementInterval,$$
On calculating this formula the prospective value of exponential distribution with ack parameter will be equal to acknowledgementInterval.
- **timeout rate** (rateTimeout): Similar to the previous formula:
$$timeout = 1/retransmissionInterval,$$
but if exponentialBackoff is true the next timeout is:
$$timeout = timeout/2,$$
thus between resend messages exponential amount of time elapses;
- **send rate** (rateSend): it can be considered as an input parameter of sensitivity analysis so a service configuration can be tested under different workload. Alternatively, it can be derived from maximum throughput parameter in a service level agreement, which expresses the number of requests received from the user.
- **resend rate** (rateResend): it should be a high value (compared to other parameters) because when timeout occurs, resent message should be sent as soon as possible.
- **reset rate** (rateReset): like resend rate, it gets a high value in order to model the transition to ClientIdle state after successful transmission.

Note again that the number of allowed retransmissions (**retransmission number**) changes the state space of the client automaton, i.e. additional retransmission states are introduced (FailNx and SentNx). This number is determined

in the following way:

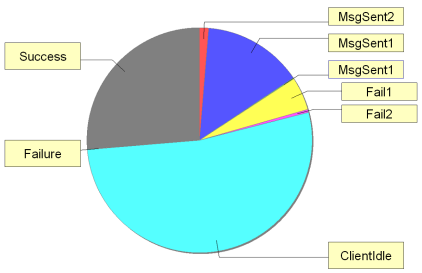
$$retransmissionNb = \left\lfloor \frac{inactivityTimeout}{retransmissionInterval} \right\rfloor$$

This expresses that the clients tries to resend the request until the inactivity timeout is exceeded.

3.3 Performability analysis objectives

The typical questions for the PEPA solvers investigate *passage time* (i.e., the expected response time of the system as a function of stochastic parameters), *utilization* of states (what percentage of total operating time is spent in a particular state). In addition, *sensitivity analysis* can also be performed to estimate the effect of changing transition rates on system-level performability attributes. The number of possible retransmissions is also an interesting parameter to investigate, however, this needs the modification of the structure of the performability model (by re-executing the transformation), while tuning of other parameters requires to modify only the rates in the generated PEPA model. Core examples for using PEPA are available in [5, 21].

We can investigate the utilization of states in order to answer questions like *"What percentage of time is spent waiting for the answer of the request?"* The result obtained from executing PEPA is listed in the pie chart of Fig. 6.



With the parameter settings of our running example (described in Fig. 5), PEPA derives that in a steady state, the system spends 23% of the time within states MsgSentX and FailX, which are exactly the states required for providing reliable messaging. In other terms, the system spends 23% of the time with fault handling.

Fig. 6. Utilization of states

Sensitivity analysis. Fig. 7 shows the (relative) change of failure rate as a function of RAMP related parameters acknowledgement time and retransmission interval based upon PEPA calculation. For the failure rate, utilization of Failure state has been used. X-axis shows different values of acknowledgment time while the different curves plot different timeout thresholds.

Our analysis results can be interpreted as early prediction of performability. For instance, one can deduce from Fig. 7 that if the rateAck rate is increased from 0.2 to 0.3 (namely acknowledgement interval decreases), then there is about 100% decrease in the frequency of errors. So it is worth improving performance of the provider if its cost is linear. Decreasing rateTimeout rate (curves with different colors) also leads to the improvement of failure rate.

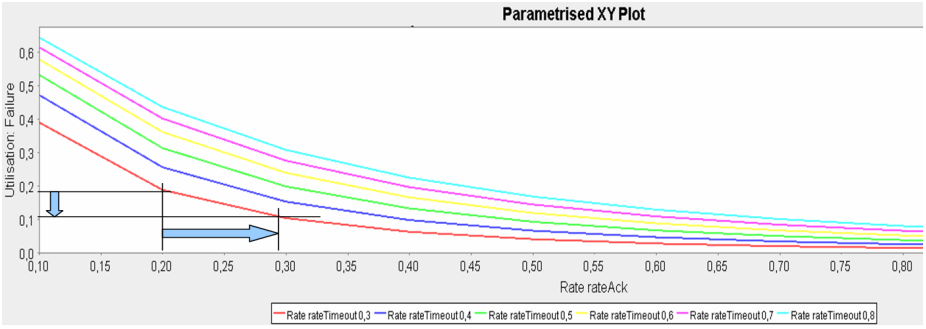


Fig. 7. Effect of RAMP-related parameters on failure

3.4 Transitive invocation of third-party services

Now we extend our performability model to handle service configurations where a service provider itself needs to call some other (third-party) service in order to serve its own request. This intermediate (mediator) component acts as a server and a client at the same time, thus we derive its behavior by combining the basic elements of our performability model (exemplified in Fig. 5) by synchronizing the core automata on send and ack messages.

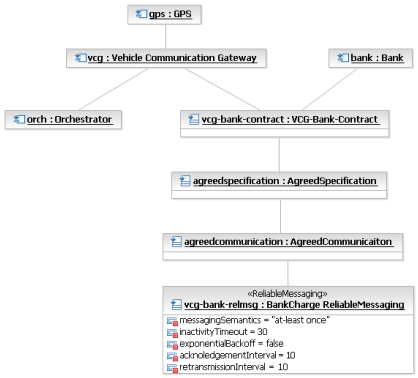


Fig. 8. Multiple parties

For our investigations, we have bounded such a transitive behavior by a depth limitation of 3, while we assume that potential cyclic service invocation is filtered out by formal model checking.

In the sample service configuration of Fig. 8 Orchestrator invokes Vehicle Communication Gateway (VCG) that acts both as a service provider and service requester. The worst performance is observed when the VCG needs to call both GPS and Bank components to fulfill a request.

Sensitivity analysis. For this scenario, we carried out a sensitivity analysis to measure the throughput of the acknowledgement action of the system. The results are depicted in Fig. 9 where the rates rateSendVC and rateSendVG stand for the sending rate between VCG - Bank and VCG - GPS components, respectively. Increasing these rates results in a faster operation of the Bank and GPS services. Observing the lower curve we can deduce that it is useless to increase the performance of Bank component if the GPS operates slowly (0.2). However the upper curve shows that it is worth increasing the speed of Bank service as

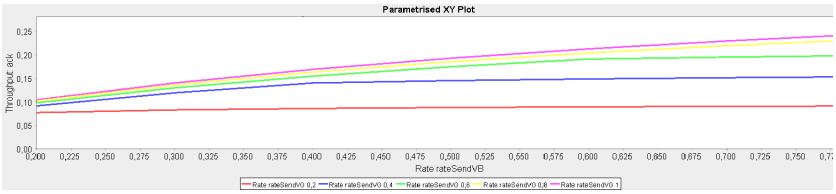


Fig. 9. Sensitivity analysis with multiple parties

it results in significant increase in acknowledgement rate, thus it decreases the acknowledgement time of the system.

3.5 Transformation implementation

The translation of the UML model to PEPA code was implemented in multiple steps shown in Fig. 10 using Model2Model and Model2Code transformations. The input of the transformation chain is a UML model using UML4SOa for modeling services and non-functional parameters of messaging. This model is imported to the internal representation of the VIATRA2 tool.

First, uml2soa transformation takes out the relevant parts of the model. This transformation is used to collect relevant information from the model representation and generates a compact model (which is semantically similar to a Domain Specific Model). This model is also used in other work as a basis of configuration generation for Web services [7].

Then uml2pepa is executed to transform relevant parts of this model (from the performance aspect) to the concepts of the PEPA tool by taking the contracts attached to the model and generating PEPA automaton. This transformation also uses a 'parameter library' (currently encoded as a set of constants) which represent typical settings of the reliable middleware. These are also used to set default values for parameters which were uninitialized in the high level model. This transformation can be considered as a "PIM-to-PSM mapping" following the MDA conventions.

Finally, pepa2out is a syntactical transformation which generates textual code from the PEPA model. Separating the syntax generation from the semantical mapping enables to develop transformations which are easier to maintain; moreover, the abstract performance model can also serve as the basis of creating input to other stochastic analysis tools.

Fig. 10. Transformation chain

Model transformations are also captured in a textual way by using a combination of (i) graph patterns for querying models, (ii) graph transformation rules for elementary model manipulations, and (iii) abstract state machines for assembling complex transformations from simple rules.

Creating transformation workflow The transformation workflow has also been integrated to the SENSORIA Development Environment (SDE) (it will be available for public download soon). SDE is an Eclipse-based tool which integrates SOA development and analysis tools in a "SOA-style" [16].

Using our transformations and the SENSORIA tools, the developer can perform complex design tasks of SOA systems where analysis questions have to be answered (for instance, whether it is worth using reliable messaging considering expected failure rate and performance constraints). All models and tools can be managed within the same Eclipse environment. Extended with deployment transformations and references to component implementations, the service configuration can directly generated to target execution environments.

4 Related work

A framework for automated WSDL generation from UML models is described in [18], using the UML extensions of MIDAS [3]. In [10], Web service descriptions are mapped to UML models, and (after using visual modeling techniques) a composite service can be created for which the descriptor is automatically generated. However, none of these works considers non-functional properties of Web services.

Non-functional aspects of e-business applications are discussed among others in [1], having some description of deployment optimization for J2EE applications, but without discussing details of model-based deployment.

Integration of non-functional aspects in the development by model transformations is also investigated in [4, 15] and [11], focusing on parts of the engineering process. However, none of these approaches address performability analysis in the context of SOA.

In a service-oriented environment, PEPA has already been used to analyze (application-specific) high-level UML models or workflow-like descriptions of services with Service Level Agreement attributes [21]. In this approach, the authors investigate performance parameters (compared to performability in our case). However, the main essential difference is the (performance-related) behavior of services needs to be modeled explicitly on the UML-level. In contrast, our technique relies only on architectural level UML models, and the core building blocks of (business-independent) performability-related behavior are instantiated in accordance with the UML model of service configurations, which allows better reusability.

Concerning previous work of the same authors, verification of the behavior of the system (i.e., checking the conformance to requirements on reliable messaging) was performed in [9], thus giving a formal semantics which can be checked

by using some basic verification techniques and tools. The process model used for performability analysis in Sec. 3 is derived as an abstraction of this work, concentrating on quantitative measures. A high-level initial overview of the current framework were introduced in [7]; however, the current paper contains significantly more details, and the performability analysis is completely novel. [8] shares some conceptually similar ideas in order to carry out a model-based performance evaluation to analyze BPEL processes with SLA requirements by using DEEM.

5 Conclusions

In the paper, we presented an integrated model-driven framework for the design and analysis of standards-compliant service configurations supporting reliable messaging. Starting from high-level, platform independent service configuration models (captured by service engineers using a UML Profile or a domain-specific editor) performability analysis was carried out by generating stochastic process models for the PEPA toolkit. We used an SLA-like description and (in contrary to existing methods were the original source model had to be enriched by performability parameters) we created a quantitative model which is the basis of precise analysis, helping the designer to estimate the cost and benefit of using reliable middleware.

Transformations were implemented using the VIATRA model transformation framework [20]. As modeling front-end, the IBM Rational Software Architect v7 UML tool was used with appropriate model importers for VIATRA.

A thorough scalability analysis of our approach on a real-life example is also part of our research activities. We will compare the provided performance characteristics of the normal message communication and that of the reliable messaging middleware. Here we anticipate that from the user point of view the average response time will be increased because of the overhead of acknowledgement and possible resending of messages, however, the number of failed messages will obviously decrease.

We also plan to work on the back-annotation of the results to the engineering model. Finally, the same model representation is used as the basis of deployment transformations to standard-compliant platform, such as IBM RAMP, Apache Axis2 extended with Sandesha module and SCA environments with implementations of the Policy framework.

References

1. A. Balogh, D. Varró, and A. Pataricza. Model-based optimization of enterprise application and service deployment. In *ISAS*, pp. 84–98. 2005.
2. L. Baresi, R. Heckel, S. Thöne, and D. Varró. Style-based modeling and refinement of service-oriented architectures. *Software and Systems Modeling*, vol. 5(2):pp. 187–207, 2006.
3. P. Caceres, E. Marcos, and B. Vera. A MDA-based approach for web information system development. In *Workshop in Software Model Engineering (WiSME@UML2003)*. 2003.

4. V. Cortellessa, A. D. Marco, and P. Inverardi. Software performance model-driven architecture. In *SAC '06: Proceedings of the 2006 ACM symposium on Applied computing*, pp. 1218–1223. ACM Press, New York, NY, USA, 2006.
5. S. Gilmore and M. Tribastone. Evaluating the Scalability of a Web Service-Based Distributed e-Learning and Course Management System. In *Workshop on Web Services and Formal Methods (WS-FM 2006)*, Springer-Verlag. 2006.
6. S. Gnesi, M. ter Beek, H. Baumeister, M. Hoelzl, C. Moiso, N. Koch, A. Zobel, and M. Alessandrini. D8.0: Case studies scenario description, 2006. SENSORIA Deliverables Month 12.
7. L. Gönczy, J. Ávéd, and D. Varró. Model-based deployment of web services to standards-compliant middleware. In I. J. M. Pedro Isaias, Miguel Baptista Nunes (ed.), *Proc. of the Iadis International Conference on WWW/Internet 2006(ICWI2006)*. Iadis Press, 2006.
8. L. Gönczy, S. Chiaradonna, F. D. Giandomenico, A. Pataricza, A. Bondavalli, and T. Bartha. Dependability evaluation of web service-based processes. In M. Telek (ed.), *in Proceedings of European Performance Engineering Workshop (EPEW 2006)*, Lecture Notes on Computer Science, pp. 166–180. Springer, Budapest, HUNGARY, 2006.
9. L. Gönczy, M. Kovács, and D. Varró. Modeling and verification of reliable messaging by graph transformation systems. In *Proc. of the Workshop on Graph Transformation for Verification and Concurrency (ICGT2006)*. Elsevier, 2006.
10. R. Gronmo, D. Skogan, I. Solheim, and J. Oldevik. Model-driven web services development. In *Proc. of the IEEE International Conference on e-Technology, e-Commerce and e-Servie (EEE'04)*, pp. 42–45. IEEE Computer Society, Los Alamitos, CA, USA, 2004.
11. H. Jonkers, M.-E. Iacob, M. M. Lankhorst, and P. Strating. Integration and analysis of functional and non-functional aspects in model-driven e-service development. In *EDOC*, pp. 229–238. 2005.
12. N. Koch and D. Brendl. D8.2.a: Requirements Modelling and Analysis of Selected Scenarios - Automotive Case Study, 2007. SENSORIA Deliverables Month 24.
13. N. Koch, P. Mayer, R. Heckel, L. Gönczy, and C. Montangero. D1.4.a: UML for Service-Oriented Systems, 2007. SENSORIA Deliverables Month 24.
14. Object Management Group. *UML Profile for QoS and Fault Tolerance*, 2006. `http://www.omg.org`.
15. S. Röttger and S. Zschaler. Model-driven development for non-functional properties: Refinement through model transformation. In *Proc. The Unified Modeling Language (UML 2004)*, vol. 3273 of *LNCS*, pp. 275–289. Springer, 2004.
16. SENSORIA Development Environment home page, 2007. `http://svn.pst.ifi.lmu.de/trac/sct`.
17. SENSORIA FP6 IST project, 2005. `http://sensoria-ist.eu`.
18. J. M. Vara, V. de Castro, and E. Marcos. WSDL Automatic Generation from UML Models in a MDA Framework. In *NWESP '05: Proceedings of the International Conference on Next Generation Web Services Practices*, p. 319. IEEE Computer Society, 2005.
19. D. Varró and A. Balogh. The model transformation language of the VIATRA2 framework. *Science of Computer Programming*, vol. 68(3):pp. 214–234, 2007.
20. VIATRA2 Framework at Eclipse GMT. `http://www.eclipse.org/gmt/`.
21. M. Wirsing, A. Clark, S. Gilmore, M. Hölzl, A. Knapp, N. Koch, and A. Schroeder. Semantic-Based Development of Service-Oriented Systems. In E. N. et al. (ed.), *Proc. of FORTE'06*, LNCS 4229, pp. 24–45. Springer-Verlag, 2006.