Towards Model-Based Communication Protocol Performability Analysis with UML 2.0

Nico de Wet and Pieter Kritzinger
Department of Computer Science
University of Cape Town
Private Bag Rondebosch 7701
South Africa
Tel: +27 21 650 2663
Fax: +27 21 689 9465
{ndewet,psk@cs.uct.ac.za}

***Abstract*— In this paper we propose a methodology for the modelling, verification and performance evaluation of communication components of distributed application building software. The methodology is centered upon model-driven development using a subset of UML 2.0 diagrams. It is supported by the proSPEX model processing tool which translates UML 2.0 specifications into executable simulation models. In our proSPEX discussion we focus on the translation from a UML 2.0 model to a simulation model. The model-based development of communication components of wireless middleware solutions is discussed as a motivational example.**

I. Introduction

The use of middleware, of which wireless middleware is a specialized subset, has been acknowledged as the principal means of simplifying distributed application building in the enterprise[5]. In recent years a number of wireless middleware products [10], [4], [20] have emerged that use proprietary network protocols when traversing low bandwidth wireless links. Here we consider the verification and performance evaluation[1] of the communication components of such middleware products using UML 2.0 and model-driven development.

It is generally accepted that distributed application building software (henceforth called middleware) needs to be validated and verified. However ensuring that such software both adheres to specifications and is error free is not sufficient when developing quality software. Performance is a fundamental quality attribute of any software and performance analysis is often neglected in the software engineering life-cycle[15]. It has also been accepted that the primary source of performance failures are due to architectural and design problems that can be detected at the early stages of the design process[19].

In this work we introduce a methodology for the modelling and performability analysis of communication components of middleware. The methodology involves the use of a subset of UML 2.0 diagrams to model the architecture and protocol interactions of a communication component. The modelling process itself should be supported by the use of design patterns for protocol system architecture[14]. The model is created in a commercial model editing tool, Telelogic Tau G2, and verified using this tool. Following the Tau-based verification a collaboration diagram depicting a simulation scenario is created by the user as a basis for defining system workloads. The proSPEX (protocol Software Performance Engineering using XMI) tool then imports the model using its filters to Tau G2. It then executes the model, hence providing dynamic verification, and gives performance measures to the user.

In Sect. II we discuss the validation, verification and performance evaluation of communication components of middleware. Model-driven development using UML 2.0 is outlined in Sect. III. Performance modelling and evaluation is discussed in Sect. IV, while in Sect. V, we describe the proSPEX methodology. The proSPEX tool architecture is discussed in Sect. VI while concluding remarks are made in Sect. VII.

II. Validation,Verification and Performance

The communication components of middleware is particularly susceptible to both errors and performance problems due to the complexity of interactions in application and network layer protocols. These errors and performance problems tend to arise primarily due to the temporal dependencies among the participating processes. It is generally accepted that network layer protocols used in middleware and the interaction protocols among enterprise information system components using the middleware should be specified using formal languages[18], [9], [21], [6], thereby allowing high level design verification. Examples of such languages are the Process Meta Language (PROMELA, the system description language of SPIN[8], [18]), the Specification and Description Language (SDL) and Estelle.

UML could also be used as a specification language, however it is a general-purpose language without formal semantics. As a work-around a common approach is to map a subset of UML diagrams to existing formal methods[2], [12], [11] in order to allow automated analysis. An alternative approach is to merge UML with a formal language, as has been done in the International Telecommunication Union Recommendation Z.109 titled ”SDL Combined with UML”[3]. Z.109 is a UML

[1]In this paper we refer to verification and performance evaluation as performability analysis[1], a term encompassing both performance and reliability (or dependability).

profile meaning that it specializes UML using stereotypes, tagged values, constraints and notational elements.

Having established that a communication component is error free, the next step in performability analysis (the construction of quality software) is performance analysis. The formally specified network and component interaction protocols would be analyzed by either analytic evaluation, experimentation or simulation. In proSPEX, the prototype tool supporting our methodology, we use process-based discrete event simulation and statistical performance evaluation. Simulation has the advantage of being able to evaluate protocol performance according to given metrics as well as being useful in aiding in the understanding of protocol interactions[13].

III. Model-Driven Development

Model-driven development[2] is an approach to software development in which the resultant implementation is automatically generated from models. In order to realize model-driven development one needs graphical programming abilities which is the ability to program directly in the modelling language. SDL has been used as a model-driven development language for some time in the telecommunication industry. Part of the attraction of SDL stems from the availability of specialized abstractions, such as signalling, that are useful in model-driven communication software development. The merger of UML 2.0[3] and SDL, via the ITU-T Z.109 Recommendation[3], is a powerful realization of model-driven development geared towards communication software development. The Telelogic Tau G2 tool uses such a merger resulting in the non-standard Telelogic *UML Syntax* that largely resembles SDL.

Using UML 2.0 as a language for model-driven development of communication software is appealing due to it being an evolution of the de facto UML 1.x standard. This evolution has been driven by the need to address deficiencies of UML 1.x noted since UML was first proposed in 1997. These deficiencies include a lack of formal semantics, inadequate semantics definition[17] and excessive size. Of the enhancements offered by UML 2.0, the architectural modelling capabilities are of particular importance when conducting model-driven development of communication components. The architectural modelling capabilities of UML 2.0 are based on mature languages such as SDL and ROOM (Real-Time Object-Oriented Modelling).

Model-driven development of communication components of wireless middleware implementations using UML 2.0 merged with SDL is appealing due to SDL being a formal language with useful protocol engineering abstractions. The appeal also derives from the fact that the language and its higher level abstractions are *target-language-independent*[3]. This means that following verification and validation of a component programming language code such as C, C++ or Java could be generated. The non-standard Telelogic Tau *UML Syntax* is target language independent, meaning that equivalent implementations and simulation models can be generated.

[2] *"The model is the implementation"*[17]

[3] Adopted as an official OMG standard specification in June 12, 2003

IV. Performance Modelling and Evaluation

The performance analysis of communication components specified in a model-driven development language such as UML 2.0[4] requires a clear definition of the semantics of time regardless of the analysis method used. For example it should be clear whether signal transfer over connectors that do not traverse network links take time. Unfortunately such semantics are not not always clearly defined.

In this work, simulation-based performance analysis is conducted to predict a communication system's performance. More specifically, we use process-based discrete event simulation. In such simulation the event list of the simulation scheduler contains processes and the order within the event list (or scheduler queue) is determined by the time of the next events in the processes' event sequences. In addition the processes interact with each other through message passing and other simulation primitives in order to realize the operational path of the system. The semantics of time is therefore embedded within the implementation of the simulation scheduler and simulation primitives. Naturally the semantics of time embedded in simulators must be validated, in the sense that the semantics must be shown to produce performance predictions that can be relied upon.

V. The proSPEX Methodology

The proposed methodology for the modelling, verification and performance evaluation of communication components of software for enterprise information systems is presented in Fig. 1. The steps in our methodology are outlined below.

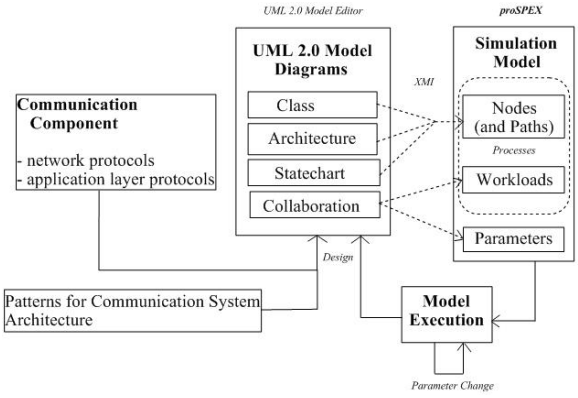

Fig. 1. The proposed methodology supported by the simulation-based proSPEX performance analysis tool

Requirements Definition: The first step is to establish the requirements of the communication component. In the case of a wireless middleware product a primary requirement would be to use the available bandwidth as efficiently as possible. Following requirement definition we identify[5] or design suitable network and application layer inter-component protocols. UML 2.0 use case and sequence diagrams could

[4] Henceforth when we refer to UML 2.0 it is assumed to be specialized for communication software development by the ITU-T Z.109 UML profile.

[5] Requests for Comments (RFC) documents could be used here.

be used to aid understanding but these are not used when generating the simulation model, as can be seen in Fig. 1.

Architecture Specification: The next step is to use a combination of UML 2.0 class and architecture diagrams (with ports, connectors and interfaces) to design the protocol architecture. The use of design patterns for protocol system architecture[14] is recommended at this stage. The focus of this stage is to identify the active classes (classes with their own thread of control) and their interfaces.

Interface-based design has the benefit of both reduced design complexity and giving distributed teams the ability to work concurrently while using the interface as a contract. In UML 2.0 an interface is a classifier representing a declaration of a set of public features and obligations[7]. Interfaces are not instantiable, instead they are either *provided* or *required* by a classifier such as a class. When a class provides an interface it carries out its obligations to clients of instances of the class. When a class requires an interface it means that it needs the services specified in the interface in order to perform its function and fulfill its own obligations to its clients. The notation introduced for a provided interface is a full-circle lollipop whilst the notation introduced for a required interface is a semi-circle lollipop.

Fig. 2 shows the architecture diagram of an active class with two *parts*, namely any number of Sessions and a single RoutingPeerProxy. The parts are linked with *connectors* that are attached to *ports*. Note that notationally ports are the squares to which the required interfaces, provided interfaces and connectors are attached. Each *port* serves the duel purpose of being used to group an active class's related interfaces and also acting as interaction (or connecting) points through which the services of a class can be accessed. In the architectural view of an active class we want to be able to distinguish between behaviour that is delegated to the class itself and behaviour that is delegated to its parts. Connectors terminating in a behaviour port mean that the signals sent to the port are handled by the containing class. Notationally a behaviour port is represented by a state symbol attached to a square port symbol, as can be seen in Fig. 2.

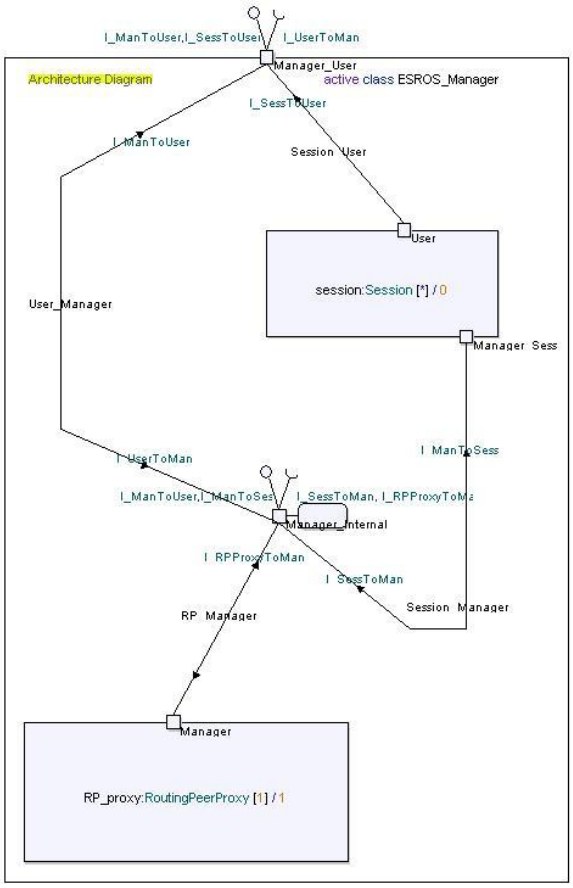

Fig. 2. Architecture specification with UML 2.0

Behaviour Specification: Following the architectural specification we specify the detailed behavior of active classes by implementing state machines using statechart diagrams. As discussed in section III, we use specialized communication abstractions derived from SDL in this model-driven development process. Fig. 3 shows a part of a UML 2.0 statechart diagram, note that the syntax used is the Telelogic Tau *UML Syntax* derived from SDL. Once this stage is complete the software is verified using facilities provided by the model editing tool, in our case Telelogic Tau G2.

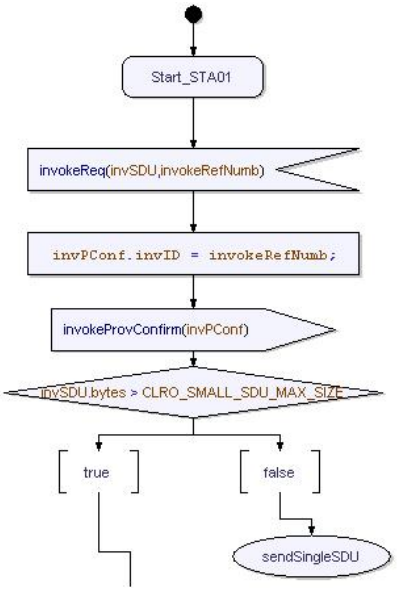

Fig. 3. Behaviour specification with UML 2.0

Simulation Scenario Specification: Once the software has been verified the performance analysis phase commences which starts with the modelling of the environment of the communication component. That is, we create client and server (or peer) active classes and their associated state machines. A collaboration diagram (see Fig. 4) is then drawn

up illustrating a simulation scenario which in combination with the statechart diagrams of the client(s) and server(s) serve as the workload. This scenario would indicate the number of clients and servers and also network link characteristics (loss probability, bandwidth and delay distribution). Once the scenario has been completed the proSPEX tool user imports the model from which a semantically equivalent simulation model is generated.

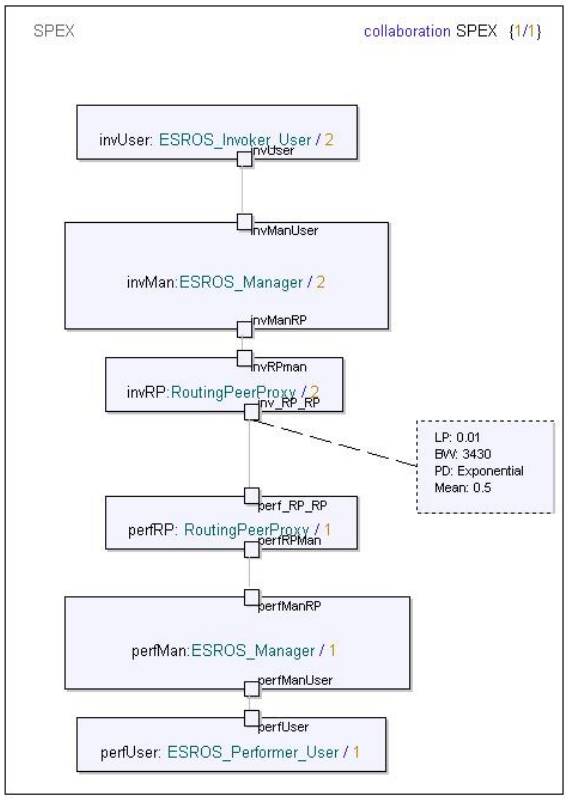

Fig. 4. Simulation scenario and workload specification

Results: The events and corresponding trace messages that the simulator is able to generate dictate the set of performance statistics that can be calculated. The simulation model generated by proSPEX is able to generate the following types of trace messages[6]:

1) Message M sent via Connector C from process P1 to process P2 at time t
2) Message M from process P1 read by process P2 at time t
3) Message M from process P1 arrives in queue of process P2 at time t
4) Process P created at time t
5) Process P destroyed at time t
6) Overflow: message M from P1 to P2 discarded at time t
7) Process P has transition from state S1 to state S2 at time t
8) Message M from process P1 discarded by process P at time t
9) Timer T set to duration d in process P at time t
10) Timer T reset in process P at time t
11) Timeout: Timer T in process P at time t

The performance measures that can be calculated from analysis of simulation traces containing the above mentioned messages includes:

1) Mean queue waiting time
2) Connector throughput
3) Mean and maximum queue length
4) Detection of queue overflows
5) Throughput of a state
6) Discarded signals
7) List Unreachable states
8) Average time spent blocked in a state for a signal
9) The lifetime of a process
10) Timeout reset and expiration ratios

Naturally any analysis results would refer to the steady-state behaviour of the system and would be computed with confidence intervals. These measures would then prompt the user to either change the simulation parameters or the model itself.

VI. The proSPEX Tool Architecture

In this section we give a general overview of the proSPEX tool architecture and certain technical issues encountered when translating a UML 2.0 model to an executable simulation representation. We also motivate our design decisions and report on the manner in which we overcame challenges.

With proSPEX our intention was to create a model-processing tool and not a model editor since developing an editor would deviate from the primary objective of the project. Telelogic Tau G2 offered an XML-based model file format which was sufficient for our purposes, although the standard XML Metadata Interchange (XMI) 2.0 file format would have been preferable, since this would theoretically allow any future UML 2.0 editor to be used. We had to filter the Telelogic Tau XML and place the filtered aspects into data structures that can be used for simulation code generation. With the Tau XML being extremely verbose this was not a trivial task.

We were faced with the option of either developing a process-based discrete event simulator from the ground up or to use existing simulation packages. A review of the available simulation packages showed that Simmcast[13], an object-oriented framework for network simulation, would be ideal. Simmcast is specifically intended to be used in research environments with limited resources, as the excerpt from [13] shows:

> ...the complete development of a dedicated simulation tool from scratch is not practical, since the amount of resources dispensed in such a project would detract the researcher's focus from the project.

[6]For brevity we use the term *process* instead of *active class*

Simmcast offers extensible building blocks (such as nodes, paths, network and packet) that are combined to describe the simulated network environment. Nodes, each of which are uniquely identified by an integer and contains at least one thread of execution, are the fundamental interacting entities and are connected via paths. The user extends the Node class, via inheritance and places protocol logic and simulation action primitives (such as send, receive, setTimer, sleep) in the extended class.

Despite offering a framework with extendible building blocks we found the need to extend the list of simulation action primitives in order to accommodate required actions such as process creation and termination. Simmcast does not offer such primitives since a Simmcast simulation experiment is defined using a simulation description file that specifies the network topology and startup parameters. We extended Simmcast to generate simulation traces with the messages mentioned in Sec.V.

An additional technical issue that had to be overcome in the translation process involved addressing. During the translation from an UML 2.0 model to a (modified) Simmcast simulation model we had to map concepts such as *Pid* (process identifier) expressions[7], which can either be *self, parent, offspring or sender*, to Simmcast simulation code.

In the Simmcast code generation process we found the need to use templates, as can be seen in Fig. 5. The templates are fed into a text templating engine in order to insert dynamic content into prewritten Simmcast source code. Text templating engines are essential tools in code generation as they solve the problem of inserting dynamic content into prewritten text. Our chosen text templating engine, the Velocity Template Engine[16]), is used for Java implementation code generation in the popular Poseidon UML tool created by Gentleware AG.

VII. Conclusion

The verification and validation of formally specified communication software has been accepted as being vital in overcoming the complexity of interactions in application and network layer protocols. In this work we have argued that in addition to correctness, performance is a vital quality attribute of communication software. In support of this view we have presented a model-driven methodology for the performability analysis of the communication components of software for EIS. This methodology is supported by the proSPEX performance analysis tool.

In developing our methodology we found that UML 2.0 class, architecture and state chart diagrams were necessary to define the architecture and behaviour of communication software. The use of patterns for communication software architecture proved to be useful in the model-driven development of the architectural aspects of network and application layer protocols. We found that simulation scenarios and network parameters (loss probability, bandwidth and delay distributions) could be specified by using UML 2.0 collaboration diagrams.

[7]These expressions are derived from SDL and incorporated into UML 2.0 via the ITU-T Z.109 Recommendation

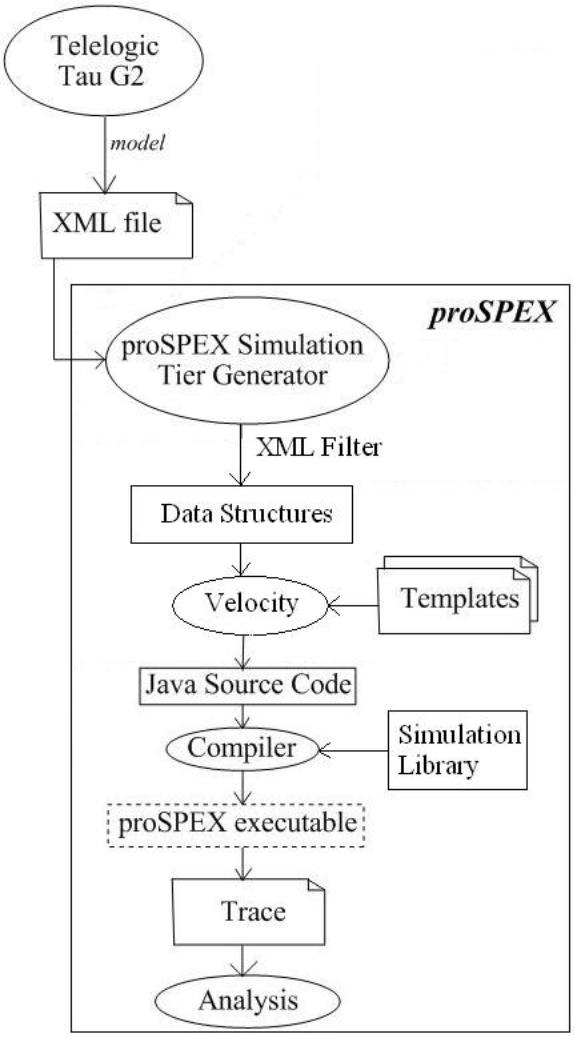

Fig. 5. The proSPEX architecture

In addition to presenting our methodology we have highlighted the architectural aspects of the proSPEX tool which takes advantage of XML-based application integration and an extendible simulation framework, namely Simmcast. We found it necessary to extend the set of simulation primitives offered by Simmcast in order to allow for dynamic node (or active class) creation and termination. At the same time we found that the UML 2.0 communication abstractions, offered by extending UML 2.0 with SDL actions, map readily to Simmcast simulation primitives.

We have used and extended an existing simulation framework and focused our efforts on building a model-processor and not an editor. We hope for these aspects to serve as an example of the efficient use of resources in the research environment. Ongoing work on proSPEX includes development on state machine code generation and a user interface.

VIII. Biography

Nico de Wet hold a BSc Honours degree in Computer Sci-

ence, *Cum Laude*, from the University of Cape Town, where he is currently completing an MSc Computer Science dissertation on *communication protocol performance engineering using UML 2.0*. He aims to become an entrepreneur following the completion of his MSc.

REFERENCES

[1] A Avizienis, J C Laprie, and Randell. Fundamental concepts of dependability. In *Proc. of the 3rd Information Survivability Workshop*, pages 7–12, 2000.

[2] S Bernardi, S Donatelli, and J Merseguer. From uml sequence diagrams and statecharts to analysable petri net models. In *Proceedings of the Third International Workshop on Software and Performance*, pages 35–45, New York, USA, 2002. ACM Press.

[3] M Bjorkander. Graphical programming using uml and sdl. *IEEE Computer*, 33(12):17–22, December 2002.

[4] BROADbeam. Broadbeam: Mobile application development, wireless software development. http://www.broadbeam.com/index.asp, 2004.

[5] W Emmerich. Software engineering and middleware: A roadmap. In *Proceedings of the conference on The future of Software engineering*, pages 117–129, New York, USA, 2000. ACM Press.

[6] X Logean et. al. On applying formal techniques to the development of hybrid services: Challenges and directions. *IEEE Communications Magazine*, 37(7):132–138, July 1999.

[7] Object Management Group. Uml 2.0 superstructure specification. Object Management Group Online Publication, August 2003.

[8] G Holzmann. *Design and Validation of Computer Protocols*. Prentice Hall, 1991.

[9] G J Holzmann. Protocol design: Redefining the state of the art. *IEEE Software*, 9(1):17–22, January 1992.

[10] Softwired Inc. Pure java messaging solutions for electronic business. http://www.softwired-inc.com/, 2004.

[11] L Lavazza, G Quaroni, and G Venturelli. Combining uml and formal notations for modelling real-time systems. In *Proceedings of the 8th European Software Engineering Conference*, pages 196–206, New York, USA, 2001. ACM Press.

[12] W E McUmber and B H C Cheng. A general framework for formalizing uml with formal languages. In *Proceedings of the 23rd international conference on Software engineering*, pages 433–442. IEEE Computer Society, 2001.

[13] H Muhammad and M Barcellos. Simulating group communication protocols through an object-oriented framework. In *Proceedings of the 35th Annual Simulation Symposium*, pages 14–18, San Diego (New York), 2002. IEEE.

[14] J Parssinen and J Turunen. Patterns for protocol system architecture. In *Pattern Languages of Programs (PLoP) Conference*, 2000.

[15] R Pooley. Software engineering and performance: A road-map. In *Proceedings of the conference on The future of Software engineering*, pages 189–199, New York, USA, 2000. ACM Press.

[16] The Apache Jakarta Project. The apache jakarta project: Velocity. http://jakarta.apache.org/velocity/, 2004.

[17] B Selic. Brass bubbles: An overview of uml 2.0 (and mda). Tutorial presented at OTS'2003 18-19 June 2003, 2003.

[18] S Sircar and A Kott. Enterprise architecture analysis using an architecture description language. In *Proceedings of DARPA Symposium on Advances in Enterprise Control*, 2000.

[19] C U Smith and M Woodside. Performance validation at early stages of software development. In *Performance 99*, 1999.

[20] Spiritsoft. Spiritsoft: Go beyond jms. http://www.spiritsoft.com, 2004.

[21] M Steppler. Performance analysis of communication systems formally specified in sdl. In *Proceedings of the First International Workshop on Software and Performance (WOSP98)*, pages 49–62. ACM Press, 1998.