Using UML Models for the Performance Analysis of Network Systems

Nico de Wet and Pieter Kritzinger

Data Network Architectures Group
Computer Science Department
University of Cape Town, South Africa
e-mail: {ndewet,psk}@cs.uct.ac.za

Abstract

The automated functional and performance analysis of communication systems specified with some Formal Description Technique has long been the goal of telecommunication engineers. In the past SDL and Petri nets have been the most popular FDTs for the purpose. With the growth in popularity of UML the most obvious question to ask is whether one can translate one or more UML diagrams describing a system to a performance model. Until the advent of UML 2.0, that has been an impossible task since the semantics were not clear. Even though the UML semantics is still not clear for the purpose, with UML 2.0 now released and using ITU recommendation Z.109, we describe in this paper a methodology and tool called proSPEX (protocol Software Performance Engineering using XMI), for the design and performance analysis of communication protocols specified with UML.

Key words: Formal Description Technique, UML, SDL, Performance Analysis, Communication Protocols, Simulation, ESRO

1 Introduction

While UML has become a *de facto* modeling standard, it is not often employed in the protocol engineering process as a specification language. This is primarily because it is a general-purpose modeling language without formal semantics.

With the emerging UML 2.0 standard the Object Management Group (OMG) appears to have addressed the shortcomings of UML in the real-time modeling and protocol engineering domains. Notably the architectural modeling capabilities of UML 2.0 has been drawn from both the ROOM modeling

language[SGW94] and SDL[Hog89]. For instance, the UML 2.0 architecture diagrams show the hierarchical subdivision of *active classes* and are composed of *parts* (which are also active classes), *provided* and *required interfaces*, *ports* and *connectors.*

While UML 2.0 does provided enhanced architectural modeling capabilities it is not a formal language and as such does not posses formal semantics or a syntax. Tool vendors have worked around the problem by applying the ITU Recommendation Z.109 [MP00] "SDL Combined with UML" to UML 2.0. This recommendation is for a UML profile meaning that it specializes UML using stereotypes, tagged values, constraints and notational elements. By applying the Z.109 profile to UML 2.0 the abstractions, tightened semantics and syntax that are found in SDL can be used when specifying the behavior of active classes using state machines.

When building performance models using UML 2.0 enhanced with the Z.109 profile, previous approaches to performance analysis [BMSK96][MTMC99] that incorporate temporal aspects into SDL specifications, can be applied. UML 2.0 offers several diagram types with different system views which may be semantically equivalent, as is the case with sequence and state machine diagrams. In the proSPEX methodology, we use a minimal subset of UML 2.0 diagrams to specify communication protocols as well as the environmental constraints and workload associated with a particular system scenario. These diagrams are then translated to a process-based discrete event simulation model.

The modeling process itself should be supported by the use of design patterns for protocol system architecture[PT00]. 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. This diagram serves as a basis for defining system workloads and for specifying non-functional time dependent aspects. The proSPEX tool then imports the model using its filters to Tau G2. It then executes the model and gives performance measures to the user.

In Section2 we discuss the validation, verification and performance evaluation of communication software. Model-driven development using UML 2.0 and SDL is outlined in Section3. We then discuss communication system performance modeling with SDL in Section4 and with UML 2.0 in Section5. The proSPEX methodology, architecture and semantic time model are discussed in Section6 to Section8 respectively. A performance analysis case-study is discussed in Section9 while concluding remarks are made in Section10.

2 Validation, Verification and Performance

Communication software 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 communication software should be specified using formal languages[SK00] [Hol92][Ste98][ea99] in order to allow automated analysis. Examples of such languages are the Process Meta Language (PROMELA, the system description language of SPIN[Hol91,SK00]), 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 [BDM02][MC01][LQV01] 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"[Bjo02]. Z.109 is a UML 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 the construction of reliable, 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[MB02].

3 Model-Driven Development

Model-driven development or "...the model is the implementation ..." according to [Sel03, Selic] 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 modeling 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 [1] and SDL, via the ITU-T Z.109 Recommendation[Bjo02], 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[Sel03] and excessive size. Of the enhancements offered by UML 2.0, the architectural modeling capabilities are of particular importance when conducting model-driven development of communication components. The architectural modeling capabilities of UML 2.0 are based on mature languages such as SDL and ROOM (Real-Time Object-Oriented Modeling).

Model-driven development of communication software 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*[Bjo02]. 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.

4 Performance Modeling with SDL

In this work we use UML 2.0 (extended with the ITU Z.109 profile) in which SDL state machines are used for model-driven behavioral specification. As such we review performance modeling issues and approaches to resolving these issues in the context of SDL.

It has been acknowledged that performance-enhanced extensions of standard SDL are necessary when modeling non-functional duration constraints. In various approaches to performance analysis with SDL, the semantic time model of SDL is enhanced by providing means of modeling non-functional time dependant aspects. Such semantic time models are realized by time related features that are needed for functional design and also by time related features that are needed for non-functional aspects and analysis.

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

Time related features required for functional design include clocks, timeouts and time dependant enabling conditions. Time related features required for non-functional design include timing restrictions due to knowledge of the execution environment and modeling the execution times of tasks.

The means of modeling non-functional time dependant aspects are missing from SDL [Gra02][Spi97], as is noted by Graf [Gra02]:

> *Non-functional primitives express timing features orthogonal to the functional behavior, and they consist in constraints on the (relative) occurrence time of events, and are completely lacking in the standard.*

It is this lack in the standard which is the subject that is addressed by the various approaches [BMSK96][Rou][Ste98] [MDMC96][MHSZ96][Spi97] to performance analysis using SDL. Each approach provides a means of modeling duration constraints that allow for the expression of timing characteristics of the environment and underlying execution system [Gra02]. Non-functional time related aspects include [2] [Gra02]:

- **Communication delays**: all communication in SDL occurs via channels which may have an associated delay. Channel attributes may include a loss rate and whether the delay is load dependent or not. A communication channel with parallelism, such as the Internet, may be regarded as load independent, while a sequential medium would be load dependent.
- **Processing times**: the processing of a signal can be divided into queueing and *treatment* [Gra02] phases. The treatment time consists of pure execution and blocking time (due to scheduling). The overall processing time can be modeled as an expression representing a time interval. With SDL an important question that arises is for which sort of behaviors duration constraints can be specified. For example are durations constraints associated with SDL behavioral primitives (i.e. tasks, output, input etc.), SDL behavior sequences (i.e. transitions or procedures), or SDL processes?
- **Execution modes**: with execution modes we consider time passage in parts of the system with no time constraints expressed. With standard SDL semantics time passage is interpreted as passing arbitrarily in such parts. A designer could specify a different execution mode, for example all non time constrained actions could be immediate.
- **Time constraints on the external environment**: the timing constraints of signals arriving from the environment must be expressible. Such characteristics include response time, inter arrival times and jitter. The environment can be modeled by processes in which the above mentioned signal characteristics can be expressed using time guards.

[2] We borrow from work [Gra02] by Susanne Graf in the list of non-functional time related aspects that are discussed.

– **Scheduling**: in order to represent scheduling algorithms in SDL information regarding the preemtability of atomic steps, or sequences of atomic steps, must be provided. Questions of how or if scheduling information should be represented in SDL is answered to varying degrees by the different approaches.
– **Local time**: the ability to express local clock time *and* global system clock time (the external reference time, or ***now***) is important in model-checking in order to detect unforseen errors such as livelock and deadlock. Moreover, the relationship between local time and the reference time must be clearly defined.

Non-functional duration constraints need to be represented to allow for automated performance analysis. The resultant issue that we consider is whether the standard SDL syntax is amended in such a representation when using a particular approach to performance analysis.

The importance of this consideration is that if one wants a tool based on an approach to be useful to the largest possible audience one would want to take existing SDL specifications and analyze them using the tool *without* having to change the given specification. In the context of this issue we examine the means of attributing performance analysis directives (e.g. delay and scheduling directives) to communication protocol models.

We review the SPECS, ObjectGEODE and QUEST approaches to performance analysis using SDL. In our review we give particular consideration to the means of representing non-functional duration constraints and whether existing SDL specifications can be analyzed without change.

4.1 SPECS: SDL Performance Evaluation of Concurrent Systems

With the SPECS tool [BMSK96] a protocol system specified using standard SDL is imported and then attributed with environmental constraints. Relative execution speed values are assigned to each block while the processes within a block are given weights. The assignment of these values, which is done using a GUI dialog box, is equivalent to annotating the model using comment symbols. Hence existing SDL/PR specifications can be analyzed since the SDL standard has not been molested.

The units of the execution speeds are *actions per time unit* meaning that the number of actions each process can execute (the process *action quota*) once scheduled is determined by its weight. In this way time, which is maintained by a global simulation clock, is advanced either when process instances have exhausted their action quotas or process instances are all waiting for input. At each advancement of the simulation clock, timers, which are maintained

separately by each process instance, are checked for expiry.

With regards to the semantics of time when considering signal transmission, SPECS enhances the standard semantics of SDL by assuming that signal transfers over channels experience a randomly distributed delay and may be lost with a certain probability.

4.2 ObjectGEODE: The SDL simulator

With the ObjectGEODE SDL Simulator [Rou] extensions to the SDL language are used to model non-functional time dependant aspects. These extensions are however used exclusively in SDL comment symbols and hence the annotated models still conform to the Z.100 standard.

Processing delay timing constraints (or delays) are associated with individual actions (and not transitions). *NODE*, *PRIORITY* and *DELAY* directive are used with ObjectGEODE. The PRIORITY directive is used as an alternative to the default scheduling algorithm which is a random uniform choice among the fireable transitions of all process instances of a node. With the DELAY directive the execution duration of SDL actions can be specified using some random distribution.

4.3 QUEST: The Queueing SDL Tool

With the QUEST [MDMC96] approach to the specification of non-functional properties, the SDL language is extended, resulting in the QSDL (Queueing SDL) language. QSDL has a QSDL/GR notation which has equivalent diagrams for each SDL/GR symbol. An SDL process is a *machine* (queueing station) in QSDL and the parameters that can be associated with a *machine* includes a name, server number, service discipline (e.g. FCFS, RANDOM), a set of offered services and service-specific speed values. Each QSDL *request* instruction is time consuming and requires a service amount attribute and an optional priority. In this way time durations and the use of resources can be associated with *certain* actions.

For workload characterization, a number of random distribution functions are provided. These functions would be used in load generators which are implemented as QSDL processes.

5 UML 2.0 and Communication Software Performance Modeling

With UML the original intention was a language for specifying, visualizing, constructing, and documenting the artifacts of software systems [BRJ98]. UML has over nine diagram types, some of which are semantically equivalent, for the mentioned purposes. As we have mentioned, UML is not often employed in the protocol engineering process primarily because it is a general-purpose modeling language without formal semantics. With UML 2.0, the Object Management Group (OMG) intended to address various shortcomings [Kob02][Dor02] [Mil02] in the standard that have been noted since its inception.

Although UML 2.0 is not a formal language, it is an attractive language in the protocol engineering field due to enhanced architectural modeling capabilities which have been drawn from both the ROOM modeling language[SGW94] and SDL[Hog89]. In addition, by applying the ITU Recommendation Z.109 [MP00] "SDL Combined with UML" to UML 2.0, the abstractions, tightened semantics and syntax that are found in SDL state machines become available.

As we have noted, when building performance models using UML 2.0 enhanced with the Z.109 profile, previous approaches to performance analysis [BMSK96][MTMC99] that incorporate temporal aspects into SDL specifications, can be applied. These previous approaches dealt with SDL and not UML 2.0 and so Use Case, Collaboration, Sequence, Activity, Deployment (and other diagrams) were not available. Questions that may therefore arise include which UML 2.0 diagrams should be used when modeling non-functional duration constraints and scenario-based workloads? In addition the UML Profile for Schedulability, Performance and Time [Gro02] (UML-RT profile) should be considered.

With proSPEX we have taken the approach of using the minimal subset of diagrams when specifying communication protocols and subsequently annotating the specifications with non-functional environmental constraints. In this approach state machine diagrams, architectural diagrams and collaborations diagrams are used as the basis of an executable performance model. We have chosen not to make any syntax changes in the state machines (as is done in [BMSK96][Rou]) and have therefore used an approach in which all non-functional constraints are specified using collaboration diagrams annotated with performance constraints. Our methodology, tool construction approach and consideration of the UML-RT profile are discussed in subsequent sections.

6 The proSPEX Methodology

The proposed methodology for the modeling, verification and performance evaluation of communication software is presented in Figure 1. The steps in our methodology are outlined below.

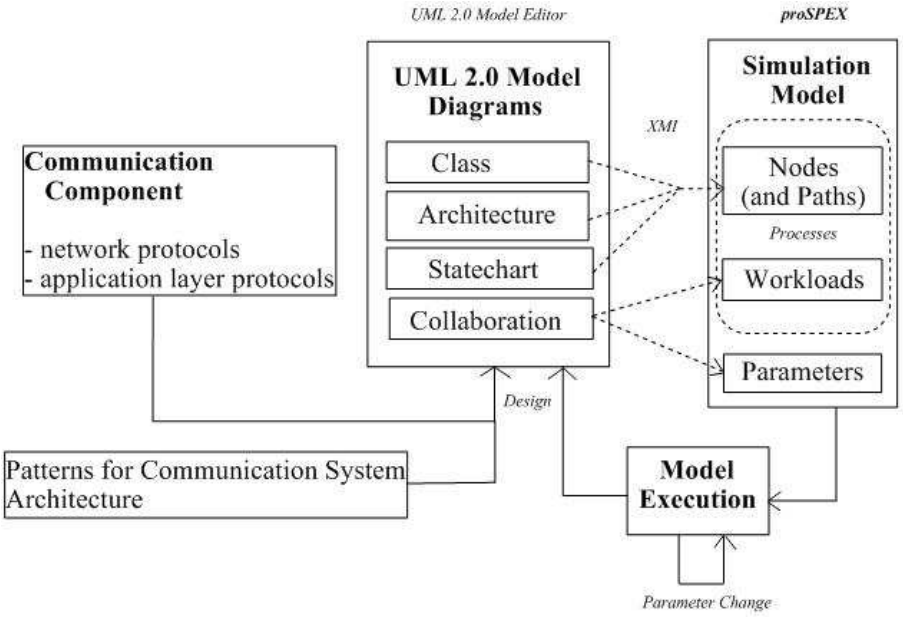

Fig. 1. The proposed methodology supported by the simulation-based proSPEX performance analysis tool

Requirements Definition: The first step is to establish the requirements of the communication component. In the case of a transport layer protocol, a requirement may be to use the available bandwidth as efficiently as possible. Following the requirements definition we identify or design suitable network and application layer inter-component protocols. UML 2.0 use case and sequence diagrams could be used to aid understanding but these are not used when generating the simulation model, as can be seen in Figure 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[PT00] is recommended at this stage. The focus is on identifying the active classes, i.e., 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[Gro03]. Interfaces are not instantiations, 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.

Figure 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 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 behavior that is delegated to the class itself and behavior that is delegated to its parts. Connectors terminating in a behavior port mean that the signals sent to the port are handled by the containing class. A behavior port is represented by a state symbol attached to a square port symbol, as can be seen in Figure 2.

Behavior Specification: Following the architectural specification we specify the detailed behavior of active classes by implementing state machines using statechart diagrams. As discussed in section 3, we use specialized communication abstractions derived from SDL in this model-driven development process. The subset of SDL employed to represent behavior is therefore finite state machines as defined in the standard [3] .

Figure 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.

Simulation Scenario Specification: Once the software has been verified the performance modeling phase commences. With proSPEX non-functional timing annotations are embedded in UML 2.0 comment symbols, thereby allowing for the use of commercial modeling tools for specification and validation.

The performance modeling phase starts with the modeling 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 Figure 4) is then drawn up illustrating a simulation scenario which in combination with the statechart diagrams of the client(s) and server(s) serve

[3] Note that the degree to which Telelogic Tau G2, the editor we used, conforms to any version (SDL-88, SDL-92 or SDL-2000) of the SDL standard is not explicitly given.

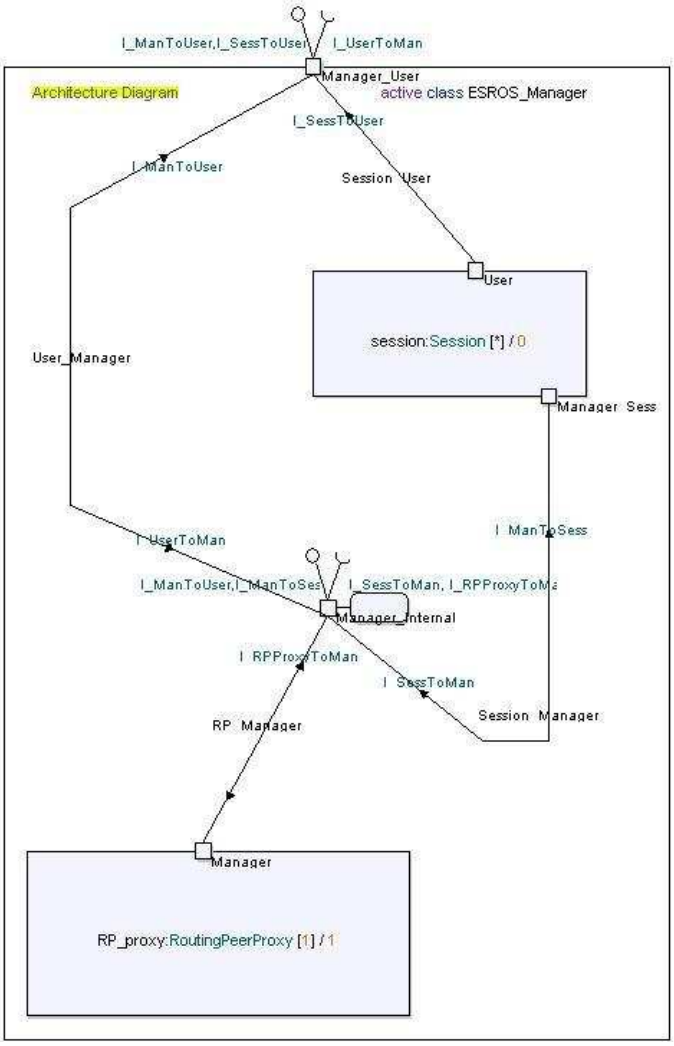

Fig. 2. Architecture specification with UML 2.0

as the workload.

This collaboration diagram would indicate the number of clients and servers and also network link characteristics (loss probability, bandwidth and delay distribution). Processing delay timing constraints (or delays) are associated with active classes and may be deterministic of randomly distributed. The network link and processing delay parameters are specified using comment symbols.

Once the scenario has been completed the proSPEX tool user imports the model from which a semantically equivalent simulation model is generated. *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

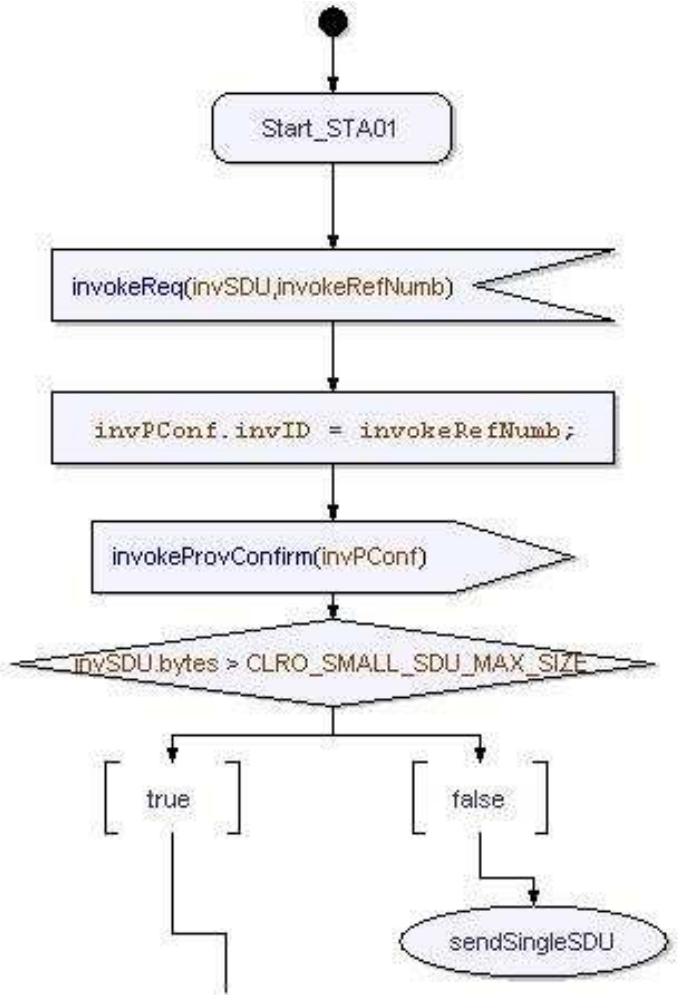

Fig. 3. Behavior specification with UML 2.0

types of trace messages [4] for some general time t_i:

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

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

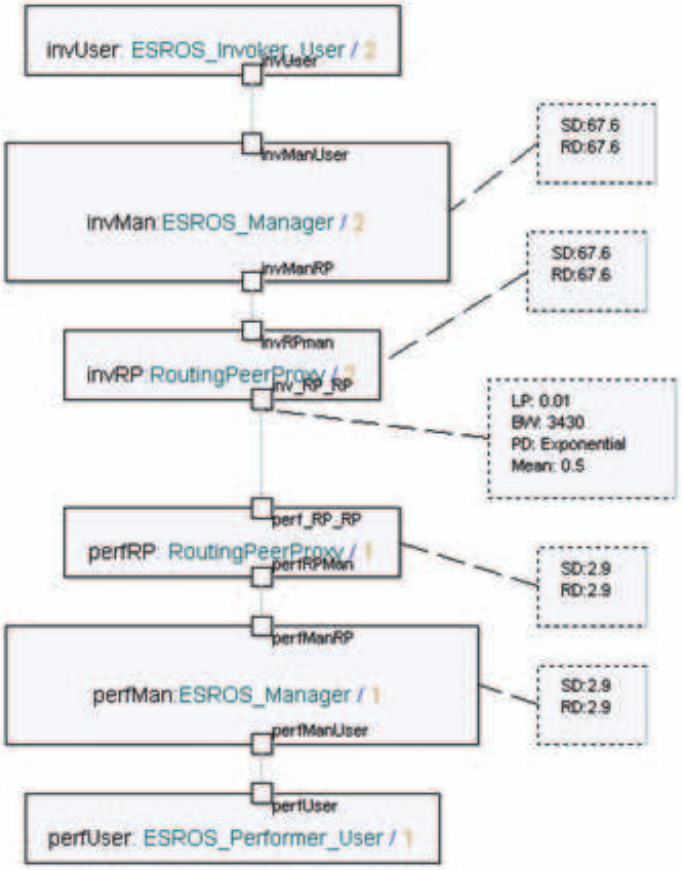

Fig. 4. Simulation scenario and workload specification

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

(1) **Mean queue waiting time**: this is the average time that a signal spends in the queue of a process. A high mean queue waiting time means that *process response time may be too slow* or that there are too many retransmission messages in the queue of a process as a result of the *timeout of the sending process being too short*. Trace messages 2, 3 and 8 are used in the calculation of this statistic.
(2) **Connector throughput**: this is *the traffic on a connector* and trace message 1 is used in its calculation.
(3) **Mean and maximum queue length**: The buffers of a communication system are often modeled using process queues. A high maximum queue length indicates that the *system requires large buffers*. Trace messages 2, 3 and 8 are used to calculate this statistic.
(4) **Detection of queue overflows**: queue overflow is indicated by trace message 6 and shows that the *process' buffers are too small*.
(5) **Throughput of a state**: this statistic shows how many times a state is reached and hence which program parts are frequently processed. States

with a *high throughput may indicate process bottlenecks*. Trace message 7 is used in the calculation of this statistic.

(6) **Discarded signals**: Such signals may either be caused by *insufficient process buffer size or as a result of being sent to processes that no longer exist*. Trace messages 6 and 8 show discarded signals.

(7) **List Unreachable states**: Process' states that are never reached indicates dead code however since simulation is not exhaustive it is not guaranteed that the code is actually unreachable. Trace message 7 is used in determining unreachable states.

(8) **Average time spent blocked in a state for a signal**: This time period shows the *idle time of a process* and records how long a process spends waiting for a signal. Trace message 7 is used in its calculation.

(9) **The lifetime of a process**: This statistic is used in other statistical calculations and uses trace messages 4 and 5.

(10) **Timeout reset and expiration ratios**: The first ratio is that of the number of timeouts set to the number of timers expired. It shows what proportion of timeouts were exceeded. The second ratio is that of the number of timeouts in a queue to the number of timers set. It shows how many timeout messages in the queue had to be reset. Both ratios *show whether timeouts in the system are set at sub-optimal values* and are very useful in improving the performance of protocol systems. Trace messages 9, 10 and 11 are used in the calculation of the mentioned ratios.

Naturally any analysis results would refer to either the steady-state or transient behavior 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.

7 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

rather 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[MB02], 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 [MB02] 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.*

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 Section 6.

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 [5] , 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 Figure 5. The templates act as input 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[Pro04]), is used for Java implementation code generation in the popular Poseidon UML tool created by Gentleware AG.

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

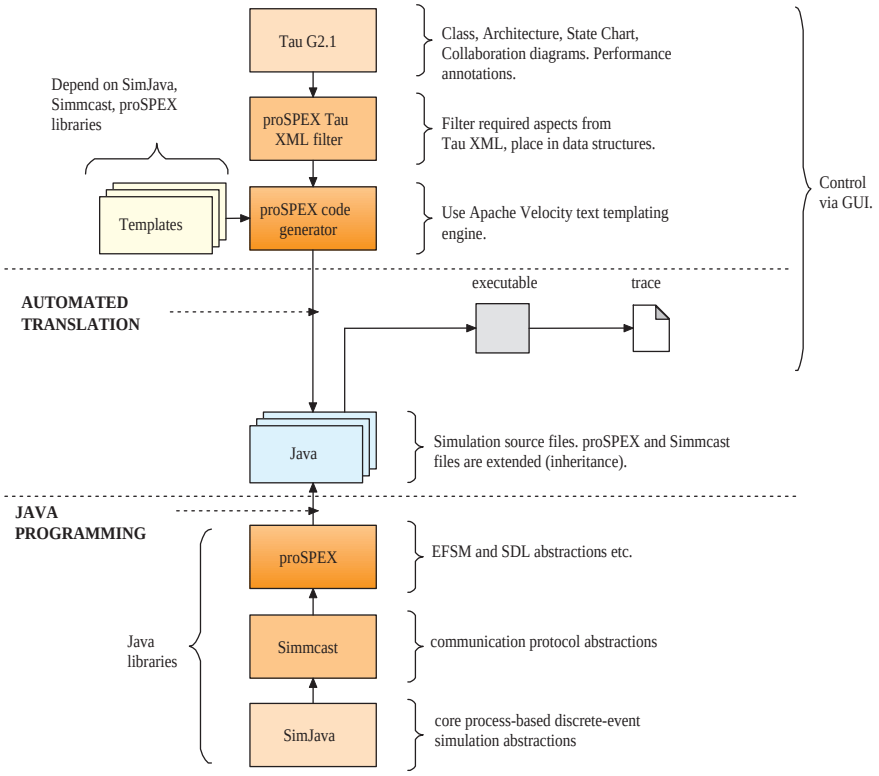

Fig. 5. The proSPEX architecture

8 The proSPEX Semantic Time Model

In Section4 we noted that performance-enhanced extensions of standard SDL are necessary when modeling non-functional duration constraints. Here we explicitly mention how these non-functional time related aspects are represented in proSPEX.

- **Communication delay**: with proSPEX communication delay is associated with packets that traverse network links. Such links have an associated propagation delay (modeled with a random distribution), bandwidth and loss probability. When a packet (or signal) is sent across a network link, delay is applied in two stages before it reaches the receiving queue of the target process. In the first stage the packet is delayed by a *sending time*, which is calculated using the bandwidth and a packet byte size attribute. In the second stage a *propagation delay* is applied and loss probability is applied, with the propagation delay being drawn from a random distribution.
- **Processing times**: with proSPEX we have used the facilities of the underlying simulation framework and hence a *sending* and *receiving* processing delay is associated with active classes. Active classes that have such delays specified have their execution blocked by the specified delay values whenever a signal is sent and received. In other words the sending and receiving

delay associated with an active class is effectively mapped to each input and output operation. Such delay is deterministic by default, although the use of random delay distributions is possible.

Future extensions to proSPEX will be to allow for processing delay to be associated with individual actions and transitions, as is done in the object-GEODE [Rou] approach, using annotated comment symbols.

- **Execution modes**: with execution modes we consider time passage in parts of the system with no time constraints expressed. With proSPEX only input and output actions are time constrained, and hence all non time constrained actions are immediate.
- **Time constraints on the external environment**: with proSPEX the environment is modeled by processes in which signal characteristics can be expressed using time guards.
- **Scheduling**: scheduling information is not represented with proSPEX. Process scheduling is determined by the order of the individual process event sequences.

9 Case study

In order to illustrate the application and utility of our methodology we studied the performance of the Efficient Short Remote Operations (ESRO) transport protocol in the context of the network scenario illustrated in Figure 6. The scenario is one in which ESRO is used in a *credit card authorization application* in which a number of clients invoke operations on a single server acting as the ESRO server. In this scenario multiple mobile stations are linked (via a GPRS network) with an e-commerce server using ESRO as the transport protocol.

The service ESRO offers is a reliable connectionless transport for *wireless* links when efficiency is of concern. The service supports applications based on a remote operations model that is largely the same as the Remote Procedure Call (RPC) model [Mic88]. Service data units (SDU) are segmented into protocol data units (PDU), each of which are encapsulated into a UDP datagram. The simplicity of the protocol lies in the retransmission strategy, which is that if an SDU is segmented, the retransmission strategy is not applied to individual lost segments, the whole SDU is retransmitted.

Since we adhered to the proSPEX methodology in the specification of the ESRO protocol, we specified static aspects using UML 2.0 class and architecture diagrams and dynamic aspects using SDL state machines. Once the specification of the protocol was complete we specified the temporal attributes of processing, network and workload delay parameters using a combination of collaboration and state machine diagrams. Processing and network delay parameters were specified in accordance with the proSPEX semantic time model

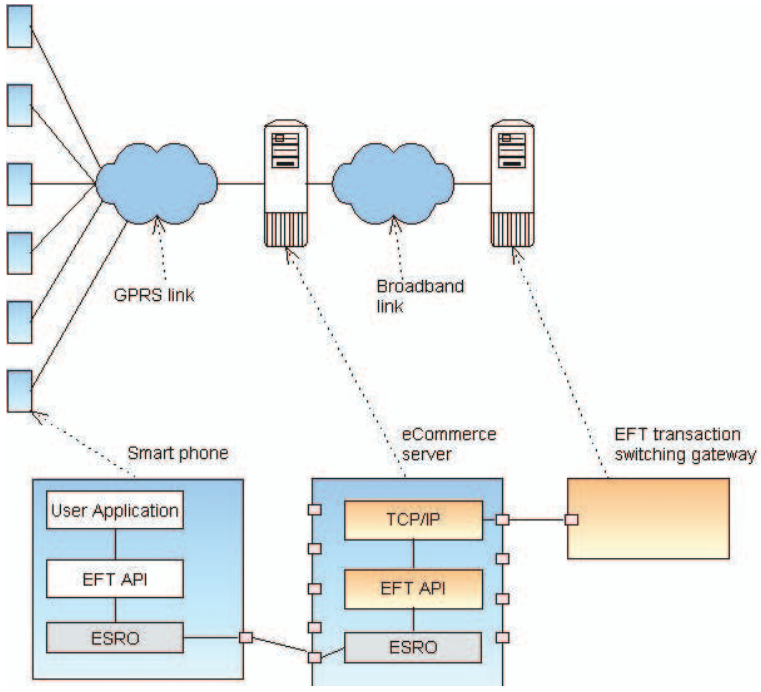

Fig. 6. Network used in the proSPEX case study

described in Section8). In other words, these parameters were placed in comment symbols in a collaboration diagram. An example of the specification of network link and processing delay parameters in the collaboration diagram was given in Figure 4. In that figure we see that a network link exists between `RoutingPeerProxy` instances with a loss probability of 0.01, bandwidth of 3430 Kb and an exponential delay distribution with an associated mean of 0.5 milliseconds. In addition active classes in the server and clients have a sending and receiving delay of 2.9 and 67.6 milliseconds respectively. Figure 4 shows that the syntax used in the specification of network link parameters is simple and intuitive. For example network link loss probability is specified with an `LP` directive. Processing delay is specified using sending delay, `SD`, and receiving delay, `RD`, directives.

In Figure 4 we also saw that an ESRO protocol entity is compose of `Manager`, `Session` and `RoutingPeerProxy` active classes. Each such class has behavior which is concerned with either performing or invoking a remote operation. An example of the behavioral specification of a `Session` instance in the performer role is given in Figure 7. Note that a peer invocation request signal, `invokReqPeer`, with an invoke PDU, `invPDU` or segmented invoke PDU, `segInvPDU` payload is expected in state `START_STA01`.

In particular, for the system illustrated in Figure 6 we asked what the expected buffer size at the server and switching nodes, respectively, will be given certain link and node parameter value scenarios. We determined the expected buffer size from the maximum queue length determined by simulating the

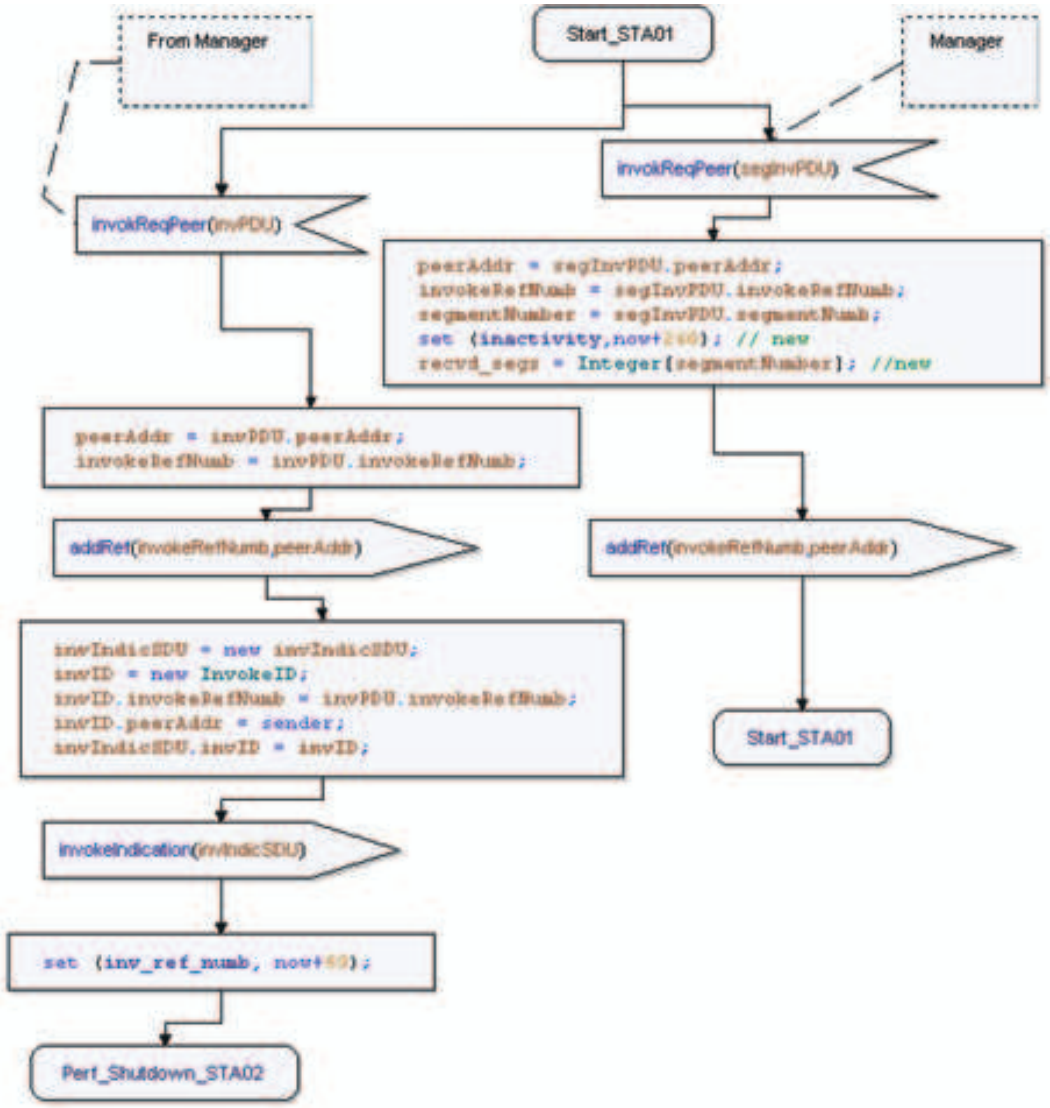

Fig. 7. A `Session` active class in the performer role

specification. The results are given in Figure 8.

With an estimated maximum queue length of 7 packets at the server and a maximum PDU length of 1500 bytes used in the scenario, the largest buffer size required in the ESRO server would be about 10Kb.

10 Conclusion

UML 2.0, a major revision of the *de facto* UML modeling standard, has emerged as a model-driven development language well suited to communication system development. By applying the ITU Recommendation Z.109

	Switch	Server
Mean Queue Length	1.00	1.33
95% Confidence Interval	(1.00 - 1.00)	(0.79 - 1.88)
Maximum Queue Length	1.00	7.00
95% Confidence Interval	(1.00 - 1.00)	(6.34 - 7.66)

Fig. 8. Estimated ESRO server buffer size

[MP00] "SDL Combined with UML" to UML 2.0, the abstractions, tightened semantics and syntax of SDL state machines become available in UML. With this enhancement previous approaches to performance analysis [BMSK96][MTMC99] that incorporate temporal aspects into SDL specifications can be applied.

In this work we have developed a methodology for the design and performance analysis of communication software. This methodology is supported by the proSPEX performance analysis tool. One of the questions we set out to answer was whether one can translate from one or more UML diagrams describing a system to a performance model.

In developing our methodology we found that the UML 2.0 class, architecture and state chart diagrams were necessary to define the architecture and behavior of communication software. We investigated means of representing non-functional duration constraints and the associated semantic time models in the case of SDL. We chose not to make syntactical changes in state chart diagrams and hence used an annotated approach in representing non-functional duration constraints. Thus network link characteristics and processing delay directives are specified by using UML 2.0 collaboration diagrams. Both network link and processing delay directives are specified using UML comment symbols using a simple syntax. Network link characteristics which that can be specified include bandwidth, loss probability and delay distribution. Processing delay directives are associated with active classes and are used to describe packet sending and receiving delay in the generated performance model.

In addition to presenting our methodology we have highlighted the architectural aspects of the proSPEX tool[6] 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. In addition, we developed a means of representing SDL signalling abstractions (e.g. *pid*, *child* and *parent*) and a means of encoding the system architecture. We also found that the UML 2.0 communication abstractions,

[6] The authors may be contacted with regard to obtaining the source code and examples.

offered by extending UML 2.0 with SDL actions, map readily to Simmcast simulation primitives.

Future work with regard to the modeling of processing delay could be to associate random delay with individual actions or transitions as is done in the objectGEODE approach to performance analysis with SDL. In addition, the syntax used for performance directives in comment symbols should be in line with that which is specified in the UML Profile for Schedulability, Performance and Time.

References

[BDM02] 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.

[Bjo02] M Bjorkander. Graphical programming using UML and SDL. *IEEE Computer*, 33(12):17–22, December 2002.

[BMSK95] M. Butow, M. Mestern, C. Schapiro, and P.S. Kritzinger. SDL performance evaluation of concurrent systems. Technical report, Department of Computer Science, University of Cape Town, 1995.

[BMSK96] M. Butow, M. Mestern, C. Schapiro, and P.S. Kritzinger. Performance modelling with the formal specification language SDL. In *IFIP TC6/6.1 International Conference on Formal Description Techniques IX / Protocol Specification, Testing and Verification XVI*, volume 69, pages 213–228. Kluwer, 1996.

[BRJ98] Grady Booch, Jim Rumbaugh, and Ivar Jacobson. *The Unified Modeling Language User Guide*. Addison-Wesley, 1998.

[Dor02] D. Dori. Why significant UML change is unlikely. *Communications of the ACM*, 45(1):82–85, January 2002.

[ea99] 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.

[Gra02] S. Graf. Expression of time and duration constraints in SDL. In *Proceedings of the Second IEEE Sensor Array and Multichannel Signal Processing Workshop*, pages 1–16. IEEE, 2002.

[Gro02] Object Management Group. UML profile for schedulability, performance, and time specification. Object Management Group Online Publication, 2002.

[Gro03] Object Management Group. UML 2.0 superstructure specification. Object Management Group Online Publication, August 2003.

[Hog89] Dieter Hogrefe. *Estelle, LOTOS and SDL.* Springer Verlag, 1989.

[Hol91] G Holzmann. *Design and Validation of Computer Protocols.* Prentice Hall, 1991.

[Hol92] G.J. Holzmann. Protocol design: Redefining the state of the art. *IEEE Software*, 9(1):17–22, January 1992.

[Kob02] Chris Kobryn. Will UML 2.0 be agile or awkward? *Communications of the ACM*, 45(1):107–110, January 2002.

[LQV01] 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.

[MB02] 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.

[MC01] 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.

[MDMC96] J. Hintelmann M. Diefenbruch and B. Muller-Clostermann. Quest: Performance evaluation of SDL systems. In *IFIP TC6/6.1 International Conference on Formal Description Techniques IX / Protocol Specification, Testing and Verification XVI*, volume 69, pages 229–244. Kluwer, 1996.

[MHSZ96] J. Martins, J.P. Hubaux, T. Saydam, and S. Znaty. Integrating performance evaluation and formal specification. In *Proceedings of IEEE ICC '96*, pages 1803–1807. IEEE Press, 1996.

[Mic88] Sun Microsystems. RFC 1050 - RPC: Remote procedure call protocol specification. http://www.faqs.org, April 1988.

[Mil02] J. Miller. What UML should be. *Communications of the ACM*, 45(1):67–69, January 2002.

[MP00] B. Moller-Pedersen. SDL combined with UML. In *Telektronikk 4.2000, Languages for Telecommunication Applications*, 2000.

[MTMC99] S Mitschele-Thiel and B Mller-Clostermann. Performance engineering of SDL/MSC systems. *Computer Networks*, 31(17):1801–1815, June 1999.

[Pro04] The Apache Jakarta Project. The apache jakarta project: Velocity. http://jakarta.apache.org/velocity/, 2004.

[PT00] J Parssinen and J Turunen. Patterns for protocol system architecture. In *Pattern Languages of Programs (PLoP) Conference*, 2000.

[Rou] Jean-Luc Roux. SDL performance analysis with objectgeode. Telelogic White Paper.

[Sel03] Bran Selic. Brass bubbles: An overview of UML 2.0 (and MDA). Object Technology Slovakia (OTS) 2003, June 2003.

[SGW94] Bran Selic, G. Gullekson, and P.T. Ward. *Real-Time Object-Oriented Modeling*. John Wiley & Sons, 1994.

[SK00] S Sircar and A Kott. Enterprise architecture analysis using an architecture description language. In *Proceedings of DARPA Symposium on Advances in Enterprise Control*, 2000.

[Spi97] *SDL* - An Annotated Specification Language for Engineering multimedia Communication Systems*, 1997.

[Ste98] 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.