In Proc. 3rd Int. Workshop on Software and Performance (WOSP), Rome, Italy, pp. 25-34, July 2002.

Performance Analysis of Time-enhanced UML Diagrams Based on Stochastic Processes

Christoph Lindemann, Axel Thümmler,
Alexander Klemm, Marco Lohmann, and Oliver P. Waldhorst
University of Dortmund
Department of Computer Science
August-Schmidt-Str. 12
44227 Dortmund, Germany
http://www4.cs.uni-dortmund.de/~Lindemann

Abstract

In this paper, we propose extensions to UML state diagrams and activity diagrams in order to allow the association of events with exponentially distributed and deterministic delays. We present an efficient algorithm for the state space generation out of these UML diagrams that allows a quantitative analysis by means of an underlying stochastic process. We identify a particular stochastic process, the generalized semi-Markov process (GSMP), as the appropriate vehicle on which quantitative analysis is performed. As innovative feature the algorithm removes vanishing states, i.e. states with no timed events active, and considers branching probabilities within activity diagrams. Furthermore, we introduce a performance evaluation framework that allows a system designer to predict performance measures at several steps in the design process. The applicability of our approach for practical performance and dependability projects is demonstrated by an UML specification of the General Packet Radio Service, a packet switched extension in GSM wireless networks.

Keywords

Model evaluation techniques, tools and techniques, QoS performance modeling, transient and steady-state analysis of generalized semi-Markov processes.

1. Introduction

To effectively design complex computer systems, software systems, embedded systems, and communication networks, the design process should be accompanied by quantitative evaluation of different design alternatives. Such quantitative evaluation considers measures like response times, queue lengths, throughput, or loss probabilities and helps understanding system performance. Although conceived as a general-purpose modeling language, the current version of the Unified Modeling Language (UML, [15]) does not contain building blocks for introducing stochastic timing into UML diagrams. Quantitative analysis of UML diagrams is particularly important for the emerging research field software performance engineering (see e.g., [22], [23]) as well as for system engineering at large.

As a first step in this direction, Douglass specified language extensions of the UML for specifying real-time constraints such as deadlines [6]. Furthermore, there are activities of the OMG to extend the current version of the UML for modeling real-time applications. Therefore, the OMG sent out a Request for Proposal (RFP) that addresses the issue of schedulability, performance, and time [16]. An initial response to the RFP that was recently adopted as a final specification was submitted by a group of OMG members consisting primarily of vendors of different kinds of real-time tools [17]. From the modeling point of view the specification mainly addresses the issue of modeling general resources. A resource is viewed as a server for which the services can be qualified by one or more quality of service (QoS) characteristics (e.g., a response time). From the analysis point of view, the response introduces modeling approaches that are tailored to schedulability analysis and performance analysis. With schedulability analysis an execution order of different entities of the system is determined for optimizing criteria such as meet all hard deadlines or minimize the number of missed deadlines. The performance analysis model defines UML extensions for e.g. modeling workloads and performance values. It is demonstrated by a web video application that is modeled with annotated activity diagrams. However, a detailed understanding of how to effectively derive performance measures from UML diagrams is missing.

Recently, Cortellessa and Mirandola developed a framework for generating a performance model from parts of UML diagrams [3], [4]. Their proposed methodology makes use of UML use case diagrams, sequence diagrams and deployment diagrams. They combined a set of sequence diagrams derived from different use cases to generate an execution graph. From the deployment diagram they derived an extended queueing network model that is parameterized by means of the execution graph. Nevertheless this approach relays entirely on traditional approaches in software performance engineering introduced by Smith in 1990 [22] and tailored to the UML in [23]. Petriu, Shousha, and Jalnapurkar developed a systematic approach to build a layered queueing network performance model from a UML description [18]. They demonstrated their approach by analyzing an existing

WOSP '02, July 24-26, 2002 Rome, Italy

telecommunication system. King and Pooley developed a methodology that considers the generation of a generalized stochastic Petri net from UML state diagrams embedded in collaboration diagrams [10]. The translation is obtained by associating to each state in the state diagram a place in the Petri net and to each transition in the state diagram a transition in the Petri net. Nevertheless, the performance analysis of UML state diagrams via the generation of a (canonical) Petri net results in an unnecessary overhead since the Petri net contains immediate transitions, that should be omitted in the quantitative analysis.

In this paper we present an approach for the automatic generation of a performance evaluation model from system specifications described through UML state diagrams or activity diagrams. To enable quantitative evaluation of UML system specifications, the building blocks of the UML must be enhanced for specifying deterministic and stochastic delays and a well-defined mapping of UML diagrams onto their underlying discrete-event stochastic system (i.e., the underlying stochastic process) must be introduced. This paper addresses these issues. The contribution is twofold: First, we propose extensions to UML state diagrams and activity diagrams to allow the association of events with exponentially distributed and deterministic delays. Subsequently, we show how to map these time-enhanced UML system specifications onto a discrete-event stochastic system. We identify a particular stochastic process, the generalized semi-Markov process (GSMP) [21], [24], as the appropriate vehicle on which quantitative analysis is performed. Second, we present an efficient algorithm for the direct and automated state space generation out of these UML diagrams that removes vanishing states, i.e. states with no timed events active, and considers branching probabilities within activity diagrams. Furthermore, we introduce a performance evaluation framework that allows a system designer to predict performance measures at several steps in the design process by the concept of nested states provided in state diagrams.

The approach presented in this paper is implemented in the new version of DSPNexpress 1.5 [11], DSPNexpress 2000, that provides tool support for the automated quantitative analysis of discrete-event stochastic systems underlying UML diagrams and Petri nets. DSPNexpress 2000 contains filters to the commercial UML design packages Rational Rose™ [19] and Rhapsody™ [20]. These filters were introduced in a previous paper [14]. The linkage of the DSPNexpress software to commercial UML design packages effectively supports the design process because these tools contain sophisticated user interfaces for user-friendly model specification that are widely used in industry.

The remainder of this paper is organized as follows. In Section 2 we present the framework for quantitative analysis of UML diagrams and describe how to integrate our methodology in the system design lifecycle. Section 3 considers the introduction of deterministic and stochastic delays into UML state diagrams and activity diagrams as well as the mapping of these UML building blocks onto stochastic processes. In Section 4, we present an UML specification of the General Packet Radio Service, a packet switched extension in GSM wireless networks. Finally, concluding remarks are given.

2. Framework for Quantitative Analysis of UML Diagrams

In order to accompany the design process of software and hardware systems with performance evaluation in different design stages a framework for quantitative analysis of UML diagrams is needed. Figure 1 presents the proposed framework for deriving performance measures for UML diagrams by analysis of their underlying GSMP. As illustrated, the derivation of performance measures is divided into four main steps. In the first step, a state diagram or activity diagram has to be specified with an UML design tool like Rational Rose™ [19] or Rhapsody™ [20]. With these tools output files can be generated with an ascii-text representation of the UML diagram (e.g., the *.sbs* file of Rhapsody™). For timed events an extra specification file is needed to include the expected waiting delays and information about the delay distribution, i.e., deterministic or exponentially distributed delay.

After adding timing specifications to the UML diagram, the derivation of the state transition graph has to be performed; see step (2) in Figure 1. Therefore, the state space of the UML diagram has to be explored as described in Section 3. Note, that the state transition graph is a formal representation of the discrete-event system specified in the UML or even other specification languages like generalized stochastic Petri nets (GSPN) [1] or deterministic and stochastic Petri nets (DSPN) [11]. Its generation is a prerequisite for the quantitative analysis of the underlying stochastic process.

The core of the quantitative evaluation process constitutes the numerical solution of the underlying GSMP with exponential and deterministic events; see step (3) in Figure 1. For a detailed mathematical treatment of stochastic processes underlying discrete-event stochastic systems, we refer to [11]. Alternatively, the transient or steady-state solution of the GSMP can be derived by discrete-event simulation e.g., as provided by the commercial simulation library CSIM [5]. The results are stored in a data structure comprising of the state probabilities for the state transition graph.

The main task of step (4) in Figure 1 constitutes the computation of performance measures like throughput, loss probabilities, mean response time of a resource etc. given the previously computed state probabilities of the underlying GSMP. Therefore, the probabilities have to be combined to performance measures of interest or even to state probabilities of the state diagram. In order to derive performance curves for a performance measure of interest the value of one delay parameter of a UML diagram is varied while the other parameters are kept fixed. Performance curves give system engineers significant insight into the system dynamics before implementing the system. The framework allows system engineers to check system performance in early design stages and to refine their model if necessary as shown in step (5) in Figure 1. As described in Section 3 the system performance can be evaluated at several steps in the development lifecycle using the concept of nesting substates into superstates available at the states diagrams palette. Therefore, it is possible to obtain rough quantitative values in early design stages as well as very accurate performance indices in a final design phase.

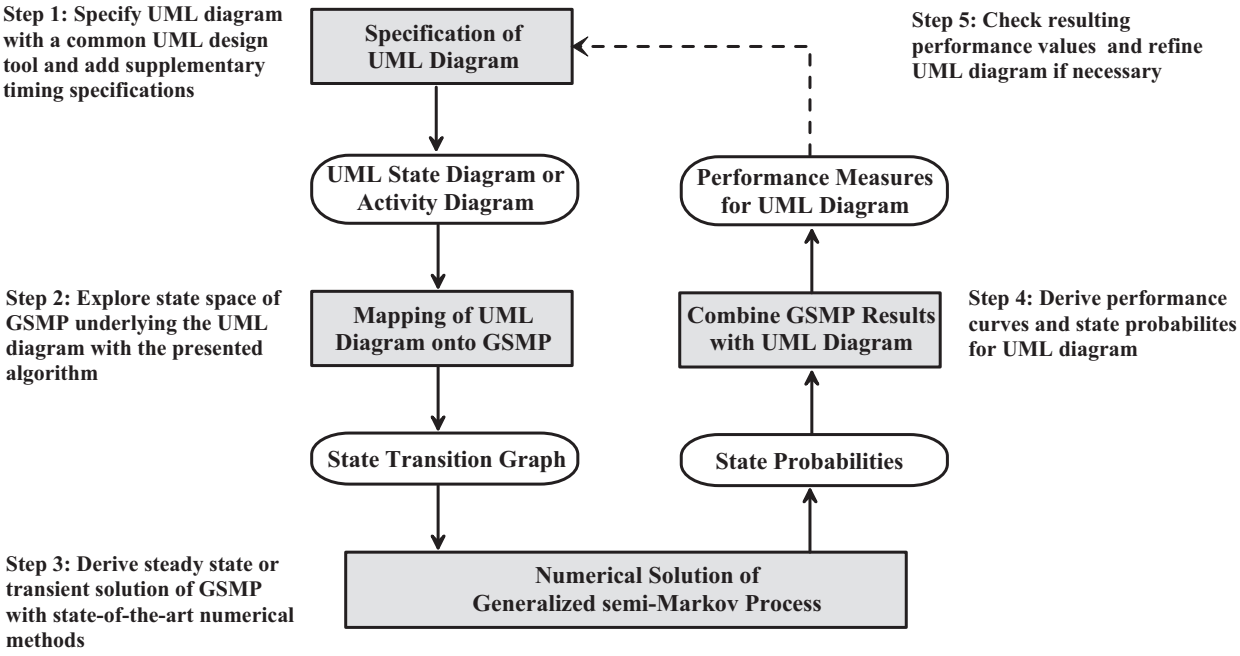

Figure 1. Derivation of quantitative performance measures out of UML diagrams

In order to automate the performance evaluation process we implemented the described methodology in the tool DSPNexpress 2000. The main goal of DSPNexpress 2000 constitutes the availability of an open interface for utilizing the highly efficient numerical solvers for GSMPs for the quantitative evaluation of systems specified in modeling formalisms other than just deterministic and stochastic Petri nets (DSPNs). In fact, the main research contribution of DSPNexpress 2000 constitute the robust implementation of an efficient numerical method for transient and steady-state analysis of GSMPs with exponential and deterministic events [12].

Currently, DSPNexpress 2000 can perform quantitative analysis for DSPNs and UML specifications, i.e., state diagrams and activity diagrams. DSPNs can be edited directly with the Petri net editor of DSPNexpress. UML state diagrams and activity diagrams can be imported through an UML filter provided by DSPNexpress 2000 as introduced in [14]. The parameter specifications for associating exponentially distributed or deterministic delays with events can be added to the UML diagrams through the DSPNexpress graphical user interface. DSPNexpress allows the user-friendly specification of performance studies (i.e., what/if studies). Performance curves derived by the report generator of DSPNexpress can be either visualized by DSPNexpress or exported in a format compatible to the common tool *GNUPLOT*.

3. Performance Analysis of UML Diagrams

3.1 Introducing Stochastic Timing into UML Diagrams

The UML [2], [15] provides different views of a model that are represented by graphical diagrams. These diagrams include *use case diagrams, class diagrams, behavior diagrams,* and *implementation diagrams.* Behavior diagrams include *state diagrams, activity diagrams* and interaction diagrams like *sequence diagrams* and *collaboration diagrams.* It is the freedom of the designer to choose the types of UML diagrams best suited for the intended representation of a system. In this paper, we choose state diagrams and activity diagrams as UML building blocks for enabling quantitative system analysis with the UML. The diagrams are presented using the notation from [6].

State diagrams of the UML provide a simple but formal means of modeling the complex event-driven system behavior. All semantics necessary to express behavior (i.e., states, historical properties, transitions, and compound transitional connectors) are available on the state diagram palette. The semantics and notation of state diagrams, also known as statecharts, are substantially those of Harel's statecharts (see e.g., [9]) with modifications to make them object-oriented. A state diagram represents a state machine; a state being a condition during the life of an object or an interaction during which it satisfies some condition, performs some action, or waits for some event.

States are shown as named rectangles with rounded corners and represent a possible situation for an object. The initial state of an object is labeled with a *default connector* that is represented by a short arc starting from a small filled circle. States can be ordered hierarchically and/or concurrently. Concurrent states are separated by dashed lines. These states are also called *or-states* because they are mutually exclusive. The hierarchy of states is represented by the nesting of states within states. The outer enclosing state is called *superstate* and the inner states are called *substates*. The substate that is visited by entering the corresponding superstate is labeled with the default connector. Furthermore, *pseudostates* are defined in UML state diagrams. The *conditional pseudostate* allows one of a set of branching transitions to be selected based on some guarding condition. A conditional pseudostate is indicated by a circled C. The *history pseudostate* indicates that

when entering a superstate the initial or default state is that last active substate of the superstate. The UML identifies two kinds of history - shallow and deep. Shallow history means that the last active substate is the active default, but if that substate is further decomposed into sub-substates, no knowledge is retained of that nested history. Deep history means that history is remembered to all levels of nesting. Shallow history is indicated by a circled H and deep history by a circled H*.

Transitions are shown as directed arcs between states. Transitions are labeled with a *trigger event*, a *guard*, and an *action.* A proper definition of a transition must contain either one of these three building blocks. The two others are optional. Actions are considered to be processes that occur quickly and are not interruptible. A guard is a logical condition that will return only "true" or "false". If the trigger event occurs and the guard resolves to "true" the action is executed and the corresponding state change is performed. Thus, we have the following notation for a transition:

```
event[guard]/action
```

Activity diagrams of the UML are used to model sequence and parallelism of activities. An activity diagram is a special case of a state diagram in which all (or at least most) of the states are action states (i.e., activities) and in which all (or at least most) of the transitions are triggered by completion of the actions in the source states. The purpose of this diagram is to focus on the flows driven by internal processing (as opposed to external events). Activity diagrams are used in situations where all or most of the events represent the completion of internally generated actions (that is a procedural flow of control). State diagrams, on the other hand, are used in situations where asynchronous events predominate.

In order to introduce timing in a UML state diagram or activity diagram, we associate trigger events with deterministic or exponentially distributed delays. Thus, *timed events* trigger a state transition. We call these transitions *timed transitions*. *Immediate events* are triggered by actions of transitions. To represent timed events in a state diagram or activity diagram, we define new syntactical expressions that can be directly derived from the performance value definition *PAperfValue* introduced in [17]. The expression `EXP_id` defines an event that triggers a state transition after an exponentially distributed delay. The identifier `id` represents the mean value of the delay, that is $1/\lambda$ in the case of an exponential distribution with parameter λ. The expression `DET_id` defines an event that triggers a state transition after a deterministic delay characterized by the identifier `id`. Actions that generate an (immediate) event `E` are denoted by `GEN_E`.

Furthermore, we introduce randomness in the sequential flow of activities in activity diagrams. Therefore we associate weights w_1, ..., w_n with outgoing transitions t_1, ..., t_n of a conditional pseudostates. If more than one guard of the outgoing transitions resolve to "true" (i.e., more than one transition is active) the weights of these transitions are used to compute branching probabilities by the following formula:

$$p(t_i) = \frac{w_i}{\sum_{j:t_j \text{ active}} w_j} \qquad (1)$$

With this definition the next state is chosen from a discrete probability distribution among the active transitions as for GSPNs [1].

3.2 Mapping UML Diagrams onto a Generalized Semi-Markov Process

We view a state diagram or an activity diagram of the UML as a discrete-state, event-driven system. That is, its state evolution depends entirely on the occurrence of asynchronous discrete events over time. For ease of exposition we describe the methodology for quantitative analysis of state diagrams only. Activity diagrams can be treated as a special case of state diagrams. The key idea of the state space exploration is to map a configuration of the UML state diagram onto an appropriate state of the underlying stochastic process and a transition in the state diagram onto a state change of the underlying process. Therefore the stochastic process can be completely represented by the *state transition graph*, a directed graph with labeled arcs.

In the following we describe the derivation of the state transition graph via the exploration of the transition system. The *transition system* consists of all possible configurations of the corresponding UML diagram. A *configuration* of a state diagram is a snapshot of its execution. One can view a configuration as the information that is needed to completely restore the "state" of the system. A configuration consists of the active substates of all concurrent states of the system, the history information and the setting of all variables. Let s_1, ..., s_n be the states of the state diagram, h_1, ..., h_r the history connectors, and v_1, ..., v_m the variables corresponding to the state diagram. Formally, a configuration C is represented by a tuple (S,H,V) comprising of a set S that contains all active states (i.e., one substate of each concurrent state), a mapping H of each history connector h_j onto a set H_j comprising of the stored history states, and a mapping V of each variable onto an appropriate value. In the case of shallow history the sets H_j contain each only one state, namely the substate that has to be restored when entering the history connector. In the case of deep history further substates have to be stored if even one substate is a concurrent state.

The configurations of the transition system are connected by directed arcs that represent a change of a configuration in the state diagram. We distinguish two causes of a change of a configuration in the transition system. First a change of a configuration can be triggered by timed transitions, with events that have exponentially distributed or deterministic delays. Furthermore, a change of a configuration can occur without delay. That is due to the evaluation of a guard or the occurrence of an immediate event generated by an action. We simply call these changes of configuration *immediate transitions*. As in generalized stochastic Petri nets [1], we define that immediate transitions have priority over timed transitions. That is if the guard of an immediate transition t_i is accepted, this transition triggers the configuration change with (branching) probability $p(t_i)$ (see equation (1)).

Configurations in the transition system can be classified as *tangible* or *vanishing* configurations. A tangible configuration is a configuration in which only timed transitions are active. A vanishing configuration is a configuration in which one or more immediate transitions are active. For the quantitative analysis of UML state diagrams only tangible configurations of the underlying transition system need to be considered. This is because immediate transitions occur without delay and therefore the probability of being in a vanishing state is equal to zero. Thus, only the tangible configurations constitute states of the stochastic

```
(1)  Import state diagram from UML design tool
(2)  Derive initial tangible configuration c_init
(3)  Label c_init as NEW and insert c_init into the empty transition graph G
(4)  FOR EACH configuration c = (S,H,V) labeled as NEW in G DO
(5)    Label c as VISITED
(6)    FOR EACH state s ∈ S DO
(7)      FOR EACH transition t originating from s DO
(8)        IF guard of t resolves to TRUE in configuration c THEN DO
(9)          Derive configuration c' reached through transition t
(10)         Derive set of tangible configurations C reachable from c'
(11)         FOR EACH configuration c" in C DO
(12)           Label c" as NEW and insert c" into G
(13)           Insert arc c → c" labeled with delay distribution into G
(14)       OD
(15)     OD
(16)   OD
(17)  OD
(18) OD
```

Figure 2. Algorithm for generation of state transition graph

process for which the quantitative analysis is performed. All vanishing configurations in a transition system have to be removed to obtain the state transition graph that comprises of only tangible configurations and directed arcs between all tangible configurations of the transition system.

Figure 2 depicts a pseudo-code algorithm for the generation of the state transition graph. The initial tangible configuration can be calculated with a top-down activation of the default states of the state diagram beginning with the top-level state. Starting with the initial tangible configuration the algorithm directly derives the transition graph without explicitly generating the transition system. The key procedure of the algorithm is performed in step (9) and (10). The configuration c' in step (9) is derived using the history information H stored in configuration c = (S,H,V) and the execution of the actions of transition t that may effect the variable setting V. Note that the configuration c' may be a vanishing configuration that should not be inserted in the transition graph. Therefore the set of tangible configurations that can be reached through c' has to be determined recursively with graph analysis methods based on a depth-first-search starting at c' (see step (10) of Figure 2). We observe that an effective method for this task can be borrowed from reachability analysis for generalized stochastic Petri nets [1].

Besides the state space exploration and the effective removal of vanishing configurations, quantitative analysis of UML specifications requires a mapping of the state transition graph onto an appropriate stochastic process for which simulation-based and/or analytical numerical analysis methods are known. This mapping is the key for the quantitative analysis of UML state diagrams and activity diagrams. In fact, every tangible configuration of the state diagram maps to a state of the corresponding state space of the underlying stochastic process and every configuration change in the state transition graph corresponds to a state change in the stochastic process.

The most general form of the stochastic process underlying a state diagram is the generalized semi-Markov process (GSMP). A generalized semi-Markov process [21], [24] is a continuous-time stochastic process that makes a state transition when one or more "*events*" associated with the occupied state occur. Events associated with a state compete to trigger the next state transition, and each set of trigger events has its own distribution for determining the next state. At each state transition of the GSMP, *new* events may be scheduled. For each of these new events, a clock indicating the time until the event is scheduled to occur is set according to an independent (stochastic) mechanism. I.e., for each new event a clock reading is generated according to its *clock setting distribution*. For each scheduled event which does not trigger a state transition but is still scheduled in the next state, its clock *continues* to run. If an event is no longer scheduled in the next state, it is *canceled*, and the corresponding clock reading is discarded.

In general, a GSMP describes the evolution of the stochastic behavior of a discrete-event stochastic system (DES). Although a GSMP constitutes a very general stochastic process, a rich body of theoretical results on monotonicity, regeneration, and continuity is available [21], [24]. In general, the analysis of a GSMP can be performed by discrete-event simulation only. For finite-state GSMPs with exponential and deterministic events, a cost-effective numerical method for the transient and steady-state analysis has been introduced in the context of stochastic Petri nets [12]. These analysis techniques are based on a general state space Markov chain (GSSMC) embedded at equidistant time points nD (n = 1,2,...) of the continuous-time GSMP. This numerical approach constitutes of two main steps: the derivation of the transition kernel and the solution of a system of multidimensional integral equations. Therefore, the well-defined derivation of the state transition graph of the GSMP is the key for quantitative analysis of UML system specifications.

To provide an example, Figure 3 shows the state diagram of a queueing system with a Markov-modulated Poisson process (MMPP) [8] as customers arrival process, one server with deterministic service time and finite buffer of size K, i.e., an MMPP/D/1/K queueing system. The MMPP is parameterized by two states representing a bursty and a less bursty mode of customer arrivals. Such an MMPP is used as the packet arrival process for the application example presented in Section 4.

Each configuration of the MMPP/D/1/K queue can be represented by a tupel comprising of the active states, the history information, and the value of the variable *Queue* (i.e., 0,1,...,K customers in the queue) as explained above. The set of states S of a configuration contains one substate of each of the concurrent superstates *Customers* and *Server*, i.e., *Normal (N)*, *Bursty (B)*, or *Decision (D)*, of superstate *Customers* and *ServerIdle (I)* or *ServerBusy (B)* of superstate *Server*. The history information is simply N or B corresponding to the *Normal* and *Bursty* substate, respectively.

The transition system of the MMPP/D/1/K queue is presented in Figure 4. Tangible configurations are drawn as white circles and are associated with state numbers. Vanishing configurations are drawn as dashed circles. Directed arcs between configurations represent feasible state transitions. Dashed arcs represent immediate state transitions. Below a state number the corresponding configuration is written in the following notation:

(customers substate, server substate; history state; queue length)

Note, that the transition system is not derived explicitly in the algorithm presented in Figure 2. Instead the transition graph is derived directly by ignoring the vanishing configurations when exploring the state space (step (10) of Figure 2). The state transition graph of the GSMP corresponding to the MMPP/D/1/K queue is shown in Figure 5. Putting it all together, the state transition graph of the GSMP consists of $2 \cdot (K+1)$ tangible

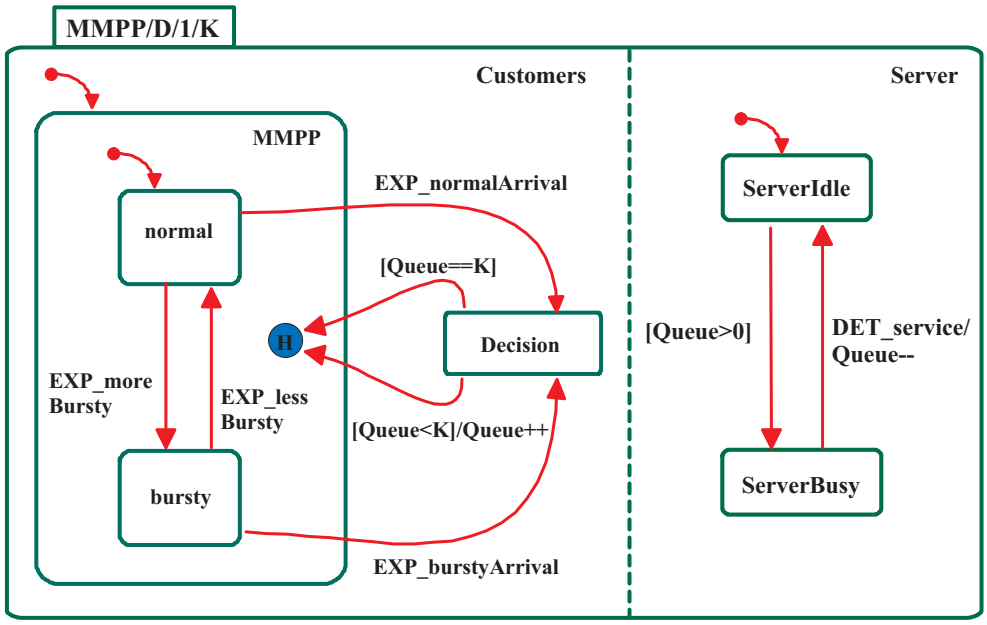

Figure 3. UML state diagram of an MMPP/D/1/K queue

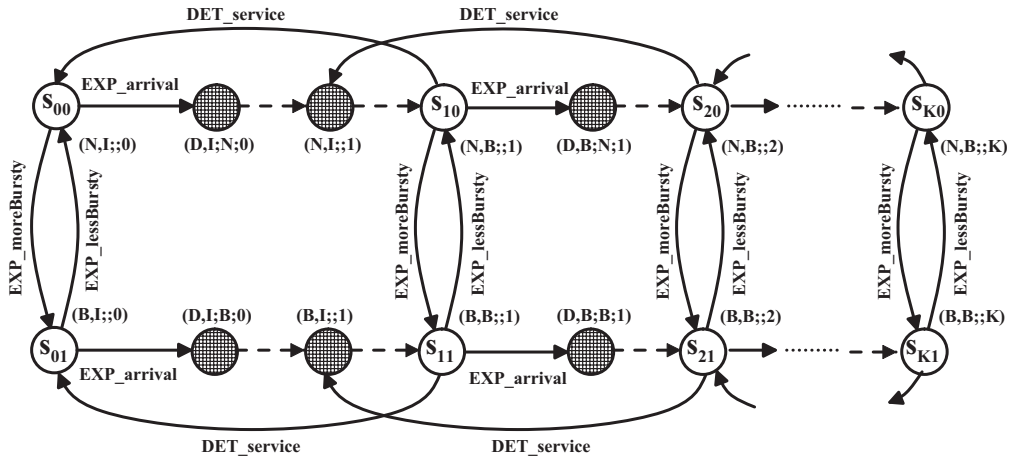

Figure 4. Transition system of the UML state diagram of an MMPP/D/1/K queue

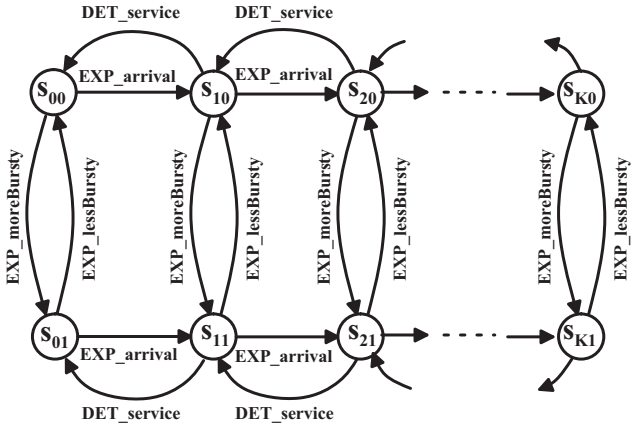

Figure 5. State transition graph of the GSMP underlying the UML state diagram of an MMPP/D/1/K queue

configurations representing 0,1,2,...,K customers in the queue with the MMPP residing in normal or bursty mode. That is, configurations s_{i0} correspond to i customers in the queue with the MMPP in normal mode and configurations s_{i1} correspond to i customers in the queue with the MMPP in bursty mode.

We want to point out that the framework for generating a performance model, i.e., the state transition graph, and subsequently deriving performance curves out of a UML state diagram or activity diagram can be applied in several steps in the system or software development lifecycle. In order to get insight in the quantitative system dynamics in early design stages we suggest a hierarchical modeling of the system with state diagrams. That is, in a first step the system designer has to identify the "main" states of the system. That can be done for example by using a set of sequence diagrams derived from different use cases. Furthermore, a rough design of the system reduces the state space size significantly and therefore performance values can be obtained very quickly. In later design stages the main states should be modeled in more detail using the concept of nesting substates into superstates. Thus, it is possible to obtain rough performance values in early design stages as well as very accurate performance indices in a detailed design phase.

4. Application Example

4.1 Modeling the General Packet Radio Service with UML State Diagrams

To illustrate the practical applicability of our approach for the quantitative analysis of UML specifications, we consider an UML state diagram of the General Packet Radio Service (GPRS) for GSM wireless networks. A detailed performance study for GPRS is published in [13].

The General Packet Radio Service is a standard from the *European Telecommunications Standards Institute (ETSI)* on packet data in the *Global System for Mobile Communications (GSM)* [7]. By adding GPRS functionality to the existing circuit switched GSM network, operators can give their subscribers resource-efficient wireless access to external Internet protocol-based networks, such as the Internet and corporate intranets. The basic idea of GPRS is to provide a packet switched bearer service in a GSM network. In conventional GSM, a physical channel is permanently allocated for a particular user during the entire call period (whether data is transmitted or not). In contrast, in GPRS the channels are allocated on a per-packet basis, i.e., only when data packets are sent or received, and they are released after the transmission. For bursty traffic this results in a much more efficient usage of the scarce radio resource because this principle allows multiple users to share one physical channel.

The GPRS model considers an integrated GSM/GPRS network, serving circuit-switched voice and packet-switched data calls. Therefore, the available physical radio channels have to be split into GSM traffic channels and channels allocated to GPRS, called *Packet Data Channels (PDCH)*. GPRS includes the functionality to increase or decrease the amount of radio resources allocated to GPRS on a dynamic basis ("capacity on demand"). Physical channels not currently in use by conventional GSM can be allocated as PDCHs and when there is a resource demand for services with higher priority, e.g. GSM voice calls, PDCHs can be de-allocated.

In Figures 6 to 8 the state diagrams considering mobile user behavior in a single cell of an integrated GSM/GPRS network are presented. The overall number of physical channels available in the cell is represented by the identifier *freeChannels*. We assume that a certain number of channels, denoted by *fixedPDCH*, are permanently reserved as PDCHs for GPRS and the remaining channels can be used either as GSM traffic channels or as "on-demand" PDCHs. In particular, the model considers the following six driving processes that may effect the state of the cell:

(1) incoming GSM calls and handovers,

(2) incoming GPRS calls and handovers,

(3) leaving GSM calls due to completion or handover,

(4) leaving GPRS calls due to completion of handover,

(5) arrivals of IP packets,

(6) transmission of IP packets.

A GSM or GPRS call can be either terminated by a handover to an adjacent cell (expressed in the model by *call dwell time*) or by completing the call in the considered cell (expressed by *call duration*). Figure 6 shows the GSM and GPRS call arrival processes and Figure 7 depicts the corresponding call service processes. We assume GSM and GPRS call arrivals as well as GSM and GPRS call service times to be exponentially distributed. If a new GSM or GPRS call is accepted in the cell an attach procedure that identifies the new user in the cell introduces a deterministic overhead. Similar, a handover call from an adjacent cell introduces a significant overhead. This overhead is modeled by the deterministic delays *GSMattach* and *GPRSattach*, respectively (see Figure 6). Note that these two deterministic events can be concurrently active. Therefore, the underlying stochastic process truly constitutes a generalized semi-Markov process with exponential and deterministic events and cannot be represented by a simpler process.

The employed traffic model constitutes a *Markov-modulated Poisson Process (MMPP)* [8] that generates IP traffic for each individual GPRS user in the cell. Opposed to an ordinary Poisson process, the MMPP can capture some of the important correlations between the interarrival times (e.g., burstiness). The UML state diagram representation of the MMPP is similar to that of the MMPP/D/1/K example presented in Section 3.

The state diagram modeling the transmission of IP packets over the wireless link is shown in Figure 8. IP packets that arrive at the base station are queued in a waiting line of limited size. For transmission the available PDCHs are fairly shared by all packets in transfer up to a maximum of eight PDCHs per IP packet ("multislot mode") and a maximum of eight packets per PDCH [7]. Note that in the state diagram of Figure 8 packets are transferred one by one. In order to emulate the parallel transfer of packets the transfer time of a packet on one physical channel is divided by the number of channels used in parallel for packet transfer. This means that the transfer time *packetService* depends on the number of packets currently queued, i.e., the identifier *packetQueue*. The algorithm for generation of the state transition graph allows such a dynamic definition of identifiers and event delays in the parameter specification file.

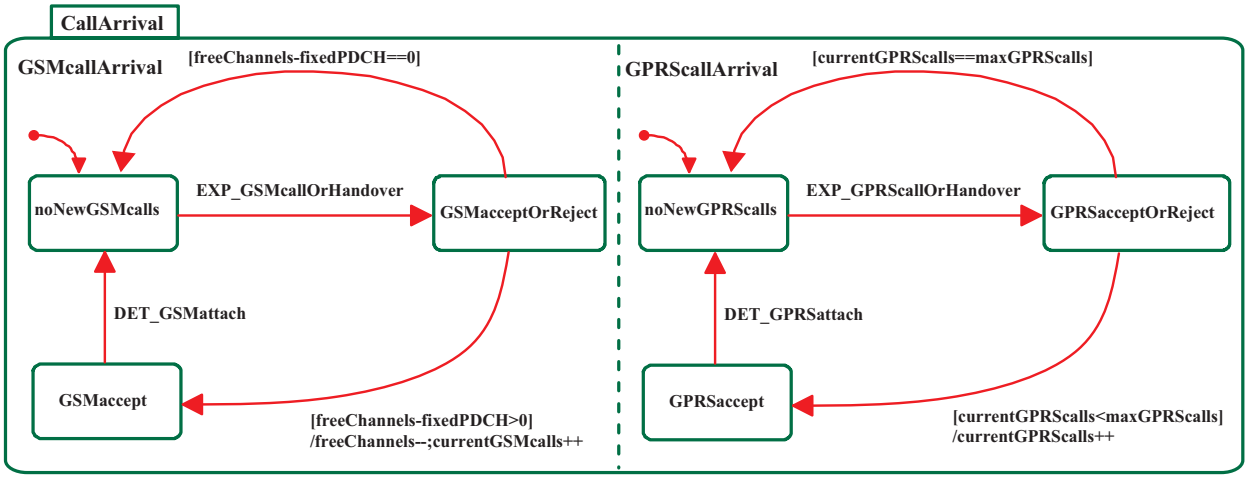

Figure 6. State diagram of GSM and GPRS call arrival process

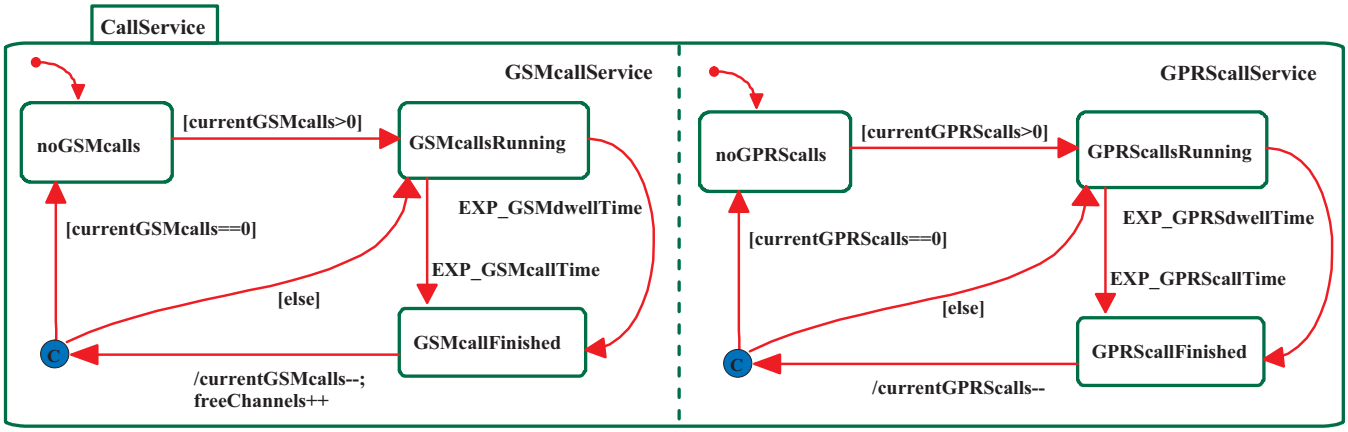

Figure 7. State diagram of GSM and GPRS call service process

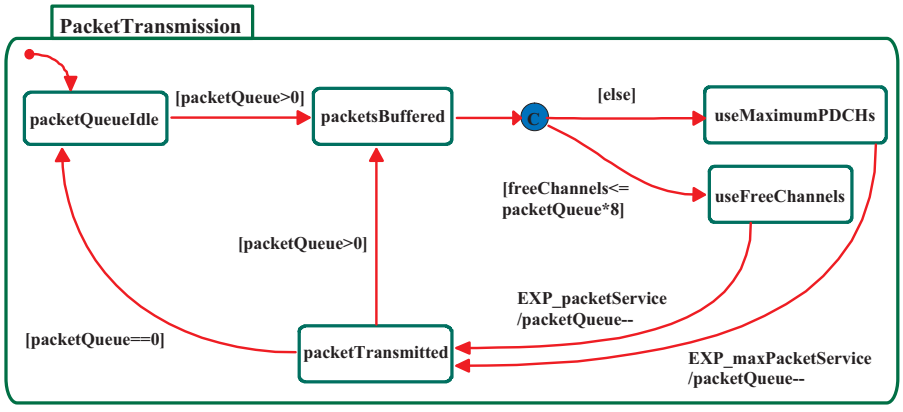

Figure 8. State diagram of GPRS packet transmission process

4.2 Performance Results

We present two curves for quality of service measures computed with DSPNexpress 2000. The curves are computed with the experiment option of DSPNexpress. For the experiments we vary the call arrival rate of GSM/GPRS users from 0 to 1.2 calls per second. Furthermore, we assume a split of 2%, 5%, 10% GPRS calls and 98%, 95%, and 90% GSM calls, respectively. The curves are computed with one PDCH permanently reserved for GPRS. A complete specification of the model parameters is given in [13]. Performance measures that are most interesting for system engineers are carried data traffic, i.e., the amount of channels that are in use for GPRS on average, and packet loss probability, i.e., the probability of a packet loss due to buffer overflow at the base station. These two measures immediately reflect the performance gained by the end-user.

Figures 9 and 10 plot carried data traffic and packet loss probability derived from the quantitative analysis of the GPRS model by DSPNexpress 2000. In the curve for carried data traffic we observe a significant decrease in allocated PDCHs when traffic load is high which is due to the priority of GSM voice calls over GPRS packet data transmissions. Such performance curves can provide system engineers with necessary insight into the system behavior under different traffic load conditions. From the curves we conclude for example that for an anticipated amount of 10% GPRS calls only 0.1 calls per second can be served if a packet loss probability of less than 10^{-3} is required.

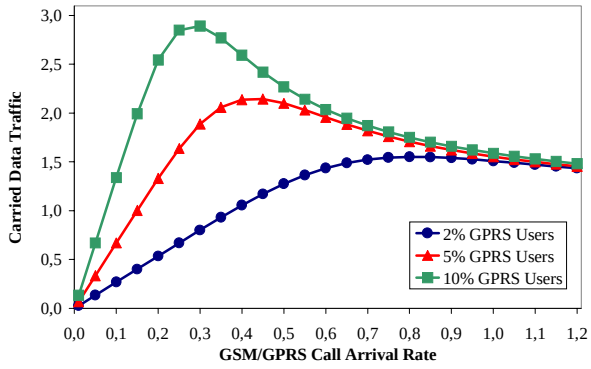

Figure 9. GPRS Performance Model: Carried data traffic and vs. offered traffic load

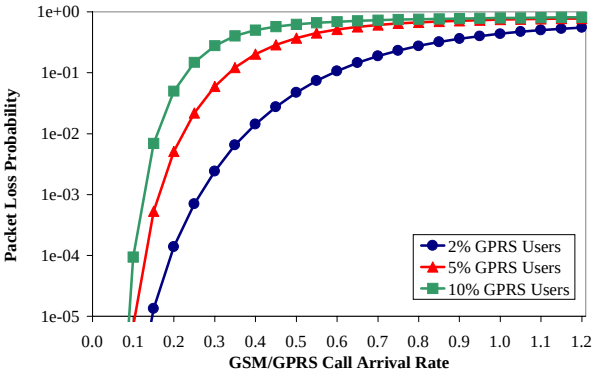

Figure 10. GPRS Performance Model: Packet loss probability vs. offered traffic load

5. Conclusions

In this paper, we propose extensions to UML state diagrams and activity diagrams to allow the association of events with exponentially distributed and deterministic delays. We identify a particular stochastic process, the generalized semi-Markov process (GSMP), as the appropriate vehicle on which quantitative analysis is performed. Furthermore, we introduce a framework for the automate quantitative analysis of UML diagrams enhanced with deterministic and stochastic delays. A main contribution of the paper is the efficient algorithm for the automated derivation of the state space underlying a UML state diagram or activity diagram that additionally deals with history connectors and branching probabilities. To illustrate the applicability of our approach for the quantitative analysis of UML system specifications, we presented a performance study for the General Packet Radio Service based on a UML state diagram.

6. References

[1] M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Francheschinis, *Modelling with Generalized Stochastic Petri Nets*, John Wiley & Sons, 1995.

[2] G. Booch, I. Jacobson, and J. Rumbaugh, *The Unified Modeling Language User Guide*, Addison Wesley 1999.

[3] V. Cortellessa and R. Mirandola, Deriving a Queueing Network Based Performance Model from UML Diagrams, *Proc. 2nd Int. Workshop on Software and Performance (WOSP), Ottawa, Canada*, 58-69, 2000.

[4] V. Cortellessa and R. Mirandola, UML Based Performance Modeling of Distributed Systems, in: *A. Evans, S. Kent, B. Selic, (Eds.), 3rd Int. Conf. on the Unified Modeling Language, York, UK*, LNCS **1939**, 178-193, Springer, 2000.

[5] CSIM18-The Simulation Engine, `http://www.mesquite.com`.

[6] B.P. Douglass, *Real-time UML: Developing Efficient Objects for Embedded Systems*, 2nd Edition, Addison Wesley 1999.

[7] ETSI, Digital cellular telecommunications system (Phase 2+); General Packet Radio Service (GPRS); Service description; Stage 2, *GSM recommendation 03.60*, 1999.

[8] W. Fischer and K. Meier-Hellstern, The Markov-modulated Poisson Process (MMPP) cookbook, *Performance Evaluation*, **18**, 149-171, 1993.

[9] D. Harel, Statecharts: A Visual Formalism for Complex Systems, *Science of Computer Programming* **8**, 231-274, 1987.

[10] P. King and R. Pooley, Derivation of Petri Net Performance Models from UML Specifications of Communications Software, in: *B.R. Haverkort, H.C. Bohnenkamp, C.U. Smith (Eds.), 11th Int. Conf. on Tools and Techniques for Computer Performance Evaluation, Schaumburg, Illinois*, LNCS **1786**, 262-276, Springer, 2000.

[11] C. Lindemann, *Performance Modelling with Deterministic and Stochastic Petri Nets*, John Wiley & Sons, 1998.

[12] C. Lindemann and A. Thümmler, Transient Analysis of Deterministic and Stochastic Petri Nets with Concurrent

Deterministic Transitions, *Performance Evaluation, Special Issue Proc. Performance 1999*, **36&37**, 35-54, 1999.

[13] C. Lindemann and A. Thümmler, Performance Analysis of the General Packet Radio Service, *Proc. 21st Int. Conf. on Distributed Computing Systems (ICDCS), Phoenix, Arizona*, 673-680, 2001.

[14] C. Lindemann, A. Thümmler, A. Klemm, M. Lohmann, and O. Waldhorst, Quantitative System Evaluation with DSPNexpress 2000, *Proc. 2nd Int. Workshop on Software and Performance (WOSP), Ottawa, Canada*, 12-17, 2000.

[15] Object Management Group, OMG Unified Modeling Language Specification, *OMG Document formal/2001-09-67*, September 2001, `http://www.omg.org`.

[16] Object Management Group, RFP: UML Profile for Scheduling, Performance, and Time, *OMG Document ad/99-03-13*, March 1999, `http://www.omg.org`.

[17] Object Management Group, Response to the OMG RFP for Schedulability, Performance, and Time, *OMG Document ad/2001-06-14*, June 2001, `http://www.omg.org`.

[18] D. Petriu, C. Shousha, and A. Jalnapurkar, Architecture-Based Performance Analysis Applied to a Telecommunication System, *IEEE Trans. on Software Engineering* **26**, 1049-1065, 2000.

[19] Rational Rose, `http://www.rational.com/products/rose/`.

[20] Rhapsody, `http://www.ilogix.com`.

[21] G.S. Shedler, *Regenerative Stochastic Simulation*, Academic Press 1993.

[22] C.U. Smith, *Performance Engineering of Software Systems*, Addison Wesley, 1990.

[23] C.U. Smith and L.G. Williams, Performance Evaluation of Software Architectures, *Proc. 1st Int. Workshop on Software and Performance (WOSP), Santa Fe, New Mexico*, 164-177, 1998.

[24] W. Whitt, Continuity of Generalized Semi-Markov Processes, *Mathematics of Operations Research*, **5**, 494-501, 1980.