In Proc. 2nd Int. Workshop on Software and Performance (WOSP), Ottawa, Canada, pp. 12-17, Sep. 2000

Quantitative System Evaluation with DSPNexpress 2000

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
{cl, at, ak, ml, ow}@ls4.cs.uni-dortmund.de
http://www4.cs.uni-dortmund.de/~Lindemann

ABSTRACT

This paper describes the software package DSPNexpress 2000, a tool for the quantitative evaluation of systems specified in stochastic Petri nets, the Unified Modeling Language (UML), or other specification languages for discrete-event systems. Linking the DSPNexpress software to commercial UML design packages allows the effective computation of performance measures like throughput and delay for UML system specifications. The unique feature of DSPNexpress constitute numerical solvers for transient and steady-state analysis of generalized-semi Markov processes with exponential and deterministic events, which may be concurrently active. The applicability of the DSPNexpress software for practical performance and dependability projects is demonstrated by an UML specification of an alternating bit protocol. The computational effort required by DSPNexpress for transient and steady-state analysis is plotted for varying model size. Furthermore, a performance curve for a measure of interest is presented.

Keywords

Performance evaluation methods, tools, and frameworks, quantitative evaluation of UML specifications, and Petri nets, numerical techniques for the analysis of discrete-event systems.

1. INTRODUCTION

To effectively employ model-based evaluation of computer systems, communication networks, and client-server systems, software packages are needed that simplify model specification, modification, as well as automate their quantitative analysis. The most popular language for model specification used in industrial projects is the Unified Modeling Language (UML, [2]). It is the proper successor to the object modeling languages of three previously leading object-oriented methods (Booch, OMT, and OOSE). The UML was invented by Booch, Rumbaugh, and Jacobson. In 1997, the UML was adopted as a standard by the Object Management Group.

There exists a large number of software packages for quantitative analysis of systems specified as queueing networks or stochastic Petri nets. Software packages for stochastic Petri nets include among many DSPNexpress [7], Möbius [12], and UltraSAN [13]. Such packages contain state-of-the-art quantitative analysis techniques and are widely distributed in academia. However, typically the recognition of such a package in industry is limited. Commercial UML design packages widely used in industry contain sophisticated user interfaces. However, such package either rely on outdated quantitative analysis methods or do not provide methods for quantitative system evaluation at all. To close the gap between commercial UML design tools and academic software packages for performance and dependability evaluation, UML system specifications enhanced by timing constraints must be transformed to stochastic Petri nets or even better to their underlying stochastic processes. Tool support for the quantitative evaluation of systems specified in the UML is particularly important for software performance engineering (see e.g., [14], [15]). Recently first approaches in this direction were made [5].

This paper describes the software package DSPNexpress 2000, a tool for the performance and dependability evaluation of systems specified in the UML, Petri nets, or other specification languages. The previous version of DSPNexpress, DSPNexpress1.5 was known for its highly efficient numerical method for steady-state analysis of discrete-event stochastic systems with exponential and deterministic events. As specification language for such discrete-event stochastic systems, DSPNexpress1.5 relies only on deterministic and stochastic Petri nets (DSPN, [1]). To outreach from Petri nets to system specification languages used in industrial projects, DSPNexpress 2000 can not only evaluate DSPNs, but also system specifications in the UML. In particular, DSPNexpress contains filters to the commercial UML design packages Rhapsody™ [11], Rational Rose™, and Together™. Such design tools allow the user-friendly specification and visualization of the artifacts of software systems and business process modeling. The package Rhapsody™ allows checking purely deterministic time constraints in UML specifications. The DSPNexpress software transforms UML specifications in which events have associated an exponentially distributed or a deterministic delay to a discrete-event stochastic system and, subsequently, maps them onto a generalized semi-Markov process (GSMP) [6], [7]. The DSPNexpress solution engine comprises of highly efficient numerical solvers for transient and steady-state analysis of GSMPs. Thus, linking the DSPNexpress software to commercial UML design packages allows the effective computation of performance measures like throughput and delay for UML system specifications enhanced by exponential and deterministic delays.

In previous work, we introduced these transient and steady-state solvers in the context of deterministic and stochastic Petri nets (DSPNs) with concurrent deterministic transitions [8], [9], [10]. The approach is based on the analysis of a general state space Markov chain (GSSMC) whose Chapman-Kolmogorov equations constitute a system of multidimensional Fredholm integral equations. The transition kernel of the GSSMC specifies one-step jump probabilities from a given state to all reachable new states. In general, a transition kernel is a functional matrix. Key contributions of the GSSMC approach constituted the observations that most of the elements of the transition kernel of the GSSMC are constants (99% for queueing systems like MMPP/D/2/K) and that the remaining elements comprise of piece-wise continuous functions. As shown in [9], transient analysis of quite complex DSPNs (i.e., with 20 thousand tangible markings for mission time T = 100) requires about 20 minutes of CPU time, steady-state analysis less than 5 minutes of CPU time. Furthermore, we showed that DSPNexpress 2000 performs transient analysis of DSPNs without concurrent deterministic transitions in a few minutes of CPU time; i.e., three orders of magnitude less computational effort than the previously known method [3].

The remainder of this paper is organized as follows. Section 2 outlines the innovative features of DSPNexpress 2000. Furthermore, the graphical user interface and the organization of the numerical solvers are explained in detail. In Section 3, the mapping of UML system specification onto generalized semi-Markov processes is explained and the filters of DSPNexpress to the package Rhapsody™ is presented. The applicability of the DSPNexpress software for practical performance and dependability projects is demonstrated by an UML specification of an alternating bit protocol in Section 4. The computational effort required by DSPNexpress for transient and steady-state analysis is plotted for varying model size. Furthermore, a performance curve for a measure of interest is presented.

2. DSPNexpress Software Architecture

2.1 Innovative Features of DSPNexpress 2000

The previous version of DSPNexpress, DSPNexpress1.5, is known for its highly efficient numerical method for steady-state analysis of deterministic and stochastic Petri nets (DSPNs, [1]) without concurrent deterministic transitions [6], [7]. Furthermore, DSPNexpress1.5 contained already a graphical user interface running under X11 allowing easy model specification, modification, graphical animation, as well as automate quantitative analysis. Novel innovative features of the DSPNexpress 2000 include:

(1) A robust implementation of an efficient numerical method for transient analysis of DSPNs without concurrent deterministic transitions based on an iterative numerical solution of one-dimensional Fredholm integral equations [9].

(2) A robust implementation of an efficient numerical method for transient and steady-state analysis of DSPNs with two deterministic transitions concurrently enabled [8], [9]. These tasks require numerical solution of two-dimensional Fredholm equations by an iterative scheme and direct quadrature, respectively.

(3) Orthogonal software architecture especially tailored to numerical analysis of the stochastic process underlying a discrete-event stochastic system with exponential and deterministic events (i.e., a Markov regenerative process or a generalized semi-Markov process) based on interprocess communication with UNIX sockets rather than writing intermediate results in files.

(4) Filters to the commercial design packages Rhapsody™ [11], Rational Rose™, and Together™. Thus, the numerical solvers of DSPNexpress can also be utilized for quantitative evaluation of system specifications with system specifications in the UML.

(5) Open interface of the numerical solvers so that they can easily be utilized for the quantitative evaluation of arbitrary discrete-event stochastic systems with exponential and deterministic events specified in other modeling formalisms than just DSPNs (e.g., embedded systems represented as Harel-statecharts or finite state machines).

2.2 The Graphical User Interface

Of course, the package DSPNexpress also provides a user-friendly graphical interface running under X11. To illustrate the features of this graphical interface, consider the snapshot shown in Figure 1. The first line displays the name of the package *DSPNexpress 2000,* the affiliation of the authors, *University of Dortmund, Computer Systems and Performance Evaluation Group,* and the year of release *2000*. A DSPN of a two-server, finite-capacity queue is displayed. The model is named *MarkovModulatedArrivalsTwoDetServers* because customers arrive according to a Markov modulated Poisson process and are serviced by two servers with deterministic service times. Recall that in DSPNs three types of transitions exist: immediate transitions drawn as thin bars fire without delay, exponential transitions drawn as empty bars fire after an exponentially distributed delay whereas deterministic transitions drawn as black bars fire after a constant delay.

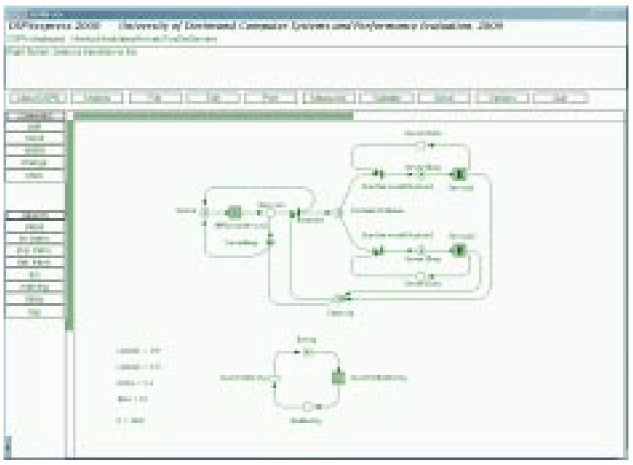
DSPNexpress 2000 University of Dortmund Computer Systems and Performance Evaluation, 2000

Figure 1. The graphical user interface of DSPNexpress

At any time, DSPNexpress provides on-line help messages displayed in the third line of the interface. The command line and the object line are located on the left side of the interface. The buttons are located in a vertical line between the on-line help line and the working area. The working area constitutes the remaining big rectangle which contains the graphical representation of the DSPN *MarkovModulatedArrivalsTwoDetServers*. This DSPN is displayed with the options *tags on*. Thus, each place and each transition of this DSPN is labeled (e.g., *Source*, *MMPoissonArrival*, *Decision*, *Accepted*, etc.). A detailed description of the features of the graphical interface is given in Chapter 10 of [7].

2.3 Organization of the Numerical Solvers

The core of the package DSPNexpress constitutes the solution engine for discrete-event stochastic systems with exponential and deterministic events. The software architecture of this solution engine and its software modules are shown in Figure 2. The solution engine is drawn as the big white rectangular box. The six software modules are drawn as rectangles. These software modules are invoked from the solution engine as UNIX processes. Interprocess communication with sockets drawn as broken ellipses is employed for passing intermediate results from one module to the next.

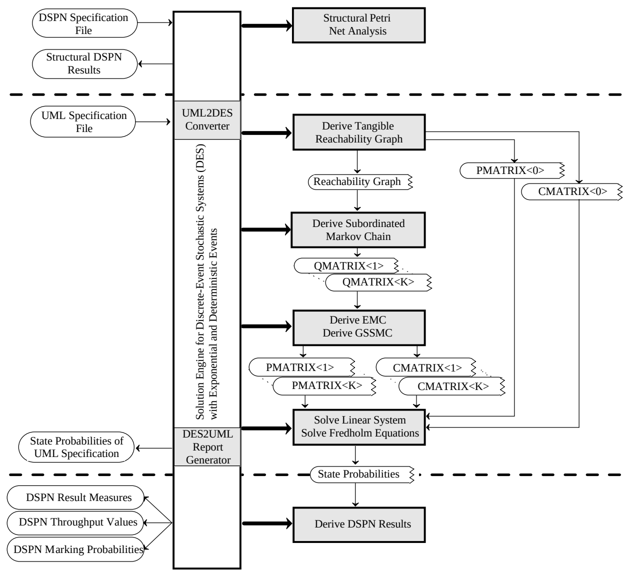

Figure 2. The software architecture of the numerical solvers of DSPNexpress 2000

The main idea when developing the solution engine of DSPNexpress was to create an open interface to utilize the numerical solvers for quantitative evaluation of arbitrary discrete-event stochastic system. In the current version of DSPNexpress discrete-event stochastic systems can be specified as DSPNs or as UML specifications especially statecharts and activity diagrams. DSPNs can be edited directly with the Petri Net editor of DSPNexpress and UML statecharts can be entered through the UML2DES converter of DSPNexpress and resulting measures are exported by the DES2UML report generator. In the following, we describe the solution engine for DES with exponential and deterministic events provided by DSPNexpress.

Steady-state analysis of DES without concurrent deterministic events relies on analysis of an embedded Markov chain (EMC) underlying such DES [1]. To efficiently derive the probability matrix of this EMC, the concept of a subordinated Markov chain (SMC) was introduced. Recall that a SMC associated with a state s_i is a CTMC whose states are given by the transitive closure of all states reachable from s_i via the occurrence of exponential events [7]. After generating the reachability graph comprising of tangible markings (states) of the DES, for each state the generator matrix of its SMC is derived. These tasks are performed in the modules *Derive Tangible Reachability Graph* and *Derive Subordinated Markov Chains*, respectively. Entries of this probability matrix are computed by transient analysis of the SMCs. A multithreaded execution can be employed for the computation of the entries of the probability matrix of the EMC using the sockets PMATRIX<1> to PMATRIX<K>. Similarly, the conversion factors required by the EMC approach are passed through the sockets CMATRIX<1> to CMATRIX<K>. Subsequently, a linear system corresponding to the stationary equations of the EMC is solved and the state probabilities of the continuous-time DES are derived using the conversion factors. These task are performed in the submodules *Derive EMC* and *Solve Linear System*. These software modules constituted the core of version 1.5 of DSPNexpress.

DSPNexpress 2000 contains two new software modules: *Derive GSSMC* and *Solve Fredholm equations*. Transient analysis of DES is based on the analysis of an embedded GSSMC. The Chapman Kolmogorov equations of the GSSMC constitute a system of Fredholm integral equations introduced in [9]. Steady state analysis of DES with concurrent deterministic events relies on the same approach [8]. Numerical computation of kernel elements of the GSSMC relies also on transient analysis of subordinated Markov chains and subsequent summation of appropriately selected transient probabilities [7]. This task is performed in submodule *Derive GSSMC.* As in case of the computation of the entries of the probability matrix of the EMC, a multithreaded execution can be employed for determining the kernel elements using the sockets PMATRIX<1> to PMATRIX<K>. Note that conversion factors are not required in the GSSMC approach. Thus, the sockets CMATRIX<1> to CMATRIX<K> are not used.

After *Derive GSSMC* has completed, for transient analysis of DES the number of iterations corresponding to the mission time is performed on the system of Fredholm equations [9] whereas for steady state analysis one large but very sparse linear system is solved using GMRES. This task is performed in the submodule *Solve Fredholm Equations*. As indicated in Figure 2, only the front end and back end of the solution engine are tailored to DSPNs.

3. Mapping UML Specifications onto GSMPs

The Unified Modeling Language (UML, [2]) provides system architects working on object analysis and design with one consistent language for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modeling. The UML provides different views of a model that were represented by graphical diagrams: *use case diagram*, *class diagram*, *behavior diagram*, and *implementation diagrams.* Among the behavior diagrams are *statecharts*, *activity diagrams* and interaction diagrams like *sequence diagrams* and *collaboration diagrams.*

In this chapter, we will show that UML specifications such as statecharts or activity diagrams can be represented as discrete-event systems. Therefore, it is possible to quantitatively analyze these specifications by means of a stochastic process. Statecharts provide a simple yet formal means of modeling the complex event-driven behavior common to embedded systems. All semantics necessary to express behavior - states, historical properties, timing, transitions, and compound transitional connectors - are available on the statechart palette. Statecharts allow a class to be defined in terms of the states it can be in and the events which cause it to move between states. They are essentially Harel statecharts [4]. States are drawn as rectangles with rounded corners and transitions are drawn as directed arcs. They are labeled with a trigger event, a guard, and an action:

`trigger[guard]/action`

All three parts of the transition label are optional. If the `trigger` event occurs and the `guard` is accepted the `action` is executed and the corresponding state change is performed. Events can be either timed events that trigger a state transition after a deterministic or exponentially distributed delay or they can be (immediately) triggered by actions of other transitions. To represent timed events, we define new syntactical expressions. The expression $\texttt{after(EXP(}\lambda_i\texttt{))}$ defines an event that triggers a state transition after an exponentially distributed delay with parameter λ_i. The expression $\texttt{after(DET(D}_i\texttt{))}$ defines an event that occurs after a deterministic delay D_i. Actions that generate (immediate) events are denoted by `GEN(Event)`. States can be ordered hierarchically and/or concurrently. Concurrent states are separated by dashed lines and the hierarchy of states is represented by embedding substates in superstates. The substate that is visited by entering the corresponding superstate is denoted with the default connector that is represented by a short arc without a starting state. An example of a statechart of an M/D/1/K queueing system is presented in Figure 3. The statechart consists of one superstate named *M/D/1/K* with two concurrent substates *Customers* and *Server*. The *Customers* state models the arriving process of customers to the queue. After an exponential distributed delay with parameter λ the state *Decision* is activated. If the queue capacity K is expired the customer will be rejected otherwise he will be put to the waiting line by increasing the variable *Queue*. The *Server* state consists of only one state *System*. A customer in the queue will be served in deterministic time D which is represented in the statechart with the construct `after(DET(D))`. This statechart has a shared variable *Queue* which consists of the number of customers waiting in the queue.

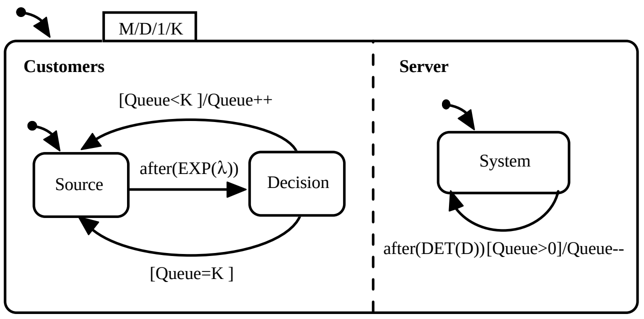

Figure 3. Statechart of an M/D/1/K queue

In Rhapsody, it is possible to graphically design the behavior of objects using statecharts and activity diagrams. The definition of the statechart can be stored in an export file with suffix *.sbs*. For timed events an extra specification file is needed to store the expected waiting delays and if the delays are deterministic or exponentially distributed. See Figure 4 for the specification of state *Source* of Figure 3. The definition of the state consists of the state ID, the name of the state, and the ID of the superstate it is nested in.

```
{ Istate
        - _id = GUID 9f22947a ...;
        - _myState = 128;
        - _name = "Source";
        - _inheritsFromHandle = { Ihandle
            - _m2Class = "";
        }
        - _parent = GUID 9f229478 ...;
        - _defaultTrans = ;
}
```

Figure 4. Statechart specification of state *Source* in Rhapsody

See Figure 5 for the specification of transition `after(EXP(λ))` of Figure 3. The specification of the transition consists of the identifier of the transition, the transition label with trigger event, guard, and action and the target and source state of the transition. This transition has no guard and no action so the variables `itsGuard` and `itsAction` are set to `NULL`. In Figure 6 the specification file to determine the expected waiting delays for timed transitions is presented. It consists of the name of the delay parameter, the type, and the parameter itself. In case of exponentially distributed delays it is the parameter λ which represents the arrival rate of customers and for deterministic timing it is the service delay D.

```
{ Itransition
        - _id = GUID 9f229485 ...;
        - _myState = 32;
        - _name = "transition_0";
...
        - _itsLabel = { ILabel
        - _id = GUID 9f229487 ...;
        - _itsTrigger = { IInterfaceItemTrigger
        - _id = GUID 9f22948d ...;
        - _body = "after(EXP(lambda))";
...
        }
        - _itsGuard = NULL;
        - _itsAction = NULL;
        }
        - _itsTarget = GUID 9f22947c ...;
        - _staticReaction = 0;
        - _itsSource = GUID 9f22947a ...;
}
```

Figure 5. Part of statechart specification of transition *after(EXP(λ))* in Rhapsody

For quantitative analysis of statechart expressions, we need to explore the state space of the statechart model. This state space consists of all possible tangible configurations. The reachability graph consists of directed connections between all configurations of the statechart. A *configuration* of a statechart is a snapshot of its execution. It consists of the active states of all concurrent states of the system and the setting of all variables. A tangible configuration is a configuration in which a timed event is activated. One can view a configuration as the information that is needed to completely restore the *"state"* of the statechart. Every tangible configuration of the statechart maps to a state of the corresponding state space of the underlying process and every configuration change of the statecharts corresponds to a state change in the underlying generalized semi-Markov process

(GSMP). This GSMP is the stochastic process that represents the evaluation of a discrete-event system over time.

```
lambda EXP    0.9
D      DET    1
```

Figure 6. Specification file for delay distributions associated with events

A GSMP 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).

The UML2DES converter shown in the software architecture of DSPNexpress (Figure 2) is the component of DSPNexpress for constructing the state space from the statechart specification. Then it is possible to evaluate the stochastic behavior of the statechart with DSPNexpress via the numerical solvers for discrete-event stochastic systems. Thus, linking the DSPNexpress software to commercial UML design packages allows the effective computation of performance measures like throughput and delay for UML system specifications.

4. Application Example

To illustrate the practical applicability of the DSPNexpress software for transient and steady-state analysis of discrete-event systems, we consider an UML-Statechart model of a reliable transport protocol. We present two QoS curves computed with DSPNexpress and one performance curve witch shows the CPU time used for solution for different model sizes. The experiments have been performed on a Sun Sparc Enterprise station with 1 GByte main memory running the operating system SunOS5.6. For the performance tests the CPU time has been measured by the UNIX system call *times*.

Figure 7 shows the alternating bit protocol modeled with UML-Statecharts as presented in [5]. The sender, receiver, and the two channels are modeled as different processes that operate concurrently. It is assumed that at most one message is in a channel as it is for example in an Ethernet. Furthermore we assume that packets that arrive from the network layer were stored in a queue with capacity K till the sender is ready to process them. The inter arrival time of packets to the sender is exponential distributed with rate *Arrival*. The timeout delay is modeled by a deterministic delay *TimeOut*. The delay of the network link is assumed to be exponentially distributed with rate *d*.

The alternating bit protocol consists of a *Sender* and a *Receiver* communicating via two channels with FIFO characteristic, a *Data Channel* for communication from sender to receiver and an *Acknowledgement Channel* for the other direction. Both can loose messages. The purpose of the protocol is to transmit data from the sender to the receiver in the correct order, although the medium can loose data and/or acknowledgement messages. Distorted messages are handled like lost ones. The protocol works as follows: The sender takes a message which is ready to be sent to the receiver. It transmits the message together with a sequence bit via the data channel to the receiver. Then the sender waits for an acknowledgement from the receiver containing the same sequence bit. If the appropriate acknowledgement arrives, the sender performs the same procedure for the next waiting message, but this time with an inverted sequence bit (i.e., 0→1, 1→0). If the appropriate acknowledgement does not arrive within a certain period of time, the sender resends the same message. This time period is controlled by a (deterministic) timer that is started as soon as the sender has transmitted the message to the receiver. The timer is stopped as soon as the acknowledgement arrives. The receiver acknowledges all incoming messages by including the sequence bit of the message received. The first time the message is received, the protocol delivers the message for processing. Subsequent messages with the same sequence bit are simply acknowledged and then discarded. The receiver now waits for the message with an inverted sequence bit.

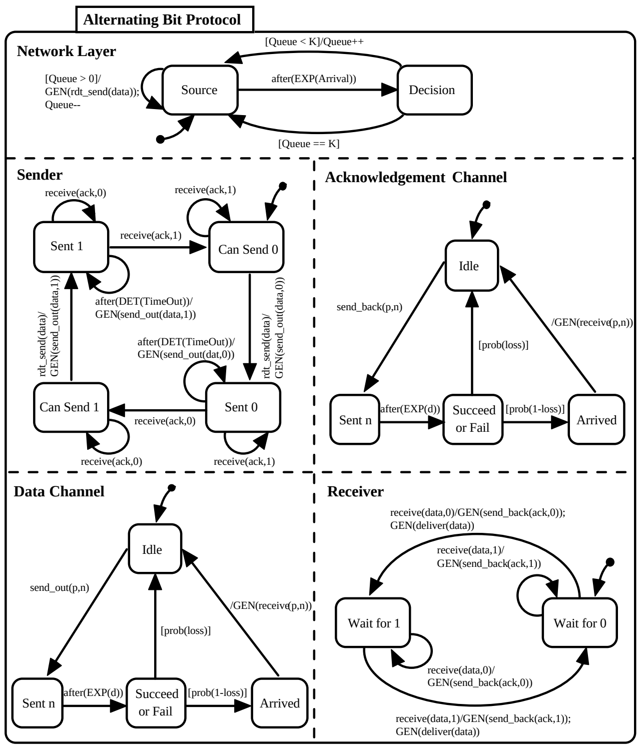

Figure 7. UML-Statechart of the Alternating-Bit-Protocol

Figure 8 presents a curve for a measure of interest of the alternating bit protocol. It plots the utilization of the data- and the acknowledgement channel for different packet arrival rates. We observe that the maximum utilization of the data channel is about 50%. The utilization for the acknowledgement channel is only 40%. This is due to the stop-and-wait characteristic of the alternating bit protocol.

Figure 9 plots the CPU time required for transient analysis at instant of time T = 100 and for steady-state analysis versus increasing model size. For this experiment, the buffer capacity K is varied from 125 to 1000. Note that the UML specification of Figure 7 does not contain deterministic events which are concurrently active. Thus, only transient analysis has to resort on the analysis of the GSMP, which requires the solution of a system of integral equations. Steady-state analysis can be performed by an embedded Markov chain and, hence, just requires the solution of a system of linear equations.

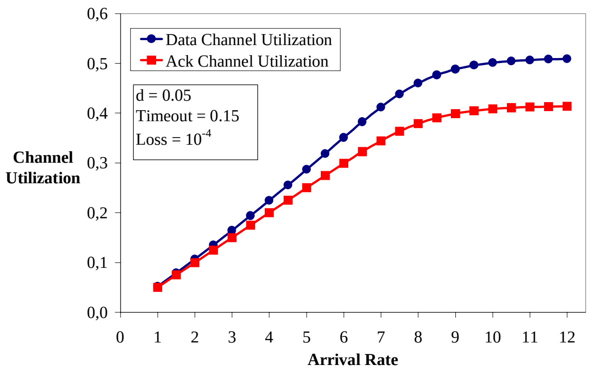

Figure 8. Arrival Rate versus Channel Utilization

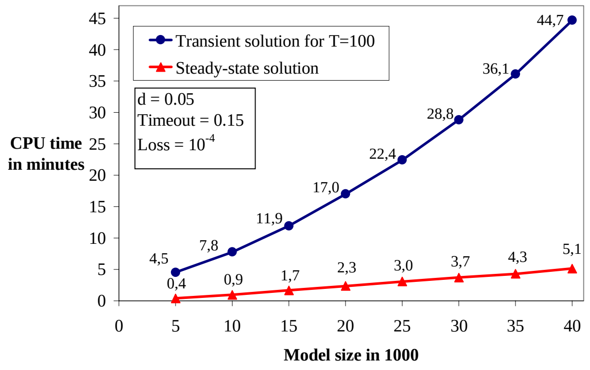

Figure 9. Transient and steady-state analysis: CPU time versus model size

5. Conclusions

This paper introduced DSPNexpress 2000, the new version of a widely distributed software package for performance and dependability modeling. While the previous version of DSPNexpress was known for its highly efficient numerical solver for steady-state analysis of DSPNs without concurrent deterministic transitions [6], DSPNexpress 2000 also provides an efficient method for transient analysis of DSPNs [9]. Furthermore, both the stationary analysis and the transient analysis is no longer restricted to the case that deterministic transitions cannot be concurrently enabled [8]. To outreach from Petri nets to system specification languages used in industrial projects, DSPNexpress 2000 can not only evaluate discrete-event systems specified as Petri nets but also system specifications in the UML.

To illustrate the applicability of the newly implemented solvers of DSPNexpress 2000, we considered a UML system model of an alternating bit protocol introduced in [5]. We presented curves for a measure of interest and for the computational effort required for transient and steady-state analysis. These curves evidently show that DSPNexpress 2000 can analyze quite complex discrete-event systems in which events have either associated exponential or deterministic delays in reasonable computing time.

6. REFERENCES

[1] M. Ajmone Marsan and G. Chiola, On Petri Nets with Deterministic and Exponentially Distributed Firing Times, in: *G. Rozenberg (Ed.) Advances in Petri Nets 1987, Lecture Notes in Computer Science 266*, 132-145, Springer 1987.

[2] M. Fowler, *UML Distilled: Applying the Standard Object Modeling Language*, Addison-Wesley 1997.

[3] R. German and A. Heindl, A Fourth Order Algorithm with Automatic Step Size Control for the Transient Analysis of DSPNs, *IEEE Trans. Softw. Engin.*, **25**, 194-206, 1999.

[4] D. Harel and M. Politi, *Modeling Reactive Systems with StateCharts: The StateMate Approach*, McGraw Hill 1998.

[5] P. King, R. Pooley, Derivation of Petri Net Performance Models from UML Specifications of Communications Software, *Proc. 11th Int. Conf. on Tools and Techniques for Computer Performance Evaluation, Schaumburg, Illinois*, 2000.

[6] Ch. Lindemann, DSPNexpress: A Software Package for the Efficient Solution of Deterministic and Stochastic Petri Nets, *Performance Evaluation*, **22**, 3-21, 1995.

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

[8] Ch. Lindemann and G.S. Shedler, Numerical Analysis of Deterministic and Stochastic Petri Nets with Concurrent Deterministic Transitions, *Performance Evaluation, Special Issue Proc. of PERFORMANCE '96*, **27&28**, 565-582, 1996.

[9] Ch. Lindemann and A. Thümmler, Transient Analysis of Deterministic and Stochastic Petri Nets with Concurrent Deterministic Transitions, *Performance Evaluation, Special Issue Proc. of PERFORMANCE '99*, **36&37**, 35-54, 1999.

[10] Ch. Lindemann and A. Thümmler, Numerical Analysis of Generalized Semi-Markov Processes, *Technical Report University of Dortmund*, (submitted for publication).

[11] Rhapsody, http://www.ilogix.com/.

[12] W.H. Sanders, Integrated Frameworks for Multi-Level and Multi-Formalism Modeling, *Proc. 8th Int. Workshop on Petri Nets and Performance Models, Zaragoza Spain*, 2-11, 1999.

[13] W.H. Sanders, W.D. Obal, M.A. Qureshi, and F.K. Widjanarko, The UltraSAN Modeling Environment, *Performance Evaluation*, **24**, 89-115, 1995.

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

[15] M. Woodside, Software Performance Evaluation by Models, in: *G. Haring, Ch. Lindemann, M. Reiser (Eds.) Performance Evaluation: Origins and Directions, LNCS State-of-the-Art Survey*, 283-304, Springer-Verlag 2000.