A Toolkit for Performability Evaluation Based on Stochastic UML State Machines

J. Trowitzsch[*]
Technische Universität Berlin
Real-Time Systems and Robotics
joni@cs.tu-berlin.de

D. Jerzynek
Technische Universität Berlin
Real-Time Systems and Robotics
danjerz@cs.tu-berlin.de

A. Zimmermann
Technische Universität Berlin
Real-Time Systems and Robotics
azi@cs.tu-berlin.de

ABSTRACT

This paper considers a sub-set of UML State Machines extended by annotations from the UML Profile for Schedulability, Performance, and Time for the modeling of technical systems and their behavior. A toolkit is presented for performability evaluation of these stochastic UML State Machine models. It extends our modeling and evaluation tool TimeNET. Performance evaluation of the resulting extended UML State Machine models is done indirectly via an automatic transformation into a stochastic Petri net, to which existing evaluation techniques are applied subsequently.

Keywords

Stochastic UML State Machines, Software Tool, Petri Nets, Model Transformation, Performance Evaluation

1. INTRODUCTION

Modeling and model-based evaluation of properties are an integral part of the development process of non-trivial systems. The resulting models allow to assess qualitative as well as quantitative system aspects in early stages of the development process already. Software tool support as well as the integration of modeling and evaluation techniques into a structured design process are necessary means for an efficient design process.

There are numerous modeling techniques which have often emerged for specific application areas. The *Unified Modeling Language* (UML) [17] is a successful attempt towards a collection of modeling techniques to describe technical systems, especially in the context of computing. UML is a widely accepted standard in industry now, but lacks some of the analysis techniques and formal clearness of approaches that have their origin in the scientific community. A combination of both advantages thus appears to be attractive.

We are mainly interested in a performance and dependability evaluation of a system, which requires models that are able to capture a system's behavior of interest. Out of the different description techniques available in the UML, we thus chose UML State Machines extended by annotations from the *UML Profile for Schedulability, Performance, and Time* (SPT) [15] for modeling systems and their characteristics.

For the resulting models the problem remains that performance measures can be obtained directly from the models with huge effort only. Several approaches can be found dealing with quantitative analysis of extended UML diagrams. These often origin from the field of software performance evaluation. Two different strategies exist: The direct strategy directly maps a UML model to the underlying stochastic process which can then be used for an evaluation. For example Lindemann et. al presented such an approach [12] based on a *generalized semi-Markov process* (GSMP). The alternative indirect strategy includes the mapping of a UML model into an established performance model such as a *Stochastic Petri Net* (SPN) or a *Queuing Network* (QN) model.

An approach for the systematic development of *Generalized Stochastic Petri Nets* (GSPNs) from UML models is proposed by Merseguer [13] and Bernardi et al. [3]. King and Pooley also use GSPNs in [11, 18] to represent the behavior of *StateCharts*. Each state is mapped into a place and each state transition becomes a transition in the Petri Net. The resulting submodels are combined using UML collaboration diagrams.

We proposed an indirect evaluation approach [24, 23] based on *Stochastic Petri Nets* (SPNs) [1, 7] in earlier papers. UML State Machine models are transformed into SPNs with equivalent behavior. Performance measures can be derived from the resulting SPN models using techniques and tools available for this model class.

This approach for modeling and quantitative evaluation of systems based on UML State Machines and Stochastic Petri Nets requires an appropriate tool support in order to be practicable. Tasks for which a support is necessary include modeling based on UML State Machines, transformation into a corresponding SPN, evaluation of the resulting SPN as well as the transformation of the computed performance results back into the UML model.

For the modeling part several commercial tools like *Poseidon for UML* [6] from Gentleware or *Rhapsody* [20] from Ilogix support modeling of UML and of UML State Machines in particular. A free tool that should be mentioned here is *ArgoUML* [2]. Besides that, several tools exist that support the evaluation of SPN models. Examples include GreatSPN tool [9] and TimeNET [21, 25, 26]. The latter integrates methods and algorithms for the numerical analysis and simulation of SPNs, especially if they contain transitions with non-Markovian delay distributions. However, it is desirable to have modeling and evaluation carried out within one tool. This includes an implementation of the mentioned transformation steps. The decision was thus made to extend the existing TimeNET software tool for both modeling and evaluation.

[*]The author's research work was supported by a PhD scholarship from the German Research Council (DFG) under grant GrK 621-2.

Throughout the work described here, TimeNET has been extended by the possibility to model (a subset of) UML State Machines. For this task a new *stochastic State Machine* (sSM) net class is introduced; net classes represent the types of models that the tool can handle. Furthermore, the model transformation is integrated in such a way that the existing SPN support of TimeNET can be used for the quantitative evaluation of the models. The results of these evaluations are not directly reported back into the sSM model yet. This is subject of further work.

The remainder of the paper is organized as follows. In Section 2 the considered subset of UML State Machines is briefly touched. Furthermore, the corresponding model transformation into SPNs is recalled. The software tool itself is introduced in Section 3. Here also the integration of the extension into the tool's software architecture is explained. Section 4 presents the use of the developed extended software tool showing the differences to other CASE tools. Finally, a summary and an outlook about future work is given in Section 5.

2. MODEL AND TRANSFORMATION

In this section we introduce the considered subset of UML State Machines and shortly explain the applied transformations into corresponding SPNs.

UML [17] is a collection of semi-formal models for specifying, visualizing, constructing, and documenting models of software systems and of technical systems. Various diagram types allow the description of different system viewpoints. Static and behavioral aspects, interactions among system components, and implementation details are captured. UML comprises an extension mechanism, which allows the definition of *profiles*. A profile for a special application domain maps aspects from the domain to elements of the UML metamodel. The *UML Profile for Schedulability, Performance, and Time* (SPT) [15] is an example for such a profile. It enables advanced annotation of quantitative system aspects such as timing and probabilistic information. For this a set of *stereotypes* and *tagged values* is provided. Among the behavioral UML diagrams our focus is on UML State Machines. In combination with the SPT profile they enable detailed modeling of quantitative system aspects. In our work we consider a subset of these extended UML State Machine. It is referred to as *stochastic State Machines* (sSMs) in the following.

SSMs comprise simple states, composite states, state transitions, pseudostates, and events. States may have optional internal activities, the so-called *entry*, *do*, and *exit* activities. The entry activity is always executed upon entering the state, i.e. prior to any other internal behavior. The exit activity is always performed whenever the state is left, regardless of which transition is taken. Pseudostates are transient vertices with a special semantics. They can be used to connect multiple transition paths into more complex ones. Pseudostates allowed in sSMs include initial, join, fork, junction, choice, entry and exit point, as well as shallow history. Furthermore the usage of counter variables, final states, and intra-synchronization between regions of composite states is allowed [23].

Sterotypes from the SPT profile are used to annotate quantitative aspects. Stereotypes `RTdelay`, `RTaction`, and `RTevent` with their tagged value `RTduration` are used to add timing information to corresponding sSM elements like internal activities of states or state transitions. The `PAstep` stereotype with its tagged value `PAprob` in combination with a choice pseudostate is used to express probabilistic branching within a sSM model. Composite states are used to build hierarchical models. In order to query the resulting sSM model we propose an additional lightweight *PQprofile* performance query sub-profile extension to the standard SPT profile. Table 1 shows the proposed stereotypes and related tagged values. Stereotype *PQstate* can be associated with simple and composite states to measure the probability for being in these states. The *PQtransition* stereotype can be associated with transitions to query their throughput. Life time of an object described by the sSM model can be derived using the *PQcontext* stereotype with its tagged value *PQlifeTime*.

Stereotype	Tagged Value	Explanation
PQstate	`PQprob`	probability for being in a state
PQtransition	`PQthroughput`	throughput of a transition
PQcontext	`PQlifeTime`	mean lifetime of the object

Table 1: Stereotypes of the Performance Query sub-profile

For the quantitative evaluation of the sSM models an indirect approach using SPNs [1, 7] was proposed in previous work [22, 23, 24]. A Petri Net is a bipartite graph. Its vertices are denoted as *places* and *transitions*. Places may include *tokens*. The distribution of all tokens over the places corresponds to the state of the model. Places and transitions are connected via *arcs*. These arcs describe the dependency of the active elements (transitions) on tokens in places and their changing due to the firing of transitions. Detailed definitions and overviews can be found for example in [14, 19].

Stochastic Petri Nets include stochastic specifications such as firing delay distributions for transitions. Thus, modeling and evaluation of quantitative system aspects is enabled, see [1]. The resulting SPNs of our transformation approach belong to the class of *extended Deterministic and Stochastic Petri Nets* (eDSPNs) [7]. Timed transitions are drawn as an empty rectangle if the firing time is exponentially distributed, as a black rectangle if the firing time is deterministic or as a gray rectangle if the firing time is a general distribution function. Immediate transitions are drawn as small rectangles and describe activities that have zero delay. Due to the assigned firing weights a nondeterministic solution of conflicts is possible.

Transformation rules for basic states as well as for pseudostates and annotations from the SPT profile were proposed in earlier work and are shortly recalled and explained in the following. A sSM model is decomposed into its basic elements. For each element a transformation into a corresponding SPN fragment is accomplished. The transformations take into account that annotations from the SPT profile may possibly be present. Based on naming conventions, the individual resulting SPN fragments are combined to one SPN, whose behavior is the same as for the original sSM model.

Transformation of simple states as presented previously [23], where dummy SPN elements were generated even for empty sSM actions, required a subsequent simplification step which deleted meaningless SPN elements structurally. In difference to this, we now decided to avoid the generation of unnecessary SPN elements on the fly. This means that if an optional internal activity is not specified for a simple sSM state, no immediate transition is generated at all. Hence, for each of the eight possible combinations of internal activities, a different transformation rule exists. The `RTduration` tagged value annotations are used to refine the timing of the resulting transitions in the SPN. A detailed explanation is given in [23] by means of the `RTdelay` stereotype. Transformations for initial, join, fork, and choice pseudostates were presented in [22]. In the resulting SPN, probabilistic branching as described by the choice pseudostate is accomplished by conflicting immediate transitions with adequate weights in the SPN model.

3. A SOFTWARE TOOL

This section presents a software tool that supports quantitative evaluation based on sSMs as described above. Basics of the software tool are introduced in the following, and the integration of the extension into our TimeNET tool will be explained in detail.

3.1 TimeNET

TimeNET is a software tool for modeling and performance evaluation of stochastic Petri nets, especially for such with non-exponentially distributed firing delays. It has been developed and maintained at the modeling and performance evaluation group of Technische Universität Berlin. Its functions are being continuously extended, mainly as a result of PhD theses and industrial projects. Thus, it is important that its overall tool architecture and its graphical user interface (GUI) are extendable and adaptable to new net classes and analysis algorithms. Analysis components are kept modular with well-defined interfaces.

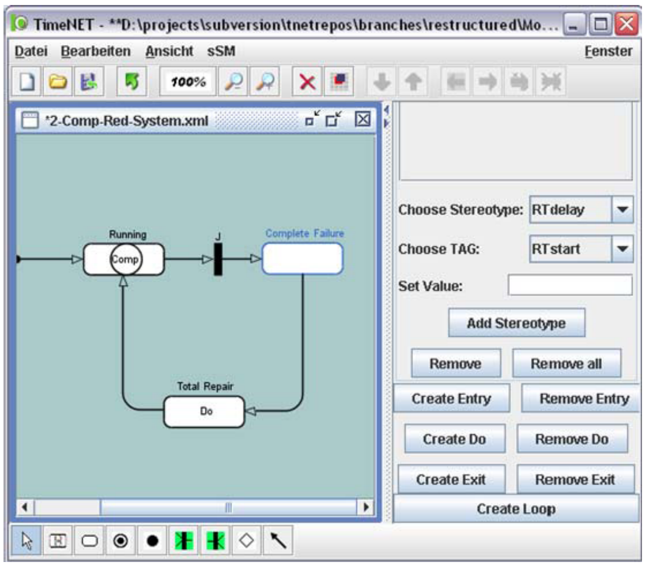
The main components of TimeNET are the GUI and the analysis and simulation algorithms. They are usually started as background processes from the GUI. Data exchange between GUI and analysis algorithms is mainly done via data files, while sockets are used between analysis processes for efficiency reasons.

The GUI is one of TimeNETs main components and has been completely rewritten in Java [10] for the current version 4.0. It can therefore be run on both Unix- and Windows-based environments. It is a generic GUI especially in the sense that any graph-like modeling formalism can be easily integrated without much programming effort. Nodes can be hierarchically refined by corresponding submodels. The GUI is thus not restricted to stochastic Petri nets. As a stand-alone program it is named PENG (*platform-independent editor for net graphs*) [26]. The tool architecture allows to run the GUI on a client desktop PC, while computationally expensive simulations may run on a remote server. Both parts may be located on the same host as well.

Program modules can be added to the tool which implement net class specific algorithms. A module has a predefined interface to the main program. It can select its applicable net class and extend the menu structure by adding new algorithms. All currently available and future extensions of net classes and their corresponding analysis algorithms are thus integrated with the same look-and-feel for the user.

Models as well as net class descriptions are stored in XML [4] format based on corresponding XML Schema [5] descriptions. Such a XML Schema defines the allowed elements of a model type. Node objects, connectors and miscellaneous others are possible elements. For each node and arc type of the model the corresponding attributes and the graphical appearance is specified. The shape of each node and arc is defined using a set of primitives (e.g. polyline, ellipse, and text). Shapes can depend on the attribute value of an object. This allows e.g. to show tokens as dots inside places in the case of SPNs. Actual models are stored in an XML file that must be consistent with the model class definition, which can be checked automatically with library toolkits for XML. Editing and storing a model can already be done with the tool after the corresponding XML Schema is available.

An example screen shot of the GUI is shown in Figure 1. The chosen net class in the example is the sSM net class corresponding to the type of models considered in this paper. Standard menus with necessary editing commands can be found in the top row. The commands are self-explaining and of typical GUI style. Frequently used menu commands may also be accessed by a set of icons below the menu bar.



Choose Stereotype: RTdelay

Choose TAG: RTstart

Set Value:

Figure 1: Screenshot of the TimeNET GUI

The main window contains the editing area. Editing is done just like in standard drawing tools with mouse-based operations for selecting, moving, and others. The lower icon bar shows all available model objects for the currently edited net class. The content of this bar is derived automatically from the net class description. Individual attributes of a model element are edited by selecting it in the drawing area and changing the values in the right tab. For each object attribute defined in the net class for that object type an entry for editing can be found in the right tab.

3.2 Integration of Stochastic State Machines in the TimeNET Tool

A new *stochastic State Machine* (sSM) net class has been developed and integrated in TimeNET. For this purpose a corresponding new net class description XML schema was implemented. It specifies the elements of a sSM model with their corresponding attributes and graphical appearances. Some sSM model element representations differ from the common standard. This is due to the specific way how models are implemented in TimeNET's GUI.

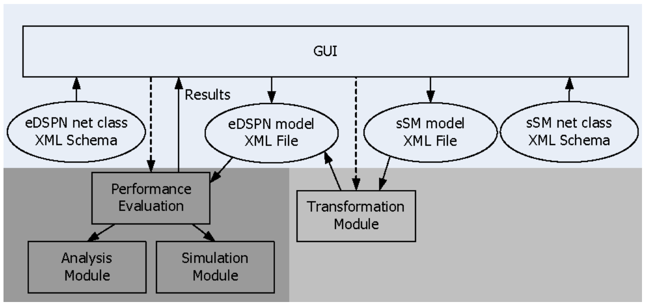


Figure 2: SSM net class integration

Figure 2 sketches the software architecture of the sSM netclass integration within TimeNET. Based on the sSM net class XML

Schema description the GUI allows to create a sSM model. Such a model is stored in a corresponding XML file. A net class specific transformation module implements the transformation of a sSM model into a corresponding eDSPN model by applying the transformations mentioned previously. The resulting eDSPN model XML file is written into a file, adhering to the eDSPN net class description XML Schema. For eDSPN models the net class specific performance evaluation including corresponding analysis and simulation modules is available. The results of such a performance evaluation are displayed in the GUI.

Stereotype	Tagged Value	sSM Elements
RTdelay	`RTduration`	Transition Activities
RTevent	`RTat`	Event (trig.)
PAstep	`PAprob`	Transition (choice)
PQstate	`PQprob`	Simple state Composite State
PQtransition	`PQthroughput`	Transition

Table 2: Supported stereotypes of the sSM net class

The sSM net class does not yet support all elements from the subset of extended UML State Machines presented earlier in Section 2. The not yet supported elements comprise the junction pseudostate, the history pseudostates, the terminate pseudostate, the entry and the exit point pseudostates. The featured modeling elements are composite states, simple states, final states, initial pseudostates, join pseudostates, fork pseudostates, choice pseudostates, and transitions. Furthermore, a set of stereotypes and related tagged values from the SPT profile and from the proposed additional lightweight *PQprofile* performance query sub-profile is included. These annotations can be added to certain elements like transitions or simple states. We point out that so far only annotations are offered that are supported by the mentioned transformation into SPNs. Among the supported stereotypes are *RTdelay*, *PAstep*, and *RTevent*. Table 2 summarizes the stereotypes that are currently supported by the sSM net class. Besides that, the sSM model class follows our recommendations for modeling UML State Machines. For example, an initial pseudostate is automatically introduced for each region of a composite state in order to ensure a correct default entry into the composite state. Furthermore, at most one final state is allowed for each region.

3.3 Tool Operation and Use

Figure 1 depicts the GUI while editing an example sSM model. The lower icon bar for the sSM net class includes the following model elements (from left to the right): selection mode, composite state, simple state, final state, initial pseudostate, join pseudostate, fork pseudostate, choice pseudostate, and state transition. The representations of these sSM elements are depicted in Figure 3. Composite states are depicted as empty rounded rectangle containing a circle and the text *Comp* in it. A simple state is depicted as an empty rounded rectangle. Final states are displayed as an empty circle with a smaller solid black circle in it. An initial pseudostate is shown as a small solid black circle. Join pseudostates are depicted as small black rectangles with a *J* above it, whereas fork pseudostates are displayed as small black rectangle with a *F* above it. The choice pseudostate is represented by a rotated empty square. The representation of transitions is not depicted in Figure 3. They are displayed as directed arcs connecting two sSM elements as can be seen in the later example.

Visible elements can be selected, their attributes be edited, and

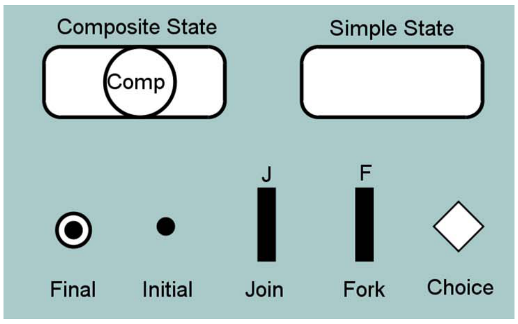


Figure 3: Representations of the sSM elements

additional action buttons be used if the selection mode is activated. These additional attributes and action buttons appear in the right tab of the GUI. Each simple state has a *text* attribute that specifies its name. Optional internal activities can be added using the *Create Entry*, *Create Do*, and *Create Exit* buttons in the right tab (see Figure 1). If specified, they are displayed within the state representation in the drawing panel. Attributes for an internal activity are a *name* and a *stereotypes* list of attached stereotypes. Internal activities can be removed from a simple state using the *Remove Entry*, *Remove Do*, and *Remove Exit* button (see Figure 1). The *Create Loop* button allows to include a self-transition for the selected simple state.

Stereotypes can be added to a selected element if an *Add Stereotype* action button is available in the right tab. After choosing the desired stereotype (*Choose Stereotype*) and related tagged value (*Choose TAG*), the value needs to be entered as a text. The value has to conform to the syntax for *RTtimeValue* from the SPT profile specification [15, Sec 5.2.]. Afterwards the action button for adding the stereotype should be clicked. The *Remove* action button can be used to remove one single selected stereotype from the stereotypes list of a sSM element. By using the *Remove all* action button the complete stereotypes list is removed from the sSM element.

In standard CASE tools like ArgoUML, UML State Machines are depicted as one connected graph. However, a sSM model features an abstraction level where each region of a composite state is described in a model part and graph on its own. Because of that abstraction the look and feel for modeling sSM composite states differs from other CASE tools. Entry and exit activities can be added using the corresponding *Create Entry* and *Create Exit* buttons for a composite state in the right tab. Do activities are not allowed for sSM composite states. The internal activities can be removed from a composite state using the corresponding *Remove Entry* and *Remove Exit* buttons. The *Create Loop* button allows to include a self-transition for a selected composite state just like for simple states. If a composite state is drawn, it is possible to specify the number of contained regions in the *numberOfRegions* attribute (the default is 1). Each composite state also has a *text* attribute specifying its name. The upper icon bar below the menu offers operations for navigating between the different regions of a composite state but also to add or remove a region. The submodel for the first region is opened by double-clicking the composite state. Elements connected to the composite state via transitions are visible in that submodel with dashed borderlines. This indicates that they are outside the current submodel, which means outside the composite state's region.

A triggering *event* can be attached to a transition. Such an added event can be selected to add a stereotype to it. These stereotypes are stored in the *eventStereotypes* list. Connected outside elements are not displayed inside submodels (regions) if the connecting transition directly points to the border of the composite state. This is the case when default entering or exiting is modeled. It is specified by setting the *connectsToBorder* and *startsFromBorder* attributes of the relevant transition to appropriate values.

The transformation module for the sSM net class can be started via the menu command *sSM → sSM to eDSPN*. The name for the resulting eDSPN model XML file can be chosen. After loading that resulting eDSPN model, all existing menu commands for the eDSPN net class, including analysis and simulation algorithms, are available.

4. A SIMPLE EXAMPLE

In this section we show the exemplary usage of the sSM net class in TimeNET. The differences to UML State Machines as supported by known CASE tools are explained. An imaginary two-component redundant system with local and global repair serves as an example.

The considered system consists of two redundant components that both are working correctly in the beginning, but may fail due to errors. If a component fails, a local repair is carried out. The first component fails every two days on average. The local repair of this component takes a fixed time of 30 minutes. The second component fails every three days on average. Its local repair takes fixed 45 minutes. If both components are failed at the same time, a complete system failure occurs and a global repair is necessary. It takes half a second until the global repair is started after the complete failure occurred. This global repair is successful with a probability of 99.95% only. In 0.05% of all cases another global repair is neccessary. The global repair itself requires a fixed time of two and a half hours to be performed. One interesting performance question is for example the probability that the system is working correctly.

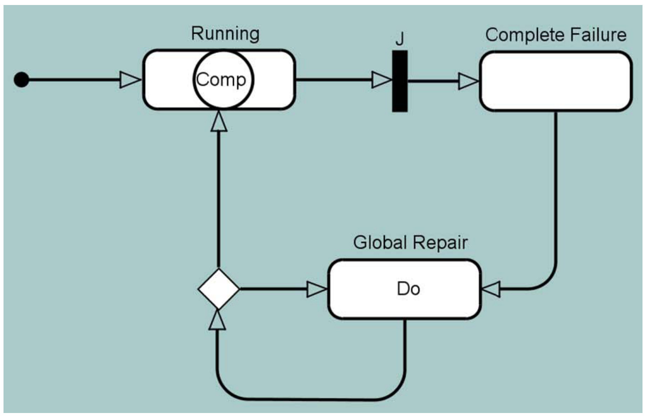


Figure 4: Top Level

The top level of the sSM model for the two-component redundant system as modeled in our software tool is depicted in Figure 4. Composite state *Running* includes two regions and models the situation when the system is working correctly, meaning that at least one component is working. A join pseudostate, which is depicted as a small black rectangle with a *J* above, is used to model the situation if both components did fail together. State *Complete Failure* is entered in that case, indicating that the whole system failed. The transition to state *Global Repair* has an *RTdelay* value attached to it, specifying its delay of 0.5 seconds. The *Do* depicted for state *Global Repair* represents the global repair activity. Its *RTdelay* value specifies the duration of the activity (2.5 hours). The subsequent rotated empty square depicts a choice pseudostate. The transition leading to the border of composite state *Running* represents a successful global repair. It has a *PAstep* stereotype attached, specifying a *PAprob* probability of 99.95%. The transition leading back to state *Global Repair* represents the case that the global repair was not successful and needs to be repeated. The *PAstep* stereotype attached to this transition specifies a *PAprob* probability of 0.05%. We chose a performance measure to evaluate the probability that the system is working correctly. It is included by attaching a *PQstate* and its related *PQprob* tag to composite state *Running*.

By double-clicking the composite state *Running*, the first of the two contained submodels (regions) is opened. Figure 5 shows this region of composite state *Running* depicting the behavior of the first of the two redundant components. Initially, the component is working correctly which is modeled by state *Ok*. Occasionally an error occurs (event *error1*) that triggers the transition to state *Failure*. The *error1* event is attached to a *RTevent* stereotype and a corresponding *RTat* tagged value specifying that the event occurs every two days on average with exponentially distributed delay (*RTat = ('exponential', 2, 'days')*). Two transitions leave from the *Failure* state. The transition back to state *Ok* is attached to a *RTdelay* specifying a *RTduration = (0.5, 'hr')*. This delay represents the time needed to perform a local repair of the first component. The transition leading to the join pseudostate from the top level (indicated by dashed border lines) is activated if the model part for the other component is in the corresponding *Failure* state as well.

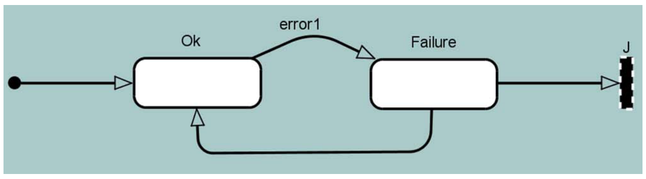


Figure 5: Orthogonal Region (one component)

The region depicting the behavior of the second component is almost identical to the first one as shown in Figure 5. The only differences are the varied timing values and the name for the triggering event indicating an error. This event is here *error2* and is attached to the corresponding *RTevent* stereotype and *RTat* tagged value specifying that the event occurs every three days on average (*RTat = ('exponential', 3, 'days')*). The transition from state *Failure* to state *Ok* is extended by a *RTdelay* specifying a *RTduration = (0.75, 'hr')* representing the time needed to perform the local repair of the second component.

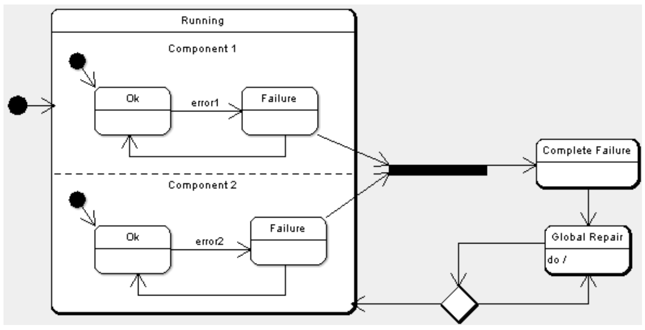


Figure 6: System in ArgoUML

A UML State Machine model for the same system is depicted in Figure 6, as it is displayed by the ArgoUML tool [2]. The same

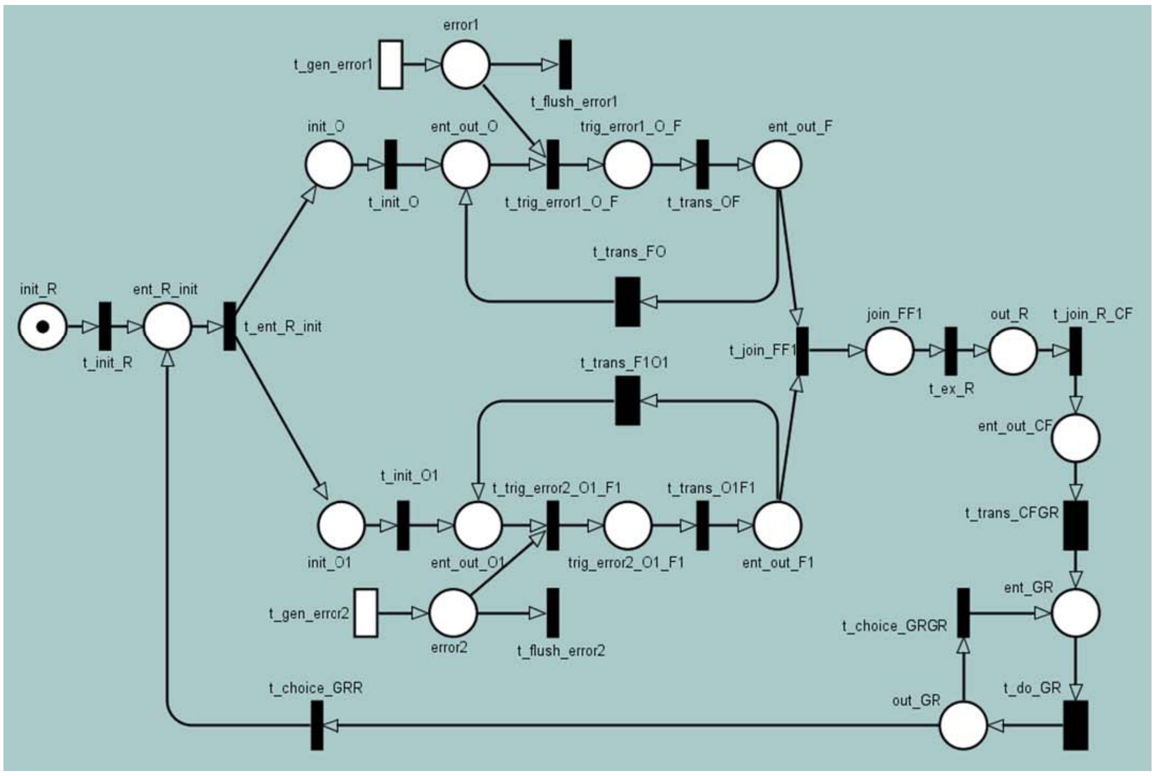


Figure 7: Resulting SPN after model transformation

names are used for the modeled elements as in our first model. The figure shows for instance the differences in displaying composite states, *Running* in our case. All regions including the submodels for the two redundant components are depicted inside the composite state as a whole connected graph in the ArgoUML tool. A hierarchical view is used in our tool, where each region of a composite state is described in a model part and graph for its own.

Performance evaluation of the sSM model of the considered two-component redundant system requires the transformation into the corresponding eDSPN model first. Figure 7 depicts the resulting eDSPN model after applying the model transformation approach as proposed in previous papers. Transitions *t_gen_error1* and *t_gen_error2* represent the occasional occurrence of events *error1* and *error2*, respectively. Probabilistic branching due to choice pseudostates is represented by the conflicting immediate transitions *t_choice_GRR* and *t_choice_GRGR*.

The probability that the system is working correctly can be calculated using the analysis algorithms or simulation component that are implemented for eDSPNs in TimeNET. The result shows that the system is working correctly with a probability of 99.91%.

5. SUMMARY AND OUTLOOK

The paper introduces a software tool for the performance evaluation based on stochastic UML State Machines. Our work extends the software tool TimeNET, comprising components for model specification as well as for performance evaluation. A *stochastic State Machine* net class that allows the modeling of a sub-set of stochastic UML State Machines has been implemented in the model-class generic graphical user interface of TimeNET for this task. From our experience people are more familiar with UML usage than with Petri net usage. The usage of the sSM net class in TimeNET does not require knowledge about Petri nets. Performance evaluation of the resulting sSM models is done by automatically translating such a model into a stochastic Petri net first, which is then subject to one of the existing quantitative evaluation algorithms of the tool. The results of the evaluations are not directly reported back into the sSM model yet. But since performance queries are specified at sSM elements the results can be related to these elements. The direct reporting of results into the sSM model is subject of further work.

With the proceeding acceptance of UML in different communities more and more UML tools came up, supporting system design, code generation, and testing. In order to accomplish a good interoperability between these varying tools the *XML Metadata Interchange* (XMI) interface [16] was adopted. XMI represents a standardized mechanism for exchanging and presenting UML models. It is an application of XML standardized by the World Wide Web Consortium (W3C). In the future the import of XMI-based UML State Machine models into our software tool will be implemented in order to enable openness to other CASE tools. Due to the restrictions from the generic GUI the imported models must be transferred into the supported sSM net class first. Performance evaluation methods can be applied afterwards. This conversion can be done by applying tools from the XML domain since XMI as well as the sSM net class schema of TimeNET are based on XML.

We are aware that other tools like ArgoSPE [8] also provide capabilities for modeling UML State Machines and performance evaluation methods. A comparison with our tool is difficult since different sub-sets of UML State Machines are supported and furthermore different Petri net classes are used for evaluation.

6. REFERENCES

[1] M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modelling with Generalized Stochastic Petri Nets*. Series in parallel computing. John Wiley and Sons, 1995.

[2] ArgoUML CASE tool. http://argouml.tigris.org.

[3] S. Bernardi, S. Donatelli, and J. Merseguer. From UML Sequence Diagrams and Statecharts to analysable Petri Net models. In *Proceedings of the 3rd Int. Workshop on Software and Performance (WOSP)*, pages 35–45, Rome, Italy, July 2002.

[4] W. W. W. Consortium. *Extensible Markup Language (XML)*. www.w3.org/xml.

[5] W. W. W. Consortium. *XML Schema 1.1*. http://www.w3.org/XML/Schema, 2001.

[6] Gentleware. Poseidon for UML. http://www.gentleware.com.

[7] R. German. *Performance Analysis of Communication Systems, Modeling with Non-Markovian Stochastic Petri Nets*. John Wiley and Sons, 2000.

[8] E. Gómez-Martínez and J. Merseguer. ArgoSPE: Model-based software performance engineering. In *ICATPN 2006*, volume 4024 of *LNCS*, pages 401–410. Springer-Verlag, 2006.

[9] The GreatSPN tool. http://www.di.unito.it/~greatspn.

[10] Java programming language. http://java.sun.com/.

[11] P. King and R. Pooley. Using UML to derive stochastic Petri net models. In *Proceedings of the 15th UK Performance Engineering Workshop*, pages 45–56, Bristol, UK, July 1999.

[12] C. Lindemann, A. Thümmler, A. Klemm, M. Lohmann, and O. Waldhorst. Performance Analysis of Time-enhanced UML Diagrams Based on Stochastic Processes. In *Proc. of the 3rd Workshop on Software and Performance (WOSP)*, pages 25–34, Rome, Italy, 2002.

[13] J. Merseguer. On the use of UML State Machines for Software Performance Evaluation. In *Proc. of the 10th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2004.

[14] T. Murata. Petri Nets: Properties, Analysis and Applications. In *Proceedings of the IEEE*, volume 77(4), pages 541–580, April 1989.

[15] Object Management Group. *UML profile for schedulability, performance, and time*. www.uml.org, March 2002.

[16] Object Management Group. *XML Metadata Interchange (XMI) Specification*, January 2002.

[17] Object Management Group. *Unified Modeling Language Specification v.2.0*. www.omg.org, August 2005.

[18] R. Pooley and P. King. The Unified Modeling Language and Performance Engineering. In *IEE Proceedings - Software*, volume 146(1), February 1999.

[19] W. Reisig. *Petri nets*. Springer Verlag Berlin, 1985.

[20] Rhapsody user guide. www.ilogix.com.

[21] TimeNET. pdv.cs.tu-berlin.de/~timenet.

[22] J. Trowitzsch and A. Zimmermann. Real-Time UML State Machines: An Analysis Approach. In *Object Oriented Software Design for Real Time and Embedded Computer Systems*, September 2005.

[23] J. Trowitzsch and A. Zimmermann. Using UML State Machines and Petri Nets for the Quantitative Evaluation of ETCS. In *Proc. of the 1st Valuetools*, Pisa, Italy, October 2006.

[24] J. Trowitzsch, A. Zimmermann, and G. Hommel. Towards Quantitative Analysis of Real-Time UML Using Stochastic Petri Nets. In *13th Int. Workshop on Parallel and Distributed Real-Time Systems*, April 2005.

[25] A. Zimmermann, J. Freiheit, R. German, and G. Hommel. Petri net modeling and performability evaluation with TimeNET 3.0. In *Proceedings of the 11th Int. Conf. on Tools and Techniques for Computer Performance Evaluation*, pages 188–202, Schaumburg, Illinois, USA, 2000.

[26] A. Zimmermann, M. Knoke, A. Huck, and G. Hommel. Towards version 4.0 of timenet. In *13th GI/ITG Conference on Measurement, Modeling, and Evaluation of Computer and Communication Systems, MMB 2006*, Nuernberg, March 2006.