# Using UML State Machines and Petri Nets for the Quantitative Investigation of ETCS

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

A. Zimmermann
University Potsdam
Hasso Plattner Institute for IT Systems Engineering
armin.zimmermann@hpi.uni-potsdam.de

## ABSTRACT

This paper proposes the modeling of technical systems and their behavior by means of Unified Modeling Language (UML) State Machines and the extending UML Profile for Schedulability, Performance, and Time. This Profile allows the detailed description of quantitative system aspects such as times and probabilistic choice. For the resulting models a transformation into a Stochastic Petri Net is established. The Petri Net's performance measures can be determined by simulation or numerical analysis. A part of the future European Train Control System (ETCS) serves as an application example. The relationship between ETCS communication quality and minimal distance between subsequent trains is investigated.

## Keywords

UML State Machines, Stochastic Petri Nets, ETCS, Model Transformation

## 1. INTRODUCTION

Developing complex technical systems requires adequate methods for modeling and evaluating the behavior of the later system. Evaluation of performance measures and reliability can be done by applying quantitative analysis methods.

The *Unified Modeling Language* (UML) [17] is a widely accepted modeling standard in industry. Without extensions UML does not allow modeling and evaluating of properties like timeliness, throughput or fault tolerance. This paper proposes modeling of technical systems my means of UML State Machines using the extending *Profile for Schedulability, Performance, and Time* (SPT) [16] to include quantitative system aspects in the model.

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

In order to allow a quantitative evaluation the resulting model is transformed into a *Stochastic Petri Net* following our approach presented in previous papers [21, 20]. The proposed transformation rules are extended in this paper. Performance measures for the Petri Net can be determined by applying simulation or numerical analysis methods. Existing results and software tools can thus be exploited for UML State Machines.

Several approaches can be found dealing with quantitative analysis of extended UML diagrams. These often origin from the field of software performance evaluation. Basically two different strategies exist. The direct strategy is to develop and apply an analysis that operates directly on the UML model. The indirect strategy includes the mapping of the UML model into an established performance model such as Stochastic Petri Nets or Queuing Network Models.

*Generalized Stochastic Petri Nets* (GSPNs) [1] are used by King and Pooley in [12, 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 sub models are combined using UML collaboration diagrams. Another approach for systematic development of GSPNs is proposed by Merseguer [14] and Bernardi et al. [2]. Extended UML diagrams are translated into labeled GSPN modules, which are merged into a complete model subsequently. In [9] also an extension of UML models with probabilistic choice and stochastic timing is proposed. These indirect approaches have in common that they are limited to exponentially distributed timing. Whereas our indirect approach also covers deterministic and even more general timing.

The idea of direct quantitative evaluation of the extended UML model without transforming it into another model class is followed by Lindemann et al. in [13]. Deterministic and exponential delays are considered. The resulting stochastic process is a *generalized semi-Markov process* (GSMP), which is numerically analyzable under certain structural constraints.

A part of the future *European Train Control System* (ETCS) is considered as application example. Operating at final implementation level it provides dynamic track assignment using radio communication, the so-called *moving block operation*. Parameters for reliability and timeliness are included in the existing specifications. But beyond it a detailed investigation of the behavior using a stochastic model for retrieving performance measures is needed.

In previous works [24, 21] a failure model for the used communication system *GSM-R* is developed and investigated. The present paper describes communication between trains and control center by means of *State Machines*. Quantitative evaluation of the resulting Petri Net shows the relationship between train distance and safe operation. Also the impact of the reliability of the communication link is shown. Considering the reliability for GSM-R required by the specifications, the results show that an efficient operation of ETCS is hardly reachable.

The remainder of the paper is organized as follows: Section 2 describes the extension of UML State Machines by quantitative information using the SPT profile. The transformation of such a model into an Stochastic Petri Net is explained subsequently. In Section 4 the application example is introduced. An UML State Machine model for the communication of ETCS is presented and transformed into a Petri Net. The interrelationship between train distance and an emergency stop forced by a communication error is calculated afterwards. Summary and outlook are given finally.

## 2. BACKGROUND

The *Unified Modeling Language* (UML) [17] is a collection of semi-formal models for specifying, visualizing, constructing, and documenting models of technical systems and of software systems. It provides various diagram types allowing the description of different system viewpoints. Static and behavioral aspects, interactions among system components and implementation details are captured. UML is very flexible and customizable, because of its extension mechanism. The extension mechanism of UML 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) [16] 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 especially the State Machines are suitable for modeling the systems behavior. Sequence charts and collaboration diagrams describe single cycles inside the system and therefor they are not directly useful for investigation of the whole behavior. UML State Machines are a variant of Harels StateCharts [7]. Because of the availability of suitable software tools they are widely accepted in industry.

At this point we abstain from giving a detailed description of UML State Machines and the SPT profile. In the following a short description is given how the relevant quantitative information may be represented within the State Machines by means of the SPT profile. For more detailed information we refer to the prior works cited before.

Figure 1 shows an example for a simple annotated UML State Machine. The most important elements within a State Machine are states (`A` and `B` in Fig. 1) as well as transitions. These model state transitions and are depicted as arrows between states. A simple State Machine is always in one state at most. To each state certain actions may be assigned. In Fig. 1 state `A` has an action `ac1` which is processed during entering of the state (*entry*). The actions `ac2` and `ac3` are processed during the stay in the state (*do*) and during leaving of the state (*exit*).

Transitions model a state transition and may depend on guards but also generate events. The complex *composite states* are divided into *regions*. Therefor they are suitable for modeling concurrent aspects. Each region specifies its own sequence having a local state. With it the description of parallel and synchronized processes is easier compared to the classical automata model. Furthermore State Machines include *pseudostates* which are abstractions that encompass different types of transient vertices.

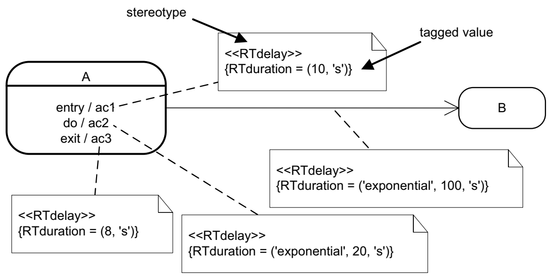

**Figure 1: Example of an UML State Machine with SPT annotations**

Fig. 1 includes examples for the usage of stereotypes and tagged values from the SPT profile. The stereotype `RTdelay` describes the delay of an action, for example 10 seconds for `ac1` in Fig. 1. Tagged values consist of a property name and an assigned value, for example `{RTduration=(8,'s')}`. For the timing information it is possible to specify distribution functions for stochastic timing, like `ac2` in Fig. 1. The number value indicates the expectation for the duration as specific parameter for the exponential distribution. The specification of probabilities at transitions can be done using the `PAstep` stereotype and its tagged value `PAprob`.

## 3. TRANSFORMATION OF UML STATE MACHINES INTO PETRI NETS

In the following we explain our approach for the transformation of UML State Machines into Stochastic Petri Nets aimed at quantitative evaluation. We introduce the transformation rules for the use of synchronization between regions of a State Machine and for the use of counter variables. In this context, basic transformation rules have been presented in [21, 20] in detail, so that we abstain from recalling them here. We are aware, that at this point we can not claim completeness for transforming all State Machine elements and constructs. However, they are subject of our ongoing work.

The approach is based on a decomposition of UML State Machines into its basic elements, like states, pseudostates, and transitions. For each element, transformation rules from State Machines into Stochastic Petri Net fragments are specified. Thereby the additional quantitative annotations from the SPT profile are taken into account. Additional timing information are of special interest, such as provided by the *RTdelay* stereotype. The resulting Petri Net fragments are finally composed following the original decomposition [21].

### 3.1 Stochastic Petri Nets

Petri Nets are a model applicable to discrete event systems where synchronized and concurrent processes play an important role. A Petri Net is a bipartite graph whose vertices are denoted as *places* and *transitions*. Places may include *tokens*. The distribution of all tokens over the places corresponds with the state of the model.

Arcs connect places and transitions and describe the dependency of the active elements (transitions) on tokens in places and their changing due to the transitions firing. For a detailed definition we refer to the extensive literature for this field. Overviews can be found for example in [15, 19].

Stochastic specifications such as firing times for the transitions were added to Petri Nets in order to enable modeling and evaluation of quantitative system aspects, see [1]. In the following *extended deterministic and stochastic Petri Nets* (eDSPNs) [6] are used. Figure 2 shows an exemplary eDSPN model describing a two-component redundant system with repairing. Components correspond to tokens in the places. At the beginning both components are intact (two tokens in *p1*).

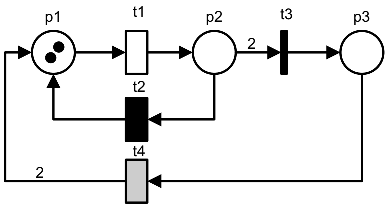

**Figure 2: Example of a Stochastic Petri Net**

Immediate transitions are drawn as small rectangles (*t3*) and describe activities that have no delays. Due to the assigned firing weights the non-deterministic solving of conflicts is possible. Corresponding to their firing times timed transitions are drawn as empty rectangle (*t1* - exponential distribution function), as black rectangle (*t2* - deterministic timing) or as gray rectangle (*t4* - general distribution function).

## 3.2 Transformation of Simple States and Transitions

The basic transformation of simple states and the involved fixed naming conventions have been introduced in detail in [21]. In the following we briefly explain how timing information is transformed into corresponding Petri Net transitions.

Time may be consumed within each state because of the optional internal *entry*, *do*, and *exit* activities. Regardless if the optional activities within a state are specified or not we always follow the temporal and logical order of the activities. If an activity is not specified or if no additional timing information is associated with the activity the corresponding transition in the resulting Petri Net is an immediate transition.

Table 1 presents possible annotations for the *RTdelay* stereotype from the SPT profile and the consequences for the transitions in the resulting Petri Net. Constant times result in deterministic transitions. Exponentially distributed timing results in exponential transitions with the corresponding rate $\lambda$. Stochastic timing with a known quantile results also in exponential transitions.

Figure 3 shows an example for the transformation of a simple state transition considering the timing annotations from the SPT profile. The missing *do* activity at state *A* results in the immediate transition *t_do_A*. The constant times for the *entry* and *exit* activities result in the corresponding deterministic transitions *t_ent_A* and *t_ex_A* respectively. The state transition from *A* to *B* is assigned by an exponentially distributed delay with the mean value of 100 seconds. The resulting exponential transition *t_trans_A_B* therefore has a rate $\lambda = 1/100$.

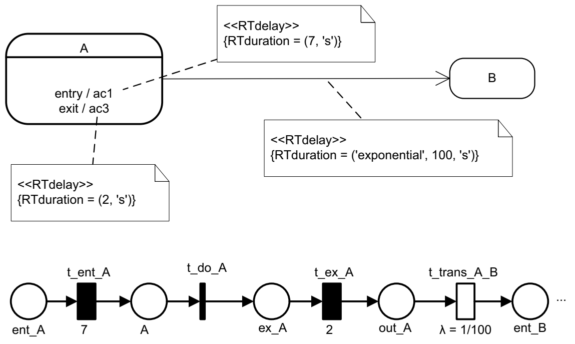

**Figure 3: Transformation of a simple state transition**

## 3.3 Transformation of special constructs

Special elements and constructs within UML State Machines include for example pseudostates, the synchronization between regions, or the usage of counter variables. Pseudostates are transient vertices with a special semantics which has to be considered during the transformation into a corresponding Petri Net fragment. In [20] transformation rules for *choice*, *join*, *fork*, *junction*, and *initial* pseudostates were introduced amongst others. In the following further extensions will be explained.

The synchronization of regions can be achieved by exchanging events. Fig. 4 displays an example for it. The regions are divided by a dashed line. In the upper region the state transition from *A* to *B* has the event *ev* as postcondition. Thus this event is generated when the state transition is taken. In the lower region, on the other hand, the state transition from *C* to *D* is only possible if the precondition *ev* is satisfied. Thus the event already has been generated by the state transition in the upper region. If the event has been generated but the state transition within the lower region is not possible due to other reasons, the generated event is discarded.

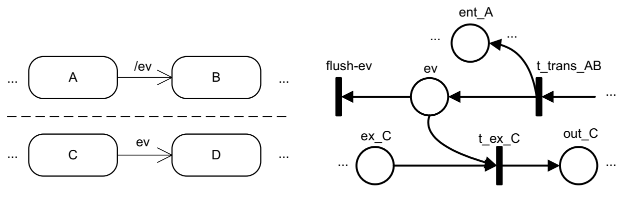

**Figure 4: Synchronization of regions using events**

The transformation results in a place with the same name as the event (place *ev* in Fig. 4). This place connects the generated Petri Net fragments for the involved single regions. In Fig. 4 this means for example that state *C* (*ex_C*) can only be left (*t_ex_C*) if place *ev* contains a token. This is the case if the state transition from *A* to *B* took place.

**Table 1: Stereotype *RTdelay* - tagged value *RTduration* transformation**

| Tagged Value | Petri Net transition |
|---|---|
| (8,'s') | deterministic - delay 8 sec |
| ('exponential', 32,'s') | exponential - rate $\lambda = 1/\text{mean}$ |
| ('percentile', 80, (5, 's'), 'exponential') | exponential - rate via $F(x) = 1 - e^{-\lambda x}$ |

If place *ev* contains a token but place *ex_C* does not the event (the token) is discarded by firing the lower prioritized transition *flush_ev.*

Using counter variables in a State Machine model is often reasonable, for example if the number of failures of a certain system component should be observed and is allowed to happen only a certain times. A simple example for the use of such a counter can be seen in Fig. 5. The variable *counter* is incremented each time a state transition from $A$ to $B$ happens, and set back to 0 during state transition from $B$ to $A$.

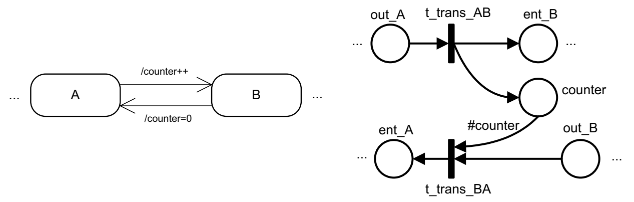

**Figure 5: Counter variable example in State Machines and resulting Petri Net**

The transformation generates for such a counter variable a corresponding place with the same name within the Petri Net (place *counter* in Fig. 5). The incrementation is represented by adding a token to this place. Resetting to 0 is represented by the firing of transition *t_trans_BA*. It removes all token from the place *counter* via the marking dependent arc inscription *#counter.*

## 4. AN APPLICATION

The future *European Train Control System* (ETCS) has been introduced in order to enable fast, efficient, and consistent train traffic across Europe. It is meant to replace the existing national systems. The traditional fixed block structure of the tracks and the release of those track blocks for a train is repealed. In the final implementation (ETCS level 3) a continuous assignment of free track blocks is introduced. Thereby an improvement of the bad track utilization because of the traditional fixed block structure of the tracks ought to be achieved. The traditional track side electromechanical infrastructure is replaced by a radio communication system. The tasks of classical railway control centers are handled by the *Radio Block Centers* (RBC). Every train actively checks its integrity and reports its position to the responsible RBC periodically. Every RBC observes the positions, speeds, and planned routes of the trains within its scope. It assigns to each train free track blocks on which the train can drive safely by transmitting *movement authority* messages to them. This method is called *moving block operation.* For it the reliable and timely data exchange via the radio interface as well as the data processing at the train and the RBC are critical issues for efficient and safe train traffic.

The data exchange between train and RBC is obviously an important issue because otherwise a train could not be informed about the free track blocks along its route. This would make the high speed operation impractical. The connection between trains and RBC is handled wireless via GSM-R (*global system for mobile communications - railway*), a variant of the known GSM system for mobile phones [3]. The radio communication was specified and designed in detail inside the EIRENE (*European Integrated Railway Radio Enhanced Network*) project [5]. The EURORADIO layer of the communication connection specifies the requirements for the radio communication [4, 11].

In the following the time critical procedure for the determination of the free track section is considered. Thereby the worst-case assumptions from the specification are used to calculate the guaranteed reachable best possible track utilization. First a train checks its integrity. This takes as per specification up to 5 seconds. Afterwards the train transmits the position of the end of the train from the beginning of the integrity check (min safe rear end) to the RBC. This is done periodically every $t$ seconds, with $t \geq 5sec$. Since the accuracy obviously becomes better if a train sends its position more often we assume in the following $t = 5sec$.

The position message is send via GSM-R to the RBC. This is specified to take between 400 and 500 milliseconds at middle. Processing of the data at the RBC takes 500 milliseconds. During this time the movement authority message for the subsequent train is generated. The transmission of this message again takes in middle between 400 and 500 milliseconds.

Communication via GSM-R is not safe. Data packages may be delayed or even get lost. Therefor each train must decide after a certain deadline if a continuation of the drive is no longer safe and an emergency braking has to be initiated. The deadline depends on the driven speed and on the length of the assigned free track section.

We consider two trains *Train 1* and *Train 2* which drive at the same speed $v$ and directly follow each other. The head-to-head distance is $s$. Our goal is the calculation of the deadline $d$ for the decision if *Train 2* has to initiate an emergency brake when no new movement authority message arrives. Fig. 6 illustrates this context. The train length (about $410m$ for German high-speed train ICE), the position error of not more than $20m$, and the braking distance (depending on actual speed between $2300m$ and $2800m$ have to be subtracted from the train distance $s$. We assume in the following the sum of these three parameters as $l = 3000m$.

In the worst-case *Train 1* crashes after an integrity check or might have lost coaches. Because of this the delay $a$ between receiving the message at *Train 2* and the integrity

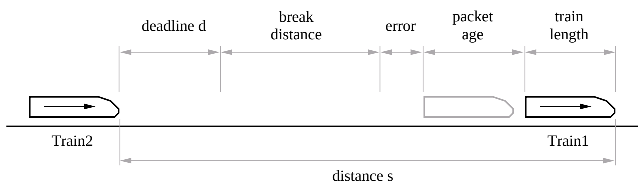

**Figure 6: Train Distance and Deadline**

check at *Train 2* also has to be subtracted from the available waiting time. According to the detailed information from the specification this delay $a$ is between 5 and 9 seconds.

The deadline $d$ now can be calculated respectively: $d = \frac{s-l}{v} - a$, whereas $v = 83ms^{-1}$ according to the speed of current ICE trains.

## 4.1 Train Communication Model

The ability to exchange data packets with position and integrity reports as well as movement authority packets is crucial for the reliable operation of ETCS. In the following we adopt worst-case assumptions based on the requirements from the ETCS specifications, because otherwise there would be no guarantee of a working integrated system. A model of the position report message exchange and emergency braking due to communication problems is developed below. The goal is to analyze the dependency between maximum throughput of trains and reliability measures of the communication system.

Fig. 7 shows the UML State Machine describing the ETCS communication. It includes five parallel regions which are explained in detail subsequently.

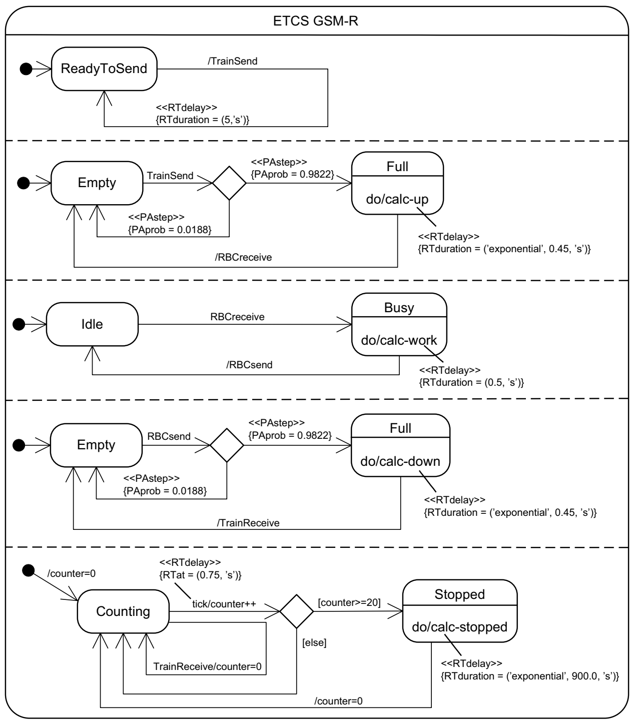

**Figure 7: UML State Machine model for ETCS communication**

The top region models the generation of position/integrity packages at *Train 1*. Such a package is generated every 5 seconds at which an event *TrainSend* is produced. The transmission of the data packages from the train to the RBC via the radio link is described in the region below. The radio link has two possible states *Empty* (no transmission activity) and *Full* (sending of data package). With the occurrence of the *TrainSend* event a new data package is ready to be send to the RBC. This data package is correctly send to the RBC with a probability of 98.22% and with a probability of 1.88% the transmission is incorrect. This is modeled using a *choice* pseudostate and the corresponding *PAprob* annotations at its outgoing transitions. These values result from the bit error rate of $10^{-4}$ given by the specification and the known package size of 190 bit: $P(error) = 1-(1-10^{-4})^{190} = 1.88\%$. The correct transmission takes 0.45 seconds in middle. This is the total transmission delay. We do not separate between the delays of the radio and the ISDN backbone transmission here. If the channel is empty again, an event *RCBreceive* is generated during the corresponding transition to state *Empty*. The next region models the behavior at the RBC. With the occurrence of event *RCBreceive* the transition from state *Idle* to state *Busy* is triggered. The processing of the received data package takes 5 seconds. During the subsequent transition to state *Idle* an event *RCBsend* is generated. The region below models the sending of a movement authority message from the RCB to *Train 2*. The only differences to the sending from train to RCB are the varying events that play a role. Event *RCBsend* activates the transition from state *Empty* to state *Full*. An error may again occur during transmission. An event *TrainReceive* is generated after a correct transmission . The lowest region models the observation of the deadline for receiving a new movement authority message at *Train 2*. A counter variable *counter* is used for it. Two states exist: *Counting* and *Stopped*. Every 0.75 seconds an event *Tick* is generated if a exemplary deadline of 15 seconds is considered. With each new *Tick* event *counter* is incremented. If *counter* has reached a value of 20 the train initiates an emergency breaking. For this a delay of 900 seconds in middle is assumed. Afterwards *counter* is set back to 0 and the train starts driving again. If *counter* is smaller than 20 state *Counting* is entered again waiting for the next *Tick*. A new movement authority message has been received if the region is in state *Counting* and the event *TrainReceive* occurs. In this case *counter* is set back to 0.

## 4.2 Resulting Petri Net

The presented model for the ETCS communication now is transformed into a corresponding Stochastic Petri Net. The resulting net includes several sequences of immediate transitions that have no influence on the behavior of the model. These sequences result from the single transformation of each State Machine element with actions or timing annotations possibly missing. Therefore the Petri Net is simplified by using two simple structural rules that avoid unnecessary *vanishing states*.

First, for all sequences *place - immediate transition - place* both places are merged and the transition is deleted. This is feasible if the immediate transition is the only connection between the places and not in structural conflict with any other transition. The second simplification considers all corresponding simple sequences *timed transition - place*

- *immediate transition*. In this case the place can be deleted and both transition can be merged so that one timed transition is left.

The resulting Stochastic Petri Net after applying the reduction rules is shown in Fig. 8. The initial state of each model part is reached via firing of transition *t_init*. Afterwards each model part has its own local behavior whose states are determined by the location of one mark. The RBC for example can be in the states *c_idle* or *busy* depending on if it currently waits or processes a message. The generation of a new message at the train in front happens every 5 seconds, so that the deterministic transition *t_trans_rts_rts* has a firing time of 5 and is immediately activated again after firing. The model parts for the up- and down link of the communication channel between train and RBC are quite similar. The probability of a package loss, for example because of incorrect transmission, is represented by the firing of one of the conflicting immediate transitions *t_choice_e* and *t_choice_f*. Both transitions have corresponding firing weights. The duration of the whole transmission is represented by the exponential transitions *t_do_full* and *t_do_full2* respectively.

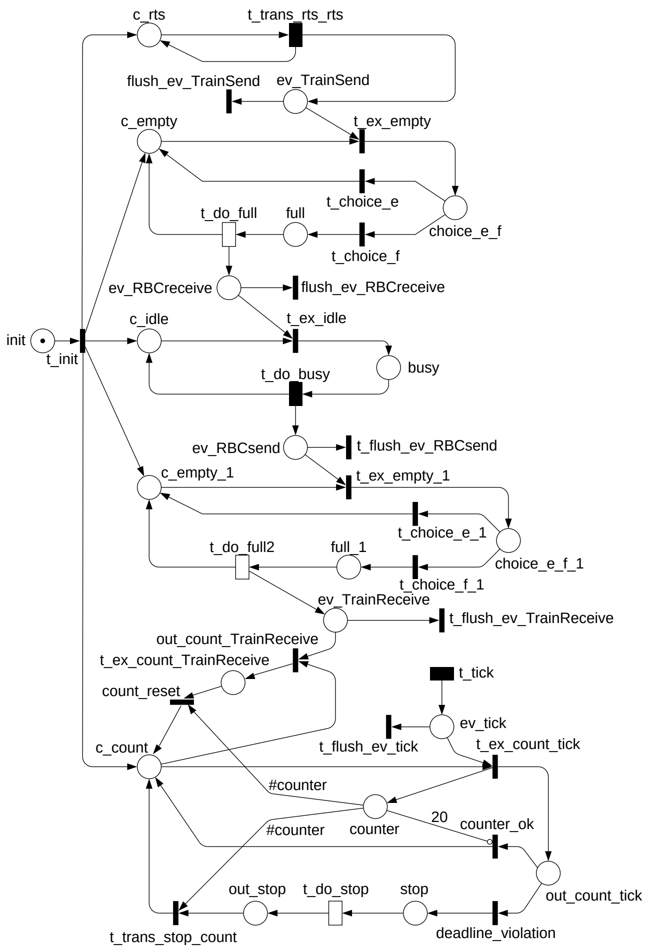

**Figure 8: Resulting Petri Net for ETCS communication**

The exchange of messages between the model parts is done using a similar construct. For example a message send from the train to the RBC is represented by a token in place *ev_TrainSend*. This is immediately send (because *t_ex_empty* has a higher priority) or dropped otherwise (firing of *flush_ev_TrainSend*). This corresponds to the semantics for the synchronization between regions of a State Machine as presented in section 3.3.

The lowest model part describes the behavior of the counter for the receiving deadline and initiation of an emergency braking. In the initial state a token is located in state *c_count*. Two events may occur: either a new message is received (*t_ex_count_TrainReceive* fires and afterwards all token from place *counter* are removed via *count_reset* using marking dependent transition inscription *#counter*) or a new *Tick* event is generated (*t_tick*). Afterwards the initial state is reached again if the counter variable has not reached a value of 20 yet and *counter_ok* fires. Otherwise an emergency stop is initiated. With the firing of *t_do_stop* it ends and the cycle starts again.

## 4.3 Results of the Quantitative Evaluation

The performance of the model can now be evaluated. The probability for a train being stopped because of a violation of the deadline can be obtained by the measure $P(Stop) = P\{\sharp counter >= 20\}$. A steady state analysis results in the mean probability during operation, i.e. the time a train spends in this undesirable state.

Calculation of this measure is not possible with one of the known numerical analytical methods because multiple non-exponential transitions are active at the same time. Use of standard simulation methods is rather limited because the relevant probabilities for an emergency stop are very small. This problem of *rare events* leads to unacceptable long calculation times. Therefore the investigations are done applying the RESTART method [22] which is a variant of importance splitting for the accelerated simulation of rare events. For the performance evaluation the TimeNET [23] tool is used. It includes an implementation of the RESTART method [10] for Stochastic Petri Nets. The number of tokens in place *counter* is used to define thresholds for the RESTART method. The tool calculates certain thresholds by using presimulation.

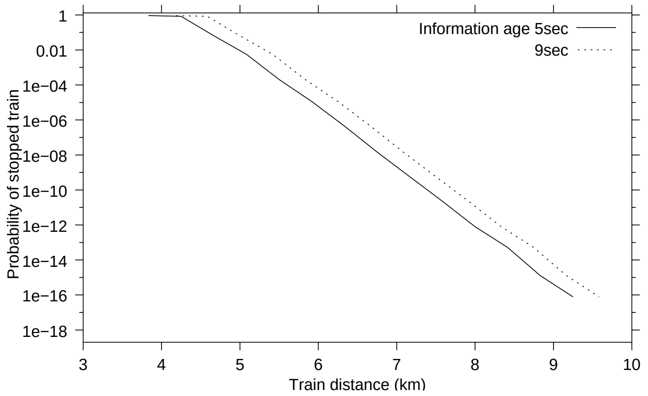

**Figure 9: Probability of train being stopped dependent on train distance**

Fig. 9 shows the relationship between train distance and the resulting probability for an emergency stop. Influence of the age of the received data is represented by two curves for 5 and 9 seconds, respectively. The curves display a nearly logarithmic dependency on the distance for the probability, from a distance $s$ of 4.5 km on. For a mean number of one emergency stop due to communication errors per train and year at most the distance must be at least $s = 6$ km.

The analysis shows the importance of the unsafe communication via GSM-R (package loss and delay) for the maximum possible track utilization when ETCS operates at level 3 implementation. The existing idea of driving at brake distance is unrealistic. Obviously larger distances are needed for safe operation. A more detailed comparison with the current fixed block operation can be found at [25]. Similar investigations have been driven out for example by Hermanns et al. [8].

## 5. CONCLUSION

The presented paper describes the modeling of the behavior of technical systems by means of UML State Machines and the subsequent quantitative investigation of such models. For this purpose, extensions from the UML Profile for Schedulability, Performance, and Time are used to introduce the necessary additional information such as delays and occurrence probabilities of actions into the UML State Machines. A method for the transformation of the resulting models into corresponding Stochastic Petri Nets is proposed. Performance measures can now be calculated by using known Petri Net software tools and algorithms.

As an application example a section of the communication between trains and Radio Block Centers (RBC) within the future European Train Control System (ETCS) is considered. In order to enable safe and efficient train traffic a safe, reliable, and timely exchange of communication data packages is needed, especially because track side infrastructure is no longer available for this purpose in the planned final implementation of ETCS. This contribution introduces a simple State Machine model of the communication as well as its transformation into a Stochastic Petri Net. The subsequent quantitative analysis by applying simulation methods for rare events shows the significant influence of the communication quality on the smallest possible distance between two trains. The results make it doubtful that efficient train traffic for high speed trains can be achieved under the given requirements. More detailed investigations are necessary in order to quantify safety and efficiency of technical solutions within this area more precisely.

The implementation of the explained methods as an extension of the new version of the TimeNET tool [26] is currently on the way. Direct editing of UML State Machine models within the TimeNET GUI will be enabled, but also the importing of models created in other UML editors. Basic transformation, analysis, and result presentation are already implemented within TimeNET.

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

[3] A. Coraiola and M. Antscher. GSM-R network for the high-speed line Rome-Naples. *Signal und Draht*, 92(5):42–45, 2000.

[4] EEIG ERTMS User Group. *Euroradio FFFIS*. UIC, Brussels, 2000.

[5] EIRENE Project Team. *EIRENE System Requirements Specification*. UIC, Brssel, 1999.

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

[7] D. Harel and M. Politi. *Modeling Reactive Systems with Statecharts: The StateMate Approach*. Wiley, New York, 1998.

[8] H. Hermanns, D. N. Jansen, and Y. Usenko. From stocharts to modest: a comparative reliability analysis of train radio communications. In *WOSP '05: Proceedings of the 5th international workshop on Software and performance*, pages 13–23, New York, NY, USA, 2005. ACM Press.

[9] R. Hopkins, M. Smith, and P. King. Two approaches to integrating UML and performance models. In *Proc. of the 3rd Int. Workshop on Software and Performance (WOSP)*, pages 91–92, July 2002.

[10] C. Kelling and G. Hommel. A framework for rare event simulation of stochastic Petri nets using RESTART. In *Proc. of the Winter Simulation Conference*, pages 317–324, 1996.

[11] D. Kendelbacher and F. Stein. EURORADIO - communication base system for ETCS. *Signal und Draht*, 94(6):6–11, 2002.

[12] 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.

[13] 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.

[14] 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.

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

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

[17] Object Management Group. *Unified Modeling Language Specification v.2.0*. www.uml.org, September 2003.

[18] R. Pooley and P. King. The Unified Modeling Language and Performance Engineering. In *IEE Proceedings - Software*, volume 146(2), March 1999.

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

[20] 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*, Erfurt, Germany,

September 2005. NetObjectDays.

[21] 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 (WPDRTS)*, Denver, Colorado, April 2005.

[22] M. Villn-Altamirano, J. Villn-Altamirano, J. Gamo, and F. Fernndez-Cuesta. Enhancement of accelerated simulation method restart by considering multiple thresholds. In *Proc. 14th Int. Teletraffic Congress*, pages 797–810. Elsevier, 1994.

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

[24] A. Zimmermann and G. Hommel. A train control system case study in model-based real-time system design. In *Proc. of the 11th Int. Workshop on Parallel and Distributed Real-Time Systems (WPDRTS)*, Nizza, 2003.

[25] A. Zimmermann and G. Hommel. Towards Modelling and Evaluation of ETCS Real-Time Communication and Operation. *Journal of Systems and Software*, 77(1):47–54, July 2005.

[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.