A Comparative Reliability Analysis of ETCS Train Radio Communications

by

Holger Hermanns[1,2] David N. Jansen[3] Yaroslav S. Usenko[4]

[1]Dependable Systems and Software Group,
Universität des Saarlandes, Saarbrücken, Germany

[2]Formal Methods and Tools Group,
Universiteit Twente, Enschede, the Netherlands

[3]Programming Logics Group,
Max-Planck-Institut für Informatik, Saarbrücken, Germany

[4]Laboratory for Quality Software,
Technical University of Eindhoven, Eindhoven, the Netherlands

email: `hermanns@cs.uni-sb.de, dnjansen@cs.utwente.nl, y.s.usenko@tue.nl`

AVACS Technical Report No. 2
February 2005
ISSN: 1860-9821

Publisher: Sonderforschungsbereich/Transregio 14 AVACS
(Automatic Verification and Analysis of Complex Systems)
Editors: Bernd Becker, Werner Damm, Martin Fränzle, Ernst-Rüdiger Olderog,
Andreas Podelski, Reinhard Wilhelm
ATRs (AVACS Technical Reports) are freely downloadable from `www.avacs.org`

A Comparative Reliability Analysis of ETCS Train Radio Communications*

Holger Hermanns[1,2] David N. Jansen[3†] Yaroslav S. Usenko[4]

[1]Dependable Systems and Software Group,
Universität des Saarlandes, Saarbrücken, Germany

[2]Formal Methods and Tools Group,
Universiteit Twente, Enschede, the Netherlands

[3]Programming Logics Group,
Max-Planck-Institut für Informatik, Saarbrücken, Germany

[4]Laboratory for Quality Software,
Technical University of Eindhoven, Eindhoven, the Netherlands

email: `hermanns@cs.uni-sb.de, dnjansen@cs.utwente.nl, y.s.usenko@tue.nl`

14 February 2005

Abstract

StoCharts have been proposed as a UML statechart extension for performance and dependability evaluation, and were applied in the context of train radio reliability assessment to show the principal tractability of realistic cases with this approach. In this paper, we extend on this bare feasibility result in two important directions. First, we sketch the cornerstones of a mechanizable translation of StoCharts to MoDeST. The latter is a process algebra-based formalism supported by the Motor/Möbius tool tandem. Second, we exploit this translation for a detailed analysis of the train radio case study.

Keywords. UML, stochastic systems, concurrency, tool support, reliability, wireless communication, European Train Control System (ETCS)

1 Introduction

The UML is pervading many challenging engineering areas including real-time and embedded system design. Embedded systems designers are usually facing various challenges if they strive for systems with *predictable quality of service* (QoS). Most QoS aspects of current embedded systems are time-related features and properties, and are of stochastic nature. While in principle the UML provides the right ingredients to model discrete event dynamic systems, it lacks support for stochastic process modeling.

Together with Katoen [17] we have proposed a QoS-oriented extension of UML statechart diagrams, StoChart, which enhances the basic formalism with two distinguished features. One enhancement allows state transitions to select probabilistically out of different effects, much like

*This work has been supported by the Dutch and German research councils as part of the bilateral cooperation program *VOSS*, as well as the German special research initiative *SFB/TR-14 AVACS* and the Dutch innovation award *NWO Vernieuwingsimpuls*.

†Current address: Formal Methods and Tools Group, Universiteit Twente, Enschede, the Netherlands

the rolling of a die can have one out of six effects, determined probabilistically. The second extension provides the "after" operator of statecharts with a stochastic interpretation, allowing the use of arbitrary probability distributions for modeling, such as EXP[10 min] for a negative exponential distribution with a mean of 10 minutes, or UNIF[10 h, 15 h] for a uniform distribution in the interval from 10 to 15 hours. The resulting statecharts dialect is called StoCharts, and contains UML statechart diagrams as a subset.

To make StoCharts a useful tool in QoS modeling, and to support trustworthy model-based QoS prediction, StoCharts are equipped with a rigid formal semantics [18]. This semantics combines concepts from timed, stochastic and probabilistic automata [1, 7, 25]. In order to associate a stochastic interpretation to collaborative collections of statecharts embedded in arbitrary environments, StoCharts are equipped with a compositional semantics, which uses concepts from Input/Output (I/O) automata [20]. The semantics associated with StoCharts is based on the requirements-level semantics of Eshuis and Wieringa [11].

The examples we studied with StoCharts so far [17, 16] show the principal modeling convenience of the formalism, but they also show that the lack of tool support is hampering its application. While drawing tools for UML statechart diagrams and StoCharts are at hand (e.g. TCM [10]), analysis tools which can digest StoCharts designs are missing. Therefore we have decided to invest in a tool environment for StoCharts. Instead of starting a new tool development, we decided to bridge to the ongoing activities in the context of the Motor tool [5]. Motor is linked to the Möbius tool set [9] for discrete event simulation-based analysis, and it uses the modeling and specification language MoDeST as an input language. MoDeST is a formal language to describe stochastic timed systems [8], equipped with a rigid formal semantics. The functional core of MoDeST can be considered as a simple process algebra enriched with some convenient language constructs, and a C-like notation. This core language is enriched with several modeling concepts tailored to model timed and/or stochastic systems. MoDeST has been successfully used in a number of nontrivial case studies, see for example [4].

The semantic basis of StoCharts and MoDeST is similar, since both map onto variations of timed and stochastic automata. Thus, at least in principle, it appears feasible to define a sound translational semantics which maps StoCharts designs onto MoDeST code. The latter can then be fed into the Motor/Möbius tool-tandem. However, several features of statecharts, such as border-crossing transitions, pose challenges to this semantics. This observation motivates the work reported in this paper, where we explore the most important elements of StoCharts and discuss their counterparts in MoDeST. The translational semantics is developed by means of a recent exemplary case study [16], which uses all intricate concepts of statecharts and StoCharts. This allows us to identify subtle issues. Furthermore, the concrete translation for this particular case, allows us to use Motor and Möbius for a detailed parametric analysis of the case study.

The case study focusses on a safety critical fragment of the European Train Control System (ETCS) standard. This standard aims at ensuring interoperability of European railway systems in the future. Communication among ETCS components (trains, trackside equipment etc.) will be based on mobile communication using GSM-R, an adaptation of the GSM protocol to railway applications. The safe and efficient operation of ETCS is, of course, of prime importance. The specifications of GSM-R and of ETCS contain various QoS requirements such as *"a connection must be established within 5 seconds with 95 % probability"*. Due to the architecture of ETCS, on-board and trackside data processing as well as the radio communication link are crucial factors in ensuring the ETCS requirements. In order to study this issue, we recently developed a StoChart model [16]. Albeit being simple, the StoChart-model enabled us to identify bounds on the distance between consecutive trains on a track, under which crucial QoS requirements of ETCS are still satisfied. To arrive at these results, the StoChart collection was manually translated into a simulation model. This model in turn was fed into the tool ProVer [26] which implements a variation of discrete event simulation. In this paper, we instead translate the StoChart-model into MoDeST, and use the tool combination Motor and Möbius to perform a much more detailed and parametric analysis, which (as a sanity check) is still consistent with the results obtained via ProVer. A particular aspect of this detailed analysis lies in the fact that we exercise a design-by-contract approach in a quantitative setting, allowing interesting insight into the general behaviour of the ETCS system.

In summary the contribution of this paper is threefold. (1) The paper sets the formal grounds for a sound and mechanizable translation from StoCharts to MoDeST, developed by means of the ETCS case. (2) The translation enables a mechanic and thus more detailed analysis of the ETCS case, owed to the power of the Möbius tool interface and the simulation engine. (3) The paper sheds light on the contractual guarantees the system can provide, and their roots in the contractual guarantees given by the GSM-R specification.

Organization of the paper. Section 2 introduces the modeling formalisms that are used in this paper. Subsection 2.1 introduces StoCharts, briefly touching upon semantic issues, and Subsection 2.2 introduces the MoDeST language. Section 3 contains the description and modeling of the ETCS case study in both StoCharts and MoDeST. The analysis is presented in Section 4. Finally, Section 5 discusses lessons learned from this case study and concludes the paper.

2 The modeling languages

2.1 StoCharts

This section gives a brief overview of StoCharts and reviews the underlying semantic model. We refer to [17, 18] for a thorough discussion of the StoChart formalism and a comparison to statechart diagrams.

Abstract syntax. A basic StoChart consists of

- a finite set of *Nodes*[1] with a tree structure, as for statechart diagrams. Nodes are of type 'basic' (leaves of the tree), 'and' , or 'or'. Each *or*-node has a distinguished, *initial* child node.
- a finite set of *Events,* as for statechart diagrams.

  Later on, we will also use *pseudo events.* A pseudo event is an expression of the form $\mathsf{after}(F)$, where $F : [0, \infty) \to [0, 1]$ is a so-called *cumulative distribution function,* i. e., a function to express a stochastic delay: $F(t)$ is the probability that the delay is at most t time units.

  Some simple cumulative distribution functions are denoted as follows: $\mathsf{DET}[t]$ means a deterministic delay of time t; $\mathsf{EXP}[t]$ means an exponentially distributed delay with mean time t.
- a finite set of (typed) *variables* or attributes together with an initial valuation, assigning initial values to the variables.
- a finite set of *P-Edges,* corresponding to the transitions of a statechart diagram. A P-edge consists of a set of source nodes, a triggering event or pseudo event, a guard, which jointly describe when the P-edge can be taken. The reaction is described by a probability distribution over pairs which consist of a set of actions and a set of target nodes.

Drawing a StoChart. A StoChart is drawn almost like a UML statechart diagram. Nodes are drawn as rectangles with rounded corners; the children of a node are drawn inside its boundary. Children of an *and*-node partition the node by dashed lines. The initial node is indicated by an arrow pointing from a small dot to the initial node.

A (nontrivial) P-edge is graphically depicted in two parts: an arrow labeled with an event and a guard directed to a P-pseudonode (drawn as Ⓟ) from which several arrows to target nodes emanate, each labeled with a probability and an action set (similar to a compound edge, where targets are chosen according to a condition).

[1]The UML specification for statechart diagrams [23] actually speaks of *states,* but we prefer to call them otherwise because the system can be in more than one node at the same time.

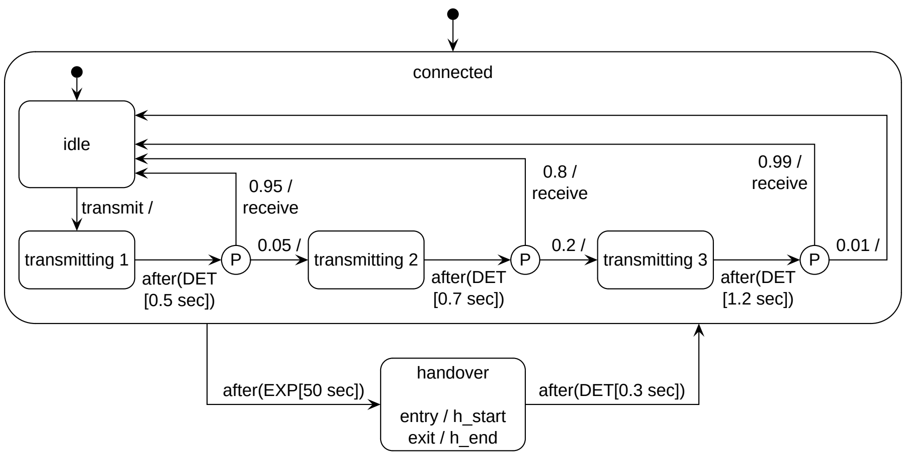

Figure 1: Example StoChart

A trivial P-edge (where probability 1 is assigned to a unique action set and target nodes set) can be drawn like a transition in a statechart diagram: a single arrow labeled with event, guard and actions.

Intuitive semantics. Like a statechart diagram, a StoChart is always in some state which consists of one or several nodes: if an *and*-node is part of the state, *all* of its children are in the state. If an *or*-node is part of the state, exactly *one* of its children is so.

A P-edge is enabled if all its source nodes are part of the current state, its guard holds, and either its event happens or the delay associated with its after operator expires. The system selects as many enabled P-edges as possible for execution (a choice between conflicting edges is often resolved by a priority scheme) and resolves the discrete probabilistic choices. Once the selected edges are taken, their source nodes are left, their actions are executed, and their target nodes are entered. To simplify the analysis, we assume that transitions are instantaneous.

The new state is completed to satisfy the rules stated above about children of *and*- and *or*-nodes; if no edge specifies which child of an *or*-node is to be entered, the initial child is chosen.

On entering a node with an outgoing (P-)edge labeled with an after(F) operation, a sample is taken from distribution F and a timer is set accordingly. The corresponding outgoing edge becomes enabled once the timer expires.

Example StoChart. Figure 1 shows a small example of a StoChart. It is a model for a fragment of the transmission medium in the ETCS system. The transmission medium reacts to transmitting a message (indicated by event transmit) by generating a reception some time later (indicated by action receive). The delay is required to be [12]

- at most 0.5 seconds with 95 % probability;
- at most 1.2 seconds with 99 % probability (i. e. more than 0.5, but $\leq$ 1.2 seconds with 4 % probability);
- at most 2.4 seconds with 99.99 % probability (i. e. more than 1.2, but $\leq$ 2.4 seconds with 0.99 % probability).

The StoChart models the worst case: the time between the event transmit and the action receive is exactly 0.5 seconds with probability 0.95, exactly $0.5 + 0.7 = 1.2$ seconds with probability $0.05 \cdot 0.8 = 0.04$, and exactly $0.5 + 0.7 + 1.2 = 2.4$ seconds with probability $0.05 \cdot 0.2 \cdot 0.99 = 0.0099$; the message is lost with the remaining probability. This behavior is modeled by three after operators, each with a deterministic delay. Figure 2 illustrates the corresponding cumulative distribution function: the solid line indicates the probability that the communication succeeds

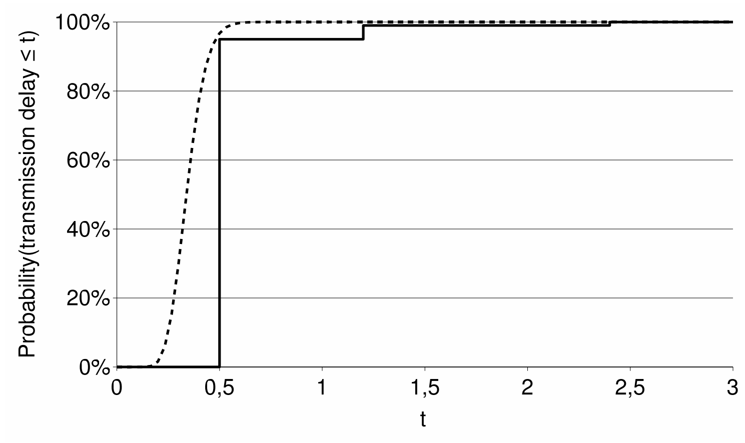

Figure 2: Cumulative distribution function for the transmission delay

within a given time (in seconds), as modelled in the StoChart. In reality, the communication delay may be shorter, for example as indicated by the dotted line in figure 2.

The above may be interrupted by a so-called handover, where the connection between the train and the radio block center is handed over from one radio cell to another. As radio cells may have different sizes, we cannot model this by a deterministic delay. We will argue later that the train moves from one radio cell to another every 50 seconds on average. If only the average of a stochastic distribution is known, the most general stochastic distribution that can be chosen is the exponential one. A cell handover lasts 0.3 seconds. When a handover starts, the current transmission gets lost.

Semantic model. The formal semantics of StoCharts [17] is defined in terms of an extension of labeled transition systems. These transition systems are equipped with *timers* to model stochastic delays, and with a set of *actions* to model system activities. The use of timers in transition systems is similar, though not equivalent to the use of clocks in MoDeST and e. g., timed automata [1, 19]. While clocks run forward at the same pace and are always reset to 0, our timers are initialized by sampling a stochastic distribution and run backwards. On the other hand, our timers are always checked for expiration (i. e., is the timer equal to zero?), while clocks can be checked against complex conditions.

Input and output actions are distinguished to allow for the composition of transition systems, like in I/O-automata [20]. Three types of transition relations are used: input transitions, output transitions, and delay transitions, the latter being enabled once a timer expires. Whereas input and delay transitions are standard ternary relations, the output transition relation is probabilistic. The resulting model is called a *stochastic I/O-automaton* (Iosa, for short).

A Iosa is a specific semantic structure that contains exactly the ingredients needed for stochart semantics [17]. Generalised Semi-Markov Processes (GSMP) are a frequently found model for stochastic processes, and often, a Iosa can be translated to a GSMP. However, GSMPs do not allow nondeterministic choice, and they also restrict the allowed stochastic distributions so as to reduce the probability that two timers expire at the same moment (introducing nondeterminism again) to zero. The Iosa associated with the StoCharts in our case study does not contain any nondeterminism.

2.2 MoDeST

MoDeST is a formal language to describe stochastic timed systems [8], equipped with a rigid formal semantics. The functional core of MoDeST can be considered as a simple process algebra enriched with some convenient language constructs. The syntax resembles that of the programming language C and the modeling language Promela [15]. Data modularization concepts and exception handling mechanisms have been adopted from modern object-oriented programming languages such as Java. Process algebraic constructs have been strongly influenced by FSP (Finite State Processes [21]), a simple, elegant calculus that is aimed at educational purposes.

This core language is enriched with several modeling concepts tailored to model timed and/or stochastic systems. We highlight three particular semantic concepts which are well-established in the context of real-time and stochastic discrete event systems:

- *Probabilistic branching* is a way to include quantitative information about the likelihood of choice alternatives.
- *Clocks* are a means to represent real time and to specify the dynamics of a model in relation to a certain time or time interval, represented by a specific value of a clock.
- *Random variables* are often used to give quantitative information about the likelihood of a certain event to happen after or within a certain time interval.

The MoDeST language allows one to specify *processes*, and to compose them in *parallel* using a '`par`' operator. Processes can manipulate *data variables* by *assignments*. Data variables are typed and must be declared, and the point of declaration determines their *scope*. In particular, they may be *local* to a process, or *global*, in which case they are shared between all processes. A particular type of variable which can be declared is the *clock* type. Clocks can be read like an ordinary float variable, but advance their value linearly to system time. All clocks run at the same speed. Clocks can only be set to zero. The language provides generic constructs to sample values from a set of predefined probability distributions. For instance, '$xd = Uniform(10, 20)$' assigns a sample from the uniform distribution on the interval $[10, 20]$ to the variable 'xd'. Other types of distributions are, *e. g., Exponential(rate)* and *Normal(mean,var)*.

Apart from manipulating data, processes can interact with other parallel processes (or the environment) by means of *actions*. Their occurrence within a process can be guarded by a '`when(.)`' clause, specifying a enabledness condition. In particular, the boolean expression in a '`when(.)`' clause may refer to clock values. In that case, an action may be enabled as soon as the `when(.)` condition becomes true (and no other action becomes enabled earlier). We assume a maximal progress semantics. – Processes in the body of a '`par`' construct perform actions and assignments independently from each other, except that common (non-local) actions need to be executed synchronously, à la CSP [14].

MoDeST provides means to raise *exceptions* inside a *try block* and to handle them. When an exception is raised, process control is handed over to the exception handler contained in a `catch` block. Another, standard way of handing over process control is by a simple process call. Upon termination of the called process, the calling process gains back control, like in an ordinary procedure call.

The '`alt`' construct is used to specify choice between different possible behaviors. In general, this choice is made nondeterministically. A variant thereof is the '`palt`' construct, which provides a weighted *probabilistic choice,* where each *weight* has the form :w:, with w a positive real number. The '`do`' keyword indicates a *repetitive behavior*. Upon termination of the body of this construct, the body is restarted, until a '`break`' is encountered.

As an example of a small MoDeST code, the following fragment describes a process C_1 which waits for five time units prior to randomly selecting between continuing as process C_2 (5 % of the cases) or performing an action c (95 % of the cases).

```
proc C_1 ()
{
clock x=0;
when (x==5)
  tau; palt
    {
    :5: C_2();
    :95: c;
    }
}
```

In this fragment, the action 'tau' stands for an internal step, in particular 'tau' is a local action which is not attainable for synchronisation.

3 The ETCS system and its Modeling

3.1 Informal Description of ETCS system

This section briefly introduces into the high-speed train radio signaling case study we considered [16], inspired by earlier work by Zimmermann and Hommel [27].

European Train Control System. The upcoming European Train Control Systems (ETCS) serves as a unifying standard of many European railways. It is promoted by the European Union to simplify access to and cross-border traffic in between different national rail networks. The main constituent is a uniform communication infrastructure across Europe. This communication infrastructure is based on GSM-R, which is an adaptation of the well-known GSM protocol for wireless communications to railway specific applications.

ETCS levels. ETCS knows multiple *levels* to enable gradual migration from the current systems. In our case study, we will only consider level 3, the highest level defined. On this level, important informations are exchanged betwen trains and trackside coordination units, so called *radio block center* (RBC) via GSM-R-based radio communication. In particular, a train needs to receive so-called *movement authorities* (MAs) from the RBC in order to continuously run at high speed. These MAs grant the train exclusive access to some physical track block, and are sent by the RBC if it is certain that the preceding train has moved ahead in its entirety. To assure this, ETCS level 3 requires that trains are equipped with an onboard devices which check train integrity. The integrity status and the current and position are reported from train to RBC at regular intervals. This enables the RBC to declare the track behind the train clear with virtually no delay, which in turn is a requirement for so-called *moving-block* operation, where the track block granted exclusively to a specific train is not a fixed unit of the track between two signals, but instead moves with the train along the track.

This moving-block operation is expected to reduce the *headway*, i. e., the time between the passage of consecutive trains at some point of the track, well below 3 minutes, which is the usual headway in fixed block operation. The minimal headway is the sum of several delays (assuming trains running at 300 km/h): a delay needed for train integrity check ($<$ 4 seconds), the communication delay itself, and a delay that reflects certain physical distances: *(i)* train length (typical value: 400 m), *(ii)* braking distance (about 2500 m), *(iii)* margin for position measurement errors (5 %). The latter is at most 50 m, if Eurobalises (a device that tells its exact position to a train passing over it) are positioned no more than 1 km apart. For simplicity, we assume that these distances sum up to 3000 m. The train travels this distance in 36 seconds. Thus, with instant communication, 40 seconds would be the ultimate lower bound on the headway between consecutive trains. In the case study, we have a closer look at the reliability of communication needed for moving-block operation. GSM-R may fail to establish a connection, a connection may get degraded or lost; during handover from one GSM radio cell to another messages may get delayed. Under normal circumstances, the train reports the safe position of its head and tail at fixed intervals, for example every 5 seconds. What happens if one or several of these reports get lost? On the other hand, MAs need to be received by the train at similar intervals. What is the probability that the train misses a movement authority?

To address these questions, we study an initial model which is based on the known guarantees provided by GSM-R, i. e., we assume that the GSM-R network functions as specified in the Euroradio specification [12, 27]. Later we will vary some of these assumptions. In particular, we assume:

- The delay to establish a GSM-R connection is at most 5 seconds with 95 % and at most

7.5 seconds with 99.9 % probability. Delays of more than 7.5 seconds are regarded as connection establishment errors.

- The end-to-end delay of a (short) message is at most 0.5 sec with 95 %, at most 1.2 sec with 99 % and at most 2.4 sec with 99.99 % probability (see the illustration in figure 2).
- Handover takes place whenever the train passes from one GSM radio cell to another. As ETCS is intended to work with train speeds up to 500 km/h, we take at first a pessimistic assumption on the time between handovers. The mean distance between handovers is specified to be 7 km; this leads to a mean time between cell handovers of 50 seconds. The communication break during handover lasts at most 0.3 sec.
- From time to time, the train may pass an area where communication is degraded and frequent transmission errors occur. These periods are more than 7 seconds apart with 95 % probability. A degraded period is required to be shorter than 1 second with 95 % probability.
- A connection loss has a probability $\leq 10^{-4}$ per hour. It shall be detected within 1 sec.

With respect to the train-specific behavior, we adhere to the following assumptions as put forward in [13, 24]:

- A passenger train completes an integrity check within 4 seconds; it reports the outcome and its position to the RBC at most once in 5 seconds.
- A typical train trip has a duration of 1 hour.

We view all the above properties as constraints to be met by the environment in which a level 3 train operates. This view can be seen as an application of the *design-by-contract* paradigm [22, 3], in the sense that the ETCS system is required to work properly if these constraints are met (or outbalanced). The question then remains what specific guarantees can be distilled from these assumptions. We intend to check whether it is possible that trains run at 300 km/h with only a small headway, for example 1 minute. In particular, we want to find answers to the following questions:

- The probability p that a message is transmitted successfully has to be at least 99.95 %. This figure is based on the availability requirement of [12]. As parts of the communication delay are distributed stochastically, the success probability depends on the time frame t we allow as maximal communication delay. Recall that the minimal headway with (hypothetic) instant communication is 40 seconds. With a 1 minute headway in mind, the question is: Is the probability $p \geq 99.95\,\%$ for $t = 20$ seconds?
- Even if 20 seconds lead to p being in the range required above (≥ 99.95), it is still not obvious that this also enables multiple trains to run at a headway of 1 minute during a complete trip. We therefore also consider the question: What is the probability that two trains (with a small headway) run for a full hour without ever braking or stopping?

3.2 The StoChart model

In our recent paper [16], we have modeled the ETCS case study using StoCharts. We construct the models used in the experiment from two components: a sender and a receiver. In Section 4, we will see how the sender and receiver are composed to answer the questions above.

Figure 3 shows the sender model for a train. In node Reporting position, a position report is prepared every 5 seconds, indicated by the after edge with a deterministic delay. It is sent as soon as possible, if the sender assumes there is a connection to the receiver. The node Connection status just stores the information the sender has about the connection: it is either disconnected (the initial node, where it tries to establish a connection by sending event try), connected normally or involved in a cell handover.

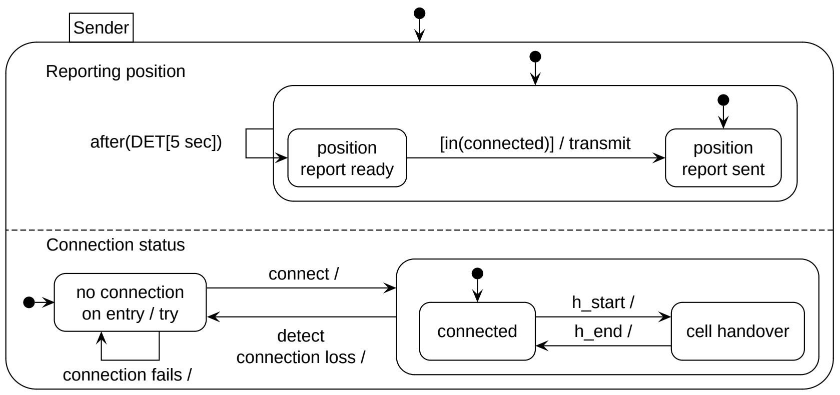

Figure 3: The sender model

There are several causes for delay and stochasticity in the communication protocol between sender and receiver. We have decided to incorporate these delays into the receiver model, while the sender model is reactive, waiting for feedback from the receiver. For example, to model the establishment phase for a radio connection, the receiver model includes after(...) operators and sends a message back to the sender system when the connection is established. Alternatively, we could have split the communication characteristics from the receiver, and let the sender and receiver interact through a transmission medium submodel.

Figure 4 shows the receiver model together with the delays. When the sender tries to establish a connection with the receiver (by sending event try), the connection establishment delay starts. It is guaranteed to be at most 5 seconds with 95 % probability and at most 7.5 seconds with another 4.9 %. We have modeled this guarantee using two deterministic delays, one of length 5 sec (on the edge leaving node connecting 1), the other one of length 2.5. Alternatively, we could have modelled it using a single more complex distribution, similar to the cumulative distribution function in figure 2.

The node correct contains as a subchart the example StoChart of figure 1; this is indicated by the @. This subchart models the communication delay and cell handover.

To this basic model of normal operation, we have added two more possibilities of perturbations: *(i)* Periods of frequent transmission errors may occur (as described above), making it impossible to correct errors in the received bitstream. In our model, this is reflected by node error burst. Both the beginning and the end of the error burst period are modeled by exponential delays, as the errors occur stochastically. The mean times of the relevant delays are chosen as to meet the requirements given above. *(ii)* All other failure types are subsumed under *connection loss,* which is required to happen at most 10^{-4} times per hour. We have modeled this by an exponential delay with an average of 10^4 hours. The sender notices the connection loss with a delay; this is modeled by waiting in node undetected connection loss for 1 second.

3.3 The MoDeST model and translation issues

The basic recipe for translating the StoChart model into MoDeST code is not difficult.

Overall structure. Based on the semantic model of StoCharts (Section 2.1), we model the collection of StoCharts as a parallel composition of processes in MoDeST in a way that each substate of an *and*-state is a separate parallel MoDeST process. So, for example, the sender StoChart is translated to two parallel processes.

```
proc Sender ()
{  :: Reporting_Position ()
   :: Connection_Status ()
}
```

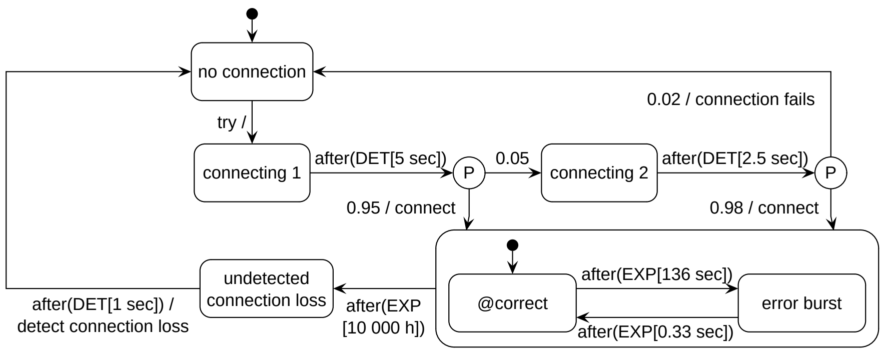

Figure 4: The receiver model, including error models

The top-level view on the resulting code is as follows:

```
par
{  :: Sender ()
   :: Receiver ()
}
```

The communication between the StoChart-nodes on the level of MoDeST processes is performed via shared variables in a straightforward fashion.

Individual processes. Each process in the translation is generated in the following way. Each *basic* node of the StoChart is translated to a state in the automata view on MoDeST. In this way we obtain a local transition system for each *and*-substate of a StoChart. This transition system can be directly encoded into MoDeST-code, because:

- MoDeST provides native support for nondeterministic and probabilistic choice.
- Drawing samples from arbitrary distributions is also supported in MoDeST directly.
- The action synchronization mechanisms of MoDeST and StoCharts are virtually the same, except that the distinction between input and output is not made in MoDeST.
- Time delays are modeled using clocks. For example, the statement after(DET[5 sec]) is modeled as:

```
clock x=0;
when (x==5) ...
```

- Random delays such as after(EXP[5 sec]) are be modeled as:

```
        clock x=0;
        float y=Exponential(1/5);
        when (x==y) ...

```

- Each pair of StoChart input/output actions “a/b” is translated into a sequence of MoDeST actions “a;b”.

In this way, the code example at the end of section 2.2 is the translation of *basic* node “connecting 1” in figure 4 (after renaming “C” into “connecting”).

In case the guard of a transition is a state predicate of another process, we model it using shared memory communication mechanism. For instance, the translation of the predicate “in(connected)” is contained in the following fragment for node “Reporting position”.

```
process Reporting_Position()
{  clock x=0;
   when(x==5) tau {= x=0 =};
   do{::par{::when (connected==1) transmit
           ::when (x==5) tau {= x=0 =}
       }
   }
}
```

In this code fragment, "connected" is a shared variable set to 1 by the parallel process corresponding to the "Connection status" state. If this variable is set to 1, then the train position report can be transmitted (the action "transmit" is enabled). But this transmission only occurs if the parallel process for the receiver is able to communicate synchronously by performing the same "transmit" action. Furthermore, the code ensures that a report can be transmitted every 5 seconds.

As an optimization, we transform tail recursion into "do" loops, which are implemented more efficiently in MoDeST than recursive process calls.

Border-crossing transitions. One of the main challenges to overcome during the translation is posed by border-crossing transitions typical for statecharts. To reflect their effect properly we use the exception mechanism of MoDeST (*cf.* Section 2.2). We combine exceptions with parallelism and obtain a powerful mechanism, somewhat similar to the disrupt mechanism in LOTOS [6]. The abstract mechanism looks as follows:

```
try{
  par{::P()
      ::disrupting_event1; throw exception_event1
  }
}
catch exception_event1{
  <do something>
}
```

In this example the "disrupting_event1" can occur at any state of process P() (because of the interleaving semantics of parallel composition). As the result, the execution of P() is aborted and control is transferred to the "catch" body. In order to model the same behavior without exceptions, one would have to add an alternative composition to every state of process P(), which would lead to error-prone code blowup.

We use this mechanism several times in the translation. For example, the "Connection status" state is translated in the following way:

```
process Connection_status()
{  do{::
       do{::action_try;
            alt{:: connection_fails
                 :: connect {= connected=1 =}; break
            }
       };

       // Connected
       try{
         par{::do{::h_start {= connected=0 =};
                    h_end {= connected=1 =}
               }
             :: detect_connection_loss {= connected=0 =};
                throw retry
         }
       }
       catch (retry){tau}
   }
}
```

The whole process is an endless "do" loop that consists of two parts: establishing a connection, and maintaining it (detecting cell handovers and connection losses). The first part is rather trivial: we try to establish a connection (action "action_try"), and then wait for the result. Once we get a handshake by the "connect" action, we set the shared variable "connected" to 1 and go to the second part; once we get a handshake by "connection_fails" action, we repeat the "action_try".

The second part demonstrates the use of the exception mechanism. We execute the process that checks for cell handovers (actions "h_start" and "h_end", which correspond to cell handover start and end, respectively). At any state of this process "detect_connection_loss" action may occur (because of the interleaving semantics of parallel composition). After that the exception "retry" is thrown immediately.

In order to translate the receiver StoChart we need to use several levels of exceptions, in the form of cascading exceptions. The problem here is that the "normal flow" of the receiver process can be disrupted by three types of events:

- cell handover,
- error burst,
- undetected connection loss.

The priority of the disrupts is in the presented order, i. e. a connection loss can interrupt both the cell handover and the error burst procedures, and an error burst can interrupt a cell handover. We use a cascading exception scheme, similar to the following, to model this:

```
1   try{
2     par{::try{
3             par{::P()
4                 ::disrupting_event1; throw exception_event1
5             }
6           }
7           catch exception_event1{
8             <do something1>
9           }
10        ::disrupting_event2; throw exception_event2
11    }
12  }
13  catch exception_event2{
14    <do something2>
15  }
```

Here we see that the inner try/catch construction (lines 2–9) is identical to the one-level exception handling example presented above.

On exit triggers. Yet another challenge lies in modeling the "on exit" construction of StoCharts, especially in the situation with exceptions. This is due to the fact that, unlike in C++, there are neither destructors, nor automatic destructor invocations in MoDeST. Therefore we propose the following solution. We remember the fact that we have to do an exit transition by setting a flag and check it when we handle the exception. For example:

```
process Transmit_with_Cell_Handover()
{  clock x; float y;
   do{::
        try{ tau {= y=Exponential(1.0/50), x=0 =};
             par{::Transmit()
                 ::when(x==y) throw cell_handover
             }
        }
        catch(cell_handover){
          h_start {= x=0, handover=1 =};
          when (x==0.3) h_end {= handover=0 =}
        }
   }
}

process Transmit_with_Cell_Handover_and_Error_Burst()
{  clock x; float y;
   do{::
        try{ tau {= y=Exponential(1.0/136), x=0 =};
             par{::Transmit_with_Cell_Handover()
                 ::when(x==y) throw error_burst
             }
        }
        catch(error_burst){
          alt{::when (handover==1)
26                  h_end {= handover=0 =}
27               ::when (handover==0) tau
28            };
29            tau {= y=Exponential(1.0/0.33), x=0 =};
30            when (x==y) tau
31         }
32     }
33  }
```

In this example the first process sets the flag "handover" whenever it is in the cell handover state (lines 10–11), and the second process checks this flag, and executes "h_end" if needed (lines 25–26). This check should also be present in all outer exception handling routines.

In this section we outlined the translation procedure based on the ETCS example, which contains all important features of StoCharts. This semantics-by-example still awaits a rigid formal proof. We are confident that this can be achieved, as the numerical analysis in the following section yields consistent results.

4 Analysis

4.1 Tools

Our original case study [16] used the tool ProVer for simulation-based analysis of a hand-crafted model of the ETCS, derived from the StoChart design. ProVer is a tool that uses *discrete event simulation* to estimate probabilities of interesting system behaviours, similar to various other tools for GSMP analysis. ProVer is, however, particular in the manner the behaviour-of-interest is specified by the user. It allows to specify two types of requirements. First, ProVer can estimate the probability with which certain path-based system requirements are satisfied by the model. These requirements are stated as path properties in the stochastic temporal logic CSL [2]. ProVer's output is then a likelihood (and an error estimate) with which the path formula is estimated to be satisfied. Second, ProVer can also be used similar to a model checker to verify whether a CSL state property is met by the model. ProVer's output is then either "yes" or "no" (and a confidence).

The Motor/Möbius tool tandem allows to perform discrete event simulation of MoDeST specifications. During the simulation the system time advances in variable steps, as long as no activity is present. As a result we get an execution trace containing the time-stamped actions the system performs. Different execution traces are different from each other due to the difference in the random values drawn by the tool in accordance to the probability distributions used in the specification. In this way a sufficiently large number of execution traces allows one to quantitatively estimate the conditions under which certain actions occur in the system.

In this case study we estimated the mean time between subsequent occurrences of "receive" actions performed by the system. Therefore, from the execution traces we distilled the time stamps of "receive" actions. Further, we used a python script to analyze this list of time stamps and to calculate the probability distributions for the delays between consecutive "receive" actions.

We considered it as a very useful sanity-check to investigate in how far the two resulting models (both derived via manual transformation steps – though on different levels) fed into different simulation infrastructures (ProVer and Motor/Möbius) lead to identical or diverging simulation results. So, we made ProVer generate a file of runs in the same format as the Motor/Möbius output, and used the same python script to analyze them. The results for the combined use of ProVer and python are also provided below, together with the results optained via MoDeST.

4.2 First Experiment: Reliability of Communication

In a first setting, we investigate the communication reliability. The model we use consists of a single sender (say, a train) and a single receiver (say, a RBC), as shown in figure 5 (in the syntax of a UML deployment diagram: informally speaking, the 3D boxes represent hardware components that run the software drawn as a box with handles inside them. A line between hardware components

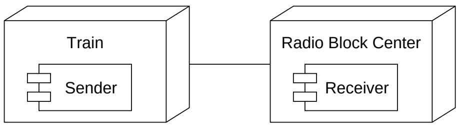

Figure 5: Model for the first experiment

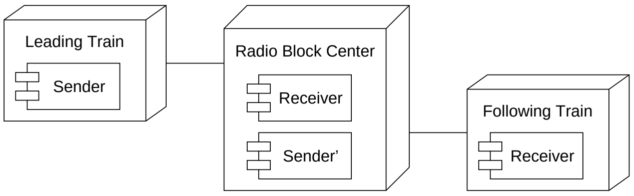

Figure 6: Model for the second experiment

indicates a communication link.). The behaviors of train/sender and RBC/receiver are as shown in the StoCharts. The train generates a position report every 5 seconds and sends it off to the RBC once it assumes the connection is working. We have checked whether the communication is reliable enough, depending on the delay until the reception of a position report. We assume that in the initial state, a position report has just been generated, sent and received without delay (an over-approximation of the best-case behavior for the preceding message).

We used the tool chains to estimate the probability p that the communication succeeds for some values of t. Table 1 gives the estimates for the three combinations of tools we have used. We can see that the three tool combinations produce (approximately) the same results.

t	ProVer	ProVer + python	MoDeST + python
5 sec	0	0	0
10 sec	0.98267±9	0.98271± 6	0.9840
15 sec	0.999700±9	0.999688± 8	0.9997
20 sec	0.9999944±6	0.9999950±10	0.9999

Table 1: Experiment 1 results

We can see that for $t = 15$ seconds, the estimated communication reliability is is large enough. As this is an estimate, the actual value may be just below the desired 99.95 %, but ProVer can help us out here. we have used the tool to check the statement: "The probability that the message is received within 15 seconds is ≥ 0.9995", stated as a formula in the input language of ProVer. ProVer then affirmed that the statement is true with a confidence > 0.9999.

4.3 Second Experiment: Delay Probability

Even if a single communication is reliable enough, it may happen too often that communication will break during a longer trip. To check for this, we have constructed a model consisting of two successive trains running at 300 km/h. The leading train sends a position and integrity report to the RBC, which in turn sends a movement authority to the following train, as illustrated in figure 6. The second train should receive a movement authority within a short time after a position report is generated, to avoid that it needs to brake. Unfortunately, it is difficult to measure (using our tools) the age of the information on which the movement authority received is based. Therefore, we measure the time between two successive receptions of a movement authority. A simple calculation (based on the fact that the communication channels have bounded capacity, and that a newer message overwrites an older one) shows that if two receptions lie away Δt seconds, the latter is based on information not older than $\Delta t + 7.4$ seconds. So, in addition to the minimal headway for instant communication (40 seconds), we allow for another 7.4 seconds headway based on the analysis method. Therefore, we have chosen to analyze the situation where the headway between two trains is 62.4 seconds. What is the probability that the following train has to brake during this trip? So, what is the probability p that the time Δt between two successive movement authorities received by the following train is more than 15 seconds at least once? Further, what is the probability that it has to stop? (At 300 km/h, a train needs about one minute to stop.) So, what is the probability p that $\Delta t > 15 + 60$ seconds at least once? ProVer and MoDeST provide us with the results shown in table 2.

max Δt	ProVer	ProVer + python	MoDeST + python
$>$ 5 sec	1	1	1
$>$ 10 sec	0.9562± 9	0.9551±13	0.9338±16
$>$ 15 sec	0.101± 2	0.1006± 9	0.0784±18
$>$ 20 sec	0.0036± 4	0.0041± 4	0.0023± 3
$>$ 25 sec	0.00034±11	0.00029±11	0.00004± 4
$>$ 75 sec	0	0	0

Table 2: Experiment 2 results: Probability that Δt is $>$ the indicated time at least once during a 1 hour trip

We can see that according to both tools, about one train out of 9 or 10 would have to brake. The simulations did not produce a single run where a train would need to stop, so this probability is too small to be measured.

The numbers obtained via the three evaluation methods are reasonably close to each other. This indicates that the translation of these StoCharts to MoDeST is faithful. Earlier in the process, the ProVer + python analysis also served as a kind of quantitative debugging tool: if significant differences were detected between ProVer's genuine analysis and the python scripts, this was a strong indication of an error in the scripts.

4.4 Experiment Series: Parametric Analysis

The power of our MoDeST translation does not only lie in its mechanizability. The simple fact that Möbius is a well-designed and well-supported modelling and analyis environment pays off. We can use the parametric analysis facilities provided by Möbius to carry out an in-depth study of the vulnerabilities of the ETCS case in a convenient way, using the experiment series support of Möbius. We have varied the second experiment and checked the influence of the following four parameters on the required headway:

Train speed. The trains in the above experiments ran at 300 km/h. ETCS should also be able to handle trains running at 400 or 500 km/h. (Please note that the train speed also influences the minimal headway for instant communication, which was 40 seconds in the original experiment.)

Delay between position reports. The specification requires that a train sends a position report at most once in 5 seconds. We also analyse the situation where the actual frequency is lower (once in 10 or 30 seconds).

Connection loss probability. The specification requires that a connection loss has a probability $\leq 10^{-4}$ per hour. However, some network operators are only willing to guarantee a lower quality, so we also checked the probabilities 10^{-3} and $10^{-3.5}$ per hour.

Cell size. We have assumed that a cell has, on average, 7 km diameter. Other sources propose other sizes, so we checked cell sizes 2 km, 6 km, and 10 km.

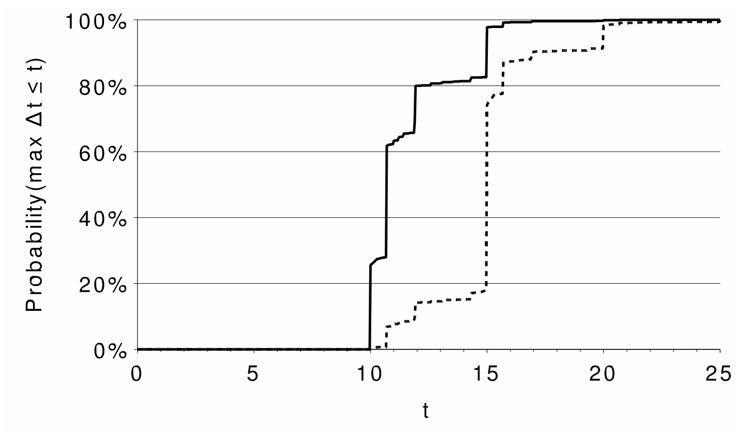

Figure 7: Comparison of two cumulative distribution functions for $\max\Delta t$

There are in total 81 possible combinations of parameters. MÖBIUS could be instructed to generate sets of runs (for PYTHON) with one simple command. We have compared the outcomes of the 81 experiments and conclude the following:

- The *connection loss probability* has almost no influence. In a few experiments, we find differences of up to 0.5 percentage points.

- Increasing the *delay between position reports* leads, more or less, to a proportional increase of $\max\Delta t$.

- The *cell size* and *train speed* are highly related. The reason is that they both influence the expected delay between cell handovers in a similar way. Higher speeds and smaller cells lead to more frequent cell handovers, which may lead to more missed communications.

  For example, assuming a connection loss probability of 10^{-3} per hour and a delay between position reports of 5 seconds, we get the following values for the probability that $\max\Delta t \leq 15$ sec:

	Train speed		
Cell size	300 km/h	400 km/h	500 km/h
2 km	0.742	0.581	0.413
6 km	0.956	0.927	0.901
10 km	0.977	0.963	0.956

The PYTHON analysis of an experiment can also be illustrated by a cumulative distribution function that shows the probability that $\max\Delta t$ is at most a given value. In figure 7, the solid line is the cumulative distribution function for the experiment with cell size 10 km, and the dotted line is the cdf for the experiment with cell size 2 km. In both experiments, the delay between position reports was 5 sec, the connection loss probability was 10^{-3}/h, and the train speed was 300 km/h. The cdfs are discontinuous because the GSM-R transmission delay is modeled by a discontinuous cdf (see figure 2). It reflects that we have modelled the the worst-case assumptions provided according to the GSM-R specification, and the discontinuities in the assumptions propagate to the guarantees provided by the ETCS system. If the service actually provided by GSM-R is better (e. g. as indicated by the dotted line in figure 2), then the resulting service of the system will be better, leading to some plot (to the left and) above the one shown in figure 7. Owed to our design-by-contract apppoach, the resulting performance is guaranteed to be bounded by the plot, as long as the input distribution satisfies the contractual assumptions.

5 Conclusion

This paper has elaborated on our efforts to define a translational semantics for StoCharts. The semantics maps a collection of StoCharts to the MoDeST language. It enables mechanization of the stochastic analysis of StoCharts, because MoDeST is connected to the discrete event simulation engine of the Möbius toolset. Our efforts have focused on the translation of a recent StoChart case study, modeling GSM-based communication in future European high speed trains. The simulation results produced by the tools ProVer and Möbius only show insignificant differences, which can be seen as a sanity-check for our work. We have used the parametric analysis features of Möbius to dive deeper into the different aspects of the ETCS case study. In particular, our design-by-contract approach allows us to distill bounds on the reliability of the ETCS case, instead of performing a case-by-case analysis.

The translational semantics reflects the hierarchical structure of a StoChart and covers all interesting aspects of StoCharts, including a few challenging issues. In particular, we have identified that the exception handling mechanism particular to MoDeST can be used to effectively model *border-crossing* transitions. Based on the insight gained in this paper, we will strive for a complete compositional translation from StoChart to MoDeST.

Acknowledgements. We are grateful to Gerald Luettgen (University of York), who pointed out the resemblance of exception handling and border-crossing statechart transitions.

References

[1] R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[2] Christel Baier, Boudewijn R. Haverkort, Holger Hermanns, and Joost-Pieter Katoen. Model-checking algorithms for continuous-time Markov chains. *IEEE Transactions on Software Engineering*, 29(6):524–541, 2003.

[3] A. Beugnard, J.-M. Jézéquel, and N. Plouzeau. Making components contract aware. *Computer*, 32(7):38–45, 1999.

[4] H. C. Bohnenkamp, H. Hermanns, R. Klaren, A. Mader, and Y. S. Usenko. Synthesis and stochastic assessment of schedules for lacquer production. In *First international conference on the quantitative evaluation of systems : QEST 2004 ; proceedings*, pages 28–37, Los Alamitos, CA, 2004. IEEE Computer Society.

[5] Henrik Bohnenkamp, Holger Hermanns, Joost-Pieter Katoen, and Ric Klaren. The Modest modeling tool and its implementation. In Peter Kemper and William H. Sanders, editors, *Computer performance evaluation : modelling techniques and tools ; ... TOOLS*, volume 2794 of *LNCS*, pages 116–133, Berlin, 2003. Springer.

[6] Tommaso Bolognesi and Ed Brinksma. Introduction to the ISO specification language LOTOS. *Computer networks and ISDN systems*, 14(1):25–59, 1987.

[7] Pedro R. D'Argenio. *Algebras and automata for timed and stochastic systems.* PhD thesis, Universiteit Twente, Enschede, November 1999. ISSN 1381–3617.

[8] Pedro R. D'Argenio, Holger Hermanns, Joost-Pieter Katoen, and Ric Klaren. MoDeST : a modelling and description language for stochastic timed systems. In Luca de Alfaro and Stephen Gilmore, editors, *Process Algebra and Probabilistic Methods. Performance Modelling and Verification: Joint International Workshop, PAPM-PROBMIV*, volume 2165 of *LNCS*, pages 87–104, Berlin, 2001. Springer.

[9] D. D. Deavours and W. H. Sanders. Möbius : Framework and atomic models. In *Proc. 9th Int. Workshop on Petri Nets and Performance Models (PNPM '01)*, pages 251–260. IEEE, 2001.

[10] Frank Dehne, Henk van de Zandschulp, and Roel Wieringa. Toolkit for conceptual modeling (TCM). `http://www.cs.utwente.nl/~tcm/`.

[11] Rik Eshuis and Roel Wieringa. Requirements-level semantics for UML statecharts. In Scott F. Smith and Carolyn L. Talcott, editors, *Formal Methods for Open Object-Based Distributed Systems IV : ... FMOODS*, pages 121–140, Boston, 2000. Kluwer Academic Publishers.

[12] Euroradio FFFIS : class 1 requirements. `http://www.aeif.org/db/docs/ccm/SUBSET-052_v200.PDF`, 2000.

[13] Functional requirement specifications : train integrity monitoring system. `http://www.aeif.org/db/docs/ccm/EEIG-TIMS-Document.doc`, 2000.

[14] C. A. R. Hoare. *Communicating sequential processes.* Prentice-Hall international series in computer science. Prentice-Hall, Englewood Cliffs, NJ, 1985.

[15] Gerald J. Holzmann. *The SPIN model checker : primer and reference manual.* Addison-Wesley, Boston, 2004.

[16] David N. Jansen and Holger Hermanns. Dependability checking with stocharts: Is train radio reliable enough for trains? In *First international conference on the quantitative evaluation of systems : QEST 2004 ; proceedings*, pages 250–259, Los Alamitos, CA, 2004. IEEE Computer Society.

[17] David N. Jansen, Holger Hermanns, and Joost-Pieter Katoen. A QoS-oriented extension of UML statecharts. In Perdita Stevens, Jon Whittle, and Grady Booch, editors, *«UML» 2003 : the unified modeling language*, volume 2863 of *LNCS*, pages 76–91. Springer, 2003.

[18] David Nicolaas Jansen. *Extensions of statecharts with probability, time, and stochastic timing.* PhD thesis, Universiteit Twente, Inmarks, Bern, October 2003. ISBN 3–9522850–0–5.

[19] N. Lynch and F. Vaandrager. Forward and backward simulations : II. timing-based systems. *Information and Computation*, 128:1–25, 1996.

[20] Nancy A. Lynch and Mark R. Tuttle. An introduction to input/output automata. *CWI quarterly*, pages 219–246, 1989.

[21] Jeff Magee and Jeff Kramer. *Concurrency : State Models and Java Programs.* Wiley, Chichester, 1999.

[22] Bertrand Meyer. Applying "design by contract". *Computer*, 25(10):40–51, October 1992.

[23] *OMG unified modeling language specification.* Object Management Group, Needham, MA, version 1.5 edition, March 2003.

[24] Performance requirements for interoperability. `http://www.aeif.org/db/docs/ccm/SUBSET-041_v200.PDF`, March 2000.

[25] Roberto Segala and Nancy Lynch. Probabilistic simulations for probabilistic processes. *Nordic Journal of Computing*, 2(2):250–273, 1995.

[26] Håkan L. S. Younes and Reid G. Simmons. Probabilistic verification of discrete event systems using acceptance sampling. In Ed Brinksma and Kim Guldstrand Larsen, editors, *Computer aided verification : 14th intl. conference, CAV*, volume 2404 of *LNCS*, pages 223–235, Berlin, 2002. Springer.

[27] Armin Zimmermann and Günter Hommel. A train control system case study in model-based real time system design. In *International parallel and distributed processing symposium*, page 118b. IEEE, 2003.