MASARYKOVA UNIVERZITA
FAKULTA INFORMATIKY

Probabilistic Extension of Message Sequence Chart

BACHELOR THESIS

Martin Křivánek

Brno, 2009

Declaration

Hereby I declare, that this paper is my original authorial work, which I have worked out by my own. All sources, references and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

In Brno, May 24, 2009
Martin Křivánek

Advisor: RNDr. Vojtěch Řehák, Ph.D.

Acknowledgement

I would like to thank my advisor Vojtěch Řehák for his patience, countless consultations and guidance throughout the whole process, my family for their support, Ľuboš Korenčiak and Martin Chmelík for valuable comments and finally Jan Krčál for the proofreading.

Abstract

Message Sequence Chart (MSC) is an appealing textual and graphical formalism for describing communicating systems. MSCs can be composed into High-level Message Sequence Charts (hMSC) with greater expressiveness. Recently, time and probability properties have been introduced into MSC. We provide a survey of the current work related to MSC extended by probability. We also suggest how to deal with stochastic information in MSC and hMSC and how to compute performance properties of the system.

Keywords

Message Sequence Chart, High-level Message Sequence Chart, time, probability, stochastic processes

Contents

Chapter 1

Introduction

In a development of a telecommunication system it is important to specify formally a desired behaviour of the system in the earliest phases of the development process. This helps to find possible flaws in the system structure or behaviour early and save the company a lot of resources.

Message Sequence Chart (MSC) is a description language widely used in industry for a requirement and design specification. Advantage of MSCs is that they have well defined both syntax and semantics. MSC specifies the system behaviour from a global perspective. Each MSC portrays one scenario of a possible behaviour of the system, by displaying message exchanges, local actions and other events ordered for each process on a single time line.

These scenarios may be composed together to a so called High-level Message Sequence Chart (hMSC), which is able to capture a more complex behaviour of the system such as branching or iteration of scenarios.

The increasing popularity of MSCs caused a standardization effort that resulted in the ITU-T Recommendation Z.120 [7].

The advantage of MSC model is that it allows designers to use intuitively looking visual description of the system, while – because of the formal background – it is possible to formally check properties of the system against possible errors and also help designer with further modeling.

To be able to specify real-time systems, MSC was recently extended with time properties, such as timers and interval delays. Also to check these time properties, time constraints were introduced.

But for even more realistic specification of a behaviour of a real-world system it is useful to extend MSC also with probabilistic properties. These probabilities may arise in a two different places. Every individual message or action may be enhanced with a stochastic time information (saying that this message or action durates for a time specified by a continuous random variable). The other place for probabilities is in the branching of scenarios within an hMSC. These must sum to 1 and specify the probability that the system execution will continue according to a particular scenario. These probabilities allow us to make a performance analysis of a system.

This area is quite recent and there has not been done much work yet. Aim of this thesis is to overview currently existing approaches, compare them and also suggest our own solutions. This is motivated by a work on a computer software SCStudio [12], which is developed in Faculty of Informatics.

Structure of this thesis is as follows: In Chapter 2 needed terms are defined and explained. Probability is introduces in Chapter 3. Chapter 4 contains an overview of currently existing approaches and their comparison. Suggestions how to solve the problem in practice are located in Chapter 5. And finally, the work is concluded in Chapter 6.

Chapter 2

Preliminaries

In this chapter we introduce and formally define needed terms, which are used in the rest of the thesis.

2.1 Basic Message Sequence Chart

Message Sequence Charts (MSC) were defined in the International Telecommunications Union (ITU-T) Recommendation Z.120 [7]. MSCs provide a standardized description technique for telecommunications system design. They are used at the early stages of the development to document the expected system behaviour.

Each MSC describes a scenario where asynchronous processes communicate by sending messages to other processes. Such a scenario contains description of the sent and received messages, local events, timers, and the ordering among them. Example of a visual description of a simple MSC with three processes and six messages is shown in Figure 2.1.

Each process in an MSC is represented in a visual description by a vertical line with a process name at the top. A message is represented by an arrow from the sending process to the receiving one. Every send or receive is an event. Events are ordered on the process lines from top to bottom according to the order of their execution. We can see from the picture that for example the receive of the message $m1$ should happen before the send of the message $m2$. This order is called the visual order.

The rectangular box $c1$ is a so called coregion. Events in a coregion may happen in any order. However, we may specify the order of some particular two events by drawing a dashed arrow (called connection) inside the coregion from one to another. In our figure, connection specifies that the receive of the message $m5$ must happen after the receive of the message $m4$ (for example because we may be using a FIFO communication channel – channel in which messages from one sender to another are received in the same order as they were sent).

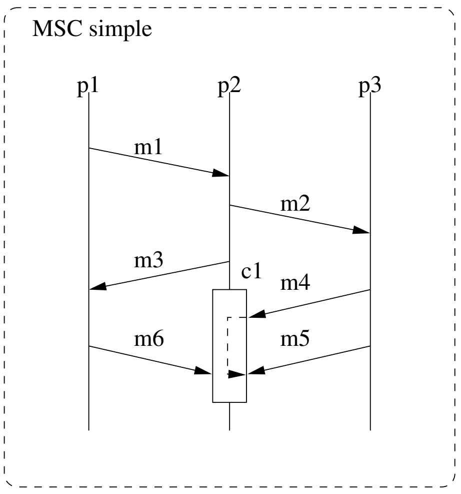

Figure 2.1: example of an MSC

Now when we have some basic intuition what an MSC is, we can define it formally. The definition we use is based on the one from [6].

Definition 1. *An* MSC *(with coregions and connections) is defined as a tuple* $(E, <, \mathcal{P}, \tau, P, \mathcal{M}, \mathcal{C}, \mathcal{G})$*, where*

- E *is a set of* events*;*
- $<$ *is a partial ordering on* E *called* visual order*;*
- $\mathcal{T}$ *is a set of timer labels*
- $\mathcal{P}$ *is a finite set of* processes*;*
- $\tau : E \to \{s, r\}$ *is a labeling function dividing events into* send *and* receive *events*
- $P : E \to \mathcal{P}$ *is a mapping that associates each event with a process;*
- $\mathcal{M} \subseteq (\tau^{-1}(s) \times \tau^{-1}(r))$ *is a bijective mapping, relating every send with a unique receive, such that for any* $e, f \in \mathcal{M}$ *we have* $P(e) \neq P(f)$ *– a process cannot send messages to itself;*
- $\mathcal{C}$ *is a set of pairwise disjoint* coregions *where a coregion* $C \in \mathcal{C}$ *is a subset of events on some process* $p \in \mathcal{P}$*, i.e.* $C \subseteq P^{-1}(p)$*;*

- $\mathcal{G} \subseteq \bigcup_{C \in \mathcal{C}} C \times C$ *is a partial ordering on events within coregions and is called* connections.

The Visual order $<$ *is defined as the reflexive and transitive closure of*

$$\mathcal{M} \cup \mathcal{G} \cup \bigcup_{p \in \mathcal{P}} <_p \ \textit{where} \ <_p = (\prec_p \setminus \{(e_x, e_y) \mid e_x, e_y \in C \ \wedge \ C \in \mathcal{C}\})$$

where $\prec_p$ is a total order on $P^{-1}(p)$ such that each coregion associated with p is a set of consecutive events wrt $\prec_p$. In other words, $<_p$ is a total ordering of events outside of the coregions and of the coregions as whole units; each two events within the coregions are pairwise unordered except of those connected by a connection.

2.2 High-level Message Sequence Charts

Message Sequence Charts are useful for describing simple behaviour. When we want to model a more complex system, where for example some choices can be made, it would be nice to be able to connect different scenarios together. This is what High-level Message Sequence Charts (hMSC) are used for.

There are three ways how to combine MSCs together – *vertical composition*, where two MSCs are combined sequentially, *alternative composition* in which the system can alternatively choose one of the MSCs to follow and *iterative composition* which composes an MSC sequentially with itself.

Intuitively we can imagine an hMSC as an oriented graph which contains in each node another hMSC or MSC. One of the nodes is initial where the system starts and one of the nodes is terminal where the system finishes.

In Figure 2.2 there is an example of hMSC modeling a very simple ATM. Initial node is depicted by a triangle oriented down, terminal node is depicted by a triangle oriented up. After the system starts either login fails and then we try to log again or login is successful and we can withdraw money or see deposit on the account.

Definition we use is the same as in [1]

Definition 2. *An* hMSC *is defined as a tuple $H = (N, B, v^I, v^T, \mu, E)$*

- *N is a finite set of nodes.*
- *B is a finite set of supernodes.*
- *$v^I \in N \cup B$ is an initial node or supernode.*
- *$v^T \in N \cup B$ is a terminal node or supernode.*

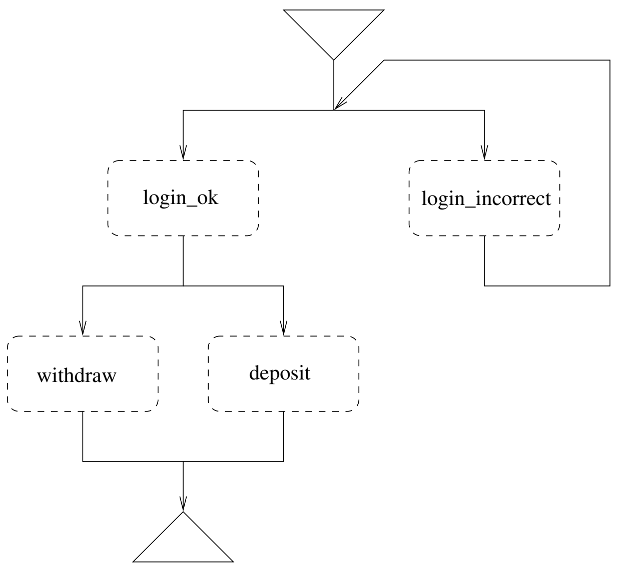

Figure 2.2: High-level Message Sequence Chart – ATM

- *μ is a labeling function $\mu : N \cup B \rightarrow \mathcal{MSC} \cup h\mathcal{MSC}$, labeling each node $n \in N$ by an MSC and each supernode $m \in B$ by an hMSC.*
- *$E \subseteq (N \cup B) \times (N \cup B)$ is an edge relation on the nodes and supernodes.*

Because it is technically more complicated to handle an hMSC which can contain another hMSCs in nodes, we "unfold" the hMSC, such that in every node there is just a basic MSC. This is called Message Sequence Graph (MSG) and it is semantically equivalent to hMSC [1].

Definition 3. *An* MSG *is defined as a tuple $G = (\mathcal{S}, \tau, s_0, s_f, c)$ where*

- *$\mathcal{S}$ is a finite set of states.*
- *$\tau \subseteq \mathcal{S} \times \mathcal{S}$ is an edge relation.*
- *$s_0 \in \mathcal{S}$ is an initial state.*
- *$s_f \in \mathcal{S}$ is a terminal state.*
- *$c : \mathcal{S} \rightarrow \mathcal{MSC}$ is a labelling function;*

Since now, we uderstand hMSC for simplicity as this MSG.

2.3 Time in MSC and Constraints

For a more realistic description of a system, time properties are introduced into MSCs [7].

We may define an interval describing how much time elapses between two events. These two events have to be in a visual order.

Each process may have associated several timers with associated time. There are four special events involving timers – *startTimer*, *restartTimer*, *stopTimer* and *timeout*. Their names are self-explanatory. Timers are local to one process, but their events may be distributed along more MSCs in an hMSC.

A similar meaning as timers have constraints – constraints link two visually ordered events and specify how much time do we want to elapse between them. A difference between the interval and the constraint is that the interval specifies how much time *really* elapses between two events, while the constraint specifies the limits in which we *want* the time between two events to be.

We also allow constraints between two MSCs in an hMSC, but it must be from the beginning of one MSC to the end of another (and there must be a *path* between them) or from the end of one MSC to the beginning of the other (and there must be an *edge* between them – it means this interval specifies a delay between them).

Also constraint has to be *proper* in some sense – because now we may specify some constraints that do not give a sense – for example starting outside a cycle and ending inside the cycle in an hMSC.

For formal definition of constraints and proper constraints please see [9]. Because of the similarity between timers and constraints we will use them interchangeably.

Chapter 3

Probability Extension of Message Sequence Chart

MSCs allow us to model behaviour of a system. But even if we extend them with a notion of a time and with constraints, it is not sufficient to capture all desired properties. Therefore it is needed for more realistic modeling of communicating protocols and posterior checking of correctness of them to extend the MSC with probabilities.

These probabilities can arise in two different places. One of them is duration of passing of one particular message. In reality duration of passing the messages is not distributed uniformly within a specified interval (as in the case of timed MSCs). It can be distributed with any continuous time probabilistic distribution. Sometime we do not know the distribution of one particular message, but instead we know the joint distribution of some events in a row. This is also allowed, but these actions have to be "non-interrupted" – it means no other action interrupts execution of these actions. We will denote this in pictures by writing a specification of a probabilistic distribution near an arrow (or a group of arrows) depicting a message.

The second use of probabilities is in a branching in an hMSC. Whenever a system can choose from more MSCs to continue, we would like to specify probability distribution over all possibilities. Every outgoing edge will then receive a real number from the interval $\langle 0, 1\rangle$ and sum of these numbers over all outgoing edges from one particular node sums to 1.

After adding these probabilities into the model, there are three natural tasks we would like to help the developer with:

1. Checking

   All probabilities, probability distributions and constraints within the system are specified and we would like to know whether the whole system, if running for a specified time t, will finish with probability at least p.

2. Synthesis of coefficients

Some probabilities, probability distributions or constraints are not specified and we would like to know how to set them to achieve wanted behaviour of the whole system.

3. Help with further modeling

   The system is again just partially described and we would like to know whether there exist some relations between some particular timers or probabilities and on which of them behaviour of the system depends the most.

Example of a problem of type 2 (synthesis of coefficients) is depicted in Figure 3.1.

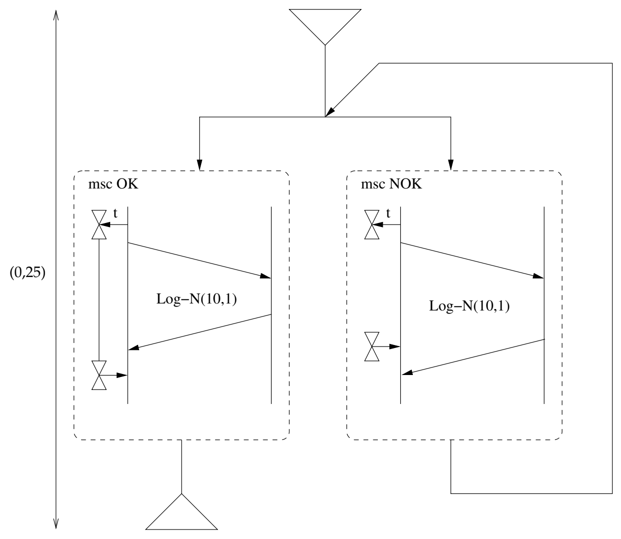

Figure 3.1: example of a small hMSC with probabilities, timers and constraints

This hMSC models a very simple system. There are just two processes. These processes exchange two messages. The message passing time needed for delivering both messages is specified with a log-normal probability distribution with median of 10 and standard deviation of 1. There is a timer specifying, that the messages have to be delivered before a time t (scenario OK). If it is not delivered before time t (scenario NOK), we send the messages again. The whole system has to finish in a time interval $(0, 25)$.

The question is how to set the time limit t such that the probability that the whole system will finish during the interval $(0, 25)$ is for example 60 %.

Chapter 4

State of the Art

There already exist three different approaches how to use probabilities in MSCs. In this chapter we shortly describe each of them and compare them with our requirements.

4.1 Reliability Prediction

Genaína Rodrigues, David Rosenblum, and Sebastian Uchitel in their paper [11] described a method how to use scenarios to predict reliability of the system based on component reliability estimates.

Their method is composed of four steps:

1. annotating a scenario specification with probabilistic properties
2. transforming the annotated scenario to the probabilistic labeled transition system (LTS)
3. constructing a stochastic matrix for LTS
4. computing the system reliability

Assumptions that have to be fulfilled for this method to apply:

- The transfer of control between MSCs has the Markov property, meaning that the transition from one scenario to another is based only on the current scenario and independent from the past history.
- Failures are independent across transitions.
- There is only one initial and one final node in the hMSC.
- There are no implied scenarios. (Implied scenario is such behaviour of the system that each process is, from its local perspective, behaving correctly, yet from a system perspective the behaviour is incorrect. Usually this happens because every process behaves according

to some valid sequence of MSCs, but not all of them follow the same sequence – it means some processes think they are in a different part of a scenario. For more details about implied scenarios see for example [2].)

Annotating Scenarios

Scenarios (both MSC and hMSC) are annotated with two kinds of probabilities, *the probability of transitions between scenarios* PTS_{ij} and *the reliability of the processes denoted by* R_p.

The transition probability PTS_{ij} is the probability that execution transfers directly from the scenario S_i to the scenario S_j. Thus, for the scenario S_i sum of all PTS_{ij} for all successors S_j is equal to one.

The reliability of processes is the probability that the process does not fail when receives a message. This reliability for a process can be a single value (process has the same reliability all the time) or it can change depending on an incoming message or scenario which is executed at the time.

Transforming to the Probabilistic LTS

This step is based on the synthesis approach of Uchitel et al. [13]. Each step is extended with adding probability weights according to probabilities in an annotated MSC.

1. For each process P_i and each MSC S_j, a labeled transition system (LTS) $P_i\_S_j$ is constructed by projecting the local behaviour of P_i in the scenario S_j. In particular, each message with an action a that P_i sends or receives in S_j is synthesized as a transition with action a in $P_i\_S_j$, and the sequence of transitions in $P_i\_S_j$ corresponds with the sequence of messages sent or received by P_i in S_j.

   Also we add for each transition in $P_i\_S_j$ a transition from the same source going into the global $ERROR$ state. This models a chance that the process P_i fails during execution of the transition. The original transition has a probability R_p, the added one $1 - R_p$.

2. For each process P_i, the set of LTSs constructed for P_i in step 1 are composed into a process LTS for P_i according to the structure of the hMSC, with hidden transitions (τ actions) linking the final state of $P_i\_S_j$ to the start state of $P_i\_S_{j'}$ whenever there is a transition from S_j to $S_{j'}$ in the hMSC. The resulting LTS includes a new start state corresponding to the start state of the hMSC.

Scenario transition probabilities $PTS_{jj'}$ are mapped to according τ transitions linking the final state of $P_i\_S_j$ with the start state of $P_i\_S_{j'}$.

3. Each component LTS constructed in the second step is reduced to a trace-equivalent deterministic minimal LTS. This is consistent with the delayed choice semantics of the ITU MSC standard [7].

   Added probabilities have to be handled correctly. Probability weights of the eliminated τ transitions have to be "pushed" to the newly accumulated outgoing transitions, with the new weight on each such outgoing transition equal to its old weight multiplied by the weight of the eliminated τ transition. It can be shown that self loops of τ transitions can be eliminated without this pushing.

   After elimination of τ transitions sum of the outgoing transitions from the resulting new state may not be equal to one. In that case weights have to be normalized.

4. The architecture model for the system is taken as the parallel composition of the minimized process LTSs constructed in step 3.

   The probability weights are computed according to the notion of *generative parallel composition* defined in [5].

Constructing a Stochastic Matrix for Probabilistic LTS

Architecture model created in a previous step is interpreted as a Markov model. The transition probability weights are mapped into a square transition matrix M', which rows sum to one. We relabel states of LTS such that the terminal state of a correct execution is in the first column, $ERROR$ state in the second, and state from which we move to the terminal state is in the last column.

Computing System Reliability

To compute the overall system reliability the Cheung model [3] is used.

Let M be a matrix obtained from the matrix M' by deleting rows and scolumns according to the terminal and $ERROR$ state and n be a number of rows and columns in a matrix M. Cheung shows that the system reliability can be computed as

$$Rel = (-1)^{n+1} \frac{|M|}{|I - M|} R_n$$

where I is an identity matrix and $|M|$ denotes a determinant of matrix M.

This concludes their method.

Sensitivity Analysis

In their following paper [10] Uchitel et al. shows how to find how much a reliability of the system depends on a particular process reliability or probability of transition between scenarios.

In case of depending on a particular process p reliability, we keep the probabilities of transition between scenarios unchanged, reliability of all other processes fix to 1 and compute reliability of the whole system with changing values of reliability of process p.

In case of depending on a particular probability of transition between scenarios i and j, we keep the reliability of all processes unchanged, all $PTS_{k,l}$ where $k \neq i$ unchanged as well. We change just the probability $PTS_{i,j}$ and recompute $PTS_{i,l}, l \neq j$ proportionally such that they sum to 1.

Summary

This approach is not very useful for time related properties of MSC, only possible use could be that reliability of each message would be replaced with a probability that the message associated with some stochastic time arrives in a specified time by a constraint. But this constraint can contain just one message.

On the other hand, construction of an LTS and the use of probabilities of transition from one scenario to another in hMSC are interesting. Their approach used for a sensitivity analysis may be useful for the task 3 – help with further modeling. We may fix all constraints and probabilistic distribution but one we are interested in and see what happens, if we change values of this one particular constraint or parameters of the distribution.

4.2 Stochastic Petri Nets

We can also transform MSCs with time information into the stochastic Petri nets. How to do this was shown by Olaf Kluge in [8]. We first recall what a Petri net is, then we show transformation from an MSC without time information into a Petri net and finally how to transform also time information in an MSC into a stochastic Petri net.

Petri Nets

A Petri net is defined as a triple (P, T, F), where P is a nonempty finite set of *places*, T is a nonempty final set of *transitions* and F is a transitioning function

$$F : (P \times T) \cup (T \times P) \to \mathbb{N}_0$$

describing how many oriented edges are between a place and a transition and vice versa.

We will not formally define semantics of Petri net, we just intuitively describe its behaviour. In each place there is a number of tokens. There also can be no tokens in some places. Transition is *enabled* when there are enough tokens on all its input places. Enough tokens means at least the same number as number of edges from this place to the transition. In each execution step one of the enabled transitions is randomly chosen and this transition "consumes" one token for each incoming edge and "produces" one token for each outgoing edge.

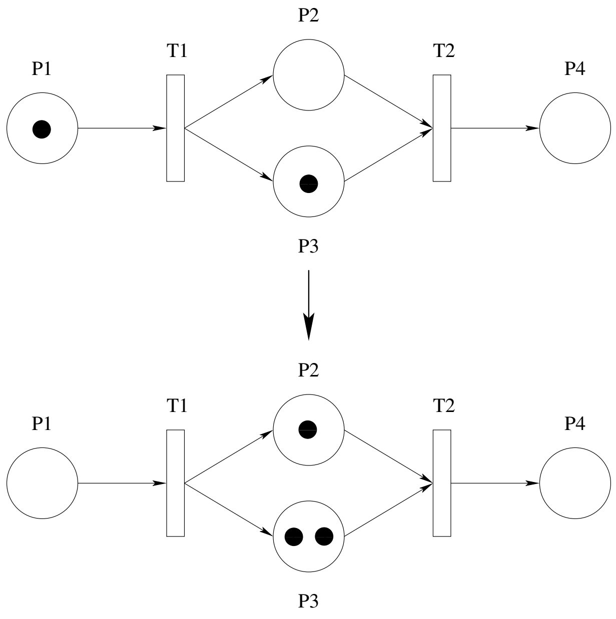

Figure 4.1: Petri net with one execution step

In Figure 4.1 there is an example of a simple Petri net with four places denoted by circles and two transitions denoted by rectangles. In an initial configuration there is one token in the place $P1$ and one in the place $P3$.

The transition $T1$ is enabled as opposed to the transition $T2$ which is not because there is no token in the place $P2$. So the only transition that can fire is $T1$ and it consumes one token from $P1$ and produces one token to each of the places $P2$ and $P3$.

Transformation from an MSC into a Petri Net

First we construct some Petri net fragments according to each event and then we compose them together.

Each MSC event is mapped to one transition in a Petri net (see Figure 4.2). The simplest situation is with termination of the process. It is mapped to one place and one transition. Local events (events that do not communicate with other processes) are mapped into two places and one transition. Non-local events – sending and receiving of messages – are both mapped into three places and one transition, just orientation of one edge differs.

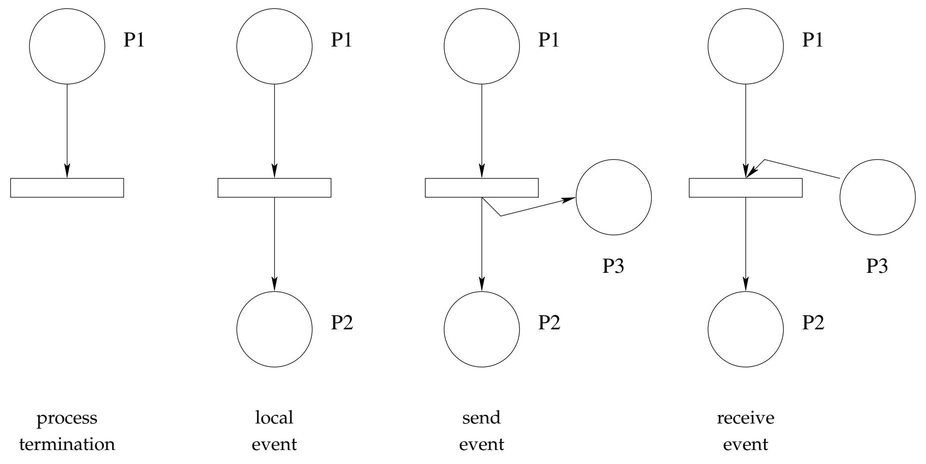

Figure 4.2: Transformation of MSC elements

For technical reasons we need two mappings p and l: $P \cup T \rightarrow String$ to be able to construct a composition of Petri net fragments. In the mapping p we remember the process this place or transition belongs to and the mapping l associates transitions with events and some places with some needed information (we do not want to go into technicalities).

There are two types of compositions to be performed – sequential and parallel. First we sequentially compose all events on one process, then parallely compose all processes.

The sequential composition of two parts of MSC (or even MSC itself) is natural – for every process the end place of the first MSC (or part) is merged with the start place of the second MSC (or part).

In the parallel composition the control flow is duplicated so that both parts can run independently. Also matching places created for send and receive events have to be merged.

Adding Time Information and Construction of the DSPN

Petri nets extended with stochastic information are called Deterministic and Stochastic Petri Nets (DSPN). In comparison with normal Petri net DSPN allow transitions not to be just instantaneous, but also to durate for a deterministic or exponentially distributed time.

DSPN is a tuple $(P, T, F, type, \Lambda, p, l)$, where P, T, F are the same as in normal Petri net, p and l are mappings introduced for sake of constructing a Petri net from an MSC, $type$ is a function: $T \rightarrow \{$**immediate**, **deterministic**, **exponential**$\}$ and finally Λ is a function defining the transition rates for exponential transitions or the delay (exact number or an interval) for the deterministic transition.

Whenever there is an event in an MSC which takes some time, we "cut" according transition into two and a timed transition is placed between.

You may see an example of a transformation of a simple MSC into a DSPN in Figure 4.3. Deterministic transitions are denoted by a black wider rectangle, exponential transitions by a wider rectangle.

However, this method is not able to transform all hMSC into a DSPN, because currently it is an open question whether also alternative composition of two MSCs can be modeled.

Summary

This approach is able to transform an MSC with a stochastic information into a DSPN. On this DSPN we may use some already existing tools as a TimeNET [15].

The main disadvantage of Petri net approach is that it is currently not able to transform all hMSCs because there is no way how to transform alternative composition of two hMSC, which is quite impracticals. Also DSPNs allow only use of deterministic or exponential distribution on messages, we are not able to model other probabilistic distributions.

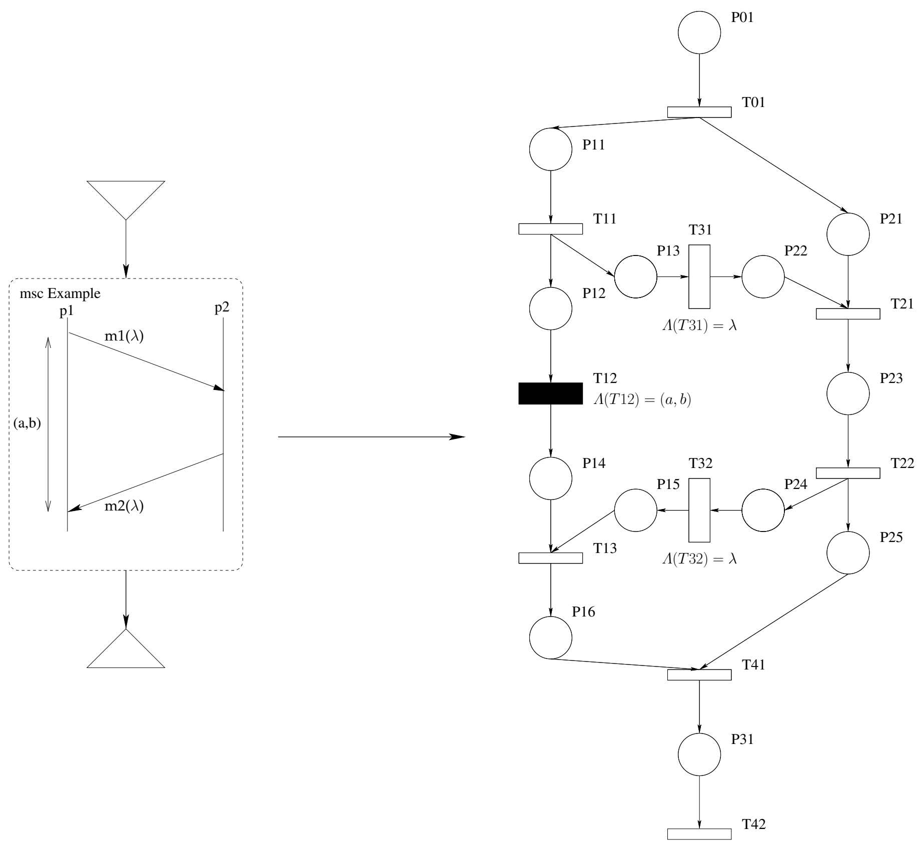

Figure 4.3: Transformation of a simple MSC into a stochastic Petri net

4.3 Stochastic Message Sequence Charts

Another approach how to extend MSCs with stochastic information was introduced by Zhihe Zhou and Frederick T. Sheldon [14] by defining Stochastic Message Sequence Chart (SMSC).

An SMSC is an MSC where all events are associated with stochastic time needed to complete the activity. An instantaneous event still can be modeled, it is an activity with associated zero time.

Stochastic time associated with activities can be of any kind of probabilistic distribution (deterministic, exponential, ...).

Message in SMSC consists of two activities – activity of sending the message and activity of receiving the message. This is done so to be able to construct ordering of events.

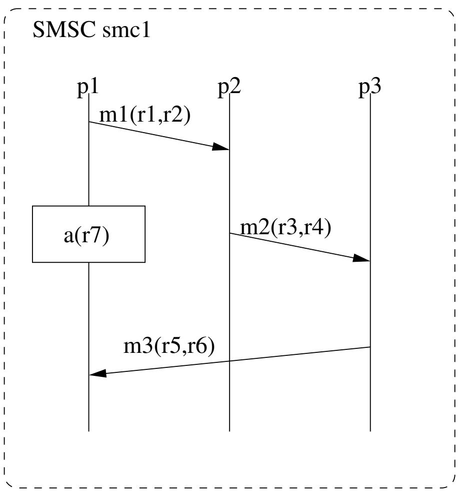

Figure 4.4: Stochastic Message Sequence Chart

In Figure 4.4 there is shown an example of SMSC with three messages and one local activity. Probability distribution used is exponential probability distribution. Each message has two associated parameters – first (in the case of message $m1$ it is $r1$) specifies the rate of an exponentially distributed random variable that gives the amount of time needed to send the message, second parameter ($r2$ in case of $m1$) assigns time to the message receiving activity. Local activities have just one parameter ($r7$ in case of the activity a).

Difference Between MSCs and SMSCs

MSCs do have time concepts to represent time passed between two events. But events are still instantaneous, time is introduced as a special event that can be inserted between two consecutive events to represent the time elapse. Moreover time event represents deterministic not random time.

There is no special time event in SMSCs. Each activity is associated with some time. An SMSC activity can mimic MSC event if the time associated is zero. In that case the SMSC activity is also instantaneous.

All constructs (processes, local actions, conditions, ...) defined on MSC are used by SMSC as well. The graphical representation of SMSC also looks the same, we just add the parameters needed to specify the time to each message or local action.

We construct High-level Stochastic Message Sequence Chart (hSMSC) the same way we did for MSC.

Ordering Rules

In MSCs there can happen just one event at the same time (because they are instantaneous). So we can construct a partial order of events. On the other hand, in SMSC each activity can last for some amount of time. It means some activity may start before some other already started finishes. In that case what is the order of these two activities?

We can decompose an activity to two events, one for starting the activity and the other for it's ending. The order of activities can be defined either as the order of starting events or that of the ending events.

There are five rules for the ordering of activities and activity events:

1. The event of starting an activity must happen before the event of finishing the same activity.
2. Activities attached to a process are executed sequentially in the same order as they are given on the vertical axis from top to bottom. An activity can only start after the previous one has finished.
3. The activity of sending a message must finish before the activity of receiving the same message can begin.
4. Activities in a coregion may happen in any order, but following the rule 1.
5. If general orderings are used, they are treated as messages in terms of ordering these activities.

All these rules are quite natural, just the rule number 3 is worth noting. A message includes two activities, and hence four events – the event of starting to send the message (SS), the event of starting to receive the message (SR), the event of finishing the sending of the message (FS) and the event of finishing the receiving of the message (FR). Naturally, SS must happen before SR and FS must happen before FR. But rule number 3 is stronger – it says that also FS must happen before SR. Why is it so?

If the receiving activity can start before the message has been completely sent, then it is possible that the receiving activity finishes before the sending activity finishes because the receiving activity takes random time to complete. Hence we cannot guarantee it finishes after the sending activity has finished.

Traces

An MSC specifies a set of valid traces that the system can take. We can define a sequence of activities as a trace, then also an SMSC specifies a set of valid traces. In addition, an SMSC also specifies a stochastic process. The stochastic process enables us to do performance analysis about the system.

Integrating SMSC into the Möbius Framework

Möbius [4] is an extensible framework which allows implementing different stochastic models and provide tools to solve them. SMSCs can be integrated within this framework and we can use its solvers. For more details about integrating SMSC to Möbius please see [14].

Summary

This approach is closest to what we want. It is able to capture stochastic time parameters, and even the coregions. Author discusses just the exponential distribution, but says that any may be used. Advantage is also the usage of an already existing tool.

The main thing that is missing are the time constraints. Also there is no way how to specify the probabilities for choice in an alternative composition in an hMSC.

Chapter 5

Performance Analysis

This chapter contains discussion what approaches may be used in practice to solve the problems mentioned in Chapter 3.

5.1 Random Variables and Probabilistic Distributions

Random variable, as we will understand it, is a function $\Omega \to \mathbb{R}_0^+$, where Ω is a sample space. Intuitively, it returns some value from the nonnegative real numbers.

Every random variable may be described either by its cumulative distribution function (*cdf*) or its probability density function (*pdf*).

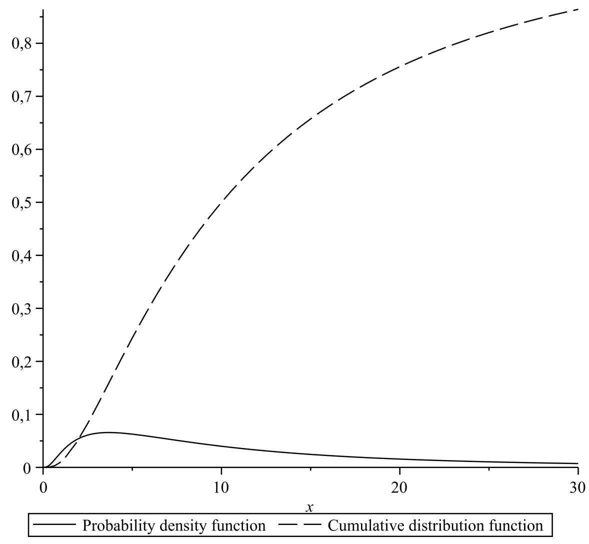

Figure 5.1: Log-normal distribution

A cumulative distribution function of a random variable X specifies the probability that the outcome of X is smaller or equal to some number. Formally $cdf_X(x) = P(X \leq x)$.

Other way how to describe a probability distribution is by a probability density function, which describes the density of probability at each point in the sample space. The probability that the outcome of a random variable falls within some interval is given as an integral of density function over that interval.

There is a close relation between a cumulative distribution function and a probability distribution function:

$$cdf_X(x) = \int_0^x pdf_X(x)\mathrm{d}x$$

Example of a cumulative distribution function and a probability density function for a log-normal distribution may be seen in Figure 5.1.

Probability density function of a sum of two independent random variables is the *convolution* of their density functions:

$$pdf_{X+Y}(x) = \int_0^\infty pdf_X(y)pdf_Y(x-y)\mathrm{d}y$$

The cumulative distribution function of a maximum of two independent random variables may be computed as:

$$cdf_{\max(X,Y)}(x) = cdf_X(x)cdf_Y(x)$$

5.2 Stochastic Analysis in MSC

Since now, we assume that every two neighboring events in the visual order have assigned a random variable specifying the time needed to complete the activity. If some pair of events does not, we assume that it happens instantaneously.

Our aim is to compute probability density function or cumulative distribution function for the whole MSC.

First we assume that there are no constraints in an MSC. If two activities with associated random variables X_1 and X_2 are sequentially composed, the time needed to complete them is determined by a random variable defined as $X_1 + X_2$.

If two or more activities meet in a coregion, time needed is determined by a random variable defined as the maximum of random variables of all these activities.

For example, for an MSC depicted in Figure 5.2, total time needed to complete all message exchanges is determined by a random variable defined as $X_1 + \max(X_2 + X_4, X_3 + X_5)$.

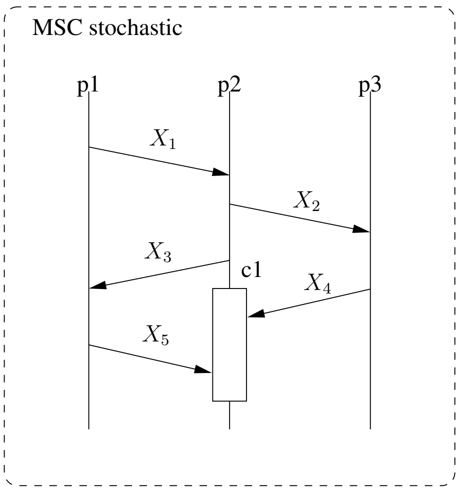

Figure 5.2: Message Sequence Chart with associated random variables

If there are constraints, we start computing them bottom up, from the ones that do not contain any other constraints inside. We first compute the probability distribution function of a time needed to complete all the activities inside the constraint (we can do this because the constraint is proper, so there are no incoming messages we need to wait for, etc.) and then truncate this distribution just to the specified interval. If the area within a constraint need time determined by a random value X and the constraint is the interval (a, b) then the probability density function of the truncated distribution is

$$pdf_{X|a<X<b}(x) = \frac{g(x)}{cdf_X(b) - cdf_X(a)}$$

, where $g(x) = pdf_X(x)$ for all x from interval (a, b) and $g(x) = 0$ elsewhere.

5.3 Stochastic Analysis in hMSC

The situation in an hMSC is more complicated, especially because of the cycles. We will show how to solve acyclic hMSCs and then we discuss how to deal with cycles.

We again compute a probabilistic distribution for a part within the constraint, starting bottom up. Because there is just a finite number of MSCs inside and no cycles, there is also only a finite number of possible traces. We take all of them, concatenate each to form an MSC, and compute the cumulative distribution function of a random variable associated with this MSC. Also we need to multiply for each trace all the probabilities of transitions from one MSC to another along the trace, this gives us the probability of this particular trace. Cumulative distribution function of the whole part of hMSC is then

$$cdf_{hMSC}(x) = \sum_{t \in Traces} p_t cdf_{MSC_t}(x)$$

where $Traces$ denotes the set of all traces in this part of hMSC, p_t is the probability of execution according to a trace t and finally MSC_t is an MSC created as a concatenation of all MSCs along the trace t.

Unfolding the cycles leads to an infinite number of traces, which is unfeasible. But there are two ways, how to restrict number of iterations of one cycle, if we are satisfied with computation with just some precision.

There is a probability p that the execution enters and repeats a cycle. Probability, that we perform exactly x iterations of the cycle is $p^x(1-p)$. As x goes higher this probability is decreasing below any limit. So we may ignore all iterations above some number of repetitions, because their impact on overall performance is negligible.

Another approach is to compute the probabilistic distribution of time needed to perform one iteration of the cycle and if there is a constraint in which this cycle is inside, we may restrict the maximal number of iterations such that more repeating of a cycle gives us only negligible chance, that the process will satisfy the constraint.

5.4 Practical Usage

The real practical usage depends very much on used probability distributions. If we use for example log-normal probability distribution, computation is unfeasible even with just a few messages.

There are several possible ways how to overcome these difficulties, such as discretization of the distributions (replacing the continuous distribution with a discrete one) or approximating the distributions with simpler ones.

Chapter 6

Conclusion

In this thesis we have studied Message Sequence Chart, formalism for describing the behaviour of a communication system. Especially time and stochastic concepts were in focus.

Survey of currently existing approaches how to deal with probabilities in MSC– reliability prediction and sensitivity analysis, transformation into a stochastic Petri net, and Stochastic Message Sequence Charts – was provided.

Also some ways how to compute properties of Message Sequence Chart with probabilities and constraints were suggested.

Bibliography

[1] R. Alur and M. Yannakakis. Model Checking of Message Sequence Charts. In *Proceedings of the 10th International Conference on Concurrency Theory*, pages 114–129. Springer-Verlag London, UK, 1999.

[2] Rajeev Alur, Kousha Etessami, and Mihalis Yannakakis. Inference of message sequence charts. In *International Conference on Software Engineering*, pages 304–313, 2000.

[3] RC Cheung. A user-oriented software reliability model. In *The IEEE Computer Society's Second International Computer Software and Applications Conference, 1978. COMPSAC'78*, pages 565–570, 1978.

[4] D. Daly, D.D. Deavours, J.M. Doyle, P.G. Webster, and W.H. Sanders. Mobius: An extensible tool for performance and dependability modeling. *Lecture notes in computer science*, pages 332–336, 2000.

[5] P.R. D'Argenio, H. Hermanns, and J.P. Katoen. On generative parallel composition. *Electronic Notes in Theoretical Computer Science*, 22:30–54, 1999.

[6] E. Elkind, B. Genest, and D. Peled. Detecting Races in Ensembles of Message Sequence Charts. *TACAS'07*.

[7] ITU Telecommunication Standardization Sector Study group 17. ITU recommendation Z.120, Message Sequence Charts (MSC), 2004.

[8] O. Kiuge. Time in Message Sequence Chart Specifications and How to Derive Stochastic Petri Nets. In *Communication-Based Systems: Proceedings of the 3rd International Workshop Held at the TU Berlin, Germany, 31 March-1 April 2000*, page 17. Kluwer Academic Publishers, 2000.

[9] L'. Korenčiak. Time Extension of Message Sequence Chart *Bachelor thesis*, Faculty of Informatics, Masaryk University, Brno, 2009.

[10] Genaína N. Rodrigues, David S. Rosenblum, and Sebastian Uchitel. Sensitivity analysis for a scenario-based reliability prediction model. In *WADS '05: Proceedings of the 2005 workshop on Architecting dependable systems*, pages 1–5, New York, NY, USA, 2005. ACM.

[11] Genaína Nunes Rodrigues, David S. Rosenblum, and Sebastián Uchitel. Using scenarios to predict the reliability of concurrent component-based software systems. In *FASE*, pages 111–126, 2005.

[12] SCStudio – Sequence Chart Studio.
`http://scstudio.sourceforge.net/`.

[13] S. Uchitel, J. Kramer, and J. Magee. Synthesis of behavioral models from scenarios. *IEEE Transactions on Software Engineering*, 29(2):99–115, 2003.

[14] Z. Zhou and F.T. Sheldon. Integrating Stochastic Message Sequence Charts into Möbius for Performance Analysis. *IEEE Tools 2003*, 2003.

[15] A. Zimmermann, R. German, J. Freiheit, and G. Hommel. TimeNET 3.0 tool description. In *Int. Conf. on Petri Nets and Performance Models (PNPM'99), Zaragoza, Spain*, 1999.