Universität Konstanz

Department of Computer and Information Science

Master Thesis

Quantitative Analysis of Concurrent System Architectures

for the degree
Master of Science (M.Sc.) in Information Engineering

by

Adrian Beer
(01/648999)

1^{st} Referee:	Prof. Dr. Stefan Leue
2^{nd} Referee:	Prof. Dr. Marc H. Scholl
Advisor:	Florian Leitner-Fischer

Konstanz, 16th March 2012

Abstract

Safety-critical software and systems development is subject to special dependability requirements. Early analysis of dependability during design and development phase is often a statutory condition for the approval of technical systems. In order to support the developers in verifying and analysing these systems the QuantUM tool was recently introduced [31]. The UML model of the system can be annotated using the presented QuantUM-profile. Afterwards, QuantUM translates the annotated models into the input language of the probabilistic model checker PRISM, which is used as an analysis back-end.

This thesis proposes an extension to the QuantUM method so that non-determinism is supported. Non-determinism is an essential paradigm when analysing concurrent system architectures. We introduce new translation rules into the QuantUM profile which are then used to transform UML or SysML models into locally uniform Continuous-Time Markov Decision Processes. Since there are no efficient model checking algorithms for CTMDPs, a discretization technique is applied to convert the CTMDP into discrete-time MDPs. The new approach is applied to two case studies, an Airport Surveillance Radar and an Airbag Control Unit.

Kurzfassung

Bei der Entwicklung sicherheitskritischer Software und Systeme müssen spezielle Sicherheitsanforderungen eingehalten werden. Die frühe Analyse solcher Anforderungen, während der Design- und Entwicklungsphasen, ist oft gesetzliche Voraussetzung für die Zertifizierung technischer Systeme. Um die Entwickler bei der Analyse zu unterstützen, wurde das QuantUM-tool entwickelt [31]. Dieses beinhaltet ein UML Profil, welches zur Annotation von UML Modellen benutzt werden kann. QuantUM konvertiert die annotierten UML Modelle danach in die Eingabesprache des probabilistischen Model Checkers PRISM, welcher als Analyse-Back-End dient.

In dieser Arbeit wird eine Erweiterung des QuantUM-tools präsentiert, welche auf den so gennanten Nicht-determinismus aufbaut. Nicht-determinismus ist essenziell wenn es um die Analyse von nebenläufigen System Architekturen geht. Es werden neue Übersetzungsregeln für QuantUM definiert. Diese ermöglichen die Übersetzung von UML bzw. SysML Modellen in lokal uniforme Continuous-Time Markov Decision Processes. Da es bisher keine effizienten Algorithmen für das Model Checking von CTMDPs gibt, wird eine Diskretisierungstechnik verwendet, um die CTMDPs in Discete-Time Markov Decision Processes zu konvertieren. Dieser neue Ansatz wird im Anschluss in zwei Fallstudien angewendet, ein Flughafen-Überwachungs-Radar und eine Airbagkontrolleinheit.

Acknowledgements

Firstly, I want to thank my supervisor Prof. Stefan Leue for his excellent guidance, his advice and all the comments he gave me about my work. Additionally I want to thank him for giving me the opportunity to work in his research group. I would like to thank also Prof. Marc Scholl for agreeing to be the second referee of my thesis.

I thank the whole research group for software engineering at the University of Konstanz for providing a great and friendly working atmosphere. Especially, I would like to thank Florian Leitner-Fischer for his outstanding mentoring and for answering my continuous flow of questions.

A special thank goes to my family for the large support and encouragement during my student days. Special thanks to my father, who taught me how to write scientifically and always had an open ear to my questions about engineering. My special gratitude goes to Carolin for her love and continuous motivation, especially during her own audit periods.

Contents

1. Introduction

Safety critical software and systems are part of our daily routine. Every system we are using in our daily routine adds a certain risk of harming someone. The higher the risks are, the more important the development of safety-critical systems becomes. The requirements of such safety-critical software and systems have special dependability demands. Dependability is a term in the field of software and systems engineering describing the reliability, availability, maintainability and safety of a system [29].

The analysis and verification of safety-critical systems at an early design phase is of high interest to the systems and software engineers. However, only few tools support the verification of the dependability requirements at early development stages. Nevertheless, the modelling of software and systems is standardised. The Unified Modelling Language (UML) or Systems Modelling Language (SysML) is an example for such a standardized notation. It is a graphical modelling language for designing software [3] and systems respectively [1]. UML and SysML give a semi-formal representation of a system. Semi-formal means that the structure of the system is defined on a formal basis and additional information to each part can be given in natural language descriptions. There are some tools supporting the simulation of such systems on an UML-basis. One well known tool is IBM Rational Rhapsody[1]. Since a simulation can never check any possible system status, it can in fact not give a reliable answer to the dependability questions. For instance one question could

[1] http://www.ibm.com/software/awdtools/rhapsody/

ask for the probability of a system to fail. With simulation only the answer to this question could not be found within reasonable time.

The QuantUM approach [31] supports the specification and analysis of system dependability requirements for UML models. Recently it has been extended to support also SysML models [30]. Using QuantUM it is possible to integrate the analysis and verification of an UML or SysML model directly at the modelling stage of a system. A benefit of this approach is that users do not have to understand the entire formal verification process in detail. This would require that the engineers have to learn the mathematical foundations of the formal verification which is a time consuming process. Another advantage is that QuantUM can cooperate with many industrial practice UML tools. This means that engineers can use the modelling tool and notations that they are familiar with and do not have to familiarize themselves with formal notations or languages needed to operate formal verification tools. The analysis provided by QuantUM is fully automatized which promises a much less time consuming verification process compared to a manual one. The results are presented in terms of failure probabilities or fault trees automatically generated from failure paths in the system[38]. This representation of the results is industrial standardized. Thus, it can be understand by the engineers without any further knowledge of the formal process.

Currently the QuantUM- approach supports the use of Continuous-Time-Markov Chains (CTMC) [9]. These models can be used to analyse probabilistic, time-depended systems in an efficient way. In CTMCs the transition from one system state to another depends on rates. The problem of the Markov Chains is that they do not support non-determinism. Non-determinism is an essential paradigm when analysing concurrent system architectures. Non-deterministic choices are state changes without a probability, which would result in a probability distribution. In concurrent systems the choice of a successor state is not modelled in the system itself. The selection depends on the scheduler which resolves the non-determinism. QuantUM is currently ignoring such non-deterministic choices. The choices are approximated by very high rates between states. This modelling trick results in an overestimation of the actual probabilities in the system [26]. This overestimation can be explained as follows: When a non-deterministic selection is replaced by a high rate, certain executions of the system become possible, though with very low probability.

In a non-deterministic interpretation of the system these executions would be absent, and thus would not contribute to the reachability probability.

The objective of this thesis is to extend the QuantUM approach to support non-determinism. For this purpose Markov Decision Processes are used. There are two types of Markov processes the Discrete-Time- (MDP) and the Continuous-Time-Markov decision process (CTMDP). MDPs and CTMDPs are the non-deterministic duals to the Discrete- and Continuous-Time-Markov Chains. The use of CTMDPs is not efficient with respect to runtime and memory consumptions. Nevertheless, we do not want to restrict the non-determinism of our approach to a discrete-time basis. Therefore we will use a discretization technique for the continuous model [36]. The discretization represents an approximation of the continuous-time model to a discrete Markov decision process. This means that the probabilities computed with the discretized CTMDP are approximate values. We will show that the error introduced by the discretization lies within an acceptable pre-computable range.

In a joined project with our industrial partner CASSIDIAN[2] we have applied the QuantUM approach to a real system. This system is an *Airport Surveillance Radar* (ASR) system modelled in SysML[1]. The ASR consists of a pure radar-signal based *primary radar* component, and a transponder based *secondary radar* component. The detected echoes are tracked by the system and graphically displayed to air traffic controllers, who are the typical users of the system. The system is safety-critical by nature, hence very stringent dependability requirements apply. The system is subject to certification requirements imposed by the German Government. The case study primarily aims at the scalability of a modeling and verification approach based on QuantUM.

The second case study applies the approach to a model from the automotive domain. An Airbag Control Unit is modeled in the UML [3] and verified using the new non-deterministic models. The original case study was done in [31] in a joined work with TRW Automotive GmbH[3]. The system can be divided into three main parts, the sensors, crash evaluation and actuators. An impact is detected by the sensors. The sensor information is evaluated by a microcontroller which decides whether there is a crash or not. A deployment

[2]EADS Deutschland - Cassidian: http://www.cassidian.com/
[3]TRW Automotive GmbH: http://www.trw.de/

of the Airbag is only performed when the microcontroller decides that there is a critical crash happening.

1.1 Related Work

In [18], Debbabi et al. present a method for automated model checking of SysML activity diagrams. Since in SysML it is possible to tag activity diagrams directly with probabilities, it is not necessary to introduces stereotypes, like it is done in QuantUM. This means the diagrams can be checked by converting them directly into Timed Petri Nets [35]. A disadvantage of this approach is that only single diagrams are checkable. The combination and checking of several concurrent diagrams is disregarded. The combination of different diagrams is addressed in [13] where Bernardi et al. have extended the UML profile for modelling and analyzing real time embedded systems with an additional profile for dependability analysis and modeling. A drawback of this method is that it relies on the MARTE[4] profile which is not well supported in most commercial UML modelling tools. Another disadvantage of the profile lies in the high number of stereotypes that have to be applied to actually prepare the model for the verification. The conversion of the UML diagrams to Petri Nets [35] that is required has to be done manually, which is an error prone and time consuming process. The application of this approach to complex systems is, hence, impracticable.

In [23] Jansen shows a stochastic extension of Statecharts. The transitions in the Statecharts are tagged with rates. Afterwards they can be checked using the approximate model checker $MC2$ [5]. With this extension only the behavior of the system is analysed and the structure is omitted completely.

The work presented in [34] by Majzik et al. introduces a new UML profile for the annotation of software dependability properties. After transforming these models into an intermediate format, they are converted into Timed Petri Nets [35]. Timed Petri Nets support non-determinism by definition. This means they are suitable to model concurrent systems. Like in the approach discussed above, a drawback is the manual conversion of the model. A second disadvantage is that it only focuses on structural aspects of the system. Similar

[4] http://www.omgmarte.org/
[5] http://www.brc.dcs.gla.ac.uk/software/mc2/

to the MARTE approach, there is a high number of stereotypes and often redundant information attached to the model.

In [14], the architecture dependability analysis framework *Arcade* is presented. Input/Output-Interactive-Markov Chains are used to model the systems [15]. A limitation of this approach is that Arcade cannot be easily integrated with industrial design processes.

In [19] Güdemann presented the formal modeling framework SAML. This framework allows the tool-independent specification of formal models for a model-based safety analysis. SAML uses finite state automata with discrete-time and non-deterministic steps (MDP). Nonetheless continuous-time rates are supported. Over a temporal resolution of the SAML model the rates can be interpreted according to their distribution. The rates are transformed into probabilities based on the resolution of the model. The probability for the time of residing in one state is represented as a self-loop in this state. A disadvantage of the SAML approach is that models have to be specified manually in the SAML language. There is no support for a direct conversion of UML or SysML models. This process can be error prone. Another drawback lies in the failure descriptions of systems modeled with SAML. They consider only the possibility of an error in general but not a specific type of failure of a component.

The paradigm of converting the rates into self-loop- and outgoing- probabilities is also discussed in [5]. In this work a uniformization technique of Continuous-Time Markov Decision Processes (CTMDP) is presented. They focus on infinite-horizon properties. An infinite-horizon property is an unbounded reachability property where no upper bound on the time is given. This is not applicable in this thesis, since we want to check the system for failure within certain time intervals.

In [10], Baier et al. presented a technique to check time-bounded reachability properties on uniform CTMDPs. They show that it is possible to compute the maximum probabilities for timed reachability efficiently. However, we cannot apply this approach to our systems, because we can not assume all failure rates to be globally uniform.

Neuhäuser and Zhang proposed a technique to compute time-bounded reachability properties on locally uniform CTMDPs [36]. They used a discretization step size to convert the CTMDP into a discrete MDP. We will show that this

approach is applicable to the systems we are analysing and that the models we are using are locally uniform by construction.

1.2 Contributions

The main contributions of this thesis can be summarized as follows:

- We propose an extension of the QuantUM approach to support non-determinism for analysing concurrent system architectures.
- An automatic translation of UML and SysML models into CTMDPs is presented. These CTMDPs are locally uniform by construction.
- We describe an automatic discretization technique from locally uniform CTMDPs to discrete MDPs in order to efficiently compute time-bounded reachability properties with the probabilistic model checker PRISM.
- On two case studies we show that the discretization algorithm generates models on which it is efficiently possible to calculate the probability of the time-bounded reachability properties.
- We show that the non-deterministic approach models concurrent system architecture more adequately.
- We present the implementation of the new non-deterministic QuantUM approach into a prototypical tool chain.

1.3 Structure of the Thesis

In Chapter 2 the foundations are explained. They include the basic ideas of UML and SysML as well as the concepts of model checking. The non-determinism and the discretization of the continuous-time models are explained in Chapter 3 and 4. The extension of the QuantUM tool is presented in Chapter 5. Afterwards the case studies are presented (Chapter 6). In the last chapter a perspective for the future is given (7). In the appendix an extra chapter for explanation and radar-related abbreviations is given for those readers who are not familiar with the radar terminology (A.1).

2. Foundations

In this chapter the foundations to the methods described in the following chapters are presented. The use of UML and SysML is described, since this specification languages are the basis of the QuantUM approach. In the QuantUM-tool chain the probabilistic model checker PRISM [21] is used. Hence the basic idea of probabilistic model checking is given and the difference between discrete-time and continuous time models is presented.

2.1 UML / SysML

The Unified Modelling Language (UML) is a specification language which is a de-facto standard in software engineering. It is used to model structural, behavioral and architectural aspects of systems. UML is standardized by the *Object Management Group* a non-profit organization founded by some well known software companies like IBM, Apple, SUN and others.

Tools supporting the use of UML for modelling software are for instance, IBM Rational Rhapsody[1], Sparxsystems Enterprise Architect[2], or ArgoUML[3]. A detailed description of the UML specification is given in [3]. Since UML was mainly developed to model software architectures, it was necessary to extend UML to systems engineering, in order to make it applicable in this domain. In systems engineering both the software and the hardware need to be considered.

[1] http://www.ibm.com/software/awdtools/rhapsody/
[2] http://www.sparxsystems.de/uml/
[3] http://argouml.tigris.org/

The *International Council of Systems Engineering* (INCOSE) added specific modeling techniques to the UML that reflect needs from the system engineering domain. The extension is called SysML [1]. The ASR case study (Section 6.1) uses the SysML language to specify the models of the system. In the second case study (Section 6.2) UML is used for the specification.

SysML is structured in two main parts. One describes the structure of a system and the other one describes the behavior of the system. The structure of a system is represented through block definition diagrams or parametric diagrams. State machines and activity diagrams are part of the behavioral specification techniques used in SysML. Overall, SysML comprises 11 diagram types, some of which can also be found in the UML specification.

2.2 Probabilistic Model Checking

Probabilistic Model Checking (PMC) is a technique for automated analysis of safety-critical systems. Azis et al. proposed PMC [9] in 1996. The term of probabilistic model checking was introduced by Kwiatkowska et al. in [28].

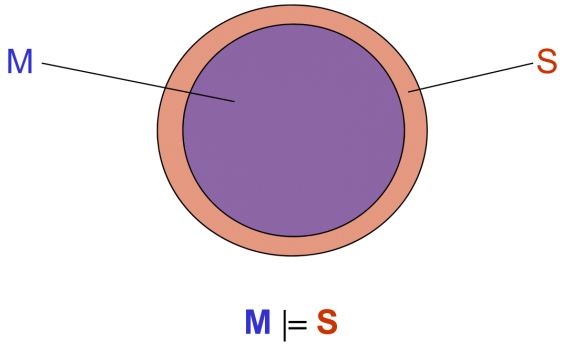

Figure 2.1: An abstract representation a model M and a specification S [33].

Model Checking is about verifying properties or specifications S against a model M. In most cases this model is represented by a transition system (TS). Every state in such a TS represents an observable valuation of the variables and program control pointer in a system. Transitions between the states represent the functionality of the system carried out. The transitions can be guarded, which means that a state change can only happen when the guard condition is satisfied.

If the transitions are labeled with quantitative information about the system, such as probabilities or rates, Probabilistic Model Checking can be applied. The probabilities or rates describe the probabilistic system behavior over discrete steps or continuous time respectively. Several types of transition systems where introduced to handle different types of probabilistic behavior. Discrete-Time and Continuous-Time Markov Chains are two types of these systems which will be discussed on the following pages.

2.2.1 Markov Chains

Definition 2.1 Discrete-Time-Markov Chain (DTMC)
A Discrete-Time-Markov Chain is a 5-tuple $\mathcal{M} = (S, \mathbf{P}, \iota_{init}, AP, L)$ *where*

- S *is a countable, nonempty set of states*
- $\mathbf{P} : S \times S \to [0,1]$ *is the transition probability function. All outgoing transitions in* $s \in S$ *have to sum up to* 1.

$$\sum_{s' \in S} \mathbf{P}(s, s') = 1$$

- $\iota_{init} : S \to [0,1]$ *is the initial distribution with*

$$\sum_{s \in S} \iota_{init}(s) = 1$$

- AP *is a set of atomic propositions and* $L : S \to 2^{AP}$ *a labeling function.*

■

The difference between transition systems and Discrete-Time-Markov Chains [11] is that in DTMCs the nondeterministic choices among successor states are replaced by probabilistic transitions. DTMCs give a time-abstract probabilistic representation of a system. Time-abstract means that the transitions between states are only chosen based on a probabilistic distribution on the outgoing transitions. In Definition 2.1 this transition probabilities are given in the form of the transition probability function $\mathbf{P}$. The given constraint on $\mathbf{P}$ says that the sum of the probabilities of each outgoing transition in one state

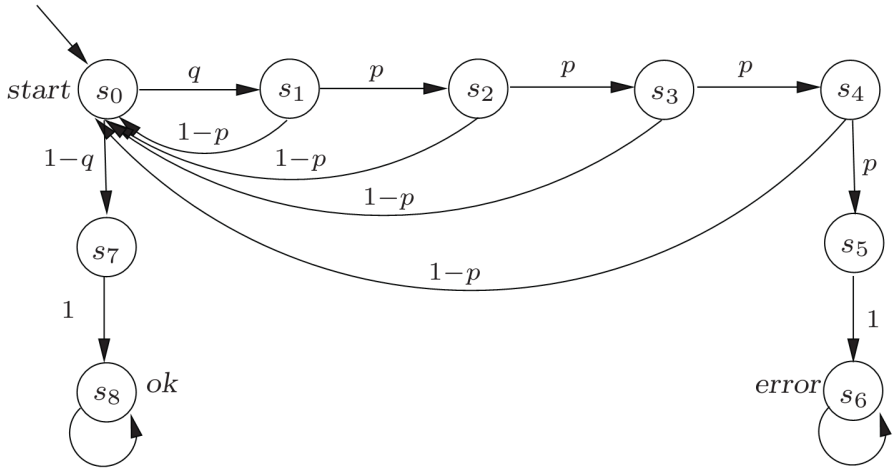

Figure 2.2: DTMC of the IPv4 zeroconf protocol for $n = 4$. [11]

has to sum up to 1. The initial state is determined accordingly using the initial distribution ι_{init}.

An application where DTMCs can be used to verify system correctness is the IPv4 zeroconf Protocol [11]. This protocol is designed to connect multiple devices in a network to enable mutual communication. The setup must be hot-pluggable and self-configuring. This means, that if a new device is connected to the network an unused, unique IP-address has to be provided to the device. The protocol starts by selecting a random address out of the 65024 possibilities. With this address a message is sent to the network asking if this address is already in use. If the message is in use the device using this address sends back a reply and the new device starts over again by selecting another random address. To increase the reliability of the protocol the new address is only taken if there is no response after n-times sending the new- address-message and not receiving an answer. In Figure 2.2 the automaton of the protocol is presented. In the start state s_0 the probability of choosing an already used address is $q = m/65024$ (m number of devices in the system). If an address is picked which is already in use the system transits into state s_1. p denotes the probability that no response is given within a given time interval. The system stops in the error state if no response arrives after 4-times sending the message with the new IP-address, despite the fact that the address was already in use. With this model of the protocol the probability of getting into the error state can be computed without timing constraints.

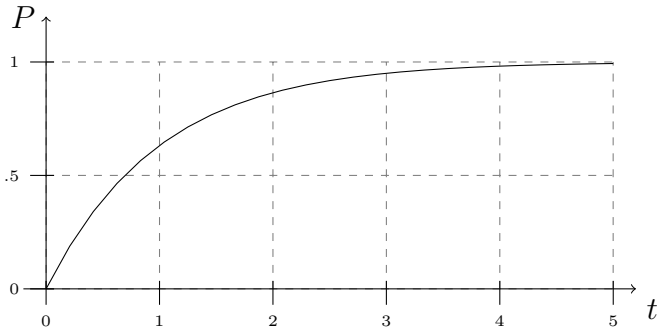

Figure 2.3: The negative exponential distribution $1 - e^{-\lambda \cdot t}$, for rate $\lambda = 1$

If the analysis of a system has to take real or continuous time into account, the use of a more advanced type of model is required. This model is called Continuous-Time-Markov Chain [9].

Definition 2.2 Continuous-Time-Markov Chain
A Continuous-Time-Markov Chain is a 5-tuple $\mathcal{M} = (S, \overline{s}, \mathbf{R}, AP, L)$*, where*

- S *is a countable, nonempty set of states, with* $\overline{s}$ *as initial state.*
- $\mathbf{R}$ *is a* $|S| \times |S|$ *transition rate matrix*
- AP *is a set of atomic propositions and* $L : S \to 2^{AP}$ *a labeling function.*

■

R is the transition rate matrix, which assigns a non-negative, real valued rate to each transition between states in the model. The probability to make a transition from state s_i to state s_j $(i \neq j)$ is interpreted as the rate of a negative exponential distribution (Figure 2.3). The advantage of using negative exponential distributed rates is that it represents a memoryless state change [12, 24]. This means that the probabilities of residing in a state are independent of the history of a path leading into this state.

The probability P to transit from s_i to s_j with rate λ and time t is given by the following formula:

$$P(\lambda, t) = 1 - e^{-\lambda \cdot t}$$

Continuous-Time-Markov Chains can be used as models for systems where probabilities over time are important for the correctness of the system, for instance a wireless communication cell for mobile phones. In this setting the mobile phone can travel around and is changing the cells while travelling. The model is constructed in such a way that the cell can accept the mobile device if there is a free channel in this cell. A possible property is that a mobile phone arriving in a cell is assigned to a free channel within a certain time bound and probability. The probability denotes the possible errors caused by the environment (e.g. houses, bad weather etc.).

2.2.2 Property Specification

In order to specify properties of a model M formally it is necessary to specify some logics. Since in DTMCs the steps between states are discrete, a logic is used which refers to discrete steps in the models. This logic is called PCTL [20]. PCTL (probabilistic CTL) is based on the Computation Tree Logic (CTL) introduced by Emerson, Clarke and Sistla [17]. While CTL is a modal (temporal) logic for reasoning about qualitative system correctness PCTL introduces (discrete) time and probability properties.

Definition 2.3 Probabilistic Computation Tree Logic (PCTL)
PCTL state formulae are defined as follows:

$$\Phi \quad := \quad true \quad | \quad a \quad | \quad \Phi_1 \wedge \Phi_2 \quad | \quad \neg\Phi \quad | \quad \mathbb{P}_J(\varphi)$$

where $a \in AP$, φ is a path formula and $J \subseteq [0,1]$ is an interval with rational bounds. PCTL path formulae are formed according to the following grammar:

$$\varphi \quad := \quad \circ\Phi \quad | \quad \Phi_1 \mathcal{U} \Phi_2 \quad | \quad \Phi_1 \mathcal{U}^{\leq n} \Phi_2$$

where Φ, Φ_1 and Φ_2 are state formulae and $n \in \mathbb{N}$ ■

State formulae are formulae that hold in specific states. This means in some states of the system they evaluate to true and in others to false. The path formulae represent the transient behavior of the system. With PCTL it is possible to verify properties like: “With a probability higher than 50% a

finish state is reached within 20 time steps". In PCTL this formula looks like the following:

$$\mathbb{P}_{>0.5}(true\ \mathcal{U}^{\leq 20} finish)$$

where $finish \in AP$. PCTL is designed to formally specify properties for discrete time probabilistic systems like DTMCs. In DTMCs the steps between states are discrete.

If the system considers continuous time, PCTL is not applicable, since it only uses discrete intervals for the path formulae. To express properties such as "within 5 minutes the finish state has to be visited with probability higher than 90%" it is necessary to come up with another kind of logic. This logic is called Continuous Stochastic Logic (CSL) [9].

Definition 2.4 Continuous Stochastic Logic (CSL)
State formulae in CSL are defined as follows:

$$\Phi \quad := \quad true \quad | \quad a \quad | \quad \Phi_1 \wedge \Phi_2 \quad | \quad \neg\Phi \quad | \quad \mathbb{P}_J(\varphi)$$

where $a \in AP$, φ is a path formula and $J \subseteq [0,1]$ is an interval with rational bounds. CSL path formulae are formed according to the following grammar:

$$\varphi \quad := \quad \circ\Phi \quad | \quad \Phi_1 \mathcal{U} \Phi_2 \quad | \quad \Phi_1 \mathcal{U}^J \Phi_2$$

where Φ, Φ_1 and Φ_2 are state formulae and J is a time interval in which the formula holds. ■

PCTL and CSL are very similar but the definition of the bounded until operator ($\mathcal{U}^J$) is different. In CSL the bounds on the operator are real time values. For example the CSL property "within 5 minutes the finish state has to be visited with probability higher than 90%" is defined as follows:

$$\mathbb{P}_{>0.9}(true\ \mathcal{U}^{[0,5]} finish)$$

It is important that the time basis in the model is based on minutes for this formula to hold, i.e. the rates in the CTMC have to be based on minutes.

2.3 Tool Support

In this thesis the probabilistic model checker PRISM [21] is used to verify the models. PRISM is a model checker which supports different models and logics. On the model side there are DTMCs, CTMCs and Markov Decision Processes (MDPs) which will be discussed in Chapter 3. PRISM supports the use of PCTL, CSL and LTL (Linear Temporal Logic) to verify the models. For the efficient verification of models PRISM has some symbolic representations built in. This means that it uses data structures based on binary decision diagrams (BDD) and multi-terminal binary decision diagrams (MTBDD) [4]. The use of BDDs helps to compress the whole model in terms of memory efficiency since they allow to perform transformations directly on the compressed data without the need to decompress them first.

2.4 Summary

In this section the foundations of probabilistic model checking were discussed. The QuantUM tool currently supports the use of Continuous-Time-Markov Chains. A disadvantage of this type of Markov Chains is that their definition compared to transition systems only considers transitions tagged with a certain probability or rate. This means that there is no transition happening without a probabilistic choice. The problem is that there are systems where such probabilistic choices are not reasonable. For instance in concurrent systems non-deterministic selections of successor states would describe the system behavior more precisely. Therefore, the introduction of non-determinism is discussed in the following chapter.

3. Non-Determinism in Markov processes

In Discrete-Time and in Continuous-Time-Markov Chains all transitions are chosen based on some probabilistic choices. One limitation of these models is that they do not support non-determinism. Non-determinism is important in cases where the probability of a choice is not known or when there is no reason for assuming that the choice follows any specific probability distribution. For example in a vending machine it is reasonable to assign probabilities to the items to be sold to make assumptions about how likely one item is sold within a certain period of time. This requires a statistical experiment to determine the probability for each item. If such an experiment is not or can not be done the transitions can be modeled as non-deterministic choices. A more common example of the use of non-determinism in system is when several components in a system are executed concurrently. Such systems have to use programs which determine the component which is allowed to make the next execution step. These programs are called schedulers. Schedulers resolve the concurrency of the components by an interleaving according to the algorithm of the scheduler.

QuantUM currently does not support non-deterministic successor selections. Hence, it is necessary to introduce these kinds of choices in order to enable the analysis of concurrent software and systems. In this chapter we will show how non-determinism can be introduced into DTMCs and CTMCs. To compute probabilities on these non-deterministic models it is necessary to define new probability measures. We will show that it is no longer possible to calculate a

unique probability, but the minimum and maximum probability of satisfying a property.

As for the Markov Chains there are two kinds of models. First the Markov Decision Process (MDP) [37] which consists of discrete time steps and probabilities like DTMCs. The other model is called Continuous Time Markov Decision Process (CTMDP) [37]. CTMDPs introduce the notion of continuous time reasoning into the model, like CTMCs discussed in the previous chapter.

3.1 Markov Decision Processes

Definition 3.1 Markov Decision Process (MDP)
A Markov decision process is a tuple $M = (S, Act, \mathbf{P}, \iota_{init}, AP, L)$ *where*

- S *is a countable finite set of states*
- Act *is a set of actions*
- $\mathbf{P} : S \times Act \times S \to [0, 1]$ *is a transition probability function such that for all states* $s \in S$ *and actions* $\alpha \in Act$*, the sum of all outgoing probability is 1:*
$$\sum_{s' \in S} \mathbf{P}(s, \alpha, s') \in \{0, 1\}$$
- $\iota_{init} : S \to [0, 1]$ *is the initial distribution which sums up to 1.*
- AP *is a set of atomic propositions and* $L : S \to 2^{AP}$ *is a labeling function.*

■

The definition for the MDP [11] differs only little from the definition of the DTMC. All parts behave the same except for the transition relation. The intuitive operational behavior of an MDP can be described as follows. According to the initial distribution a starting state is chosen. In every state of the system a non-deterministic choice between the different enabled actions is performed. After an action is taken there is a probabilistic choice between all outgoing transitions with this action. If there is only one outgoing transition with this action this transition is taken with a probability equal to one. An example

of an MDP is depicted in Figure 3.1. The automaton consists of three states $\{t, s, u\}$, three actions $\{\alpha, \beta, \gamma\}$ and one initial state s. In the initial state s a non-deterministic selection between the two actions α and β takes place. If action α is taken the MDP changes into state t. In the other possibility action β is taken and a probabilistic selection of the successor state has to be done. There are two possible outcomes. The first choice is stay in state s with 50% probability and the other choice is to go into state u also with probability 50%. This example illustrates how non-determinism is modelled in Markov Decision Processes.

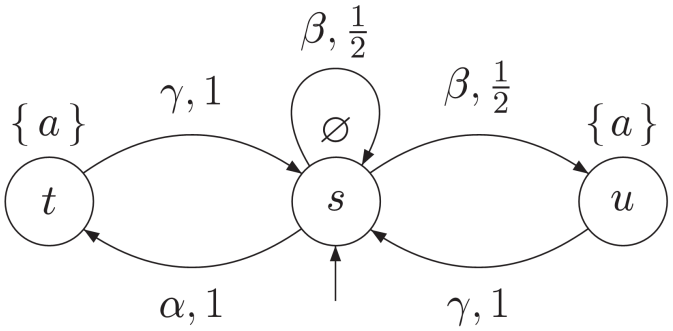

Figure 3.1: An example MDP from [11]

In DTMCs the probability of an execution path can be intuitively calculated by multiplying all probabilities along the path. In simplified terms one can sum up the probabilities of all paths ending in an end state to get the probability of getting into this end state eventually. In MDPs the calculation of the probability to get into a specific state can not be calculated in such a way. For example in the MDP in Figure 3.1 we want to calculate the probability of getting into a state labeled with a. These states are t and u. This question cannot be answered, since the probability depends on whether action α or action β is chosen. How the action is selected is not specified in the model. What can be calculated, however, is the maximum and minimum probability of getting into a state labeled with a. The maximum probability is 1, if action α is chosen because the probability is 1. The minimum probability is 0 if action β is always taken and the self loop is taken infinitely many times. This outcome is in fact very unlikely but it is the minimum probability.

The example shows that MDPs do not have a unique probability measure. We have seen that reasoning about probabilities of paths of an MDP relies on the resolution of the non-determinism. This resolution is done by a scheduler.

Definition 3.2 Scheduler
Let $M = (S, Act, \mathbf{P}, \iota_{init}, AP, L)$ be a Markov decision process. A scheduler $\mathfrak{S} : S^+ \to Act$ is a function that maps a path in an MDP to an action.

$$\mathfrak{S}(s_0, s_1, \ldots, s_n) \in Act(s_n), \qquad s_0 s_1 \ldots s_n \in S^+$$

■

The definition of a scheduler gives a function which calculates the action which is taken in a given state s_n based on the path $s_0 \ldots s_{n-1}$ leading to s_n. By resolving of the non-determinism the scheduler induces a Discrete-Time-Markov Chain. More precisely, the behavior of an MDP M under the decisions of scheduler $\mathfrak{S}$ can be formalized by a DTMC $M_{\mathfrak{S}}$. Note that the induced Markov Chain $M_{\mathfrak{S}}$ is infinite, even if M is finite. This is due to the fact that the scheduler unfolds the Markov decision process into a tree or forest if the initial distribution is given over more than one state. Every path in the tree corresponds to an infinite path in the DTMC with a defined sequence of actions according to the strategy of the scheduler.

Definition 3.2 gives the general description of a scheduler. We are interested in the maximum probability of a system to fail. Hence, it is sufficient to consider only memoryless schedulers. Memoryless schedulers are a subclass of schedulers which are independent of the history or path leading into a state. In [11] it was proven that there exists an optimal memoryless scheduler maximizing the probability of reaching a state.

Definition 3.3 Memoryless Scheduler
Let $M = (S, Act, \mathbf{P}, \iota_{init}, AP, L)$ be a Markov decision process. A scheduler $\mathfrak{S}$ is called memoryless or simple iff for each sequence $s_0 s_1 \ldots s_n$ and $t_0 t_1 \ldots t_m \in S^+$ with $s_n = t_m$:

$$\mathfrak{S}(s_0, s_1, \ldots, s_n) = \mathfrak{S}(t_0, t_1, \ldots, t_m)$$

$\Longrightarrow \mathfrak{S}$ *is a function* $\mathfrak{S} : S \to Act$ ■

The definition states that a scheduler $\mathfrak{S}$ is memoryless if it always selects the same action in a given state $s \in S$. This choice is independent from the path which led to s. To compute the probability of such a path in the system a probability model containing a σ-algebra has to be defined.

Definition 3.4 σ-Algebra
A σ-algebra is a tuple $\{\Omega, \mathcal{F}\}$ where Ω is a set of possible outcomes and $\mathcal{F} \subseteq 2^{\Omega}$ is the set of possible events which satisfy the following constraints:

1. $\Omega \in \mathcal{F}$*: All events are possible.*
2. $A \in \mathcal{F} \Rightarrow (\Omega - A) \in \mathcal{F}$*: The complement of all sets of possible events are also possible.*
3. $(\forall i \geq 0. A_i \in F)$*: The countable union of all sets of possible events are also possible.*

■

The pair $\{\Omega, \mathcal{F}\}$ is called a measurable space. With the definition of a σ-algebra it is possible to introduce the notion of a probability model. The model is also called probability space. In this space there is a probability assigned to each event to occur.

Definition 3.5 Probability Model
A probability model is a triple $\{\Omega, \mathcal{F}, Pr\}$ with,

- $\{\Omega, \mathcal{F}\}$ *is is a σ-algebra*
- $Pr : \mathcal{F} \to [0, 1]$ *a probability function where*
  1. $Pr(\Omega) = 1$*, with Ω being the certain event and*
  2. $Pr\left(\bigcup_{i \in I} A_i\right) = \sum_{i \in I} Pr(A_i)$ *for any $A_i \in \mathcal{F}$ with $A_i \cap A_j = \emptyset$ for $i \neq j$. $\{A_i\}_{i \in I}$ is finite or countably infinite.*

  The elements in $\mathcal{F}$ of a probability space $\{\Omega, \mathcal{F}, Pr\}$ are called measurable events.

■

In order to apply the definition of a probability model to an MDP the set Ω and the probability function Pr have to be defined. For Ω we choose on an infinite path:

$$\Omega = S \times Act \times S \times Act \times \ldots = \{S \times Act\}^{\infty}$$

A sample path $\omega \in \Omega$ is defined as follows:

$$\omega = (s_1, a_1, s_2, a_2, \ldots)$$

The probability function is constructed by introducing two random variables $X_t(\omega) = s_t$ and $Y_t(\omega) = a_t$, for $t = 1, 2, \ldots$. This means that X_t and Y_t denote the state and the action at step t respectively. We define a history process Z_t by

$$Z_1(\omega) = s_1 \text{ and } Z_t(w) = (s_1, a_1, s_2, a_2, \ldots, s_t)$$

In software and systems there is only one initial state, which is the defined entry point of the system. This means that the initial probability for s_1 is $P_1(s_1) = 1$ for one $s_1 \in I \subseteq S$. For an arbitrary scheduler π with the selection sequence $\pi = (d_1, d_2, \ldots)$ the probability can be calculated by the following formula:

$$Pr^{\pi}\{X_1 = s\} = P_1(s) \tag{3.1}$$

$$Pr^{\pi}\{Y_t = a | Z_t = h_t\} = q_{d_t(h_t)}(a) \tag{3.2}$$

$$Pr^{\pi}\{X_{t+1} = s | Z_t = (h_{t-1}, a_{t-1}, s_t), Y_t = a_t\} = \mathbf{P}(s_t, a_t, s) \tag{3.3}$$

The three formulae state that the probability for the variable X_t to be s in the first step of the path is equal to the initial distribution of s (Formula 3.1). The second formula (3.2) denotes the probability of selecting action a based on the history h_t. The last formula (3.3) describes the probability of getting

into state s when action a_t is selected under a history Z_t. A sample path probability can then be calculated by the following equation:

$$P^{\pi}(\omega) \quad = \quad P_1(s_1) \cdot q_{d_1(s_1)}(a_1) \cdot \mathbf{P}(s_1, a_1, s_2) \cdot q_{d_2(s_2)}(a_2) \cdot \ldots \tag{3.4}$$

Since memoryless schedulers suffice to calculate the maximum probability of a path we can omit the probability of the decision making $q_{d_t(h_t)}(a)$. In this scenario the probability of taking action a_t in step t is always 1. Therefore, the sample path probability reduces to:

$$P^{\pi}(\omega) = P_1(s_1) \cdot \mathbf{P}(s_1, a_1, s_2) \cdot \mathbf{P}(s_2, a_2, s_3) \ldots$$

3.2 Continuous-Time MDP

Discrete-Time-Markov Decision Processes are like DTMCs discrete time abstract models. When continuous-time aspects are important for a systems safety, a continuous time model has to be considered. For the deterministic case this is the Continuous-Time-Markov Chain as described in the previous chapter. Introducing non-determinism yields a new model called Continuous-Time-Markov decision process (CTMDP) [36, 37]. A CTMDP combines non-determinism and continuous time reasoning into one model.

Definition 3.6 Continuous-Time-Markov Decision Process
A CTMDP is a tuple $M = (S, Act, \mathbf{R}, \iota_{init}, AP, L)$ *where*

- S *is a countable finite set of states*
- Act *is a set of actions*
- $\mathbf{R} : S \times Act \times S \to \mathbb{R}_{\geq 0}$ *is a three dimensional transition rate matrix.*
- $\iota_{init} : S \to [0, 1]$ *is the initial distribution which sums up to 1.*
- AP *is a set of atomic propositions and* $L : S \to 2^{AP}$ *is a labeling function.*

■

The non-deterministic choice is done over the set of possible actions enabled in a state. If there are multiple transitions with the same action, a race condition occurs. This race condition is solved like in the CTMC [37]. The rates are again interpreted over a negative exponential distribution for which an example was given in Figure 2.3.

The syntactic definition of CTMDPs is known for long time but a single common semantic interpretation has not been given so far. Such a semantic definition is important to interpret different logics over the model and furthermore to come up with a model checking algorithm. We will now present a semantic interpretation of CTMDPs based on [37].

3.2.1 Semantic Interpretation of CTMDPs

Let S denote the finite state space of a CTMDP M. In our case we assume that there is only one initial state so $\iota_{init}(s_0) = 1$ for one state $s_0 \in S$. This definition can be easily extended to multiple initial states. In the systems we want to analyse the initial state is always given as the entry point of the system. Therefore, only one initial state is considered. In every state the scheduler has to take one action. The selection of the actions are called decision epochs and the time of residing in a state is called sojourn time. In CTMDPs decision epochs can occur at random points in time determined by the model description. If the system is in a state $s \in S$ the scheduler must choose an action α from the set of actions $A_s \subseteq Act$ which are enabled in state s. As a consequence of choosing action $\alpha \in A_s$, the next decision epoch occurs at or before time t with the maximum probability

$$Pr(\lambda, t) = 1 - e^{-\lambda \cdot t}, \quad \lambda = max\{\lambda_i | \lambda_i = \mathbf{R}(s, \alpha, j), j \in S\}$$

according to the highest rate λ of all outgoing transitions in s labeled with action α. This means the probability of the next decision epoch depends on the rates assigned to all transitions where action α is enabled.

A run of a CTMDP is denoted as an infinite sequence

$$h = (t_0, s_0, \alpha_0, t_1, s_1, \alpha_1, \ldots)$$

where the t_i denotes the time when the system occupies state s_i. After time t_i, action α_i is selected by the scheduler. A finite prefix of h is called history:

$$h_n = (t_0, s_0, \alpha_0, t_1, s_1, \alpha_1, \ldots, t_n, s_n)$$

This means after n decision epochs the system resides in state s_n. In contrast to the discrete-time models the history contains also the sojourn times $t_0, \ldots$. These times result from the rates which are attached to each transition. In fact there are infinitely many possible runs. The probability of each run varies depending on the rates attached to the transitions and the time of residing in the states along the run. Since the time is continuous there are infinitely many possible time intervals t_i in one state. If all t_i are fixed to some constant t for all i, the runs can be interpreted as runs from a discrete-time model. Hence, Discrete-Time-Markov Decision Processes are a special case of the continuous model. We will use this relationship between MDPs and CTMDPS later to discretize the CTMDP [37].

3.2.2 Probability model for CTMDPs

In this section the definition of a probability space introduced in 3.5 is applied to an CTMDP M. Therefore the set Ω and the probability function Pr have to be defined. The essential difference to the discrete probability measures applicable to MDPs or DTMCs is that the description includes a continuum T representing time. For the next step it is assumed that the number of states S and the number of actions Act in M are finite and $T = [0, \infty)$. The possible outcomes Ω are defined as follows:

$$\Omega = T \times S \times Act \times T \times S \times Act \times \ldots = \{T \times S \times Act\}^{\infty}$$

A path $\omega \in \Omega$ may be represented by

$$\omega = (t_0, s_0, a_0, t_1, s_1, a_1, \ldots)$$

where $t_0 = 0$ and $s_i \in S, a_i \in Act$ and t_i T for all i. We define random variables X_n, Y_n and τ_n which take values in S, Act and T respectively by

$$X_n(\omega) = s_n, \qquad Y_n(\omega) = a_n, \qquad \tau_n(\omega) = t_n$$

This means that the random variable X_n takes the value s_n at the nth position in path ω. A history process Z_n is defined by

$$Z_0 = (t_0, s_0) \text{ and } Z_n(\omega) = (t_0, s_0, a_0, \ldots, t_n, s_n)$$

With the defined variables and functions it is now possible to set up the probability measure Pr on the CTMDP: For each path ω in the model M under the scheduling policy $\mathfrak{S}$ the probability measure $P^{\mathfrak{S}}$ on the σ-algebra $\{\Omega, \mathcal{F}\}$ is defined as follows.

Definition 3.7 Probability measure $P^{\mathfrak{S}}$ on CTMDP M
Let $P_0(s)$ denote the initial distribution and $\pi = (d_1, d_2, \ldots)$ a decision sequence of decisions done by the scheduler. Then define conditional probabilities by

$$P^{\mathfrak{S}}\{X_0 = s\} = P_0(s) \qquad \textit{and} \qquad P^{\mathfrak{S}}\{T_0 = 0\} = 1$$

and for $i \geq 0$

$$P^{\mathfrak{S}}\{Y_i = a | Z_n = h_n\} = q_{d_{n+1}(h_n)}(a)$$

$$P^{\mathfrak{S}}\{X_{n+1} = j, \tau_{n+1} \in (t, t+\delta t) | Z_n = h_n, Y_n = a_n\} = 1 - e^{-\lambda \cdot \delta t}$$

where $\lambda = \mathbf{R}(s_n, a_n, s_{n+1})$ is the rate for changing the state from s_n to s_{n+1} under action a_n within time δt. $q_{d_{n+1}(h_n)}(a)$ denotes the probability of the scheduler choosing action a when the system is in state s_n with a history h_n. ■

The probability of a simple path ω under the decisions π equals

$$P^{\pi}(\omega) = P_0(s_0) \cdot q_{d_1(h_0)}(a_0) \cdot \left(1 - e^{-\lambda_1 \cdot \delta t_1}\right) \cdot q_{d_2(h_1)}(a_1) \cdot \left(1 - e^{-\lambda_2 \cdot \delta t_2}\right) \ldots$$

For deterministic schedulers (Definition 3.3) this simplifies to

$$P^{\pi}(\omega) = P_0(s_0) \cdot \left(1 - e^{-\lambda_1 \cdot \delta t_1}\right) \cdot \left(1 - e^{-\lambda_2 \cdot \delta t_2}\right) \cdot \ldots$$

Example

In Figure 3.2 a simple example for a CTMDP is depicted. There are two states $S = \{s_0, s_1\}$, one action $Act = \{\alpha\}$ and two rates $\mathbf{R}(s_0, \alpha, s_1) = \lambda_1$,

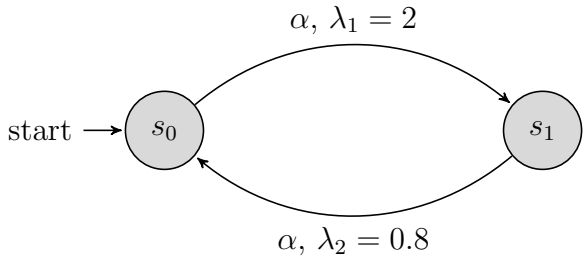

Figure 3.2: An example CTMDP M with one action α and two transition rates λ_1, λ_2.

$\mathbf{R}(s_1, \alpha, s_0) = \lambda_2$ defined in the model. The initial state is s_0 with a probability $P_0(s_0) = 1$.

As discussed above there are infinitely many runs in this model, since the time is a continuous variable. Possible runs are, for example:

$$\begin{aligned} h^a &= (t_0, s_0, \alpha, 1\,sec, s_1, \alpha, 2\,sec, s_0, \alpha, \ldots) \\ h^b &= (t_0, s_0, \alpha, 0.5\,sec, s_1, \alpha, 5\,sec, s_0, \alpha, \ldots) \\ &\vdots \end{aligned}$$

where $t_0 = 0$. In this example the scheduler is deterministic. Even if it were not deterministic, there is only one action which can be taken. Thus in every state action α will be selected by the scheduler. The calculation of the probability for a finite history h_n of the runs above can be done using the formula from Definition 3.7.

$$\begin{aligned} Pr^\pi(h_2^a) &= P_0(s_0) \cdot \left(1 - e^{-\lambda_1 \cdot \delta t_1}\right) \cdot \left(1 - e^{-\lambda_2 \cdot \delta t_2}\right) \\ &= 1 \cdot \left(1 - e^{-2 \cdot 1\,sec}\right) \cdot \left(1 - e^{-0.8 \cdot 2\,sec}\right) \\ &= 0.69 \end{aligned}$$

This example calculates the probability for the path h^a restricted to two steps of the system h_2^a. The probability is 69% when the system resides 1

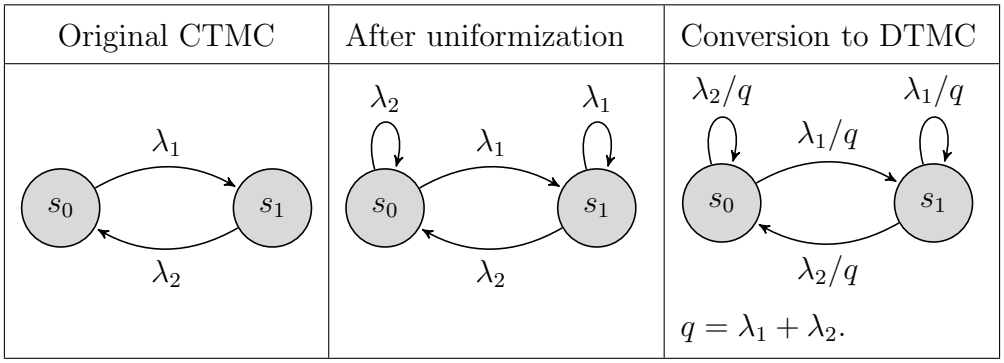

Figure 3.3: This example illustrates the uniformization of a CTMC in three steps.

second in the first and 2 seconds in the second state. The same calculation applied to the second run h_2^b results in a probability of 62%. Here the system is 0.5 seconds in the first and 5 seconds in the second state. This means that the first run h^a is more probable than the second one h^b.

3.2.3 Model Checking CTMDPs

In the recent past several approaches to formulate model checking algorithms on different subsets of the CTMDPs were presented.

In [16] Brázdil et al. presented a method to compute the time-bounded reachability in arbitrary CTMDPs. They showed that the algorithm has an exponential complexity in terms of the largest rate of the CTMDP, the time bound and the precision of the analysis itself. This approach was implemented in the MRMC model checking tool [26] and was tested on a case study in which a number of tasks with exponentially distributed processing time have to be scheduled on two identical processors. The property that was analyzed was the maximum probability that all tasks are completed within a certain deadline. In the test set 4 tasks were considered yielding only 24 states in the model. The results showed that memory and time consumptions are growing exponentially and thus are not efficiently applicable to real world scenarios.

In 2004, Baier et al. proposed a technique to efficiently compute time-bounded reachability probabilities in uniform CTMDPs [10] which was extended to locally uniform CTMDPs in [36] by Neuhäußer and Zhang. Uniformization is a well known technique in model checking of Continuous-Time-Markov Chains. The CTMCs are uniformized and reduced to DTMCs to com-

pute probabilities efficiently. In Figure 3.3 an example for the uniformization process is depicted. In the first picture the original CTMC is given. The second image shows the uniformed CTMC, while in the last picture the corresponding discrete model is given. While in CTMCs the transient probabilities after the uniformization are the same, this process can not be applied to arbitrary CTMDPs, since, the minimum and maximum probabilities could be changed [10]. Therefore, uniformization can not be applied to our systems. The transition rates can not be guaranteed to be globally uniform. Nevertheless, the systems we want to analyze are locally uniform CTMDPs by construction. Therefore we will apply a technique described in [36] to discretize the locally uniform CTMDP generated from the UML or SysML model. The resulting discrete time MDP will then be checked using the probabilistic model checker PRISM.

4. Discretization of Continuous-Time Markov Decision Processes

The negative exponential distributed rates used in Continuous-Time Markov Decision Processes and CTMCs are an industry standard to model failure rates of components in a system. Since we want to analyze real world software and systems it is necessary to support the use of these standardized failure rates. Therefore, the use of CTMDPs as the non-deterministic extension to CTMCs is necessary in order to support continuous-time reasoning in the models. While properties on Discrete-Time Markov Decision Processes can be computed efficiently, the methods to do such computations on Continuous-Time Markov Decision Processes are inefficient.

In this Chapter a conversion from CTMDPs into discrete MDPs is presented so that an efficient computation of time-bounded reachability properties is possible.

4.1 Differentiation of Uniformization and Discretization

The uniformization of a model and the discretization of it are similar processes. Nevertheless, the processes gain different results. In the discretization the transition rates are transformed into probabilities using a time step. In

the process of model checking CTMCs uniformization is used to convert the continuous model into an equivalent DTMC by scaling the rates according to a maximum rate in the system. During this process the transient probabilities are preserved [25].

Definition 4.1 Uniformization [25]
For a CTMC C, the uniformized DTMC is given by

$$unif(C) = (S, \mathbf{P}, AP, L)$$

where

$$\mathbf{P} = \iota_{init} + \frac{\mathbf{Q}}{q}, \qquad \mathbf{Q} = \mathbf{R} - diag(\mathbf{R}), \quad q \geq max\{\lambda(s) | s \in S\}$$

■

The uniformization rate q is determined by the state with the shortest residing time, i.e. the largest transition rate. All delays are normalized with respect to q. The result can be transformed into an DTMC where each state s with a rate $\lambda(s) < q$ is equipped with a self loop representing the probability of residing in the state. This probability is computed with the following formula:

$$Pr((s,s)) = 1 - \frac{\lambda_s}{q}$$

The transient probabilities in the DTMC $unif(C)$ resulting from the uniformization of a CTMC are equivalent to the transient probabilities in the original CTMC. An example of the uniformization of CTMCs is given in Figure 3.3.

According to [10] the uniformization process cannot be applied to Continuous-Time Markov Decision Processes to gain an equivalent discrete MDP. Within the non-deterministic models the process would change the extremal probabilities. To solve this problem we now introduce the discretization of a continuous time model.

4.2 Discretization of Continuous Time models

In the previous chapter we have introduced a semantic interpretation and a probability model for arbitrary CTMDPs. The discretization technique de-

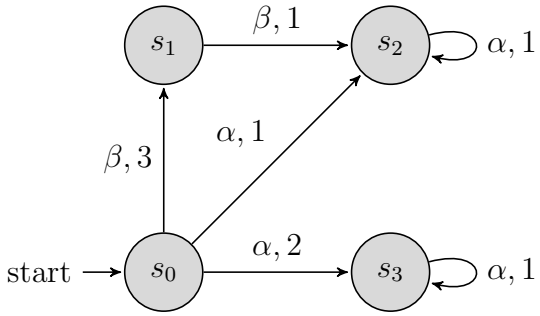

Figure 4.1: Example for a local uniform CTMDP. [36]

scribed in [36] is only applicable to locally uniform CTMDPs. Locally uniform CTMDPs are a subset of arbitrary CTMDPs. The definition states that a CTMDP is locally uniform if the sum of all rates for each action that can be taken in a state s is equal.

Definition 4.2 Locally uniform CTMDP
A CTMDP $M = (S, Act, \mathbf{R}, \iota_{init}, AP, L)$ *is locally uniform iff* $\forall s, s' \in S$ *and* $\forall \alpha, \beta \in Act(s)$. $\mathbf{R}(s, \alpha, s') = \mathbf{R}(s, \beta, s')$. ■

In Figure 4.1 a locally uniform CTMDP is depicted. Note that in state s_0 there are two possible actions α and β and their rates sum up to 3. We will now use the definition of the probability measure of CTMDPs to reduce the probability measure of an locally uniform CTMDP to a probability measure of an MDP.

Let $h_n^{DT} = (s_1, a_1, \ldots, s_n)$ and $h_n^{CT} = (t_0, s_1, a_1, t_1, \ldots, t_n, s_n)$ be paths in the history of a discrete time MDP and a locally uniform CTMDP, respectively. From [37] we know that by by setting all $t_i \in h_n^{CT}$ to a constant t_c an equivalence on the paths in the discrete and continuous case can be generated. In [36] the definition for the discretization of a locally uniform CTMDP was formalized.

Definition 4.3 Discretization of an CTMDP
Let $t_c \in \mathbb{R}_{>0}$ *be a constant time,* $M = (S, Act, \mathbf{R}, \iota_{init}, AP, L)$ *a CTMDP. Then* $M_{t_c} = (S, Act, \mathbf{P}_{t_c}, \iota_{init}, AP, L)$ *denotes the discretized Markov decision process with*

$$\mathbf{P}_{t_c}(s,a,s') = \begin{cases} (1-e^{\lambda(s)\cdot t_c})\cdot \mathbf{P}(s,a,s'), & \text{if } s \neq s' \\ (1-e^{\lambda(s)\cdot t_c})\cdot \mathbf{P}(s,a,s') + e^{\lambda(s)\cdot t_c}, & \text{if } s = s' \end{cases} \tag{4.1}$$

where $\mathbf{P}(s,a,s')$ *denotes the time abstract probability to move from state* s *to state* s'

$$\mathbf{P}(s,a,s') = \frac{\mathbf{R}(s,a,s')}{\lambda(s,a)}, \quad \lambda(s,a) = \sum_{s' \in S} \mathbf{R}(s,a,s')$$

Further, for all $a \notin Act(s)$ *we define* $\mathbf{P}_{t_c}(s,a,s') = 0$. ■

With this discretized definition of the probability model for CTMDPs the probability of a simple path ω under the decisions $\pi = (d_1, d_2, \ldots)$ equals

$$P^{\pi}(\omega) = P_0(s_0)\cdot q_{d_1(h_0)}(a_1)\cdot \overbrace{\left(1-e^{-\lambda_1\cdot t_c}\right)}^{\mathbf{P}_{t_c}((s_0,a_1,s_1)}\cdot q_{d_2(h_1)}(a_2)\cdot \overbrace{\left(1-e^{-\lambda_2\cdot t_c}\right)}^{\mathbf{P}_{t_c}((s_1,a_2,s_2)}\ldots$$

Since the time is constant in every state, time can be abstracted away by calculating the probability directly in every step of the system. This is done according to the exponential distribution by the first formula in Equation 4.1. This equation yields the following probability equation for a simple path ω

$$P^{\pi}(\omega) = P_0(s_0)\cdot q_{d_1(s_0)}(a_1)\cdot \mathbf{P}_{t_c}(s_0,a_1,s_1)\cdot q_{d_2(h_2)}(a_2)\cdot \mathbf{P}_{t_c}(s_1,a_2,s_2)\ldots$$

which is equivalent to Formula 3.4 for the probability of a path in an MDP defined in the previous chapter.

4.2.1 Stationary Probabilities

So far only transient probabilities were considered. Transient probabilities are the probabilities of taking a transition within a certain time interval and with a defined rate. In continuous settings there exists another kind of probability which does not exist in the discrete settings. This probability is called stationary probability [5] and denotes the probability of a system residing in a state within a certain time interval and after taking a certain action. In other

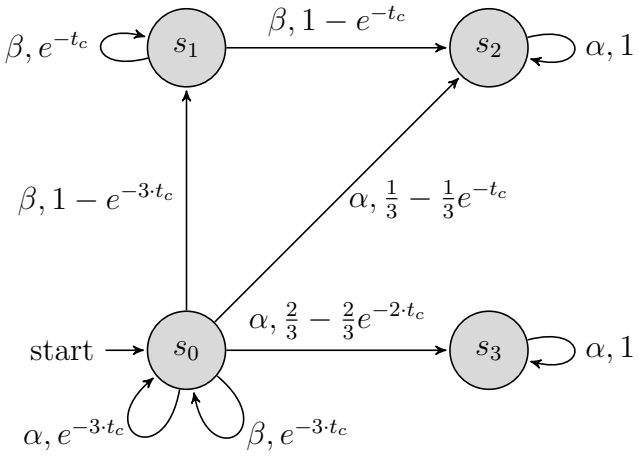

Figure 4.2: Example for a discretized local uniform CTMDP. [36]

words, the probability of not taking the transition within the time interval. This probability is calculated with the second formula in Eq. 4.1.

In discretized locally uniform CTMDPs, the stationary probability cannot be encoded directly into the existing transitions and states. For this purpose a new transition has to be introduced. For every transient probability calculated from a rate a new fictitious self-loop is generated with the stationary probability attached to it. In Figure 4.2 the discretized example from Figure 4.1 is depicted. In this example it becomes clear that stationary probabilities depend on all rates of the outgoing transitions under a selected action a. In state s_0 there are two actions labeled with α. This means after selecting action α in state s_0 a stochastic race between the rate 1 and 2 takes place. This race is resolved by weighting the discretized probabilities according to the distribution of the rates in the original CTMDP using the second case in Eq. 4.1.

In the UML/SysML models of the real world systems we are analyzing, multiple stochastic transitions per action do not occur, since, the specifications of UML and SysML do not cover such behavior. Thus, in UML and SysML as well as in our QuantUM profile there is no scenario where a decision between different successor states has to be made after an action is selected non-deterministically or probabilistically. This means in our case that only one transition with one rate for each action has to be considered. For the discretization formula for $\mathbf{P}_{t_c}(s, a, s')$ this means that the $\mathbf{P}(s, a, s')$, in Definition 4.3, reduces to a characteristic function:

$$\mathbf{P}(s,a,s') = \begin{cases} 1, & \text{if } \mathbf{R}(s,a,s') > 0 \\ 0, & \text{if } \mathbf{R}(s,a,s') = 0 \end{cases}$$

This characteristic function simplifies the conversion from a locally uniform CTMDP to a discrete MDP, since in the discretization algorithm each transition can be processed independently from all others. This means that the transient probability and the stationary probability of each transition in the discrete model can be calculated without taking other transition rates into account.

4.2.2 Granularity of the Discretization

One disadvantage of the discretization process is that the calculation of the probabilities on the model is not possible down to an arbitrary granularity anymore. In the original model it was possible to calculate the probability of reaching a set of states within time $t \in \mathbb{R}_{>0}$. In the discretized model the time interval depends on the chosen time step t_c. An example is depicted in Figure 4.3. The displayed CTMDP is also equal to a CTMC, since there are no real non-deterministic selections between successor states. For example we want to calculate the probability of getting into state s_3 within one hour. The probability is 25.258% according to the *MRMC* model checker. To calculate this probability in a discretized CTMDP the choice of the step size is crucial. In Figure 4.4 the discretized CTMDP is presented with a step size $t_c = 10ms$. With this step size of $10ms$ it is necessary to calculate 360.000 steps in the discretized CTMDP to get paths with a runtime of one hour. The probability of getting into state s_3 within 360.000 steps is also 25.258%.

The two probabilities have mostly the same value, even though, there is an error introduced through the discretization. This error can be calculated according to [36] using the following formula.

$$\varepsilon = \frac{(\lambda \cdot t_c \cdot k)^2}{2 \cdot k} \tag{4.2}$$

where k is the number of iterations necessary to get the desired runtime of the system. In the example above $k = 360.000$. t_c is the discretization time step

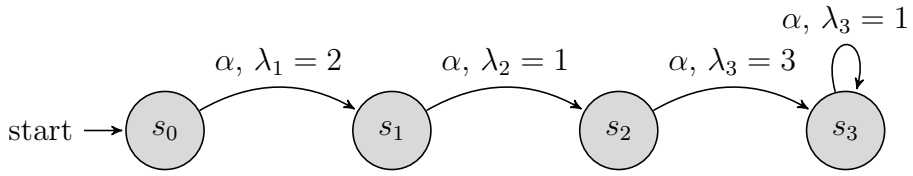

Figure 4.3: Example for a CTMDP representing a chain of 4 nodes with different rates.

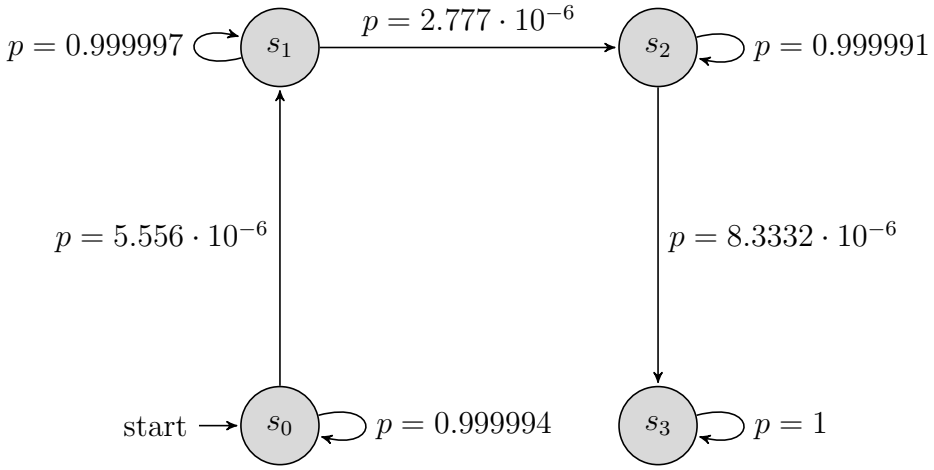

Figure 4.4: The discretized CTMDP from Figure 4.3 with probabilities p.

and λ is the maximum rate in the system, which is $\lambda = max_{s \in S} \lambda(s) = 3$ in the example in Figure 4.3. Calculating the error yields a value of $\varepsilon = 1.25 \cdot 10^{-5}$. With this formula it is possible to calculate the error introduced into the model. Furthermore this formula can be taken as a guidance to a well chosen time step t_c.

In the following chapter the implementation of the discretization process and the newly introduced QuantUM profile changes are discussed. Furthermore the local uniformity of the automatically generated model is explained in detail.

5. QuantUM Extensions

The systems we want to analyze are specified at the levels of UML or SyML models. To facilitate the analysis at this stage of development a quantitative extension to the UML and SysML models was introduced in [31] and [30], respectively. With this extension it is possible to annotate the model with quantitative information such as probabilities or rates. Furthermore, it is possible to specify undesired behavior of the system by introducing failure patterns in terms of new diagrams directly into the model. The annotated model is then exported into an intermediate format called XMI[2]. Afterwards it is automatically translated into the PRISM language which is used to check the behavior of the systems. The results of the checking process are subsequently presented in the well known format of a Fault tree [38]. The approach to automatically generate such Fault trees from probabilistic counterexamples [7] is presented in [27]. The tool chain of QuantUM is depicted in Figure 5.1.

In this chapter, we present an extension to the QuantUM approach as proposed in [31]. Furthermore, we show that the translated models are locally uniform by construction.

5.1 QuantUM

The QuantUM profile is a description of the information required to analyze a UML or SysML model. The application of the profile is carried out using stereotypes [1, 3] which are attached to the elements of the diagrams in the

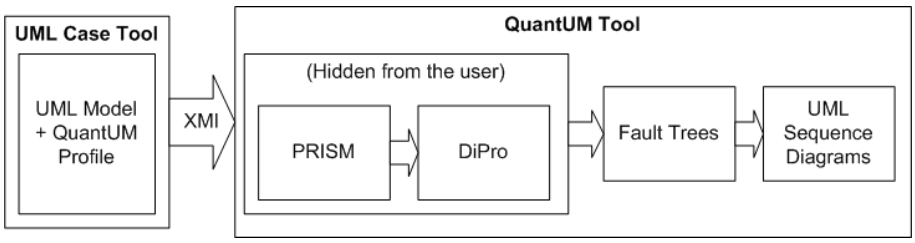

Figure 5.1: The QuantUM tool chain.[31]

model. A stereotype is a limited meta-class which can not be used as a stand-alone element. This means a stereotype has to be used in combination with the metaclasses it extends. Each stereotype extends one or more classes and thus can be used to add similar kinds of information to different classes. Any model element in the UML or SysML can be extended using a stereotype. For instance , states, transitions, activities, state machines or even a whole model description can be extended with a stereotype.

The advantage of such a profile is that it can be used within any UML/SysML case tool. The users can build models in the modeling framework of their choice and do not have to familiarize themselves with new tools. The second advantage is that the implementation of a profile does not interfere with the original UML/SysML specification of the model, hence, the original behavior of the modeled system is preserved.

We will now give a description of the parts of the QuantUM profile for SysML which we are using in our case studies. For further information to the stereotypes we refer to [31] and [30].

QSymComponent

The QSymComponent stereotype (Figure 5.2) can be applied to all SysML elements that represent building blocks of the real system, for example blocks in a block definition diagram[1]. The QSymComponent includes up to one state machine, representing the normal behavior of the component it is applied to. Additionally one up to finitely many failure patterns have to be added. A failure pattern describes the undesired behavior of the system in terms of another state machine. In some components it is sufficient to attach one failure pattern. In such a case, for example, a component only describes behavior of an external system. All state machines can be constructed directly for the

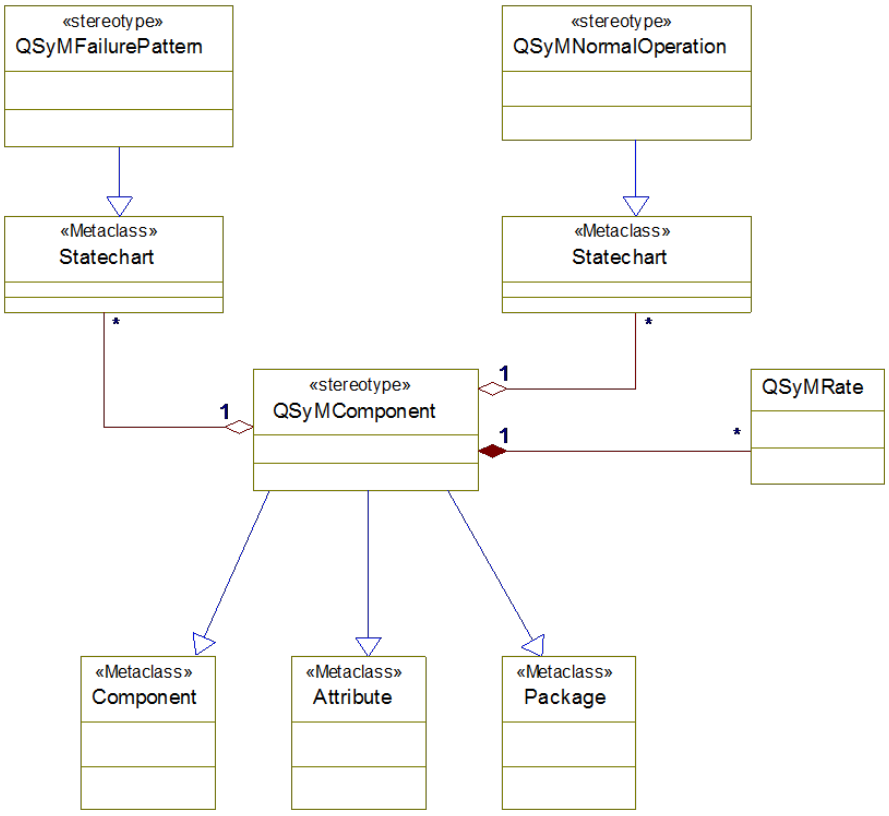

Figure 5.2: The QSymComponent stereotype with its dependencies.

QSymComponent or can be taken from a repository of state machines to avoid redundant modelling of the same behavior. Another tag in this stereotype denotes a list of rates. These rates are globally defined for the whole component and can be used in the transitions of the state machines associated with this component.

QSyMAbstractStochasticTransition

The normal behavior and the failure patterns of the system are modeled as state machines. These machines consist of states and transitions to describe the transient behavior of the component. To tag the transitions with quantitative information about the system several transition stereotypes were introduced. The dependencies between all transition stereotypes are shown in Figure 5.3. The QSyMAbstractStochasticTransition is used to indicate that a transition between two states is tagged with a rate from the list of rates defined in the QSyMComponent. In the QSyMStochasticTransition stereotype

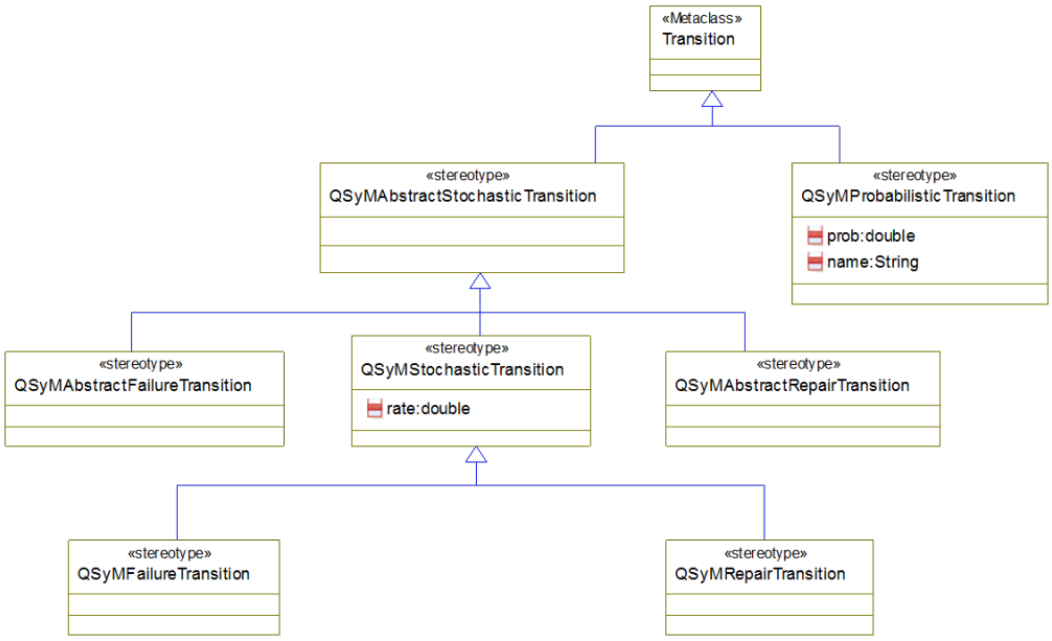

Figure 5.3: The transition stereotypes.

the rate for the corresponding transition can be set directly. The other abstract transitions are specializations of the QSyMAbstractStochasticTransition. A QSyMAbstractFailureTransition denotes a transition from the normal behavior to a failure pattern. This means failure transitions are only traversed if an error in the system occurs. The repair transitions are used to indicate a repair rate of failed components.

5.2 Extensions of QuantUM

In this section we will present the extensions to the QuantUM profile which are necessary to analyze concurrent systems.

QSyMProbabilisticTransition

The QSyMProbabilisticTransition stereotype is used to indicate that a transition has a certain probability to be executed. The probability is time abstracted. This stereotype is used in the failure pattern of a component. After a QSyMStochasticTransition is executed the QSyMProbabilisticTransition determines the next successor state based on the probability distribution. All outgoing QSyMProbabilisticTransitions with the same name have to sum up to one to

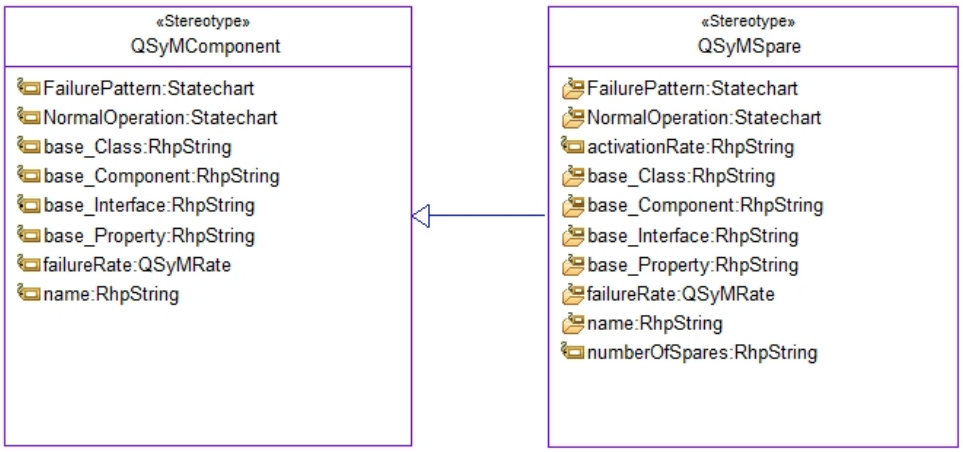

Figure 5.4: The changed QSymSpare stereotype.

generate a proper MDP. The stereotype relationship with the other QuantUM transition stereotypes is shown in Figure 5.3.

QSyMSpare

The QSyMSpare stereotype is used to indicate redundant system components. Redundant components are necessary in cases where the system availability is important. In [31] the QSyMSpare stereotype was applied to self-loop associations on components tagged with QSyMComponent to indicate that there are other components which are activated in case the first one fails. We have changed this stereotype to be an extension of the QSyMComponent stereotype. In Figure 5.4 the extension is depicted. This change can be used more intuitively and the failure rates and the number of spare components can be set directly in the component.

QSyMProject

The new QSyMProject stereotype is used to indicate which kind of analysis of the system is done. Currently there are two possible types of analysis. The first one is the pure stochastic analysis of the system using CTMCs. For details of this analysis method we refer to [31]. The second one is the approach introduced in this thesis to analyse concurrent systems with CTMDPs.

```
module samplemodule
    var1: bool init false;
    var2: [0..10] init 0;
    [Count] (var2 < 10) ->    0.6: (var2'= var2 + 1)
                            + 0.4: (var2'= var2 + 2);
    [End] (var2 >= 10) -> 1 : (var1'= true);
endmodule
```

Figure 5.5: An example for a PRISM module.

5.3 Semantics of the Extension

In [31] the semantics of the conversion from the QuantUM profile to the PRSIM input language was explained for the pure stochastic case of Continuous-Time-Markov Chains. In this section we will discuss the rules for the conversion of the UML/SysML models to an MDP. The PRISM input language is a state-based, guarded command language based on the formal description given in [8]. We use this input language in order to specify the Discrete-Time Markov Decision Processes which results from the discretization process. First, a short description of the basic elements in the PRISM language is presented.

A PRISM model consists of one or more modules which can interact with each other. A module includes the definition of variables and transitions. The values of the variables at any given time constitute the state of the module itself. The global state of the system is determined by the local state of all modules. The behavior of a module is encoded in commands. A command for a transition in an MDP is specified in the following way:

$$\begin{aligned} [\text{transition\_label}]\ \text{guard} \quad \rightarrow \quad & (\text{prob}_1 : \text{update}_1^1 \,\&\, \ldots \,\&\, \text{update}_i^1) \\ & + \ldots \\ & + (\text{prob}_n : \text{update}_1^n \,\&\, \ldots \,\&\, \text{update}_i^n) \end{aligned}$$

where prob_a denotes the probability of making the updates update_1^a to update_i^a. From Definition 3.1 of an MDP the following condition has to hold.

$$\sum_a \text{prob}_a = 1$$

Each update in the command corresponds to one transition in the module.

```
%module_id%_state: [0..%#normstate% + %#failstate%] init 0;
```

Figure 5.6: PRISM translation of the state encoding.[31]

An example MDP module is given in Figure 5.5. In the example a variable is increased until it reaches the limit of 10. In every counting step two possible results can be achieved. With a probability of 60% the variable `var2` is increased by 1 and with a probability of 40% it is increased by 2. If the evaluation of `var2` is less than 10 the increasing continues and If it is greater or equal to 10 `var1` will be set to true. Possible properties which can be checked on this module could be: How high is the probability of getting into a state where `var1` evaluates to true within 10 steps?

5.3.1 Translation Rules

In the following presentation of the translation rules for the UML/SysML to MDP conversion rules we will use the notation explained in [31]. Every element which is enclosed by a % character will be rewritten by the QuantUM tool. Statements in the << ... >>-notation are optional. Other notations are part of the PRSIM language [21]. We will only give the translation rules which have to be changed or are new in the non-deterministic analysis, for the remaining rules, such as the QSyMComponent, we refer to [31].

5.3.1.1 Translating State Machines

The state machines describing the normal behavior and the failure pattern are combined into one hierarchical state machine. A simplified result is shown in Figure 5.7. For the translation we assume that transitions without a QSyMStochasticTransition stereotype have infinite rates. This means, that such transitions are executed instantaneously in the CTMDP. For the discretization this yields a probability of 1 for every possible time step according to Equation 4.1.

The states of the UML/SysML state machine are numbered and encoded into integer variables. This encoding in the PRISM code is shown in Figure 5.6. States representing the normal behavior are always assigned a value between 0 and the total number of states representing the normal behavior (`#normstate`). Failure states are represented by numbers greater than `#normstate`. If there

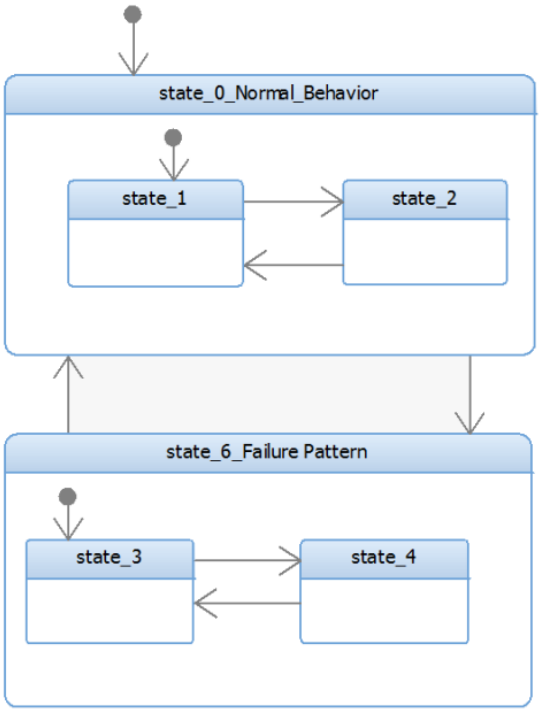

Figure 5.7: Combination of the normal behavior and the failure pattern

are hierarchical state machines the parent states are represented by the interval of integers representing the children in the underlying state machine. The placeholders `#normstate` and `#failstate` denote the number of states in the normal behavior state machine and the failure pattern, respectively. The variable `%module_id%_state` represents the status of the state machine. This means the state in which the machine is at any time. All transitions from the UML/SysML model are translated into PRISM commands according to the rule depicted in Figure 5.8. A transition is enabled and is taken with a probability `%prob_2%` if the following conditions are fulfilled.

- The state machine is in a state where the transition is an outgoing transition or in a sub-state of a hierarchical state machine of this state. The conditions can be represented by the expression:

$$\begin{aligned} &((\text{state\_id\_parent} = \text{transition\_source\_id}) \quad \wedge \\ &(\text{state\_id\_parent} \leq \text{module\_id\_state} \leq \text{state\_id\_substate}_n)) \quad \vee \\ &(\text{module\_id\_state} = \text{transition\_source\_id}) \end{aligned}$$

```
[%module_id%_%transition_name%]
    ((%module_id%_state >= %state_id_parent%)
        & (%module_id%_state <= %state_id_substate_{n}%))
    | (%module_id%_state = %transition_source_id%))
    << & (%guard%) >> ->
      %prob_1%: ( %module_id%_state'= %module_id%_state)
    + %prob_2%: ( %module_id%_state'= %transition_target_id%)
        << & (%actions%) >> << & (%events_fired%) >>;
```

Figure 5.8: PRISM translation of the transitions in the system.

where module_id_state denotes the current active state, state_id_parent represents the id of the parent state p, state_id_substate$_n$ indicates the sub-state of p that has the highest id and transition_source_id denotes the id of the state where the transitions starts.

- The event causing the execution of the transition has been fired
- The transition guard evaluates to true.

The probability `%prob_1%` is only considered when failure transitions are translated. For the correctness of this approach we will restrain transitions with rates to be failure transitions. This assumption is valid since in real systems such rates are only known for the errors occurring in components and they are not given for the normal behavior. The probabilities `%prob_1%` and `%prob_2%` are calculated according to Eq. 4.1.

The other translation rules are not changed or are changed similar to the presented ones, e.g. the `QSyMSpare` stereotype. Therefore, we refer to [31] for a detailed description of the remaining elements.

5.3.2 Local Uniformity by Construction

In the remainder of this chapter we will show that the translation from UML/SysML yields locally uniform CTMDPs. There are three constraints that have to be fulfilled for the translation:

1. Only failure transitions have rates attached to it.
2. Every transition without a rate is assumed to have a rate which has an infinite value.

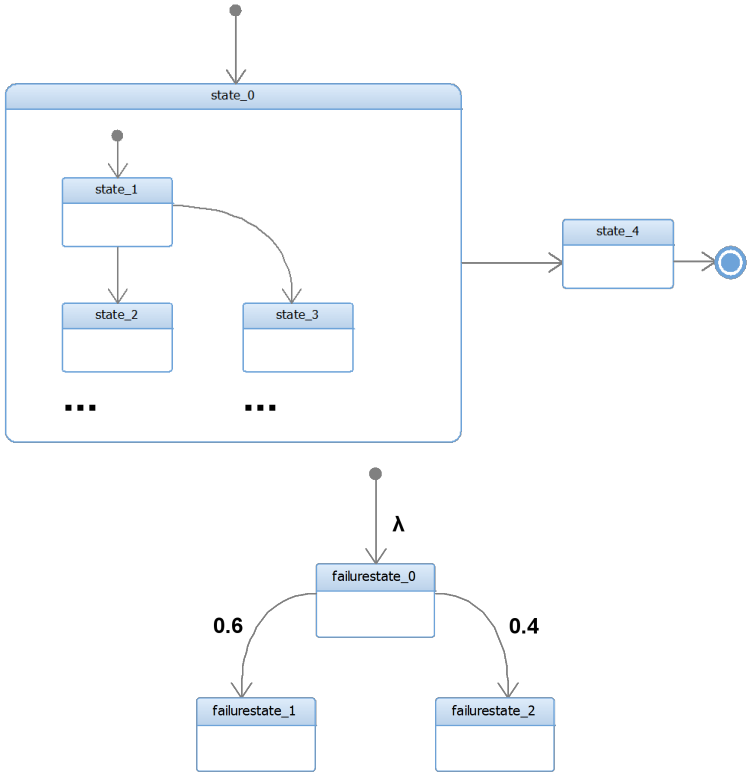

Figure 5.9: An arbitrary state machine (above) and a simple failure pattern for the state machine (below).

3. Each failure state can have transitions into sub-failure states which indicate the probability of getting into the sub-failure state with the given failure rate. The probabilities sum up to one.

An arbitrary UML/SysML model satisfying these constraints is depicted in Fig. 5.9. λ denotes the rate of the component to fail. The CTMDP corresponding to the state machine is presented in Figure 5.10. In this CTMDP the two state machines representing the normal and the failure behavior of the system are already combined. This is indicated by the λ-transitions leading into the failure state f0 from every state. All other transitions in the CTMDP have infinite rates, according to the second constraint above. This constraint ensures that every transition with an infinite rate is executed instantaneously and, hence, does not add any time to the whole execution sequence of the CTMDP. As a consequence these transitions do not add probability to the reachability of the failure state. Hence, all possible runs within the normal behavior do not add any probability to the maximum reachability probabil-

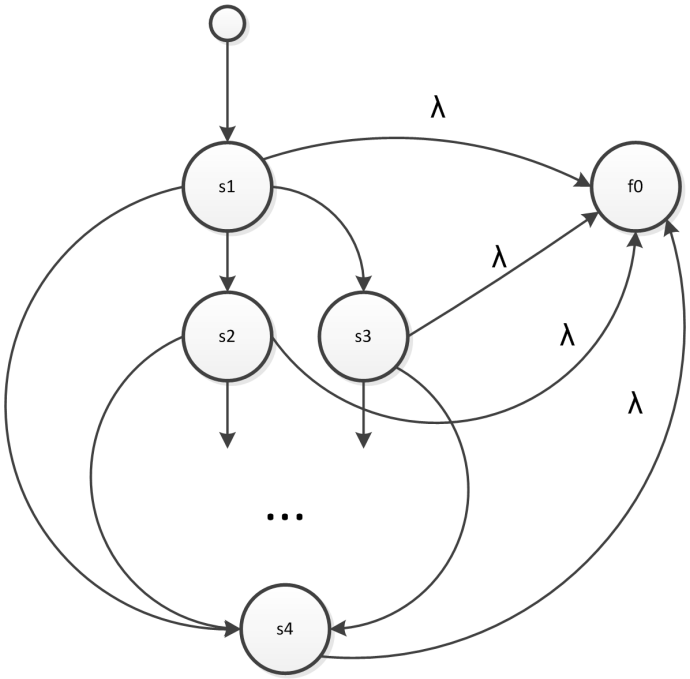

Figure 5.10: The CTMDP resulting from the state machine in Figure 5.9.

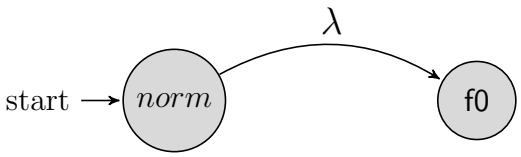

Figure 5.11: The collapsed representation of the normal behavior of the CTMDP from Fig. 5.10.

ity. Collapsing the normal behavior into one single state does not change the probability of reaching a failure state either and yields a equivalent CTMDP in terms of reachability probability. The result of this collapsing process is shown in Fig. 5.11. From Chapter 4 we know that the depicted CTMDP is locally uniform and thus can be transformed into an discrete MDP. The discretized version of the CTMDP from Fig. 5.10 is depicted in Figure 5.12. In the discretized version the sub-failure states f1 and f2 are considered as well. The transitions leading into the sub-failure states are directly tagged with probabilities. These transitions result from the QSyMProbabilisticTransition stereotype and are used to indicate the probabilities of getting into a sub-failure state after a stochastic transition took place.

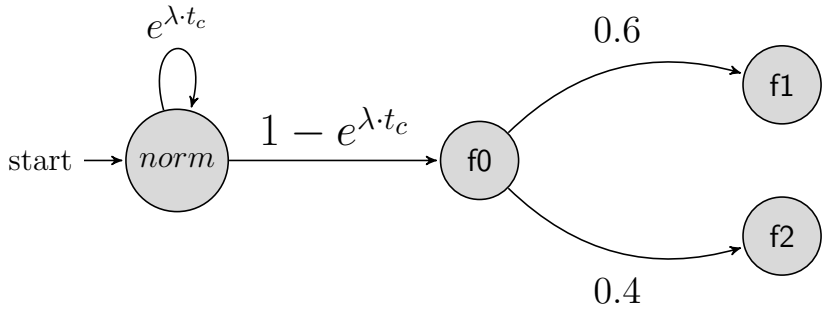

Figure 5.12: The collapsed representation of the discerte MDP from Fig. 5.11.

In the discrete MDP all transitions in the *norm* state collection have the transition probability one. This results from the infinite rate which is assumed for every transition. The collapsing of the *norm* states does not affect the probabilities in the discretized CTMDP as well. From the definition of the probability calculation on paths in CTMDPs and MDPs we know that every transition with probability equal to one does not affect the overall probability (Def. 3.5 and 3.7). Regarding the introduced error (Equation 4.2) this means that only the rate of the failure transition has to be taken into account. As a consequence we know that the CTMDPs we are using as an intermediate step for the translation from UML/SysML to discrete MDPs are locally uniform by construction. This justifies the use of the discretization technique described in Chapter 4.

5.4 Automatic Property Generation

In the non-deterministic extension of QuantUM is possible to automatically generate PCTL properties. Probabilistic computational tree logic [11] was discussed in the Chapter 2. There are two possibilities to generate the properties. The first one is to automatically generate a set of PCTL formulae out of the UML/SysML model. The second possibility is to specify state configurations in the model. The automatically generated formulae check for simple component failures by trying to get into a failure state within a given time bound. The state configurations allow the user to manually specify some states which should be visited or should not be visited within a certain step bound. Since we are using non-deterministic models the maximum probability of satisfying the formulae is computed as discussed in Chapter 3. For details of the automatic generation process we refer to [31].

6. Case Studies

We have applied the extended QuantUM approach for analysing concurrent systems to two case studies. The first one is an Airport Surveillance Radar (ASR) developed by Cassidian[1]. The analysis model used in this case study is a SysML model.

The second case study is an airbag control unit developed at TRW Automotive GmbH[2]. This case study is adapted from [31]. The analysis of this system is based on a UML model.

6.1 Airport Surveillance Radar

The Airport Surveillance Radar (ASR) is a system for monitoring the airspace in the vicinity of an airport. It is used by airtraffic controllers who guide aircrafts according to their flight plan. Each channel processes the data provided by the PSR and the SSR.

First the main functionality is described and some SysML diagrams are presented to show the interaction between the different components in the ASR. Afterwards, we will discuss the verification results of the analysis of the one- and the two-channel model.

[1]EADS Deutschland - Cassidian: http://www.cassidian.com/
[2]TRW Automotive GmbH: http://www.trw.de/

6.1.1 Functionality of the ASR

The ASR consists of three main components. The first component is the Primary Surveillance Radar (PSR) which uses radar signals that are reflected from the body of an object. In the normal case this objects are aircrafts that are to be tracked, in order to locate the object. The second component is the Secondary Surveillance Radar (SSR) which communicates with the transponder of the aircraft in order to obtain information about the aircraft. This information is, amongst others, the height and the identification number of the aircraft. The internal structure of the radar consists of two redundant channels. To explain the functionality of the ASR only one channel is considered. In a second step the extension of the system to a second channel is described. A channel of the ASR consists of five components:

- Receiver (not shown in the diagram)
- Signal Processor
- Parameter Extractor
- Sensor Tracker
- Validation Unit

The receiver collects the reflected radar signals from the environment. It forwards them to the signal processor, which is an analog-digital converter for the signals. The digitalized signal is analyzed inside the parameter extractor (PEX). In order to determine where plots have to be initialized. This is accomplished by discriminating the reflections of aircrafts from the reflections of the surrounding terrain (houses, trees, etc.) by means of what is referred to as a clutter map. After identifying the plots the data is sent to the sensor Tracker (ST). The sensor tracker combines the plots from the primary radar and the secondary radar and tries to discern possible tracks of aircrafts. If the ST finds a track it is added to the list of tracks that the system maintains. A track is observed until the aircraft leaves the range of the radar, or a failure happens. In the last step the validation unit (VU) checks whether the outcome of the sensor tracker is plausible or not. After performing these steps the calculated data is presented on the graphical display of the ASR system.

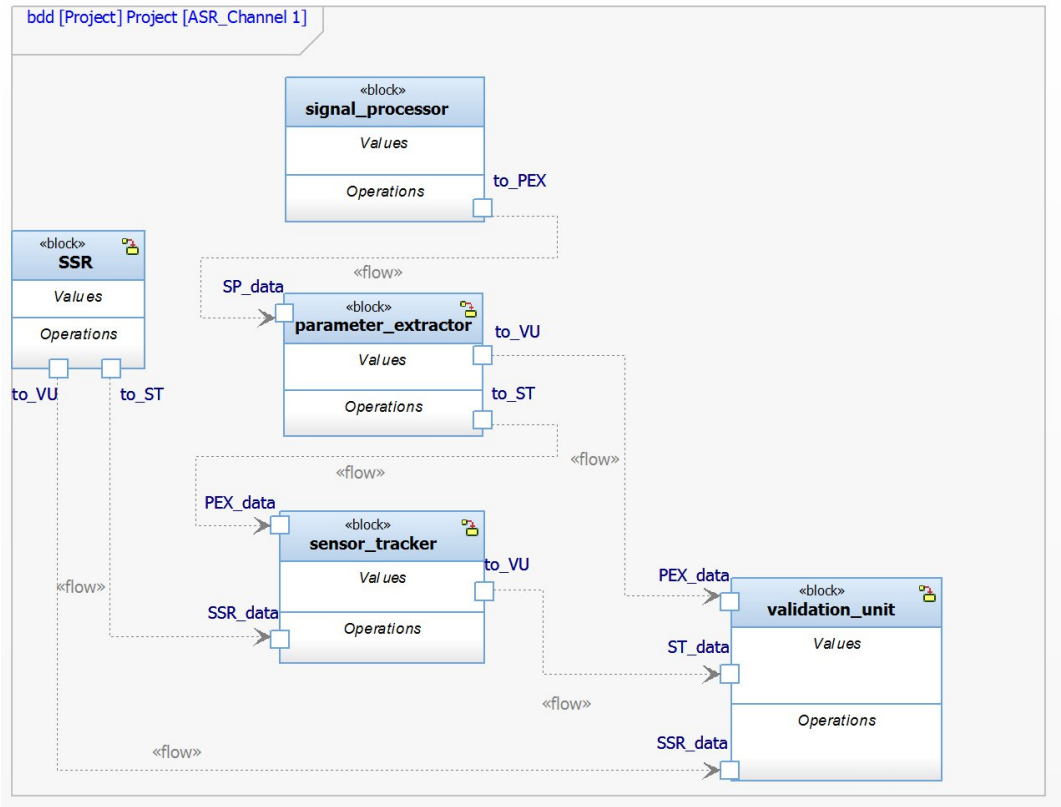

Figure 6.1: Overview block-definition-diagram of the ASR system.

In Figure 6.1 the overview block-definition-diagram [1] of one ASR channel is shown. The receiver and the signal processor are omitted from the analysis since we are focusing on the digital part of the system. A state machine containing the normal behavior of the components is defined for all other components in the block-definition-diagram. This is indicated by the yellow boxes in the upper right corner on each component in Figure 6.1. In addition to the components depicted in the window there is an environment component called `FlightObject`. It simulates the behavior of the system to be analyzed as well as the behavior of the environment in which it operates. In other words, the external functionality has to be defined within the model. We will now discuss the behavior of each component in detail.

6.1.1.1 Secondary Surveillance Radar

The first component is the SSR. The state machine is presented in Fig. 6.2. This component contains 3 states: `sending`, `waiting` and `init_plot`. Initially the state machine is in the `sending` state where the SSR sends a

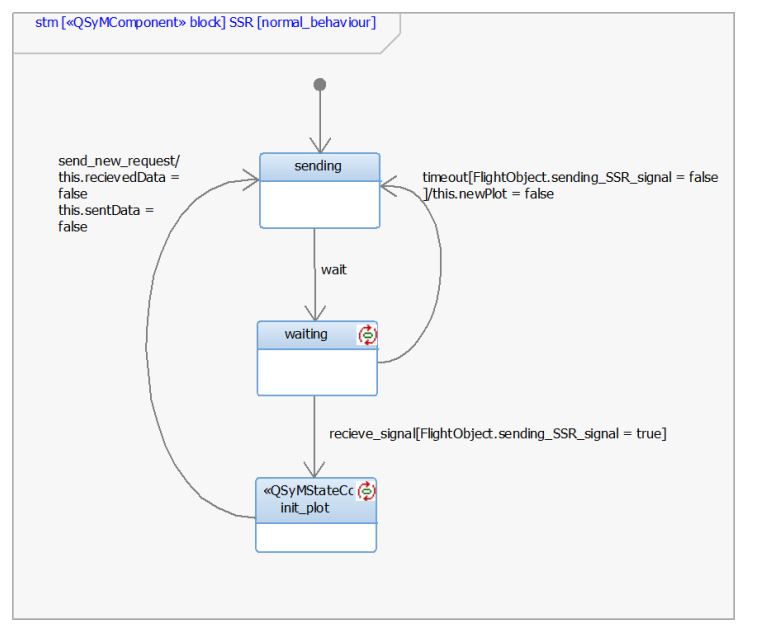

Figure 6.2: The state machine of the SSR

request to the environment. After sending the message the SSR transitions into the `waiting` state and waits for an answer from the transponder of an aircraft. If after a certain timeout period no answer is received, the message will be resent. Otherwise a message is returned from the aircraft and the SSR initializes a plot (`init_plot`). The new plot is then sent to the sensor tracker and the validation unit. The next cycle of the SSR starts with the sending of a new message to the environment.

6.1.1.2 Parameter Extractor

This component processes the already digitized signal reflections from the PSR. It contains three states: `dot_sighted`, `plot_recognized` and `clutter`. In the beginning, when a dot is sighted the parameter extractor (PEX)) is in the `dot_sighted` state. When there is a flight object which can reflect the radar signal the PEX changes into the `plot_recognized` state. The parameter extractor remains in this state as long as the Flight object is moving or generates a DOPPLER-effect, for example a hovering helicopter with rotating rotors. If these constraints are not fulfilled, the plot is categorized as clutter and is saved in the clutter-map so that it is not displayed on the next cycle.

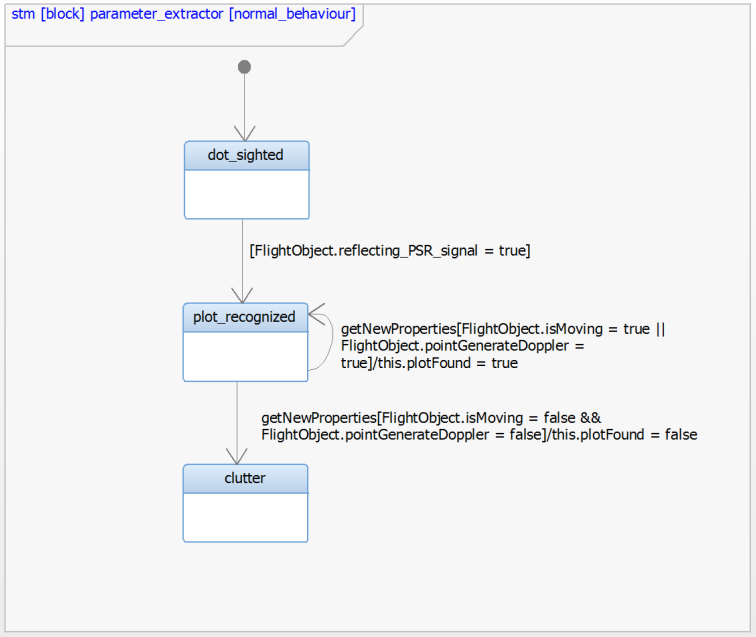

Figure 6.3: The state machine of the parameter extractor.

6.1.1.3 Sensor Tracker

The state machine of the sensor tracker is depicted in Figure A.1 in the appendix. In this component the digital and the analog plots are combined into tracks representing the flight paths of the tracked objects. The sensor tracker consists of 4 states: `first_plot`, `plotDetected`, `track` and `coasted-Track`. The states `first_plot`, `track` and `coastedTrack` contain additional hierarchical state machines where the track labeling is handled. Since all of these sub-state machines have more or less the same functionality, only one of them is given in Figure A.2. The Figure shows the sub- state machine of the state `track`.

In the beginning, the sensor tracker is in the `first_plot` state where an initial labeling of the plot is done. There are three different labels for a track, namely *combined*, *SSR* and *PSR* only. When the system is in the *SSR only* state then there is only one SSR-signal attached to the plot. The *PSR only* is defined accordingly and the *combined* label says that both signals are found. After labeling the first plot with one of the possible labels the sensor tracker is transiting into the `plotDetected` state, where two possible successor decisions can be made. If there is a *PSR only*-plot found, the sensor goes back to the `first_plot` and increments a counter. If this happened more than three

times, a new track tagged *PSR only* is initialized. The other possibility is that a *combined* or *SSR only* plot is found. In this case a new track is initialized immediately, since it is very unlikely that there is a SSR signal without an aircraft. This can only happen, if an SSR failure occurs and this failure is handled in the failure pattern of the SSR.

When a new track is found, new plots matching this track are added after each cycle of the system. If there is a cycle where no new plot is found, the track changes its labeling in a bad manner. For instance a labeling can change from *combined* to *SSR only* or *PSR only*. "Bad" in this case means that information about the track is lost in comparison to the last cycle.Then the track will be coasted. Coasting a track means that the possible position of the aircraft is extrapolated from recent data. The system tries to recover the track into its original labeling state through three more cycles. If this is not possible the track is closed, which means that the aircraft is no longer appearing on the screen. This is one of the worst scenarios which can occur in the system: The aircraft is still up in the air, but does not appear on the screen. When there is a new plot found within three more cycles, then the track is recovered to its original status before the coasting.

Another possibility is that the track changes its labeling in a positive way, for example, when the track was labeled with *PSR only* and in the next cycle the label is changed to *combined*. Of course, this is not a bad behavior and thus the track is continued under the new label.

6.1.1.4 Validation Unit

The last component in the channel is the validation unit which checks whether the output of the sensor tracker is plausible or not. The normal behavior state machine is therefore very simple. It contains a basic decision between the two states `OK` or `NOK`. The attribute which holds the value ot the plausibility is set in the failure pattern of the sensor tracker. The meaning of the failure patterns is explained in the following paragraph.

6.1.1.5 Failure Patterns

Failure patterns are used to determine whether a component has a failure or not. Since we later want to analyze the correct as well as the failure behavior of the system, both behaviors need to be modeled. In the model the

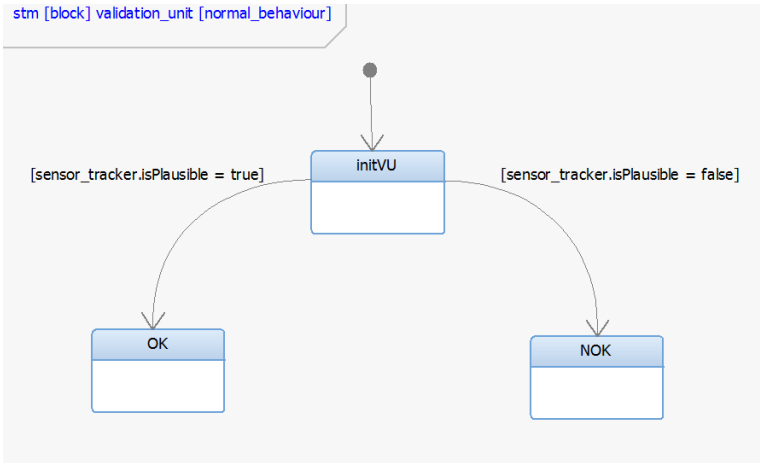

Figure 6.4: The state machine of the validation unit

failure patterns are attached to the components in the form of additional state machines using the QSyMComponent stereotype. An example failure pattern state machine for the SSR is shown in Figure 6.5.

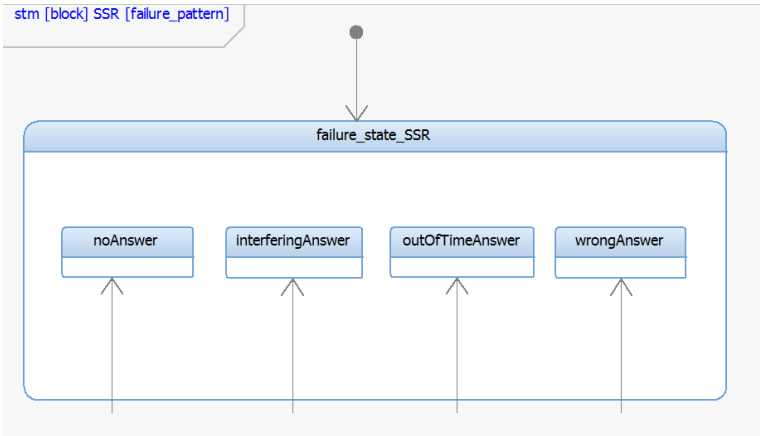

Figure 6.5: The Statechart for the failure pattern of the SSR

The SSR contains four possible failure modes:

1. **noAnswer:** This failure can occur when there is an aircraft, but it is not responding to any requests of the SSR.

2. **interferingAnswer:** In this case two or more aircrafts are simultaneously sending a response to the SSR so that the messages are interleaved. This is not handled in the implementation of the SSR protocol, because it is very unlikely that such an interference is lasting for more than one cycle.

3. **outOfTimeAnswer:** The message which is sent to the aircraft was not answered within one cycle.

4. **wrongAnswer:** The message was returned with wrong parameters.

6.1.2 Extension to a Second Channel

To extend the one-channel model to a second channel we have used the QSyMSpare stereotype. This stereotype replaces all QSyMComponent which have one spare components. In the ASR these are the following three components:

- Parameter extractor
- Sensor Tracker
- Validation unit

In this radar system the spare components are identical copies of the original components that run concurrently. In the QuantUM model we represent the switching from a component to its spare copy by incrementing a counter. The switching can only occur if the counter is less than the amount of spares available. In the ASR system the number of spares was set to 1.

6.1.3 Application of the QuantUM Profile

The first step in the preparation of the model is to identify the components which are safety critical. We disregard the signal processor since it is merely an analog-digital converter and hence considered irrelevant for the safety proof of the digital part of the system. All other components are crucial, since we are interested in the digital part of the system. All crucial components are tagged with the `QSyMComponent` tag so that QuantUM can convert them into individual modules for the model checking process. Figure 6.6 depicts a part of the QuantUM specification of the parameter extractor in IBM Rational Rhapsody[3], which we are using to model the system. Notice that the specification includes a normal behavior and a failure pattern. The two parameters set in the specification refer to the state machines defined as illustrated in the previous

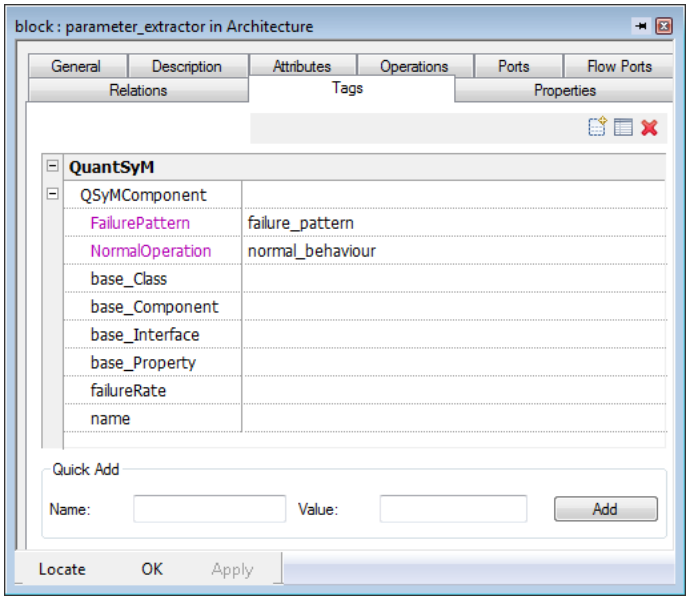

Figure 6.6: The stereotype `QSyMComponent` in IBM Rational Rhapsody.

section. The other tags of the stereotype are optional and filled automatically during the parsing step in QuantUM.

These are the basic steps necessary to obtain an output model from QuantUM that can be analyzed with the PRISM model checker. The transitions in the failure pattern are tagged with the QSyMFailureTransition stereotype to attach the rates to the component failures. For this survey we use 10^{-6} as a typical error rate of all components. For the PSR error rate we use a higher value of 0.01 since this component is less reliable.

Another stereotype we are using is the QSyMAttributeRange. This is added to the attribute using integer domains. Since the size of the domain of integer variables in programs are finite, but very large, we restrict the domain of the variables on the SysML model to small ranges. This helps to contain the state space size of the PRISM model.

6.1.4 Analysis Results

There are two properties we want to present in this thesis. The first one is a general SSR failure within 100 hours. The second failure is the possibility of getting a coasted track based on a track which is tagged *PSR only* within 100 hours. Both properties are generated automatically by QuantUM. The

[3]http://www.ibm.com/software/awdtools/rhapsody/

discretization step of the model is set to 1 hour per step. The step size can be chosen quite large, because according to Formula 4.2 the introduced error is low.

The PCTL formulae for the properties are automatically generated as follows.

$$P_{max=?}[(\text{true})\mathcal{U}^{<=T}(SSR\_failure)]$$

$$P_{max=?}[(\text{true})\mathcal{U}^{<=T}(coastedTrack)]$$

where $SSR\_failure$ denotes the states defined in the failure pattern of the SSR and $coastedTrack$ refers to the states tagged with the QSyMStateConfiguration for the coasted track property. The experiments where performed on an Intel QuadCore i7 processor with 3.3Ghz and 24GB of RAM.

6.1.4.1 Computing Probabilities for Safety Requirements

After exporting the one channel model to an XMI-file[2] it can be opened in QuantUM. We did two calculations: computing the probability of a given hazard or error and generating the fault tree for a given hazard or error. The resulting PRISM model for the one-channel system consists of approximately 2 million states and 21 million transitions. The direct computation of the SSR failure took 39.78 seconds with a memory consumption of 86.1 MB and results in a failure probability of $9.9995 \cdot 10^{-5}$. This probability is as expected the maximum failure probability of $1 \cdot 10^{-6}$ for the component. The error computed with Equation 4.2 is $5 \cdot 10^{-11}$. For the probability of getting a coasted track based on a *PSR only* track we have computed a probability of 0.5768 within the same time and memory. The error computed was $5 \cdot 10^{-3}$. By comparing, we can conclude that the high probability of getting a coasted track from an *PSR only* results from the unreliability of the PSR itself.

In a second step the one-channel model was extended to a two channel model using the QSyMSpare stereotype. This model consists of approx. 132 million states and 1.1 billion transitions. The memory consumption of the model checking process was 3.2 GB. The calculation of the SSR failure took 3148.72 seconds and the probability is $9.9995 \cdot 10^{-5}$. For the coasted track property the memory consumption was 3.2 GB as well and the calculation took 4073.58 seconds. The probability is 0.5725.

In advance we applied the QuantUM approach presented in [31] to the ASR system to get reference values for the non-deterministic approach. The probability for the SSR-failure was equal to the failure calculated with the new approach except for the calculated error stated above. This is due to the fact, that the SSR failure was a one-step general failure of a component which does not rely on the functionality of the system. This means the calculation for the general SSR failure can be done without taking the functionality of the ASR into account. However, the failure probability of getting a coasted track in the CTMC differs significantly from the non-deterministic approach. The calculated probability for the coasted track after 100 hours was 0.067. In the ASR model used to do the calculation all transitions where no QSyMFailureTransition was applied were assumed to have a rate of 1.0. This assumption leads to a probability which is lower than the real value. This is due to the fact that a transition with a rate of 1.0 has a probability of 0.632 per hour to be fired. Hence, a path in the system where a rate of 1.0 is assumed for all non-failure transitions becomes less probable than a path where higher rates are applied. We verified this assumption by redoing the experiment with higher rates of 100 for these transitions. The new computed probability for a coasted track was 0.108.

The other modification we did was the introduction of the QSyMProbabilisticTransition. In [31] all sub-failure states were modeled with additional failure rates resulting in a lower probability of the sub-failures. We replaced the rates from the general failure states to the sub-failure states by probabilities using the new stereotype. The result was a higher probability for the sub-failure states, since, the transitions into these states were assumed to be timeless. If, for example, a component transits into a failure state, it does not reside in the general failure state, but directly gets into a sub-failure state. The sub-failure state is chosen according to the time-abstract probability distribution of all probabilistic transitions. The experiment showed that the assumption of setting add rates to infinity and modelling transitions as non-deterministic choices as well as changing the sub-failure transitions to probabilistic transitions is modelling the system more adequately.

The direct computation yields the probability for an error to happen. However, it does not provide any insights in how or why the hazard or error hap-

pened. In order to gain insight in how the hazard happens, QuantUM can compute the fault tree [38] for the hazard we are interested in.

6.1.4.2 Automatic Fault Tree Generation

In our analysis we are using time bounded probabilistic reachability properties. These qualify as safety properties in the commonly used topological classification scheme of safety and liveness properties [11]. In order to automatically generate fault trees for those safety properties we use the approach proposed in [27]. The fault tree is computed from a full set of counterexamples, which contain all bad executions and all corresponding good executions of the system. Consequently, the fault tree generation algorithm presumes that the full state space of the model has been explored.

In [6] two approaches to explore the state space of an MDP were presented:

1. The first approach resolves the non-determinism of the MDP in advance to come up with a deterministic DTMC. On this DTMC the counterexample generation can be done in an on-the-fly manner. The creation of the deterministic DTMC with a probability maximizing scheduler (see Chapter 3) requires the full state space of the MDP to be explored as well. This exploration is again a very time and memory consuming process.

2. The second method works directly on the MDP. The non-determinism is resolved on-the-fly by using an And-Or-Tree [6]. This process is more time consuming than the first one and the memory consumption is only reduced linearly in the size of the MDP.

An experimental evaluation showed that for a model of the IPv4-protocol (Fig. 2.2) with only $n = 8$ tries for connection no counterexample could be generated within 4 hours using the second approach.

We applied the second approach for the counterexample generation for MDPs [6] to the ASR model. The computation ran out of memory after 4 hours without providing a counterexample. This memory issue was expected due to the experiments done in [6]. To address this problems some improvements to the methods will be discussed in future work.

6.2 Airbag Control Unit

The second case study is an Electronic Control Unit for an Airbag system (AECU). The system is developed at TRW Automotive GmbH[4]. The AECU can be divided into three main parts: sensors, crash evaluation and actuators. An impact is detected by acceleration sensor in the front, rear and the side of the car. Additionally there are pressure sensors in the side of the car to detect side impacts. Angular sensors are used to detect rollover accidents. A microcontroller is used to evaluate whether a situation is critical enough to trigger the deployment of the airbag or not.

6.2.1 Functionality of the AECU

In the architecture of this case study only two acceleration sensors are considered. Their task is to detect front and rear crashes. A microcontroller is used to evaluate the sensor information and an actuator controls the deployment of the airbag. The system consists of the following components:

- **The Field Effect Transistor (FET)** controls the power supply for the airbag squibs. If the FET is not in the high state, the squib has not enough power to ignite the airbag.
- **The Firing Application Specific Integrated Circuit (FASIC)** controls the airbag squib directly. Only if it receives an arm command followed by a fire command from the microcontroller, it will deploy the airbag.
- **The Microcontroller** is used to evaluate crash situations.
- **The MainSensor** is used to measure the acceleration of the car. This sensor is aligned to the driving direction.
- **The SafetySensor** is used to compare the results with the *MainSensor* to get a more reliable result. This sensor is aligned against the driving direction.

In Figure 6.7 the UML class diagram of the system is depicted. All the components in this diagram are tagged with the `QUMComponent` stereotype, which is the UML correspondence to the `QSyMComponent` stereotype explained above. For a detailed functional description of the components we refer to [31].

[4]TRW Automotive GmbH: http://www.trw.de/

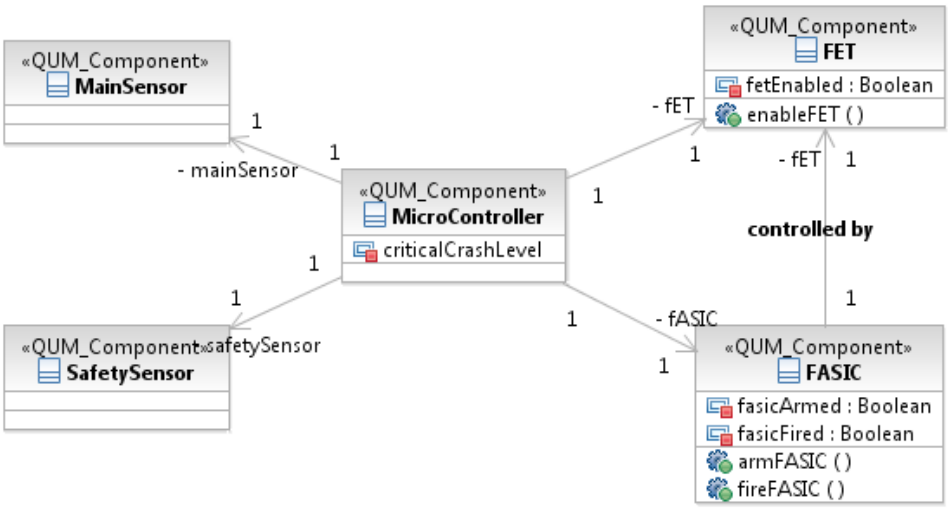

Figure 6.7: The class diagram for the AECU [31].

6.2.1.1 Failure Pattern

While airbags can save lives in critical crash situation, they may cause fatal injuries when deployed inadvertently. The driver could lose control of the car when such a deployment occurs. A main safety requirement is therefore that the airbag is only deployed in critical crash situations. The possible error situations are presented in the state machines for the different components: Microcontroller (Figure A.6), FASIC (A.7) and FET (A.8). For the microcontroller and the FET component we assume a failure rate of $3.28 \cdot 10^{-4}$ and for the FASIC component we assume a failure rate of $3.0 \cdot 10^{-3}$. All rates and probabilities used in the model are not real, since the real values are intellectual property of our industrial partner TRW Automotive GmbH that we are not allowed to publish.

6.2.2 Analysis Results

We want to check one property on the Airbag model. This property is the probability of an inadvertent deployment of the Airbag within 100 hours. For the discretization step size we chose 1 second.

The PCTL formula for this property is automatically generated from the QUMStateConfiguration:

$$P_{max=?}[(\text{true})\mathcal{U}^{<=T}(inadvertentDeployment)]$$

where *inadvertentDeployment* is replaced by the actual encoded state numbers in the PRISM model. The experiments where performed also on an Intel QuadCore i7 processor with 3.3Ghz and 24GB of RAM.

6.2.2.1 Computing Probabilities for Safety Requirements

After exporting the UML model of the Airbag Control Unit to an XMI-file it was converted into an MDP using QuantUM. The resulting PRISM model consists of 1461 states and 7389 transitions. The probability of the inadvertent deployment of the Airbag was 0.1297 and the time to compute the property was under one second. The memory consumption was 342KB. The approximation error computed with Equation 4.2 was $2.5 \cdot 10^{-9}$.

In [31] the computed probability for the inadvertent deployment was 0.032. As discussed above, this calculation assumes all transitions without a QUMFailureTransition to have a rate of 1.0. Redoing the experiment from [31] with higher rates of 100.0 for these transition resulted in a new probability of 0.04. The rates from the general failure states to its sub-failure states were also replaced by probabilities. The result was a higher probability for the sub-failures, since, the transitions into this states were now assumed to be timeless. The experiment showed again that the assumption of setting all rates to infinity and modelling transitions as non-deterministic choices as well as changing the sub-failure transitions to probabilistic transitions models the system more adequately.

6.2.2.2 Automatic Fault Tree Generation

Since the model of the Airbag Control Unit has a significantly smaller size than the ASR model, we tried to apply the automatic fault tree generation to it. As stated above the property we wanted to check is the inadvertent deployment of the Airbag. Due to the issues discussed in 6.1.4.2 the counterexample generation using DiPro [7] ran out of memory after 4 hours of processing. The partial result after the 4 hours contained only one counterexample. This counterexample was the shortest path of the system to an electrical short circuit in the *FASIC* component, which would lead to an immediate deployment of the Airbag. The probability of this counterexample was $1.43 \cdot 10^{-3}$.

6.3 Summary

The two case studies showed that it is possible to model and verify complex systems efficiently. The direct computation of the properties proceeds in a reasonable amount of time and memory consumption. We showed that non-deterministic QuantUM approach models concurrent system architectures more adequately. This was achieved by replacing high rates with non-deterministic choices. As we expected, the computation of the fault trees was not possible due to memory and timing issues. Therefore, new methods will be discussed in future work to address this problem.

7. Conclusion

In this thesis an extension of the QuantUM approach introduced in [31] is presented. The extension supports the verification of non-deterministic models which is necessary for the analysis of concurrent system architectures. We have discussed two forms of Markov models, the Discrete-Time and Continuous-Time Markov Chains. We showed how non-determinism can be introduced into these models resulting in Discrete-Time and Continuous-Time Markov Decision Processes (MDP and CTMDP). The negative exponential distributed rates used in CTMDPs are an industrial standard to indicate failure rates of components. Thus, the use of CTMDPs became necessary to correctly include these rates. A problem with CTMDPs is that there are currently no efficient model checking algorithms to analyze these models.In order to address this drawback we have used a discretization technique to convert the CTMDPs into a discrete MDP. This conversion process was then implemented into QuantUM so that an automatic translation of UML and SysML models into discretized CTMDPs became possible.

On two case studies the analysis showed that the direct calculation of the failure probabilities is efficiently possible. Furthermore, we showed that the non-deterministic approach can model systems where concurrent execution of components is important more adequately. The automatic Fault Tree generation failed, as we expected, due to time and memory consumption of the methods. This issues will be addressed in future work.

7.1 Future Work

We succeeded in computing probabilities for the safety requirements of the systems directly. The application of the automatic fault tree generation was hampered by the size of the models, the resolving of the non-determinism and the need for a full state space exploration. In future work we want to integrate functional model checking techniques, such as implemented in the explicit state model checker SPIN [22], to perform the fault tree generation. First experiments show that a functional model checking of the two-channel ASR model look quite promising. We are planing to integrate causality checking algorithms [32] with the on-the-fly algorithms used for functional model checking. Furthermore, we plan to derive heuristics automatically from the functional model checking to speed up the probabilistic model checking process in general.

Currently, QuantUM supports the analysis of UML and SysML models. We plan to extend the supported model types to MathWorks Simulink[1]. This tool is widely used in the automotive domain to model safety critical systems and analyze them using an integrated simulation engine. We are planning to determine how we can adapt the QuantUM profile to work on Simulink models.

We plan to analyze and assess different safety patterns often used in safety critical system development. The objective is to come up with adequate modelling techniques and translation rules to low level modelling languages such as the PRISM input language. Furthermore, we plan to derive patterns from this analysis to evaluate possible design alternatives of systems.

[1] http://www.mathworks.de/products/simulink/

A. Appendix

A.1 Abbreviations and Explanations

ASR Airport Surveillance Radar. This is the complete radar system.

SSR Secondary Surveillance Radar. This is the digital radar system which communicates with the transponders of the aircrafts.

PSR Primary Surveillance Radar. The classical radar system where the rays are emitted from the antenna and then are received by the receiver.

Clutter-Map A map which saves the environmental reflections (houses, trees, etc.) from the *PSR*.

SP Signal processor. The analog-digital converter which converts the high frequency radar signals from the *PSR* to digital values.

ST Sensor tracker. It tracks the aircraft's ways through the range of the radar

PEX Parameter extractor. Here the digitized data from the PSR is separated into aircrafts and environment.

VU Validation unit. It checks the plausibility of the Sensor Tracker.

XMI XML Metadata Interchange. It is a standard defined by the Object Management Group (OMG) to save and exchange UML-diagrams and metadata between different programs.

A.2 State machines of the ASR

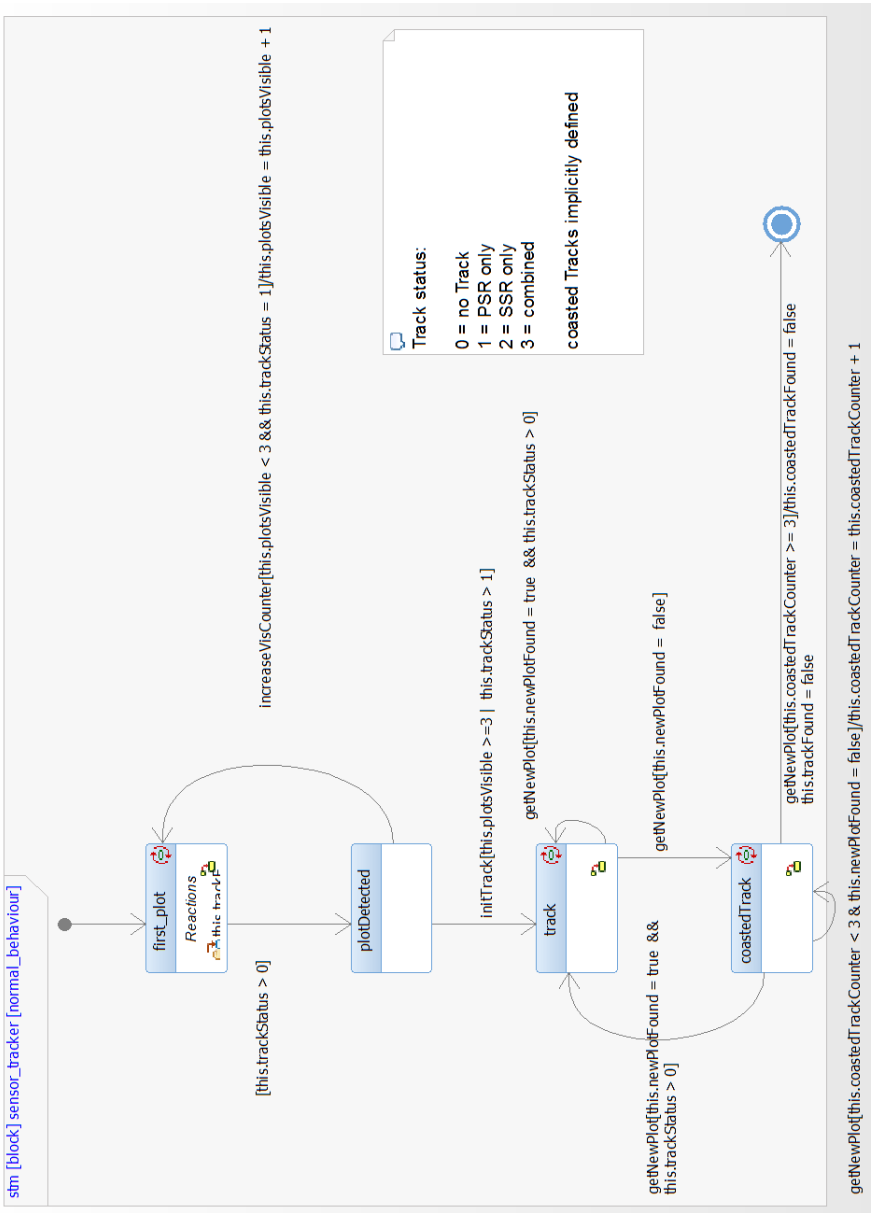

Figure A.1: The state machine of the sensor tracker

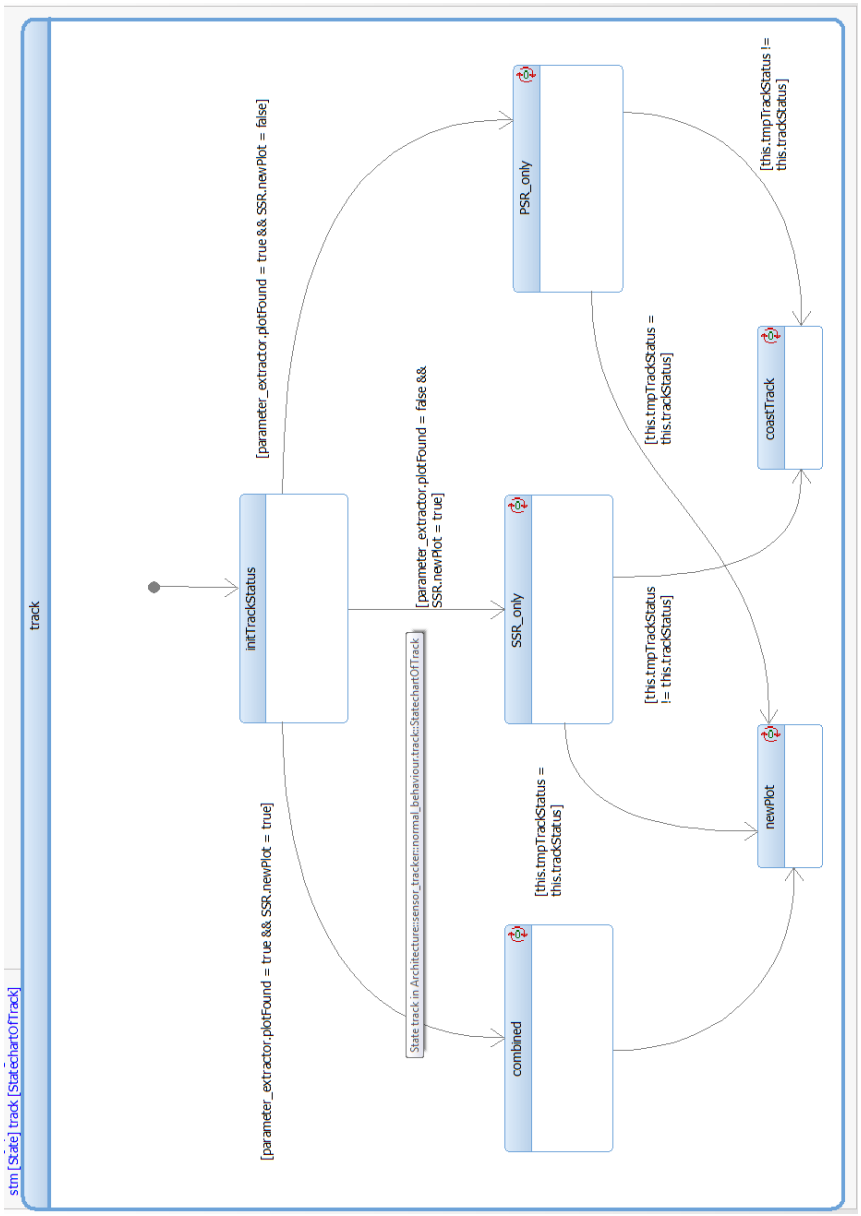

Figure A.2: The sub-state machine of the state track from the sensor tracker

A.3 Additional Failure Patterns of the ASR

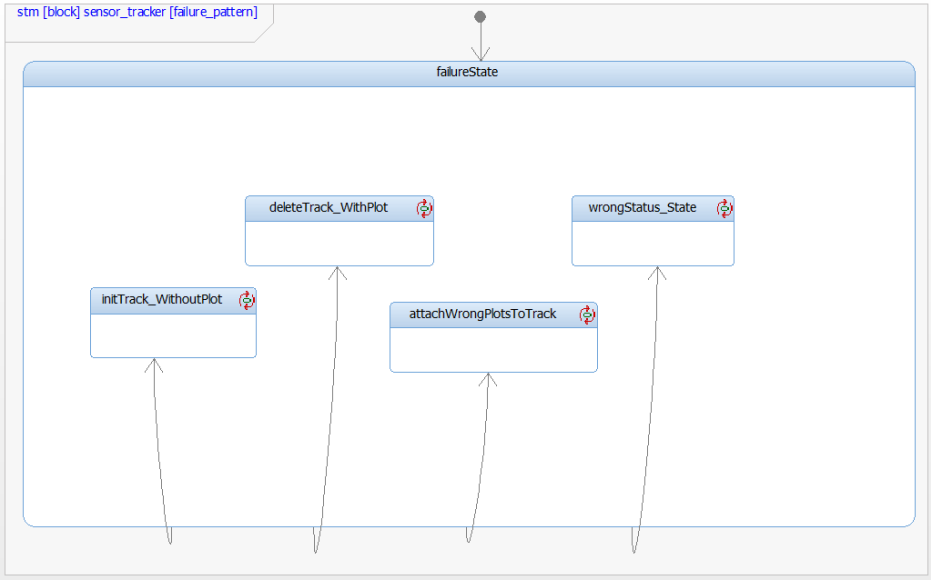

Figure A.3: The state machine for the failure pattern of the sensor tracker

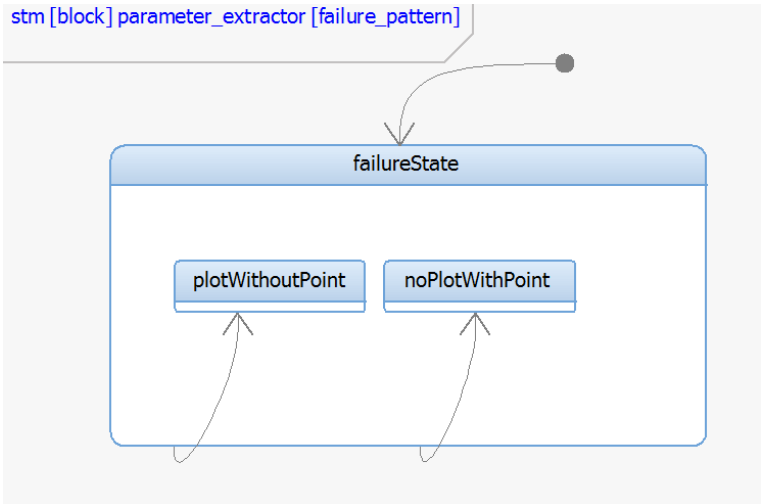

Figure A.4: The state machine for the failure pattern of the parameter extractor

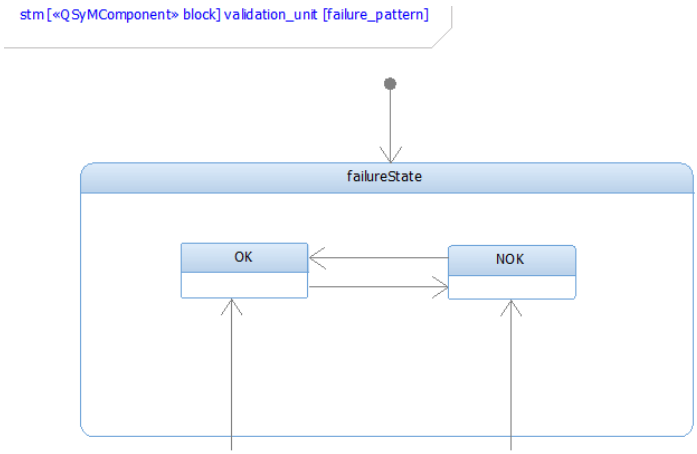

Figure A.5: The state machine for the failure pattern of the validation unit

A.4 Failure Patterns of the Airbag Control Unit

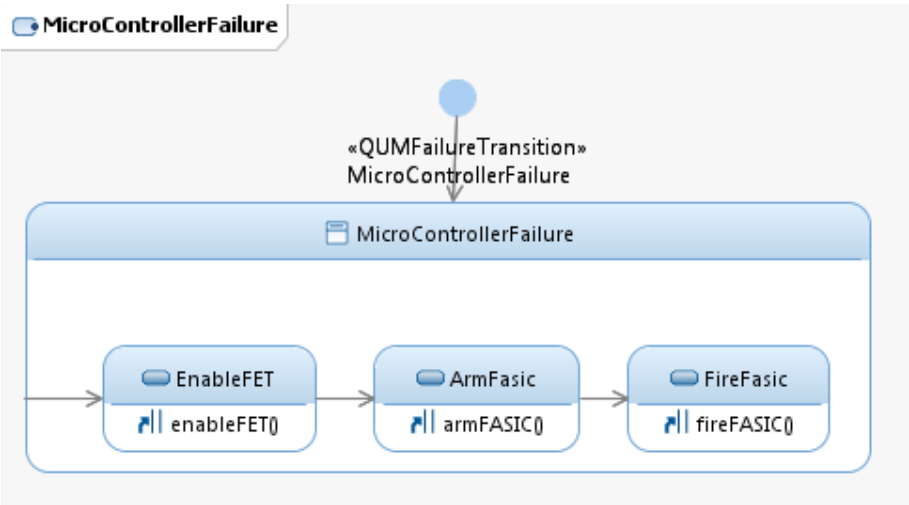

Figure A.6: The state machine for the failure pattern of the microcontroller [31].

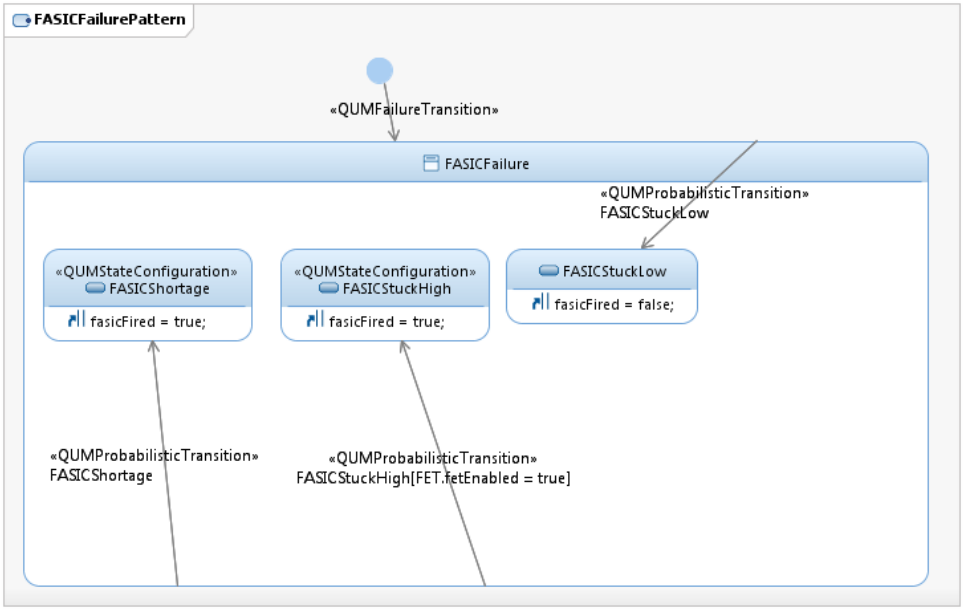

Figure A.7: The state machine for the failure pattern of the FASIC[31].

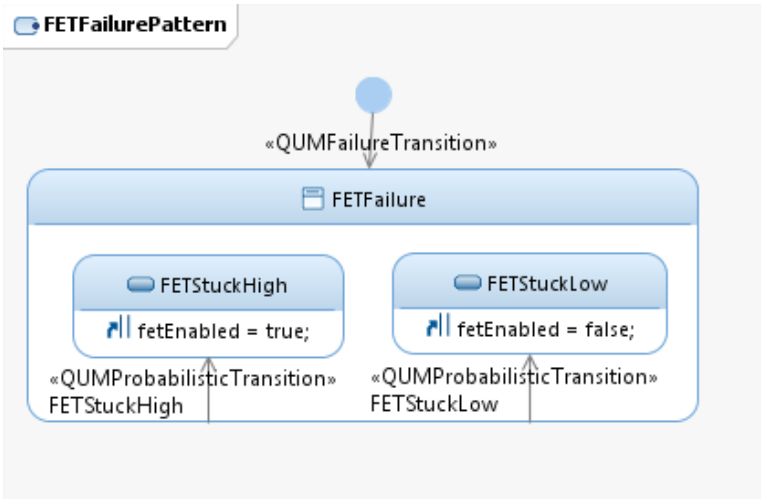

Figure A.8: The state machine for the failure pattern of the FET[31].

Bibliography

[1] Systems modelling language, specification 1.2, Jun. 2010. URL http://www.sysml.org/specs/.

[2] Xml metadata interchange, specification 2.4.1, 2010. URL http://www.omg.org/spec/XMI/.

[3] Unified modelling language, specification 2.4.1, August 2011. URL http://www.uml.org/.

[4] S. B. Akers. Binary decision diagrams. *IEEE Trans. Comput.*, 27:509–516, June 1978. ISSN 0018-9340. URL http://dx.doi.org/10.1109/TC.1978.1675141.

[5] O. Alagoz and M. U. Ayvaci. *Uniformization in Markov Decision Processes*. John Wiley & Sons, Inc., 2010. ISBN 9780470400531. URL http://dx.doi.org/10.1002/9780470400531.eorms0934.

[6] H. Aljazzar. *Directed Diagnostics of System Dependability Models*. PhD thesis, Universität Konstanz, 2009. URL http://kops.ub.uni-konstanz.de/volltexte/2009/9188/.

[7] H. Aljazzar, F. Leitner-Fischer, S. Leue, and D. Simeonov. Dipro - a tool for probabilistic counterexample generation. In *Model Checking Software*, volume 6823 of *Lecture Notes in Computer Science*, pages 183–187. Springer Berlin / Heidelberg, 2011. ISBN 978-3-642-22305-1. URL http://dx.doi.org/10.1007/978-3-642-22306-8_13. 10.1007/978-3-642-22306-8_13.

[8] R. Alur and T. A. Henzinger. Reactive modules. *Formal Methods in System Design*, 15:7–48, 1999. ISSN 0925-9856. URL http://dx.doi.org/10.1023/A:1008739929481. 10.1023/A:1008739929481.

[9] A. Aziz, K. Sanwal, V. Singhal, and R. Brayton. Verifying continuous time markov chains. In *Computer Aided Verification*, volume 1102 of *Lecture Notes in Computer Science*, pages 269–276. Springer Berlin / Heidelberg, 1996. ISBN 978-3-540-61474-6. URL http://dx.doi.org/10.1007/3-540-61474-5_75. 10.1007/3-540-61474-5_75.

[10] C. Baier, H. Hermanns, J.-P. Katoen, and B. R. Haverkort. Efficient computation of time-bounded reachability probabilities in uniform continuous-time markov decision processes. *Theoretical Computer Science*, 345(1):2 – 26, 2005. ISSN 0304-3975. URL http://www.sciencedirect.com/science/article/pii/S030439750500383X.

[11] C. Baier and J.-P. Katoen. *Principles of Model Checking*. The MIT Press, 2008.

[12] T. Bedford and R. Cooke. *Probabilistic risk analysis: foundations and methods*. Cambridge University Press, 2001.

[13] S. Bernardi, J. Merseguer, and D. Petriu. A dependability profile within marte. *Software and Systems Modeling*, 10:313–336, 2011. ISSN 1619-1366. URL http://dx.doi.org/10.1007/s10270-009-0128-1. 10.1007/s10270-009-0128-1.

[14] H. Boudali, P. Crouzen, B. R. H. M. Haverkort, G. W. M. Kuntz, and M. I. A. Stoelinga. Architectural dependability evaluation with arcade. In *Proceedings of the 38th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, Anchorage, USA*, pages 512–521. IEEE Computer Society, Los Alamitos, 2008.

[15] H. Boudali, P. Crouzen, and M. Stoelinga. A compositional semantics for dynamic fault trees in terms of interactive markov chains. In K. Namjoshi, T. Yoneda, T. Higashino, and Y. Okamura, editors, *Automated Technology for Verification and Analysis*, volume 4762 of *Lecture Notes in Computer Science*, pages 441–456. Springer Berlin / Heidelberg, 2007. ISBN 978-3-540-75595-1. URL http://dx.doi.org/10.1007/978-3-540-75596-8_31. 10.1007/978-3-540-75596-8_31.

[16] T. Brazdil, V. Forejt, J. Krcal, J. Kretinsky, and A. Kucera. Continuous-time stochastic games with time-bounded reachability. In R. Kannan and

K. N. Kumar, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2009)*, volume 4 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 61–72. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 2009. ISBN 978-3-939897-13-2. ISSN 1868-8969. URL `http://drops.dagstuhl.de/opus/volltexte/2009/2307`.

[17] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Program. Lang. Syst.*, 8:244–263, April 1986. ISSN 0164-0925. URL `http://doi.acm.org/10.1145/5397.5399`.

[18] M. Debbabi, F. Hassaïne, Y. Jarraya, A. Soeanu, and L. Alawneh. *Verification and Validation in Systems Engineering Assessing UML/SysML Design Model*. Springer Berlin / Heidelberg, Nov. 18, 2010. ISBN 978-3-6421-5227-6. 153-166 pp.

[19] M. Güdemann. *Qualitative and Quantitative Formal Model-Based Safety Analysis*. PhD thesis, University of Magdeburg, 2011. URL `http://nbn-resolving.de/urn:nbn:de:gbv:ma9:1-385`.

[20] H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6:512–535, 1994. ISSN 0934-5043. URL `http://dx.doi.org/10.1007/BF01211866`. 10.1007/BF01211866.

[21] A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker. Prism: A tool for automatic verification of probabilistic systems. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 3920 of *Lecture Notes in Computer Science*, pages 441–444. Springer Berlin / Heidelberg, 2006. ISBN 978-3-540-33056-1. URL `http://dx.doi.org/10.1007/11691372_29`. 10.1007/11691372_29.

[22] G. J. Holzmann. *The SPIN Model Checker: Primer and Reference Manual*. Addision–Wesley, 2003. ISBN 0-321-22862-6.

[23] D. N. Jansen. *Extensions of statecharts : with probability, time, and stochastic timing*. PhD thesis, University of Twente, Enschede, October 2003. URL `http://doc.utwente.nl/58230/`.

[24] J.-P. Katoen. Modeling and verification of probabilistic systems, summer term, 2011. URL http://www-i2.informatik.rwth-aachen.de/i2/fileadmin/user_upload/documents/MVPS11/lec14_handout.pdf.

[25] J.-P. Katoen, M. Kwiatkowska, G. Norman, and D. Parker. Faster and symbolic CTMC model checking. In L. de Alfaro and S. Gilmore, editors, *Proc. 1st Joint International Workshop on Process Algebra and Probabilistic Methods, Performance Modeling and Verification (PAPM/PROBMIV'01)*, volume 2165 of *LNCS*, pages 23–38. Springer, 2001.

[26] J.-P. Katoen, I. S. Zapreev, E. M. Hahn, H. Hermanns, and D. N. Jansen. The ins and outs of the probabilistic model checker mrmc. *Performance Evaluation*, 68(2):90 – 104, 2011. ISSN 0166-5316. URL http://www.sciencedirect.com/science/article/pii/S0166531610000660.

[27] M. Kuntz, F. Leitner-Fischer, and S. Leue. From probabilistic counterexamples via causality to fault trees. In *Computer Safety, Reliability, and Security*, volume 6894 of *Lecture Notes in Computer Science*, pages 71–84. Springer Berlin / Heidelberg, 2011. ISBN 978-3-642-24269-4. URL http://dx.doi.org/10.1007/978-3-642-24270-0_6. 10.1007/978-3-642-24270-0_6.

[28] M. Kwiatkowska, G. Norman, and D. Parker. Probabilistic model checking. In *On probabilistic model checking*. Springer, N.Y., 2001.

[29] J. C. Laprie. *Dependability: Basic Concepts and Terminology: in English, French, German, Italian and Japanese (Dependable Computing and Fault-Tolerant Systems)*. Springer, 1991. ISBN 3211822968.

[30] D. Lehle. Quantitative safety analysis of sysml models. Bachelor's thesis, Universität Konstanz, Nov. 12, 2011.

[31] F. Leitner-Fischer. Quantitative safety analysis of uml models. Master's thesis, Universität Konstanz, 2010. URL http://nbn-resolving.de/urn:nbn:de:bsz:352-opus-125206.

[32] F. Leitner-Fischer and S. Leue. Causality checking for complex system models, Jan. 14, 2012. Available from http://www.inf.uni-konstanz.de/soft/research/publications/pdf/soft-12-02.pdf.

[33] S. Leue. Model checking of software and systems, lecture, summer term, 2011. URL http://www.se.inf.uni-konstanz.de/teaching/summer-term-2011/course-model-checking-of-software-and-systems/.

[34] I. Majzik, A. Pataricza, and A. Bondavalli. Architecting dependable systems. In *Architecting dependable systems*, chapter Stochastic dependability analysis of system architecture based on UML models, pages 219–244. Springer-Verlag, Berlin, Heidelberg, 2003. ISBN 3-540-40727-8. URL http://dl.acm.org/citation.cfm?id=1768179.1768192.

[35] M. Marsan and G. Chiola. On petri nets with deterministic and exponentially distributed firing times. In *Advances in Petri Nets 1987*, volume 266 of *Lecture Notes in Computer Science*, pages 132–145. Springer Berlin / Heidelberg, 1987. ISBN 978-3-540-18086-9. URL http://dx.doi.org/10.1007/3-540-18086-9_23. 10.1007/3-540-18086-9_23.

[36] M. Neuhäußer and L. Zhang. Time-bounded reachability in continuous-time Markov decision processes. Technical Report AIB-2009-12, RWTH Aachen, May 2009. URL http://aib.informatik.rwth-aachen.de/2009/2009-12.ps.gz.

[37] M. L. Puterman. Chapter 8 markov decision processes. In D. Heyman and M. Sobel, editors, *Stochastic Models*, volume 2 of *Handbooks in Operations Research and Management Science*, pages 331 – 434. Elsevier, 1990. ISSN 0927-0507. URL http://www.sciencedirect.com/science/article/pii/S0927050705801720.

[38] W. E. Vesely, F. F. Goldberg, N. H. Roberts, and D. F. Haasl. Fault tree handbook, 2002. URL http://handle.dtic.mil/100.2/ADA354973.