Bachelor Thesis for the degree
Bachelor of Science (B. Sc.) in Information Engineering

Quantitative Safety Analysis of
SysML Models

by
Dominic Lehle
(Matr.-Nr. 01 / 678352)

1st Referee: Prof. Dr. Stefan Leue
2nd Referee: Prof. Dr. Oliver Deussen
Konstanz, 12. November 2011
Abstract

Throughout the implementation of safety-critical systems the decision which requirements should be prioritized have to be balanced. It has been often ignored that many failures of a system could have been addressed in early design processes. Despite the lack of well-engineered methods for safety analysis, the costs and complexity often even keep industry giants away from the safety analysis in early stages of the development. Another criteria is that immersed knowledge is required and not often found in industrial methods. In QuantUM, a successful approach of Florian Leitner-Fischer to bridge this gap and improve the integration of quantitative safety analysis methods into the development process, all inputs needed for this type of analysis can be specified at the level of an UML model. However the gap still exists for models which are not principally software-systems. The SysML is a language based on UML and is an established standard in modeling complex and embedded systems in the industry today. Due to the fact that UML includes, in the manner of speaking, the basic principles for SysML, it is convenient to apply and adjust an existent approach in order to bridge the gap in systems engineering.

As a result we propose a profile for SysML to enable the specification of all inputs needed for quantitative safety analysis in the early development process, and extend the functionality of the existent software on the level of SysML.
Contents

Abstract 2

1 Introduction 5
  1.1 Contributions ........................................ 7
  1.2 Structure of the thesis ................................. 7

2 Foundations 8
  2.1 Unified Modeling Language ............................. 8
  2.2 Systems Modeling Language ............................. 9
  2.3 Probabilistic Model Checking .......................... 10
  2.4 The QuantUM Approach .................................. 12

3 The QuantSyM Approach 13
  3.1 Motivation .............................................. 13
  3.2 Main differences between UML and SysML ............... 14
  3.3 Extension of SysML ...................................... 17
  3.4 SysML-Profile for Quantitative Analysis ................ 18
  3.5 Stereotypes and example ................................ 19
    3.5.1 QSyMComponent ..................................... 19
    3.5.2 QSyMAtributeRange ................................. 21
    3.5.3 QSyMTransitions .................................... 21
    3.5.4 QSyMNormalOperation and QSyMFailurePattern .... 22
    3.5.5 Failure patterns and state configurations ........... 26
    3.5.6 Requirements ...................................... 29
    3.5.7 Failure Propagation ................................ 29
    3.5.8 Repair and Spare Management ....................... 31
  3.6 Discussion .............................................. 34

4 From Quantitative SysML to PRISM 35

5 The resulting QuantSyM Tool 37

6 Case Studies 38
  6.1 Airbag Control Unit ................................... 38

7 Related Work 46
  7.1 Probabilistic Model Checking of SysML
    Activity Diagrams ..................................... 46
8 Conclusion
  8.1 Conclusion .............................................. 47
  8.2 Future Work ............................................. 47

9 Appendix
  9.1 CD ......................................................... 48
List of Figures

2.1 screenshot of the PRISM graphical user interface ........ 11
3.1 Dependencies between UML 2.0 and SysML ................ 14
3.2 SysML Diagram Taxonomy ............................. 16
3.3 Dependencies of the QuantSyM Profile .................... 17
3.4 Definition of the QSyMComponent stereotype .............. 19
3.5 Block diagram of the Railway Crossing System .......... 20
3.6 Definition of the QSyMAttributeRange stereotype .......... 21
3.7 Definition of the stereotypes for abstract and concrete stochastic transitions. ..................................... 22
3.8 State machine representing the normal operation for the Train 23
3.9 State machine representing the normal operation for the Car 24
3.10 State machine representing the normal operation for the Gate 25
3.11 State machine representing the normal operation for the Stop-Light .............................................. 25
3.12 State machine representing the ”Stuck” failure pattern for the Gate ................................................ 26
3.13 State machine representing the ”Stuck” failure pattern for the Light .................................................. 27
3.14 State machine representing the ”Broken” failure pattern for the Gate ................................................ 28
3.15 State machine representing the ”Stuck” failure pattern for the Light .................................................. 28
3.16 The QSyMFailureFlowRule stereotype derived from QuantUM 29
3.17 Example of a failure propagation ............................ 30
3.18 Definition of the stereotypes used for repair and spare management in SysML ............................... 31
3.19 Internal block diagram example with a QSyMSpare. .... 32
3.20 Railway Crossing block diagram illustrating the RepairUnit mechanism ............................................. 33
5.1 QuantUM / QuantSyM GUI .................................. 37
6.1 Airbag Control Unit in SysML .............................. 39
6.2 MicroController normal behaviour state machine .......... 40
6.3 MicroController failure pattern state machine ............ 41
6.4 State configuration attributes ............................... 41
6.5 FASIC failure pattern state machine ...................... 42
6.6 FASIC normal behaviour state machine .................. 42
6.7 FET normal behaviour state machine ..................... 43
6.8 FET failure pattern state machine ......................... 44
1 Introduction

In a recent joint work with an industrial partner the authors of [1] have proven that probabilistic verification techniques can be applied to safety analysis in an industrial setting [1]. The approach they implemented had problems with the missing connection of their analysis to common existing high-level architecture models and the modeling languages that they are typically written in. One of the common used languages that supports analyzing, specifying, designing, verification and validation for system modeling is the Unified Modeling Language (UML) [2]. During their analysis approach they had to use the language provided by the analysis tool they used, in this case the input language of the stochastic model checker PRISM [5]. The Translation from UML to the formal modeling language was a manual and thus a time-consuming procedure hence a way to "bridge the gap between architectural design and formal stochastic modeling languages" had to be found.

In Order to bridge this gap Florian Leitner-Fischer [10] provides within the scope of his master thesis "Quantitative Safety Analysis of UML Models" an extension of the Unified Modeling Language that offers possibilities to capture probabilistic and error behavior information that are relevant for a formal stochastic analysis. Additionally he provides a translation process from UML models to the PRISM [5] language and developed a tool to fully automate this process. However if we look at the complete life-cycle of a system, especially in regards to systems that do not only consist of software, the design language UML has some major gaps in respect to the modeling of systems:

1. UML serves to describe "software-heavy" systems
2. UML does not address the relations between the requirements e.g. there is no possibility to trace a systems specifications down to design elements and test cases.
3. UML does not allow to allocate parts of the systems onto each other. Hence there is no possibility to associate a piece of software to the hardware deploying it.
4. UML models do not provide possibilities to model continuous physical circumstances

For this gaps between UML and Systems Engineering[17] and other reasons the SysML Partners, a group of software tool vendors and industry leaders
created and developed a profile of UML for systems engineering called Systems Modeling Language SysML[7]. The SysML is based on UML 2.0 and extends the language to specify systems containing hardware, software, data, personnel, assets and procedures. It reduces the UML to its system relevant diagrams and stereotypes and leaves out the more software-specific. Overall it provides mechanisms to address the gaps in regard to system modeling. As a result, SysML is gaining broader recognition and acceptance across different industries.

The objective of this thesis is to assign and extend the QuantUM Profile on SysML in order to fill the gap between architectural design and formal stochastic modeling languages in complex system environments. We take the extension of Florian Leitner Fischer’s QuantUM and try to apply it to the SysML Profile to capture probabilistic and error behavior information that are relevant for a formal stochastic analysis. Based on the existing extension we evaluate the profile on different system models and add additional possibilities to annotate the SysML models with quantitative information.
1.1 Contributions

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

1. In regard to the differences between UML and SysML we evaluate how the existing QuantUM approach for UML can be adapted to SysML models, in order to enable quantitative safety analysis for SysML models.

2. We extend the already existing profile to certain new elements in order to provide the possibilities to annotate structural, behavioral diagrams and requirement diagrams with quantitative information. We name the resulting notation QSyM.

3. We describe the development of a prototypical tool chain for quantitative system analysis.

4. We evaluate our SysML extension based on an industrial case study.

1.2 Structure of the thesis

In this thesis we analyze the portability from QuantUM to SysML and provide an extension for SysML to enable quantitative system analysis to SysML models.

In Chapter 2 we provide a short introduction to the Unified Modeling Language, SysML, probabilistic model checking and the QuantUM approach. In Chapter 3 the differences between the two modeling standards UML and SysML are described in order to analyze the applicability of QuantUM to SysML, furthermore the quantitative extension of SysML is presented. In Chapter 4 the changes for the translation rules from SysML to PRISM are described. Subsequently what has been maintained from the QuantUM approach tool chain is shown in Chapter 5. That followed, the resulting profile extension is demonstrated and evaluated on a case study in Chapter 6. Finally, related work is discussed in Chapter 7 followed by possible future work and conclusions in Chapter 8.
2 Foundations

2.1 Unified Modeling Language

The Unified Modeling Language (UML)[2] is a specification for visualizing and documenting models in the field of software and system engineering. The Object Management Group (OMG)[3] established UML as a standard for the design of software applications and manages the structure of the language. UML provides a large number of graphical elements, in order to design visual models of software-intensive systems. The current released version of UML is 2.4 -Beta 2 released in March 2011 [3] and it contains two categories of diagrams. The category structural information includes seven diagram types representing information about the systems structure. The category behavioral information contains another seven diagrams representing the behaviour of certain objects in the system or their general communication, with four of them including interaction aspects. The overall overview of all language elements can be seen in the current UML Standard Specification[4]. Although the OMG is a non-profit computer industry consortium since 1989 they work continuously on improvements to the language.

So far, UML is the dominating standard for today’s system and software architecture modeling and a significant number of tools which support UML exist. The most commonly used tools are IBM Rational Software Architect¹, Sparxsystems Enterprise Architect² and the open source tool ArgoUML³. The UML is standardized in ISO IEC 19501 Version 2.1.2. Additionally other standards for the development of safety-critical systems like ISO IEC 61508 [8] or ISO CD 26262 [9] highly recommend the usage of UML to specify the design of software systems.

¹http://www.ibm.com/software/rational/
²http://www.sparxsystems.com/
³http://argouml.tigris.org/
2.2 Systems Modeling Language

The Systems Modeling Language SysML is an UML extension in order to bridge the gaps between the domain-specific usage of UML for Systems Engineering. It is the result of a joint initiative of OMG and the International Council on Systems Engineering (INCOSE) and built to verify and validate complex systems which are not necessarily built in a software-domain. The SysML is based on UML 2.0 and extends the language to specify systems containing hardware, software, data, personnel, assets and procedures. It reduces the UML to its system relevant diagrams and stereotypes and leaves out the more software-specific. Overall it provides the following important mechanisms to address the gaps in regard to system modeling:

1. A possibility to define requirements modeling in a more complex way.

2. Extensions to system structure modeling by defining additional stereotypes like blocks and value properties.

3. The possibility to define parametric models.

4. Allocations that allow to establish relations between different sub-systems or levels of the system e.g. structure and behavior, logical and physical...

5. Extensions to enable modeling of time-continues activities and object flows

Today SysML is used by more than just system engineers on complex systems and a large desire to continue using SysML in organizations exists[14]. In order to apply the QuantUM approach to SysML we will go into more specific differences between the UML and the SysML in [3.2].
2.3 Probabilistic Model Checking

Complex systems consume more time and effort on verification than on construction. Probabilistic Model Checking [11] is an automated verification method, which is used to analyze safety-critical systems.

Like in general Model Checking, two inputs are required: a model of the system to be analyzed and a specification of the quantitative properties of the system that are to be analyzed.

In Probabilistic Model Checking the Model Checker constructs a probabilistic model out of the system model. The result is a probabilistic state-transition system, with states representing a possible configuration of the system and transitions that represent a possible evolution of the system from one configuration to another over time. In Probabilistic Model Checking the transitions are additionally labeled with quantitative information which specifies the probability and/or timing of the transition’s occurrence.

In Leitner-Fischer’s QuantUM approach he uses continuous-time Markov chains CTMC [11] which is one of several methods to check a model. The

\[ P_{0.1} \text{ is a true state in the CTMC} \]

state-transition system is extended to probabilities in that case discrete probability distributions are assigned to the transitions. The CTMC also assumes that no deadlock state exists, therefore every state has at least one outgoing transition. The probabilistic model checker explores the model and constructs all possible occurent paths, for a formal specification given. The specification for the Probabilistic Model Checker is declared using a variant of temporal logic. The temporal logic used in this thesis as well as in Leitner-Fischer’s QuantUM approach to describe the Markov chains is the Continuous Stochastic Logic (CSL) [12, 13]. CSL enables to specify the probability of a state, in order to satisfy some temporal property including the time interval in which this property must hold. It is an extension of the Computation Tree Logic (CTL) [14] and adds a probabilistic operator in order to refer to the probability of the occurrence of particular paths in
the CTMC. A steady state operator is added as well in order to refer to the probability residing in a particular set of states specified by a state formula at long sight. The state and path formulas are based on [15]. The state formulas are interpreted over states of a CTMC, whereas the path formulas are interpreted over paths in a CTMC. The steady-state operator refers to the probability in which it converges in. The time span of a certain path, is expressed by the path operators until (U) and next (X), extended with a parameter that specifies a time interval.

There are two possible ways for approximating probability measure: numerical and statistical. Probabilistic model checking uses numerical methods for the approximations in form of a truncation error, hence it is better to obtain more accurate and exact results compared to discrete-event simulation techniques, which samples model’s executions and produces an estimation of the measure of interest.

In this thesis we use PRISM [5, 6], a probabilistic model checker developed at the University of Oxford. It accepts probabilistic models like CTMCs in its simple high-level state-based modeling language.

Figure 2.2: screenshot of the PRISM graphical user interface

PRISM has also been used in Florian Leitner-Fischer’s QuantUM approach, which served as a foundation for this thesis.


2.4 The QuantUM Approach

The QuantUM approach [QuantUM] is Leitner-Fischer’s approach to extend the UML with a profile for quantitative analysis, where all inputs of the quantitative system analysis are specified in UML. He extended the standard UML so that all information that is needed to perform a quantitative analysis, in that case probabilistic model checking, can be specified at the level of UML. In order to extend UML he added structural description capabilities to the language so it is possible to present dependencies of different components, e.g. that a failure of a specific component influences a failure of another component. The behavioral description is extended too, to provide a possibility to include stochastic behavior in the model. A model annotated with the QuantUM extension, allows to specify all information needed to perform the safety analysis. Additionally a tool is provided to parse the analysis model and translate it into the input language of the probabilistic model checker PRISM. With the resulting PRISM model counterexamples for the probabilistic properties are computed throughout the PRISM extension DiPro. The results are then represented as a fault tree which can be mapped onto UML sequence diagrams.

In this thesis we will extend the existing QuantUM UML profile so it allows the application to the SysML.
3 The QuantSyM Approach

3.1 Motivation

The QuantUM approach already specifies all information needed for a quantitative analysis of UML models. Additionally the QuantUM Tool automatically translates the corresponding annotated model in the analysis model and represents the results in a sequence diagram. Since SysML is based on several elements of UML it is suitable to use the existing approach and adjust it to several other needs for the system engineering domain.

In order to do so, we define our requirements of the extension for SysML on the same requirements that were crucial for the acceptance of the QuantUM approach:

1. Applicability is provided on system and software architectures defined in SysML.

2. It is possible to specify dependability objectives / requirements.

3. Means for the specification of dependability characteristics of the system components, such as failure modes and rates are existent.

4. Means to specify failure propagation paths and dependencies between different system components are existent.

5. There are possibilities to model safety mechanisms such as redundancy structures and repair management.

6. Experienced users (i.e. system engineers) are easily able to acquire the knowledge about how to use the extension.

7. The additional modeling with QuantSyM shall be as inexpensive as possible.
3.2 Main differences between UML and SysML

SysML is built on UML 2.0 and satisfies the needs of system engineers by adapting UML when it is necessary. Therefore SysML removes the software-specific diagrams and focuses more on system relevant’s. Thus UML is split in the UML concepts necessary for SysML and the ones not. The dependencies between both modeling standards is shown in Figure 3.1. The four diagram types: use-case diagram, state machine diagram, package diagram and sequence diagram haven’t changed in SysML and therefore act the same as they did in UML. The package diagram describes the system and the projects structure, the use-case diagram allows to describe the systems function from a user perspective, the sequence diagram allows for example to model the course of events or interactions between system parts and finally the state machine diagram, which enables to model the dynamic behaviour of the system or its parts. Some diagrams from UML are already useful but need to be modified to fit into the systems engineering environ-
The QuantSyM Approach

Main differences between UML and SysML

ment. The class diagram for instance is simplified: All the software specific elements have been removed, furthermore classes are now blocks and build a unifying basis element for systems. Another diagram modified from UML 2 is the internal block diagram, which specifies the interconnection of parts. Like its UML 2 predecessor, the composite structure diagram, the internal block diagram specifies the interconnection of parts. Additionally to the normal ports, flowPorts are introduced in SysML which enable the modeling of physical or continuous flows. The direction of flowPorts plays an important role and is indicated by an arrow. The activity diagram is the last diagram taken from UML 2 and modified to the needs of system engineering. In SysML’s activity diagrams still specify the controlled sequence of actions, the main addition is the possibility to control the execution of actions, for instance running actions can be disabled. The limited control of actions in UML 2 is extended by control operators making it possible that a behaviour may not terminate by itself, but instead externally.

When representing requirements, system engineers early met their limits of UML. Although the current version of UML provides many possibilities to specify requirements for a system, it is not possible to trace requirements of a system from informal specifications to specific design elements. Use cases for example help build up a base for understanding the expected behavior of a system and can validate its architecture, but requirements often only trace to the use cases but not to the other parts of the system. The addition of information which captures reasons for specific design decisions made during the development process and linking them to the requirements helps to analyze consequences that appear if a requirement changes. Additionally to the traceability limitation, no model element provides the possibility to declare non-functional requirements. With the requirement diagram SysML introduces a tool to define several kinds of relationships for improving the requirement traceability. Requirements can now be linked to the design and to the test suite and can be build up in a similar way than class diagrams.

SysML also introduces the concept of assembly, a stereotyped class which describes a system as a structure of interconnected parts. An assembly provides a domain neutral modeling element that can be used to represent the structure of any kind of system, regardless of the nature of its components. The concept of allocation in SysML is a more abstract form of deployment than in UML. It is defined as a design time relationship between model elements which maps a source into a target. An allocation provides the generalized capability to allocate one model element to another. For example, it can be used to link requirements and design elements, to map a behavior into the structure implementing it, or to associate a piece of software and
The QuantSyM Approach
Main differences between UML and SysML

The hardware deploying it. The SysML support for flows between assemblies is meant to provide an equivalent and domain-neutral modeling capability. In particular the control of execution is extended such that running actions can be disabled.

Finally the focus on system engineering leads to the removal of pure software specific diagrams. Therefore the communication diagram, interaction diagram and timing diagram are not used in SysML. An overview of the SysML diagram taxonomy in regards to the important diagrams used in QuantUM can be seen in figure 3.3.

Figure 3.2: SysML Diagram Taxonomy
3.3 Extension of SysML

SysML as well as UML includes built-in mechanisms: model libraries and profiles to customize the language. In general a modeling language contains a number of distinct language concepts, which are represented by a collection of metaclasses. Metaclasses define the behavior of certain elements and instances and include a set of properties and constraints to do so. The collection is called metamodel and the underlying metamodel for SysML is UML for SysML (UML4SysML) [figure 3.1].

A profile is some kind of package for specific domains and platforms. It serves as the container for a set of stereotypes and supporting definitions. A stereotype [4] is an extensibility mechanism providing the possibility to add certain properties to an already existing metaclass. Typically a profile contains a set of stereotypes representing a contiguous set of concepts for a specific modeling domain. QuantUM defines concepts that add further information to the UML model in order to perform a quantitative analysis safety of UML Models. Any UML model element can be extended by Leitner-Fischer’s defined stereotypes. In SysML several elements used in QuantUM are not existent because SysML references the UML4SysML metamodel to extend its metaclasses, therefore we have to change and extend the profile in order to keep the compliance of SysML.

Figure 3.3: Dependencies of the QuantSyM Profile
3.4 SysML-Profile for Quantitative Analysis

In this subsection we will describe the SysML profile for quantitative analysis in regard to changes we made to QuantUM. We will briefly explain the stereotypes introduced by Leitner-Fischer and additional stereotypes we added in order to add required information to SysML. In accordance with UML, SysML splits the model into two fundamental parts: the structural and the behavior part. Static elements and the architecture or structure of the system are described in the structural part, dynamic behavior between elements and parts and their associated relationships however are described in the behavior part of SysML. In addition SysML introduces new cross-cutting constructs: allocations and requirements[18]. Whereas the last one provides possibilities to attach problems and rationales to any model element in order to capture issues and decisions. Therefore we could use it to directly trace the requirements onto blocks or other parts of the system. The capture of the dependency of a model’s structure, is described in UML as well as in SysML, in the structural part of the model. Hence we can assign the extension of the structural description capabilities of QuantUM to SysML in regards to specific changes to elements throughout SysML. The same applies to the extension of the behavioral description to capture the stochastic behavior. In course of this thesis we use the prefix QSyM for the names of stereotypes that belong to our SysML profile. Subsequently we define the new stereotypes, properties and their changes from the original QuantUM profile as well as new stereotypes from scratch used to specify the information needed to perform stochastic analysis. We demonstrate the usage of the profile on a railway crossing example: the model consists of a train, a car, a gate, and a stop-light. Whenever a train is approaching the gate is requested to close and the stop-lights are requested to start flashing showing the car that it has to stop. As soon as the train left the crossing, the gate is being signaled that it should be opened again and the stop-lights should stop flashing and the car can cross the gate.
3.5 Stereotypes and example

3.5.1 QSyMComponent

The stereotype QSyMComponent is the SysML version of the QUM-Component in QuantUM. It can be assigned to all SysML elements that represent building parts or blocks of the real system, therefore to the block element that replaces UML classes, and to packages and parts. The QSyMComponent does not differ from the basic structure of the QUMComponent, hence it comprises up to one (hierarchical) state machine representing the normal behavior and one to finitely many (hierarchical) state machines that represent the possible failure patterns. As already mentioned in previous chapters the state machine diagrams and their corresponding elements have been adopted from UML, therefore state machines can be treated the same as in QuantUM. The QSyMComponent still includes the list of Rates containing the rates with the names identifying them. The final QSyMCompo-
The QuantSyM Approach
Stereotypes and example

*nent* stereotype, its dependencies and attributes is shown in Figure 3.4. The

![Block diagram of the Railway Crossing System](image)

**Figure 3.5:** Block diagram of the Railway Crossing System

block diagramm of our railway crossing system is shown in Figure 3.6. In order to analyse the behavior of all the blocks, all of them are tagged with the stereotype *QSyMComponent*. The *Gate* block contains the operation *switchGate* and a boolean attribute with the name *closed* represents the gate’s state. The *Stop-Light* block contains the boolean attribute *flashing* indicating whether or not the stop light is flashing. The dependency of the Train block to the Gate block can be seen as continuous flow, as a train constantly crosses the gate. The same applies to the car, but it can only cross the Gate when it is not closed. With the tagged stereotype *QSyMComponent* we now allow the association with the state machines representing the normal and failure behavior like in QuantUM.
3.5.2 QSyMAttributeRange

The QSyMAttributeRange keeps the same functionality as the QUMAttributeRange in QuantUM: It allows for the specification of the range of integer attributes of the QSyMComponents needed for finite state verification methods like probabilistic model checking.

![Figure 3.6: Definition of the QSyMAttributeRange stereotype](image)

3.5.3 QSyMTransitions

As in the UML the possibility to specify quantitative information like failure rates in SysML is not possible because state machine diagrams and therefore transitions haven’t been changed. Hence we apply the stereotypes used in QuantUM to the transition element in state machines. QSyMAbstractStochasticTransition and QSyMStochasticTransition extend the capability of transitions (see Fig. 3.7) like their QuantUM predecessors and enable the user to specify rates as well as a name for it. These transition rates are used for the continuous-time Markov chains generated for the stochastic analysis. The stereotype QSyMAbstractStochasticTransition acts as a superclass for the different specializations and it does not contain a default rate. Consequently there has to be a rate in the Rates list of our QSyMComponent, if a state machine is connected to this QSyMComponent, in order to specify a rate for this transition. This enables the individual specification of rates for each component out of a repository. Basically we need the same two transition types used in QuantUM. Thus we define the QSyMFailureTransition describing a transition to a failure pattern and the QSyMRepairTransition which describes the transitions back to a normal behaviour, explained later on 3.5.8. The ”Abstract” Version of each of the transitions fulfills the prior mentioned function: it does not contain a default rate.
3.5.4 QSyMNormalOperation and QSyMFailurePattern

A QSyMComponent consists of one Normal Operation state machine, describing the normal behaviour and one to infinitely Failure Pattern state machines. In the Tool and hence not visible for the user, these state machines are combined in exactly one hierarchical state machine. The functionality of this in QuantUM established manner of modeling works as followed: First of all the QSyMComponent executes the normal operation state machine. While executing this operation an error can occur. For instance if a QSyM-FailureTransition is enabled, the QSyMComponent enters the corresponding failure pattern state machine with the specified rate.

In our Railway Crossing example we tagged all of the blocks with the QSyMComponent stereotype. The Train block and the Car act only as environment input, therefore they only have a normal behaviour. In order to keep the complexity manageable we consider them as a sub-system, not
The QuantSyM Approach
Stereotypes and example

important for the correct behaviour of the gate and the stop-light. Therefore we do not specify any failure patterns in these blocks. Our \textit{Train} has three possible states (Figure 3.8). In the driving state the train is not near the gate, hence the approaching variable is set to false. If he is approaching the gate the corresponding variable is set to true. This keeps until he is not crossing the gate anymore.

![State machine representing the normal operation for the Train](image)

\textbf{Figure 3.8:} State machine representing the normal operation for the Train

The state machine for the normal operation (Figure 3.9) of the \textit{Car} shows that the \textit{Car} is initially driving. Whenever it is near the gate, it takes the transition to \textit{Approaching}. In this state the variable \textit{closed} is the guard for the next possible transition. The car can only cross the gate if the gate is open, otherwise it has to stop.

Two blocks are more important for the behaviour of our system, namely the \textit{Stop-Light} block and the \textit{Gate} Block. In our example we assume that the \textit{Gate} is controlling the \textit{Stop-Light} and that there is no additional Sensor block that indicates that a train is approaching.
The QuantSyM Approach

Stereotypes and example

Figure 3.9: State machine representing the normal operation for the Car

The state machine responsible for normal operation of the Gate block is shown in Figure 3.10. It switches the GateClosed state whenever a train is approaching indicated by the trainApproaching operation. When the operation trainCrossed is executed the train has passed the gate and the state is changed to GateOpen. The variable closed is changed when entering the respective state to the appropriate value.
Figure 3.10: State machine representing the normal operation for the Gate

Figure 3.11: State machine representing the normal operation for the Stop-Light
3.5.5 Failure patterns and state configurations

In Figure 3.12 one of the two possible failure patterns of the Gate block is shown. The Stuck failure pattern can either go to the StuckOpen state where the gate keeps open or to the StuckClosed state where the gate keeps closed. The closed variable changes its value to false upon entry to the StuckOpen state and true upon entry to the StuckClosed state. Additionally we assign the QSyMStateConfiguration stereotype to the StuckOpen and StuckClosed state in order to define one state configuration. A state configuration in QuantUM can be seen as a boolean formula: each state can either be true or false depending on whether the system is in this state or not. Additionally an operator variable indicates how the boolean variables representing the states are connected by each other. This operator can be either a logic or-operator (OR) or a logic and-operator (AND). In our running example we name these state configuration SystemStuck and select the or-operator to connect them. Our defined state configuration SystemStuck is true whenever the system is in at least one of the "Stuck" states.

Figure 3.12: State machine representing the "Stuck" failure pattern for the Gate block.
The QuantSyM Approach
Stereotypes and example

Figure 3.13: State machine representing the "Stuck" failure pattern for the Light

The state configuration is applied to the Stop-Light Stuck failure pattern as well which allows us to determine the probability of reaching a "Stuck" failure in the further progress of the analysis. QSyMStateC onfiguration having the same name are treated as one. Another QSyMStateConfiguration with the name SystemDown is applied to two other failure patterns, whereas one of each corresponds to a different block (i.e. Gate and Stop-Light). Figure 3.14 shows the failure pattern GateBroken which consists of one failure state where the Gate is Broken and the variable closed is set to false by the entry action, and a state tagged with the QSyMStateConfiguration named SystemDown. The state machine describing the Stop-LightBroken failure pattern (Figure 3.15) consists of one failure state where the Stop-Light is broken and the variable flashing is set to false by the entry action, therefore it can’t flash whether or not the train is approaching. The SystemDown state configuration is now true whenever the system is in one of the "Broken" states. Since we are interested in the analysis of unanticipated failures, that is the car is crossing and the train is crossing, we tagged the Crossing-Gate state in both, the normal behaviour state machine for the car and the train, with the state configuration named Crash (Figure 3.8 and 3.9)
**Figure 3.14:** State machine representing the "Broken" failure pattern for the Gate

**Figure 3.15:** State machine representing the "Stuck" failure pattern for the Light
3.5.6 Requirements

In SysML it is possible to trace requirements to every design element in the model. Hence in our QuantSyM tool we add the possibility to check all model elements for requirements. These requirements can then be used as additional checks for all elements. For instance failure rates can then be specified by requirements. This enables to trace the design decisions made during the creation of development artifacts to the roots and helps to analyze consequences in the process. In our example we add the requirement "StopLight\_failureRate\leq0.8" (Figure 3.15). The tool then checks if the failureRate for the StopLight is below that rate, if not it sets the rate to 0.8. This ensures that in the tool certain variables can be restricted to reasonable values.

Beside of that we also offer an additional possibility to automatically create property specifications out of requirements: In QuantUM the Continuous Stochastic Logic (CSL) [?] [16] is used to specify the properties that want to be verified. These CSL properties are generated automatically out of the SysML model by the QuantSyM tool, and the state formulas variable for time (t) can be restricted to a certain value. In QuantSyM it is now possible if for instance the requirement "P<\text{x} \text{ in t}=2" is linked to a state configuration (stc), that the resulting CSL property: P<\text{x}[(true) U\leq2 (stc)] is generated for that state configuration. In the railway example one requirement is added (see Figure 3.15) to the state configuration System\_Down which leads to the construction of the CSL property P<0.7[(true)U\leq2 (System\_Down)] automatically by the tool.

3.5.7 Failure Propagation

The QuantUM approach offers two ways to specify a failure propagation, whereas specifying the behaviour by state machines is only one. In our railway crossing example a failure of the Gate block does not automatically propagate to the Stop-Light block. In our case the Train block is responsible for the indication of a train approaching. Whenever this indication

Figure 3.16: The QSyMFailureFlowRule stereotype derived from QuantUM
is not working the Gate and the Stop-Light do not work because they will never know if a train is approaching or not. Therefore we want to propagate a failure of the Train block to other blocks. In order to achieve that we use the main idea behind the QUMFailurePropagation in QuantUM. The result is the stereotype QSyMFailureFlow which is applicable to flows. A QSyMFailureFlow enables to specify that the failure of one block propagates onto another. The direction of this propagation is indicated by the direction of the flow. Therefore if a flow port has only the direction from block A to B the failure only propagates from A to B. In addition Items with the stereotype QSyMFailureFlowRule can be added to the QSyMFailureFlow, to define the rules for the propagations. In Figure 3.16 the stereotype QSyMFailureFlowRule and its tags are shown.

In our example we added an additional QSyMFailureFlow from the Train block to the Gate block and added an Item called Approaching which has the stereotype QSyMFailureFlowRule with the flowEnd1 = approachingSensorDoesNotWork, the rate 1.0 and the flowEnd2 = GateBroken (Figure 3.17). Now whenever the Train executes the approachingSensorDoesNotWork failure transition the Gate will execute the GateBroken transition. If the flow is bidirectional it would imply that if the GateBroken transition is executed, the Train is executing the approachingSensorDoesNotWork transition.

![Figure 3.17: Example of a failure propagation](image-url)
3.5.8 Repair and Spare Management

The QuantUM UML extension also includes possibilities to model repair management strategies. These strategies are enabled throughout the definition of repair units which replace or repair failed system components. In order to enable that modeling enhancement we apply the same functionality to SysML. Hence the $QSyMSpare$ component, derived from the $QUMSpare$ in QuantUM, is added to our SysML extension with slight changes. The stereotype can be assigned to parts in the internal block diagram, and its associated part then functions as a spare. Therefore whenever the main block activates a failure, the $QSyMSpare$ is activated with a rate which is specified in its $activationRate$.

Supplementary to the $QSyMSpare$ stereotype we also add the $QSyMRepairUnit$ derived from the QuantUM counterpart $QUMRepairUnit$ with minor changes in order to be suitable to SysML. Applicable to blocks it can be associated to other $QSyMComponents$ by a flow or an association with the $QSyMRepairAssociation$ stereotype. The block tagged with this stereotype has the functionality to repair the components having a connection in form of an association or flow to it. The two different repair strategies from QuantUM are adopted in the $QSyMRepairStrategies$ and have to be specified in the strategy tag in the $QSyMRepairUnit$. The dedicated strat-
The QuantSyM Approach
Stereotypes and example

ey repairs exactly one component per repair unit and the first come first serve (FCFS) strategy can repair more than one component, processing the components in the rule the name implies. The QSyMRepairUnit includes a set of rates which stand for the components repair rates. Repair transitions are defined in the wanted failure patterns of the corresponding component and are tagged with the QSyMRepairTransition stereotype.

In our railway crossing example we added an internal block diagram to the Stop-Light block which shows the Stop-Light from a white-box-perspective (Figure 3.19). The SpareLight part is tagged with the QSyMSpare stereotype, thus two spare instances of the LightBulb are activated after each other, with the specified activatorRate whenever the main block (i.e. LightBulb) is in the failure state Stop-LightBroken, which is specified in the QSyMSpare attribute repairedFailure within the SpareLight part.

**Figure 3.19:** Internal block diagram example with a QSyMSpare.

Additionally we add a repair unit named Stop-LightRepairUnit to our Stop-Light with a QSyMRepairAssociation attaching it to the Stop-Light. The StopLightRepairUnit executes the QSyMAbstractRepairTransition which
brings the *Stop-Light* back to its normal behavior state machine with the in the association specified repair rate. The resulting block diagram is shown in Figure 5.1.

**Figure 3.20:** Railway Crossing block diagram illustrating the RepairUnit mechanism.
3.6 Discussion

To see if our appliance of QuantUM to SysML maintains the in Section 3.1 stated requirements, we discuss the degree of fulfillment in this section.

The first requirement (1) "Applicability is provided on system and software architectures defined in SysML" has been maintained because the proposed extension is applicable to the SysML elements used in SysML. The second requirement (2) "It is possible to specify dependability objectives/requirements.” is fulfilled throughout the added QSyMStateConfiguration stereotype, allowing the specification of state configurations which are used in QuantUM to specify ability objectives and requirements. Beside of that the possibility to define requirements for dependability is enabled. The profile does not differ from QuantUM in terms of the possibility to specify all dependability characteristics needed for the analysis. Thus the requirement (3) "Means for the specification of dependability characteristics of the system components, such as failure modes and rates are existent” is also maintained. The adopt and changes to the QUMPropagationRule leading to the QSyMPropagationRule for SysML assures the fulfillment of requirement (4). The same approach to handle means to ”model safety mechanisms such as redundancy structures and repair management” with the introduction of the from QuantUM derived stereotypes: QSyMSpare, QSyMRepairUnit, QSyMRepairAssociation and QSyMRepairStrategies ensures the fulfillment of requirement (5). Since the profile is extended to concepts which can be simply inferred from standards in system engineer, it should be easy to use the profile for an experienced users in system engineering. Therefore we maintain (requirement (6)) throughout our ”SysML - QuantUM” approach. Finally, the fact that we kept the basics of the QuantUM approach and extended it to the system engineering domain needs, shows that ”The additional modeling with QuantSyM is as inexpensive as possible” (requirement (7)).

\footnote{comparison to Florian-Leitner Fischer’s requirements}
4 From Quantitative SysML to PRISM

In order to analyse a system we modeled with the QuantSyM approach with the model checker PRISM, the model has to be translated to the PRISM language. In QuantUM there are already translation rules specified for the UML to translate its elements into the input language of the probabilistic model checker PRISM. In QuantSyM some translation rules are enhanced or changed so that the analysis possibilities are maintained. We will give a short introduction on how the PRISM language is structured based on the PRISM Manual, for further information we refer to [22, 21, 10].

A PRISM model consists of a number of modules which contain local variables that constitute the state of the module at any given time. A modules behaviour is described by a set of commands. These commands have the following form:

\[ \text{[transition label]} \text{ guard } \rightarrow \text{ rate}_1 : \text{ update}_1 + \ldots + \text{ rate}_n : \text{ update}_n; \]

The \textit{guard} is a predicate over all the variables in the model (including those belonging to other modules). Every update describes a transition which the module can make if the guard is true. The transitions are specified by giving the new values of the variables in the module, possibly as a function of other variables. Each update is also assigned a probability (or in some cases a rate) which will be assigned to the corresponding transition.

Codesample 4.1 shows an example module named \textit{samplemodule} containing two variables, whereat variable 1 is of type Boolean and initially true, and variable2 is numeric has initially the value 0. The guard (variable 2 \(\leq 12-1\)) \& (variable2-1 \(\geq 0\)) \& (variable2 \(< 2\)) needs to be true, so that the update (variable2' = variable2 +1) is executed with the rate 0.8. Whenever the guard (variable 2 = 2) is true, the variable1 is set to false (variable1'=false) with the rate rate 1:0.

```
module samplemodule
   variable1: bool init true;
   variable2: [0..12] init 0;
   [Count] (variable2<=12-1) \& (variable2-1 >=0) \&
   (variable2 < 2 ) \rightarrow 0.8 : ( variable2' = variable2 +1);
   [End] (variable2 = 2) \rightarrow 1.0: (variable1' =false);
endmodule
```

Codesample 4.1: a module in the PRISM language
In QuantSyM, the translation rules for the *QUMComponent*, *QUMFailurePropagation*, *StateMachine* and *QUMRepairUnits* are the same for the corresponding QSyM elements, whereas the *QSyMSpare* is changed slightly. So far the *QUMSpare* stereotype is translated by adding a counter that counts the active spares, and one transition command that sets the module to the initial state, whenever any failure state is entered and the left spare count is not zero. With the *QSyMSpare* we assume that not any System failure can be replaced by a spare, therefore we restrict it to specific failure states. The resulting translation rule is shown in Codesample 4.2

```plaintext
%module_id%_activespares: [0..%spares%];

[%module_id%_SpareActivated]
   (%module_id%_state = %#failurestate%)
   & (%module_id%_activespares < %module_id%_nuofspares)
   => %rate%: (%module_id%_state = %id_init%)
   & (%module_id%_activespares = %module_id%_activespares + 1);

Codesample 4.2: PRISM translation rule for QSyMSpare
```

Besides the PRISM analysis model, the CSL properties to be analyzed are automatically generated by the tool. The construction of the main CSL properties can either be changed throughout the adding of further state configurations or by just adding requirements (Chapter 3.5.6) to the state configurations. In addition a way to manually specify CSL properties is possible in the tool.
5 The resulting QuantSyM Tool

Within the scope of this thesis, the QuantUM tool has been restructured and further improvements have been made. The possibility to parse SysML models has been added to the tool, thus it is possible to translate SysML models with the QSyM profile to the PRISM language. In our case studies we used the IBM Rational Rhapsody v7.6 to annotate the model and export it to the XML Metadata Interchange (XMI) format, a standard format for exchanging models and objects based on meta-models [20]. The QuantSyM tool has the same "process chain" as the QuantUM tool. Therefore after the translation to the PRISM language, it generates the counterexamples and computes the fault tree using the DiPro tool [15], and maps it on a sequence diagram, stored in the XMI format. A Graphical user interface exists in order give the user information about the current progress and to have the possibility to change several attributes in the QSyMComponents. An opportunity to choose between the different state configurations is also provided, so a specific analysis can be made.

![QuantUM / QuantSyM GUI](image)

**Figure 5.1: QuantUM / QuantSyM GUI**
6 Case Studies

6.1 Airbag Control Unit

In order to evaluate our SysML profile, we applied it to the case study from the automotive software domain, Florian Leitner-Fischer used in case studies within his thesis: An Electronic Control Unit for an Airbag System developed for TRW Automotive GmbH has been modeled in UML and the QuantUM extension, with the objective of analyzing whether or not the Airbag System is safe. Since we want to test the QuantSyM approach we need to transfer the model to SysML, whereat we use the CASE tool IBM Rational Rhapsody. We will explain the resulting model based on the explanations in [10].

In general the airbag systems consists of three fundamental parts. The first part are sensors: the System has different sensors detecting an impact. Acceleration sensors detect front, rear and side impacts whereas pressure sensors are in addition to side impact detection. Rollover accidents are detected by so called angular rate or roll rate sensors. The second part is the microcontroller: the microcontroller decides if the sensors information stand for a crash situation. Unless the decision does not fall into the category "critical crash", the airbags are not deployed. However, if they do fall in that category, the third part: actuators control the deployment of the airbag.

In the in [10] modeled airbag system architecture, two acceleration sensors exist to detect front or rear crashes. Exactly one microcontroller is responsible for the crash evaluation and an actuator controlling the airbag deployment.

Two redundant mechanisms, the Field Effect Transistor (FET) and Firing Application Specific Integrated Circuit (FASIC) secure the deployment of the airbag system whereas the FET is the power management of the airbag squibs and the FASIC controls the squib itself. The FET has to be armed and therefore the FET-Pin high in order to provide the airbag enough electrical power to ignite. The FASIC has to receive an arm command from the microcontroller and a fire command in advance so that it will ignite the airbag squib.

The structure of the QuantSyM model derived from the QuantUM model is shown in Figure 6.1 and has not changed in its basic structure. The before modeled classes: MainSensor, SafetySensor, MicroController, FET and FASIC are now changed to blocks and are tagged with the stereotype QSyMComponent.

\[1\]

\[3\] Comparison to [19]
Case Studies

Airbag Control Unit

Figure 6.1: Airbag Control Unit in SysML

The MicroController QSyMComponent still consists of one normal behaviour state machine and one failure pattern state machine. In the normal behaviour state (see Figure 6.1) after every 20ms passed time the Microcontroller evaluates if a crash condition exists. Two thresholds are checked, the MainSensorAcceleration and the SafetySensorAcceleration. Whenever both of them are equal or above the threshold (in that case >= 3) the criticalCrashLevel value is increased by one and whenever this level is equal or higher than 3 a transition to the Crash state is executed, thus a critical crash occurred and the deployment of the airbag has to be executed. This happens by arming the FASIC in state one, then enable the FET in state two and in end fire the FASIC. The failure pattern state machine represents the behaviour of an inadvertently deployed airbag that can happen anytime with a rate specified in the MicroController Component for the QSyMAbstractFailureTransition. It executes the same sequence of states the Crash state does and is shown in Figure 6.3.
Figure 6.2: MicroController normal behaviour state machine
The *FASIC* Component has also one failure pattern (Figure 6.5) and one normal behaviour state machine (Figure 6.6). The failure pattern for the *FASIC* has three different failure modes than can be entered after the failure state is entered with the rate specified throughout the *QSyMAbstractFailureTransition*. *FASICStackLow* is the failure mode which occurs if *FASIC* gets never fired (fasicFired=false) and therefore the airbag cannot be fired. *FASICShortage* is the failure mode where the mechanism which should protect the *FET* is disabled and the airbag is fired. Finally the *FASICStackHigh* state represents the failure mode whenever *FET* was enabled before and the *FASIC* is fired. Each of these failure modes is entered with a specific rate, defined in the corresponding *QSyMFailureTransition*. Additionally, in order to analyse the specific cases when the airbag is deployed inadvertently the failure states where the *FASIC* is fired are tagged with a *StateConfiguration* named inadvertent_deployment and the or-operator (Figure 6.4).

![Figure 6.3: MicroController failure pattern state machine](image)

![Figure 6.4: State configuration attributes](image)
Figure 6.5: FASIC failure pattern state machine

Figure 6.6: FASIC normal behaviour state machine
There are no failure patterns specified for the sensors, because the goal for Leitner-Fischer was to analyze the correct behavior of the other three components. Therefore the final component interesting for the analysis is the FET. The normal behavior of the FET switches between the Enabled and Disabled state (Figure 6.7). Initially disabled, the Enabled state is triggered whenever the enableFET() operation is called. However, if it is already in the Enabled state and the operation triggers the transition to the Disabled state is taken. The failure pattern state machine of the FET, can be either in the $FETStuckHigh$ or in the $FETStuckLow$ state. Whereat in the former case the FET is enabled and in the latter case it is disabled, hence the variable $fetEnabled$ has the corresponding value (Figure 6.8). The rate for the FET failure pattern is set throughout the $QSyMAbstractFailureTransition$. The rates for the failure modes are specified in the $QSyMFailureTransitions$ leading to them.

![Figure 6.7: FET normal behaviour state machine](image)
As the resulting SysML model of the airbag system shows, there is no important difference to the model generated in the QuantUM approach. So in the end we proceeded with the export of the model into an XMI file and imported it in the QuantSyM tool. The translation to the PRISM model results in the same amount of states and transitions and the same CSL formula has been generated by the tool. We have also compared the PRISM code of the UML version with the SysML version and noted no differences.

In order to check the runtime for the computation of the counterexamples (Runtime CX) and the runtime for the fault tree generation (Runtime FT), we performed small tests on the QuantSyM tool. We kept the mission time $T=10$, $T=100$, and $T=1000$ since we wanted to see if the results differ from the QuantUM runtime tests. The experiments were performed on a PC with an Intel Core i7-720QM processor with 2.8 Ghz and 8 GBs of RAM. The results of the test are shown in Table 1.

<table>
<thead>
<tr>
<th>$T$</th>
<th>Runtime CX (sec.)</th>
<th>Paths in CX</th>
<th>Runtime FT (sec.)</th>
<th>Paths in FT</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>580.247 (approx. 9.67 min)</td>
<td>738</td>
<td>2.21</td>
<td>5</td>
</tr>
<tr>
<td>100</td>
<td>594.633 (approx. 9.91 min)</td>
<td>738</td>
<td>2.74</td>
<td>5</td>
</tr>
<tr>
<td>1000</td>
<td>772.319 (approx. 12.87 min)</td>
<td>738</td>
<td>3.12</td>
<td>5</td>
</tr>
</tbody>
</table>

Table 1: QuantSyM runtime test results for $T=10$, $T=100$ and $T=1000$

The results approve that the fault tree generation is build in a few sec-
Case Studies
Airbag Control Unit

onds whereas the computation of the counterexamples takes several minutes. Overall the running times keep the same for the QuantSyM approach of the airbag system.
7 Related Work

Beside the QuantUM approach there exist several other work in the literature that deal with quantitative analysis of UML models. However not many are this we found only one other approach for probabilistic model checking in SysML models. In this chapter we will discuss its procedural method and compare it to QuantSyM, with the goal to find possible improvements or ways to integrate the approach.

7.1 Probabilistic Model Checking of SysML Activity Diagrams

In SysML, the in UML established activity diagrams have been extended to probabilistic features. The authors of [23] use that feature and propose a translation from SysML activity diagrams into the input language of the probabilistic model-checker PRISM.

The flow and control of data during the systems operations can be specified in the activity diagram in SysML. The "Activity Diagram Translation" then uses Markov decision process [25] (MDP) to analyze probabilistic and non-deterministic behaviours. There is no introduction of new stereotypes, instead the translation to PRISM is ensured through an algorithm which enables the transformation of activity diagrams directly into a MDP. Therefore there is no need to tag additional values and annotations to the model. The major drawback of this approach lies in the focus on the solely use of the activity diagram. Hence the only used aspect of the SysML is the behavioral structure and the requirement as well as structural aspects of SysML are left behind.

The QuantSyM approach uses primarily state machines to model the behavioral part of the system. Although we use the structural, behavioral and requirement aspect of SysML we generally use state machines and block diagrams for our approach. However, there are several modeling elements we did not take in consideration (i.e. activity diagrams). Because of the fact that some of the control flows are difficult to model with state machines, the integration of the activity diagram into the QuantSyM approach can be very valuable for system engineers. In order to achieve that, the basic concept of the state machine could be assigned to the activity diagram. For instance objects and activities could act like a state did in QuantSyM. Conditional branches could be realized by translating them as transitions with guards et cetera. At first glance the applicability seems to be possible, but further examination is necessary to verify that.
8 Conclusion

8.1 Conclusion

The adaption of QuantUM to SysML has been successful. Therefore the introduced QuantSyM profile allows to specify quantitative information in SysML models. The tool has been upgraded to the possibility to automatically translate a SysML model into the PRISM language and perform the a quantitative analysis with PRISM. Several additional stereotypes have been added to enhance the modeling possibilities to fulfill system engineers needs. The automatic generation of CSL properties has been further improved, and furthermore, the process chain in QuantUM and the requirements to the tool have been maintained. Hence the results keep the established fault trees, containing the crucial information about critical behavior.

Finally, the improved QuantSyM tool allows for a fully automated probabilistic analysis of SysML models, where the analysis level is complete hidden from the user. The implementation of the approach on the case study shows that the tool can be very useful to analyse system critical processes. With the fast translation of the model into PRISM and automatic generation of CSL formulas, the tool can solve tasks where system engineers usually spent hours of time.

8.2 Future Work

In future work we plan to evaluate how the QuantSyM / QuantUM approach can be integrated into an industrial process model. So far the model has been created and exported to XMI with IBM Rational Rhapsody, but more industrial tools for life-cycle and requirements management exist. Hence we plan to integrate the tool chain to other tools as well. In spite of all opportunities of the tool, the fact that the system engineering domain keeps developing further and systems increase in complexity, the tool needs to be tested for its applicability on real data and systems. Finally, we want to analyze the possible integration of modeling elements that haven’t been used for the analysis yet.
9 Appendix

9.1 CD
References


[25] Handbook of Markov Decision Processes Methods and Applications Eugene A. Feinberg,SUNY at Stony Brook, USA, Adam Shwartz,Technion Israel Institute of Technology, Haifa, Israel