Entwurf und Analyse sicherheitsrelevanter Kommunikationsarchitekturen

Dem Fachbereich Mathematik/Informatik der
Universität Bremen
zur Erlangung des akademischen Grades eines
Dr.-Ing.

eingereichte Dissertation

von
Herr M.Sc. Oliver Schulz
aus
Hildesheim

Datum der Einreichung: 2011/01/31

Referent: Prof. Dr. rer. nat. habil. Jan Peleska
Koreferent: Prof. Dr. rer. nat. Jens Braband

Tag der mündlichen Prüfung: 2011/05/03

Abstract

In the future, system engineers of railway control systems have to consider digital communication between components more than ever. The number of communicating systems rises steadily, while the offer of commercial off the shelf products for digital transmission networks is also growing. Safety protocols be included in the architecture in order to guarantee that safety-critical systems could be used on unsafe transmission channels. Safety layers have to detect different types of message errors to grant functional safety. It is highly recommended to prove the specification of a safety protocol with model checking methods to ensure a correct specification. The safety reaction on such errors must be a safe state, which usually stops the communication service until the system is reinitialised or reset by an operator. Therefore a safe communication reduces the fault tolerance against arbitrary transmission errors and lowers the reliability of the communication architecture. To improve the fault tolerance against message errors it is necessary to use a reliable message transmission service before the safety check is executed. A reliable transmission service can be included in the safety layer, in the upper protocol layer of the grey channel or in both layers. A naive combination of fault-tolerance mechanisms in the grey channel and safety layers will not necessarily increase the overall fault-tolerance: if, for example, lost messages in the grey channel lead to retransmissions after timeouts, the message eventually passed to the receiving safety layer may be out-dated and therefore has to be discarded. As a consequence, it is necessary to perform analyses whether the design of safety related communication architectures is safe and reliable.

This thesis describes a common concept for reliability and safety analysis of communication architectures in safety-critical systems. Case studies of industrial sized communication architectures evaluate this new approach. Besides, the analysis results are used to improve the design.

Keywords:

Communication Architecture, Railway Interlocking, Safety Protocol, Domain Specific Modeling, Reliability Analysis, EN 50159

Zusammenfassung

In der Zukunft müssen Systemdesigner von Streckensicherungstechnik die digitale Kommunikation zwischen Komponenten der Eisenbahndomäne mehr denn je berücksichtigen. Die Anzahl von kommunizierenden Systemen steigt stetig, während gleichzeitig Commercial-off-the-Shelf Produkte für digitale Übertragungsnetze ebenfalls zunehmen. In der Kommunikationsarchitektur werden Safety-Protokolle eingesetzt, so dass nicht sichere Übertragungskanäle verwendet werden können. Die funktionale Sicherheit wird dann vom Safety-Layer garantiert. Dieser Safety-Layer muss verschiedene Fehlerarten bei der Übertragung aufdecken. Es ist empfohlen, die Spezifikation eines Safety-Protokolls mit Model Checking Methoden zu überprüfen, um die korrekte Reaktion auf Übertragungsfehler nachzuweisen. Bei der Reaktion auf Übertragungsfehler wird ein sicherer Zustand eingenommen, in dem üblicherweise der Kommunikationskanal geschlossen wird, bis das System reinitialisiert wird. Der sichere Zustand reduziert damit die Fehlertoleranz gegenüber zufälligen Übertragungsfehler und wirkt sich nachteilig auf die Zuverlässigkeit Kommunikationsarchitektur aus. Um die Fehlertoleranz zu verbessern wird ein zuverlässiger Übertragungsdienst verwendet, bevor der Safety-Check ausgeführt wird. Ein zuverlässiger Übertragungsdienst kann dabei in das Safety-Protokoll oder in unterlagerte Protokollschichten nicht-sicherer Komponenten integriert werden. Eine ungeeignete Kombination von Mechanismen steigert jedoch nicht die Fehlertoleranz: Wenn zum Beispiel verlorene Nachrichten durch ungeeignete Mechanismen nicht rechtzeitig wiederholt werden, sind Echtzeitanforderungen nicht erfüllt, was ebenfalls zur Sicherheitsreaktion und damit dem Beenden der Kommunikation führt. Demnach ist es notwendig, das Design von sicherheitsrelevanten Kommunikationsarchitekturen auf Sicherheit und Zuverlässigkeit zu überprüfen.

In dieser Arbeit wird ein Konzept zur kombinierten Analyse von Sicherheits- und Zuverlässigkeitseigenschaften beschrieben. Im letzten Teil dieser Arbeit sind Fallstudien von realen Kommunikationsarchitekturen aufgeführt, die diesen neuen Ansatz evaluieren. Zudem liefern die Analyseergebnisse Hinweise auf Schwachstellen im Design, wodurch die Architekturen verbessert werden.

Schlagwörter:

Kommunikationsarchitektur, Streckensicherungstechnik, Safety-Protokoll, domänenspezifische Modellierung, Model Checking, Zuverlässigkeitsanalyse, EN 50159

Danksagung

Die vorliegende Arbeit entstand während meiner Tätigkeit als Stipendiat an der Universität Bremen im Fachbereich Informatik. Meine Forschungsgruppe war die Arbeitsgruppe Betriebssysteme (AGBS), unter der Leitung von Prof. Dr. Jan Peleska. Für die Aufnahme in die AGBS und das entgegengebrachte Vertrauen möchte ich mich sehr bedanken. Die großartige Unterstützung, die vielen Anregungen und Leitimpulse verhalfen mir zu einer geraden Linie während meiner Forschungen. Ich möchte mich ebenfalls bei meinem Zweitgutachter Prof. Dr. Jens Braband für das Stipendium von der Siemens AG herzlich bedanken. Zudem waren die regelmäßigen und gut organisierten Workshops ein ideales Forum zum Ideenaustausch mit Anderen und zur Kontrolle der eigenen Arbeit. Ich bin sehr froh, über das RA!GS Programm von Siemens eine Beschäftigung im Unternehmen gefunden zu haben.

Zum Gelingen der Arbeit hat maßgeblich auch Dr. Peter Ziegler von der Siemens AG beigetragen, wofür ich ebenfalls großen Dank aussprechen möchte. Die vielen Hintergrundinformationen, regelmäßigen und intensiven Gespräche verhinderten Irrwege und sorgten stets dafür, das Ziel im Blick zu halten. Zudem entstand dadurch der wichtige praktische Aspekt dieser Arbeit.

Des Weiteren möchte ich mich bei meinen Kollegen aus der AGBS bedanken, mit denen ich Ideen austauschen und Fragen klären konnte. Mein Dank gilt auch meiner Familie, meiner Freundin Kim Anne und allen anderen Freunden, die mich während dieser nicht immer einfachen Zeit unterstützt haben. Für die Durchsicht dieser Arbeit und die Korrekturvorschläge möchte ich mich bei meiner Mutter, Peter, Florian, Elena, Tanja und Berit bedanken.

Bremen, im Januar 2011

Oliver Schulz

Inhaltsverzeichnis

1 Einleitung **1**

1.1 Motivation . . . 2
1.2 Verwandte Ansätze . . . 3
1.3 Lösungsansätze und Übersicht dieser Arbeit . . . 4
1.4 Wissenschaftliche Beiträge . . . 5

2 Technische Grundlagen **7**

2.1 Systemarchitektur Streckensicherungstechnik . . . 7
2.1.1 Die Leittechnik . . . 10
2.1.2 Die Stellwerke . . . 10
2.1.3 Die Außenanlage . . . 10
2.2 Anforderungen an Kommunikationsarchitekturen . . . 11
2.2.1 Normative Anforderungen . . . 12
2.2.2 Safety . . . 16
2.2.3 Security . . . 18
2.2.4 Reliability . . . 19
2.3 Fault, Error, Failure . . . 20
2.3.1 Verwendung der Begriffe in Bezug auf Kommunikationsarchitekturen 22
2.3.2 Message Errors - Übertragungsfehler . . . 23
2.4 Zuverlässigkeitsmechanismen . . . 24
2.4.1 Redundanz . . . 25
2.4.2 Quittierungsverfahren . . . 25
2.4.3 Übertragungswiederholung . . . 26
2.4.4 Flusskontrolle . . . 26
2.4.5 Staukontrolle . . . 27
2.5 Kommunikationstechnik und Trends . . . 28
2.5.1 Hard und Software von Endgeräten . . . 29
2.5.2 Lokale Netze (LAN) . . . 32

2.5.3 Netzzugangstechnik . . . 32
2.5.4 Backbonenetze . . . 33
2.5.5 DB Systel propagiert All-over-IP Netzwerke . . . 33
2.6 Zusammenfassung . . . 34

3 Grundlagen zur Modellierung und formale Verifikationsmethoden 37
3.1 Entwurf von Kommunikationsarchitekturen . . . 38
3.2 Analyse- und Beschreibungsmethoden . . . 39
3.3 Domänenspezifische Modellierung . . . 40
3.4 Domänenspezifische Modellierung mit MetaEdit+ . . . 41
3.5 Model Checking und Tools . . . 42
3.6 Model Checking und Temporallogik . . . 44
3.7 Der Uppaal Model Checker . . . 45
3.7.1 Beispiel Ampelsteuerung . . . 45
3.7.2 Formale Syntax und Semantik des Uppaal Model Checkers . . . 46
3.7.3 TCTL Temporallogik von Uppaal . . . 51
3.8 Der probabilistische Model Checker PRISM . . . 53
3.8.1 PRISM Modelle . . . 54
3.8.2 Formale Syntax und Semantik des PRISM Model Checkers . . . 55
3.9 Quantitative und Qualitative Verifikation . . . 57
3.9.1 Der Unterschied zwischen TCTL und PCTL . . . 58

4 Analyse sicherheitsrelevanter Kommunikationsarchitekturen 61
4.1 Generische Modellierung von Kommunikationsarchitekturen . . . 62
4.2 Analyse-Framework . . . 65
4.3 Communication Architecture Modelling Language . . . 68
4.3.1 CAMoLa - System-Ebene . . . 68
4.3.2 CAMoLa - Prozessspezifikation . . . 70
4.4 Workflow Phase 1: Modellieren von Kommunikationsarchitekturen und Formalisieren von Anforderungen . . . 76
4.4.1 Modellieren von Kommunikationsarchitekturen . . . 77
4.4.2 Abstraktionen . . . 77
4.4.3 Formalisieren von Anforderungen . . . 79
4.4.4 Design Transformation und Modellverifikation . . . 80
4.5 Phase 2: Verifikation qualitativer Eigenschaften . . . 82
4.6 Phase 3: Analyse quantitativer Eigenschaften . . . 83
4.6.1 Fehlerwahrscheinlichkeit aufgrund der Fehlerhypothese . . . 88

4.6.2 Beispiel zur quantitativen Analyse (Phase 3) . . . 91
4.6.3 Verteiltes Model Checking mit Modell Mutationen . . . 93
4.6.4 Auswerten der Ergebnisse . . . 94

5 Design und Analyse sicherheitsrelevanter Kommunikationsarchitekturen: Fallstudien 97
5.1 Das SAHARA Protokoll . . . 98
5.1.1 Funktionsweise von SAHARA . . . 98
5.1.2 Modellierung und Formalisierung von Anforderungen . . . 99
5.1.3 Abstraktionen und Fehlerhypothesen . . . 99
5.1.4 Ergebnisse der Phase 2 . . . 103
5.1.5 Analyse und Ergebnisse der Phase 3 . . . 104
5.1.6 Komplexität der Verifikation . . . 106
5.2 Das Safety Protocol HASP . . . 108
5.2.1 Ergebnisse der Phase 2 . . . 109
5.2.2 Analyse und Ergebnisse der Phase 3 . . . 111
5.3 Die SAHARA-SCTP Architektur . . . 113

6 Zusammenfassung und Ausblick 119
6.1 Die generische Modellierung sicherheitsrelevanter Kommunikationsarchitekturen . . . 119
6.2 Kombination von UPPAAL und PRISM . . . 120
6.3 Analyse sicherheitsrelevanter Kommunikationsarchitekturen . . . 120
6.4 Ergebnisse der Fallstudien . . . 121
6.5 Ausblick . . . 122
6.5.1 Verifikation von Safety mit Modellmutationen . . . 124

A Fallstudie: Kombination von UPPAAL und PRISM 125
A.1 CMM - Common Meta Model . . . 126
A.1.1 Timed Location. . . . 128
A.1.2 Probabilistic Transitions. . . . 128
A.1.3 Synchronisation Transitions. . . . 128
A.1.4 Parallele Prozesse. . . . 129
A.2 Bewertung der Kombination. . . . 129

B Architekturmodelle 131
B.1 SAHARA Architekturmodell . . . 131
B.2 HASP Architekturmodell . . . 132

B.3 SAHARA-SCTP Architekturmodell 133

C Prozessmodelle **137**

C.1 Prozess Watchdog 137

C.2 Prozess Application 137

C.3 Prozess Buffer 138

C.4 Prozess SCTP_AckTimer und SCTP_BundlingTimer 140

C.5 Prozessmodell des HASP Protokolls 140

C.6 Prozessmodell des SCTP Protokolls 147

Abbildungsverzeichnis

2.1 Systemarchitektur der Streckensicherungstechnik 8
2.2 Geltungsbereich ausgewählter Normen (entnommen aus [BBSGS06]) 14
2.3 Auswirkungen von Ausfällen innerhalb eines Systems (aus [CEN00]) 15
2.4 Referenzarchitektur nach EN 50159 (aus [CEN10]) 17
2.5 Zusammenhang zwischen Übertragungsfehlern und Zuverlässigkeit 21
2.6 Fehlerfortpflanzung: Fault, Error, Failure 22
2.7 Fehlerarten bei der Datenübertragung 23
2.8 Token Bucket Filter 28
2.9 Klassifizierung von Kommunikationstechniken 29

3.1 Prinzip des Model Checkings 43
3.2 Systemmodell einer Ampelsteuerung 47
3.3 TCTL Aussagen über den Computation Tree eines Modells M 52
3.4 Qualitative versus quantitative Eigenschaften 58
3.5 Der schwedische Koch und seine Mettbällchen 59

4.1 Modellbasierte Verifikation von Safety-Protokollen 63
4.2 Einordnung der Modellierung in des V-Modell 64
4.3 Verifikation und Analyse von Architekturmodellen 65
4.4 Mögliche Systemzustände nach dem Auslösen einer Transaktion 66
4.5 Modellierung, Verifikation und Analyse, Workflow 67
4.6 CAMoLa - Systemspezifikation 68
4.7 CAMoLa Fault-Switch 74
4.8 Trace-Controlled Fault-Switch 75
4.9 Übertragungskanal mit generischen Fehlerarten 76
4.10 Transformation von CAMoLa Modellen 81
4.11 Generieren und Prüfen von Modell Mutationen 87
4.12 Beispiel eines Mutationsbaums, mit $\mathcal{FS} = \{0, 1\}, b_0 = \infty, b_1 = 3$ 90

4.13 Mutationsbaum des Beispiels 92
4.14 Verteiltes Model Checking von Mutationen 94

5.1 Vereinfachte Darstellung der SAHARA Zustandsmaschine 100
5.2 Skizze der SAHARA Architektur 100
5.3 Ermittelte Zuverlässigkeitsfunktion von SAHARA 106
5.4 Ausfallwahrscheinlichkeit pro Stunde der SAHARA Kommunikation 107
5.5 Skizze der HASP Architektur 109
5.6 Erfolgreiche Übertragungswiederholung des HAS Protokolls 110
5.7 Übertragungswiederholung mit Verlust von zwei Nachrichten 110
5.8 Ermittelte Zuverlässigkeitsfunktion des Hybrid Acknowledge Safety Protocol 113
5.9 Ausfallwahrscheinlichkeit pro Stunde. Vergleich zwischen der HASP- und der SAHARA Architektur 113
5.10 Skizze der SAHARA-SCTP Architektur 115
5.11 Ermittelte Zuverlässigkeitsfunktion der SCTP Architektur 117
5.12 Ausfallwahrscheinlichkeit pro Stunde. Vergleich zwischen der SCTP und der HASP Architektur 117

6.1 Vergleich der Zuverlässigkeitsfunktionen von SAHARA, HASP und SCTP . 123
6.2 Ausfallwahrscheinlichkeit pro Stunde von SAHARA, HASP und SCTP . . . 123

A.1 CMM - Gemeinsames Metamodell für Uppaal und PRISM 127

B.1 Modell der SAHARA Architektur 132
B.2 Modell der HASP Architektur 133
B.3 Modell der SCTP Architektur 135

C.1 Modell des Prozesses Application 137
C.2 Modell des Prozesses Application 138
C.3 Modell des Prozesses Buffer 139
C.4 Modell der Timer Prozesse von SCTP 140
C.5 Modell des Porokolls HASP 141
C.6 Modell des Porokolls SCTP 147

Tabellenverzeichnis

2.1 Ausgewählte CENELEC Standards der Eisenbahndomäne (aus [Fen07]) . . 13
2.2 Gefahren und Abwehrmaßnahmen der Datenübertragung 16
2.3 Bitfehlerwahrscheinlichkeiten verschiedener Übertragungstechnologien . . . 24
2.4 Vergleich von SCTP, TCP und UDP 31

4.1 Prozesse der Beispiel-Architektur 69
4.2 Transformation von Locations und Transitionen 72

5.1 Verifikation der SAHARA-Architektur 104
5.2 Parameter der quantitativen Analyse 105
5.3 Komplexität der SAHARA-Architektur 108
5.4 Verifikation der HASP-Architektur 111
5.5 Komplexität der HASP-Architektur 112
5.6 SCTP - Spezifische Parameter 115

Abkürzungsverzeichnis

BER ..... Bit Error Ratio
BMC ..... Bounded Model Checking
CAMoLa .. Communication Architecture Modelling Language
CDD ..... Clock Difference Diagrams
CENELEC. Comité Européen de Normalisation Électrotechnique
CMM ..... Common Meta Model
CC ..... Congestion Control
COTS ..... Commercial off the Shelf
CRC ..... Cyclic Redundancy Check
CTL ..... Computation Tree Logic
CTMC .... Continuous-Time Markov Chains
DBM ..... Difference Bound Matrix
DOCSIS ... Data Over Cable Service Interface Specification
DoS ..... Denial of Service
DSL ..... Domain Specific Language
DTMC .... Discrete-Time Markov Chains
xDSL ..... Digital Subscriber Line
FIFO ..... First In First Out
FPGA ..... Field Programmable Gate Array
FTTH ..... Fibre to the Home
GOPPRR . Graph Object Property Port Role Relationship
GPL ..... General Purpose Language
HASP ..... Hybrid Acknowledge Safety Protocol
ISDN ..... Integrated Services Digital Network
IP ..... Internet Protocol
ISO ..... International Standards Organization
LTL ..... Linear Temporal Logic
LZB ..... Linienförmige Zugbeeinflussung

MDP ..... Markov Decision Processes
MEF ..... Metro Ethernet Forum
Merl..... MetaEdit+ Reporting Language
MPLS..... Multiprotocol Label Switching
MSTT .... modulares Stellteil
MTBDD .. Multi Terminal Binary Decision Diagram
NAU ..... Network Access Unit
OSI ..... Open Systems Interconnection
PCTL..... Probabilistic Computation Tree Logic
PDU ..... Process Data Unit
PMC..... Probabilistic Model Checking
Promela... Process Meta Language
RBC ..... Radio Block Center
RTT ..... Round-Trip-Time
SAHARA.. Safe Highly Available and Redundant
SCTP..... Stream Control Transmission Protocol
SDL..... Specification and Description Language
SDH ..... Synchronous Digital Hierarchy
SIL..... Safety Integrity Level
SIMIS-W.. "sicheres Mikrocomputer System von Siemens für den Weltmarkt"
TBF..... Token Bucket Filter
TCP ..... Transmission Control Protocol
TCTL..... Timed Computation Tree Logic
THR..... Tolerable Hazard Rate
UDP..... User Datagram Protocol
UML..... Unified Modeling Language
XML..... Extensible Markup Language

Kapitel 1

Einleitung

Es kommt nicht darauf an, mit dem Kopf durch die Wand zu rennen, sondern mit den Augen die Tür zu finden.

Werner von Siemens, 1886 - 1892

Moderne Kommunikationsnetzwerke für die Datenübertragung haben in den letzen Jahren neue Möglichkeiten und kostengünstige Lösungen für vernetzte Systeme im Industrie- und Automobilbereich hervorgebracht. In dem traditionell konservativen Bereich der Eisenbahnautomatisierung steht dieser Wandel in der Kommunikationstechnik unmittelbar bevor. Verscheide Hersteller dieser Branche bemühen sich derzeit, die vorhandenen Kommunikationssysteme zu vereinheitlichen und mit einer herstellerübergreifenden Kompatibilität die Chancen kostengünstiger Lösungen zu nutzen. Diese Interoperabilität entspricht den Kundenwünschen und gewinnt im Zuge von Modernisierungen der Streckensicherungstechnik zunehmend an Bedeutung. Gleichzeitig werden aufgrund der fortschreitenden Miniaturisierung und stark sinkender Kosten von digitalen Komponenten, klassische analoge Feldelemente durch Mikrocontroller unterstützte Systeme abgelöst. Mit modernen Backbonenetzen, die weltweit bei Bahnbetreibern entlang von Schienenwegen aufgebaut werden, wird die Möglichkeit eröffnet, bislang begrenzten Entfernungen zwischen den Systemen aufzuheben. In der Streckensicherungstechnik gelten jedoch andere Anforderungen an die digitale Kommunikation, als bei der klassischen Industrieautomatisierung. Während üblicherweise geringere Echtzeitanforderungen gestellt werden, so gelten höchste Anforderungen an Sicherheit und Zuverlässigkeit. Insgesamt machen die Anforderungen der Streckensicherungstechnik das Definieren von speziellen Kommunikationsarchitekturen nötig. Weitestgehend soll dabei von Commercial off the Shelf (COTS) Technik profitiert werden. Mit den vielfältigen am Markt erhältlichen Standardtechniken und den nötigen Kommuni-

kationsprotokollen zum Garantieren von Sicherheit und Zuverlässigkeit, sind viele Architekturszenarien möglich. Es ist daher nötig, verschiedene Szenarien genauer zu analysieren, um Kommunikationsprotokolle optimal auf die Gegebenheiten von Übertragungsnetze und Infrastrukturen abzustimmen. Dieses soll kostspielige spätere Änderungen aufgrund von Designschwächen verhindern. Hierbei werden die gegenläufigen Aspekte Sicherheit (Safety) und Zuverlässigkeit (Reliability) von sicherheitsrelevanten Kommunikationsarchitekturen betrachtet. Safety-Mechanismen garantieren den sicheren Austausch von Prozessdaten. Zufällige Übertragungsfehler, wie zum Beispiel verfälschte Bits oder ein verlorenes Datenpaket, müssen von einem Safety-Protokoll erkannt werden. Diese Fehlerarten dürfen die Sicherheit des Automatisierungsprozesses nicht gefährden. Das System nimmt einen sogenannten sicheren Zustand ein, was zum Beispiel das Sperren des Streckenabschnitts bedeutet. Dieser sichere aber unerwünschte Zustand reduziert die Verfügbarkeit der automatisierten Streckensicherungstechnik. Neben der Garantie von Safety sind daher Mechanismen nötig, die die Übertragungsfehler korrigieren und somit die Zuverlässigkeit erhöhen. Je nach Art des Übertragungsfehlers und des Zuverlässigkeitsmechanismus begrenzen Echtzeitanforderungen und weitere Sicherheitsbedingungen die tolerierbare Häufigkeit der Fehler.

1.1 Motivation

Das komplexe Zusammenspiel zwischen Fehlerarten, Zuverlässigkeits- und Sicherheitsmechanismen kann nur mit Computergestützten Methoden analysiert werden. Da Sicherheitseigenschaften nachzuweisen sind und diese Analysen zu einem frühen Design-Zeitpunkt durchgeführt werden, ist das Model Checking eine adequate Methode. Das Model Checking wird seit den 70er Jahren bereits durchgeführt und stetig weiterentwickelt. Im Gegensatz zum universitären Umfeld, etablieren sich im industriellen Bereich Model Checking Methoden nur langsam. Die erforderlichen speziellen Kenntnisse sowie die für Systemdesigner unbrauchbaren und rudimentären Tools verhinderten bislang eine großflächige Anwendung im “Tagesgeschäft”. Zudem ist das Model Checking aufgrund der hohen Komplexität der Systeme nur eingeschränkt anwendbar. In den letzten Jahren gab es einige Verbesserungen der Werkzeuge und der dahinter stehenden Theorien, so dass sich neue Möglichkeiten eröffnen, das Model Checking für praxisgerechte Kommunikationsarchitekturen anzuwenden.

Die Herausforderung dieser Arbeit besteht darin, ein Framework zu entwickeln, das eine intuitive Modellierung von praxistauglichen Kommunikationsarchitekturen ermöglicht. Des Weiteren steht die Kombination aus Safety-Verifikation und Zuverlässigkeitsanalyse gemeinsamer Modelle im Vordergrund. Die Reduktion der Modellierung durch die kombi-

nierte Analyse verringert den zeitlichen Aufwand und erhöht die Akzeptanz, dieses Framework für Kommunikationsarchitekturen entwicklungsbegleitend einzusetzen. Im Vordergrund steht hierzu der Entwurf einer domänenspezifischen Modellierungssprache, die eine verständliche und grafische Modellierung ermöglicht. Modelle dieser Sprache werden in einem automatischen Schritt für weitere Betrachtungen in entsprechende Analysemodelle transformiert.

Derzeit befindet sich ein Safety-Protokoll für den Bahnbereich im Standardisierungsprozess. Dieses, empirisch entworfene Protokoll, ist auf Safety- und Reliability-Eigenschaften zu analysieren. Die gewonnenen Ergebnisse tragen zur Verbesserung der Architektur in zweierlei Hinsichten bei: Zum einen werden Designschwächen in Bezug auf Safety und Reliability ermittelt und zum anderen zeigen die Ergebnisse der Zuverlässigkeitsanalyse die Anforderungen an unterlagerte Übertragungsnetze auf. Letzteres ist ein wichtiger Hinweis auf Infrastrukturen, die in die sicherheitsrelevante Kommunikationsarchitektur integriert werden können. Gleichzeitig stellt die Architektur mit einem realen Safety-Protokoll eine Evaluation des eingesetzten Frameworks dar. Mit der Demonstration über die Eignung zum Nachweisen der genannten Eigenschaften, gilt das Framework als praxistauglich.

Ein wichtiger Punkt bei der Verifikation und Analyse ist das Berücksichtigen von Übertragungsfehlern. Im Gegensatz zu simulativen Ansätzen ist die auftretende Fehlerkombinatorik beim Model Checking ein Problem, das bisherige Verfahren vor große Herausforderungen stellte. Ein neuer Ansatz ist hierbei die Grundlage für weitere Forschungen, dem kombinatorischen Problem zu begegnen.

1.2 Verwandte Ansätze

Diese Arbeit stützt sich auf drei wesentliche Bereiche: Das Erstellen eines domänenspezifischen Modells, Verifikation von Safety-Eigenschaften und die Analyse von Zuverlässigkeitseigenschaften von Kommunikationsarchitekturmodellen. Eine domänenspezifische Sprache wird von einem Domänenexperten entworfen, um eine an die Domäne angepasste Modellierungssprache zu definieren. Mit dem Erstellen einer Domain Specific Language (DSL) kann das Abstraktionsniveau ohne überflüssige Elemente an die Domäne angepasst werden. Seit den 90er Jahren verbesserte sich vor allem die Tool-Unterstützung im Bereich der grafischen Meta-Modellierungssprachen [KLR96, KT08]. Der Ansatz, die Modellierung von Kommunikationsarchitekturen mit einem Metamodell zu ermöglichen, profitiert von den praktischen Erfahrungen und Diskussionen in der Arbeitsgruppe Betriebssysteme der Universität Bremen. Die domänenspezifische Modellierung ist zum Beispiel in [PBH00, HP07, Mew09] untersucht worden und stellte sich mit begleitenden Fallstudien zu [SP10] als geeignete Beschreibungsmethode heraus.

Der zweite Bereich dieser Arbeit, die formale Verifikation von Eigenschaften reaktiver Systeme, ist in einer Vielzahl von Publikationen, wie zum Beispiel [Pin02, Wan04, Mos09] beschrieben. Im Vordergrund stehen dabei Echtzeiteigenschaften, die mit Timed Automata Tools modelliert und verifiziert werden. Die formale Verifikation von Kommunikationsprotokollen und Architekturen ist zum Beispiel in [DKRT97, DY00, MM02, SB07] nachzuschlagen.

Der dritte Bereich dieser Arbeit beschäftigt sich mit der Analyse von Zuverlässigkeitseigenschaften von Kommunikationsarchitekturen. Die Zuverlässigkeit ist von probabilistischer Natur, so dass für die formale Verifikation ein probabilistisches Modell und ein probabilistischer Model Checker benötigt wird. In zum Beispiel [MS87, KNS02, DFH+04] ist ein solches Vorgehen genannt. Die Kombination von Timed Automata und Probabilistic Model Checking (PMC) ist ein aktuelles Forschungsthema. So wird zum Beispiel in [DKN+07] ein Probabilistic Timed Automata vorgeschlagen, um Real-Time und probabilistische Eigenschaften in Kombination zu verifizieren[1]. Aufgrund der Komplexität sind bei dieser Methode starke Einschränkungen nötig. Diese Dissertation nimmt sich dem Problem an und beschreibt ein Framework, mit dem Kommunikationsarchitekturen auf einem geeigneten Abstraktionslevel modelliert und anschließend auf Basis formaler Verifikationsmethoden sowohl qualitative Eigenschaften, als auch die probabilistische Eigenschaft Zuverlässigkeit von Kommunikationsarchitekturen verifiziert werden.

1.3 Lösungsansätze und Übersicht dieser Arbeit

Im Kapitel 2 auf Seite 7 ist in Grundzügen die Streckensicherungstechnik beschrieben sowie Rahmenbedingungen für Kommunikationsarchitekturen. Hierzu gehören normative Anforderungen, Grundlagen von Kommunikationsarchitekturen und eine Auswertung von Trends in der Kommunikationstechnik. Das Kapitel 3 auf Seite 37 erläutert Beschreibungsmethoden von Kommunikationsarchitekturen und Grundlagen zum Model Checking sowie die Model Checking Werkzeuge Uppaal (Timed Automata) und PRISM (Probabilistic Model Checker). Im Kapitel 4 auf Seite 61 ist ein neues Framework zum Modellieren und Verifizieren von Kommunikationsarchitekturen vorgestellt. Dieses kombiniert die qualitative Verifikation und die quantitative Analyse von Architekturen und definiert damit eine Methode für das Design und Analyse sicherheitsrelevanter Kommunikationsarchitekturen. Die Basis ist die domänenspezifische Modellierungssprache Communication Architecture Modelling Language (CAMoLa), wobei CAMoLa Modelle automatisch zu Verifikations-

[1] Im Dezember 2010 ist der probabilistische Model Checker PRISM für die Modellierung von Probabilistic Timed Automata erweitert worden [KNP09, KNP10]. Dieses konnte aufgrund der finalen Phase dieser Arbeit nicht mehr berücksichtigt werden.

modellen transformiert werden. Das Kapitel 5 auf Seite 97 wendet das Framework anhand realer Kommunikationsarchitekturen an. Die daraus gewonnenen Ergebnisse evaluieren die vorgestellte Methode und zeigen gleichzeitig Erkenntnisse zum Verbessern der eingesetzten sicherheitsrelevanten Kommunikationsarchitektur. Im letzen Teil (Kapitel 6 auf Seite 119) sind alle Ergebnisse noch einmal zusammengefasst und ein Ausblick beschreibt die identifizierten Möglichkeiten zur Verbesserung der vorgestellten Methode.

1.4 Wissenschaftliche Beiträge

Das in dieser Arbeit vorgestellte Framework ermöglicht die Entwicklung, Verifikation und Analyse von sicherheitsrelevanten Kommunikationsarchitekturen. Es handelt sich um eine Industrie-Promotion, so dass die praxisorientierte Modellierung mit der formalen Verifikation verbunden ist. Die wissenschaftlichen Beitrage dieser Arbeit sind:

1. Das Erstellen einer domänenspezifischen Sprache, die die Modellierung von Kommunikationsarchitekturen auf einem intuitiven Level und geeignetem Abstraktionsniveau ermöglicht.

2. Eine Fallstudie, in der Timed Automata und Probabilistic Model Checking kombiniert sind. Die Bewertung der Kombination führte zum Verwerfen dieses Ansatzes und ist daher im Anhang A auf Seite 125 als ungeeignet beschrieben.

3. Ein Konzept, das probabilistische Eigenschaften mit dem nichtprobabilistischen Timed Automata analysiert. Mit diesem Konzept wird durch Partitionierung des Zustandsraums in Subbereiche die prüfbare Modellkomplexität gesteigert und gleichzeitig, durch das Verteilen der Subbereiche auf mehrere Model Checking Prozesse, die parallele Verifikation von Modellen ermöglicht.

4. Fallstudien zeigen zuverlässige Mechanismen von (Safety-) Protokollen und deren Kombination unter Berücksichtigung von Rahmenbedingungen der Streckensicherungstechnik. Diese Fallstudien evaluieren das gesamte Framework an der Komplexität von industriellen Architekturen. Die Analyse von proprietären Protokollen zeigen Zuverlässigkeitsschwächen auf, die mit diesem Frameworks ermittelt wurden. Die Ergebnisse zeigen zudem eine Architekturvariante, die bei gleichen Safety-Mechanismen die Zuverlässigkeit deutlich steigert.

Kapitel 2

Technische Grundlagen

Man sollte alles so einfach wie möglich sehen - aber auch nicht einfacher.

Albert Einstein, 1879 - 1955

Dieses Kapitel gibt Hintergrundinformationen zu Kommunikationsarchitekturen in der Eisenbahndomäne sowie zu Kommunikationsmechanismen und -techniken vor. Zunächst wird die Hierarchie der Streckensicherungstechnik erläutert und mit der industriellen Automatisierungstechnik verglichen. Des Weiteren wird ein Überblick über die Applikationsanforderungen und normative Anforderungen an Kommunikationsarchitekturen für die Streckensicherungstechnik gegeben. Die normativen Anforderungen haben einen entscheidenden Einfluss auf den Entwicklungsprozess von Kommunikationsarchitekturen. Im weiteren Verlauf wird ein Überblick über derzeit aktuelle Kommunikationstechnologien gegeben, wobei zukünftig der Profit von COTS Produkten eine wichtige Rolle spielt. Beschrieben werden sowohl konkrete Technologien, als auch grundsätzliche Mechanismen zum Informationsaustausch.

2.1 Systemarchitektur Streckensicherungstechnik

Die Systeme der Streckensicherungstechnik gliedern sich in Deutschland in drei hierarchische Ebenen: Leittechnik, Stellwerkstechnik und Außenanlagen (siehe Abbildung 2.1 auf der nächsten Seite). In anderen Ländern sind ähnliche Konzepte zu finden. Die Systemarchitektur ist mit der Netzhierarchie aus den industriellen Kommunikationsebenen vergleichbar. In [Sch06] werden die drei hierarchischen Bereiche Prozessleitebene, Feld-Ebene und Sensor-Aktor-Ebene unterschieden. Jede Ebene stellt unterschiedliche Anforderungen

an die Kommunikationsinfrastruktur. Die aktuellen industrielle Feldbussysteme sind mit verschiedenen Ausprägungen an die Anforderungen der jeweiligen Ebene angepasst.

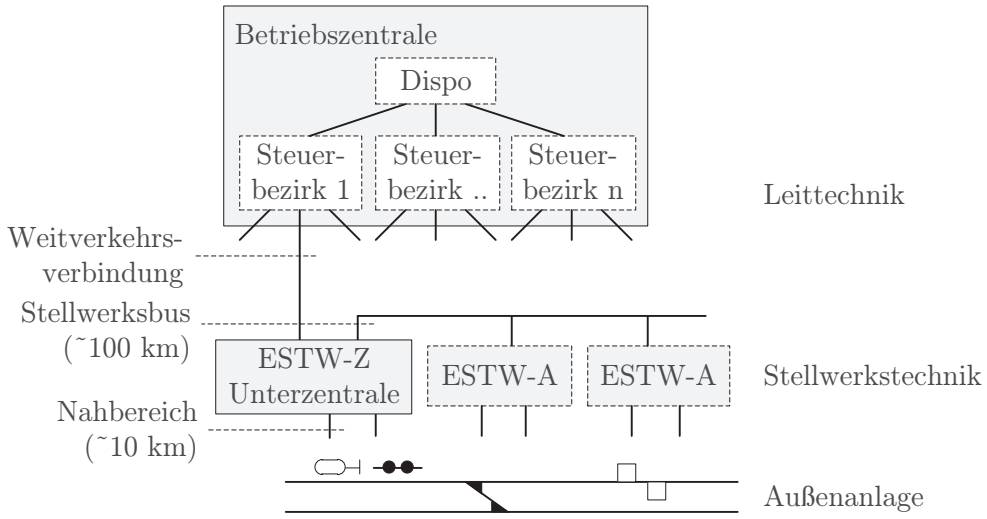


Abbildung 2.1: Systemarchitektur der Streckensicherungstechnik

Ein entscheidender Unterschied zwischen der Industrieautomatisierung und der Streckensicherungstechnik ist die geographische Ausdehnung. Die Rechnersysteme von Stellwerken sind über einen Stellwerksbus vernetzt. Es handelt sich dabei nicht um ein klassisches Bussystem, sondern um logische Punkt-zu-Punkt Verbindungen mit proprietären Übertragungsprotokollen, die über exklusive Telekommunikationskabel, öffentliche Netze (zum Beispiel G.703) oder Lichtwellenleiter getunnelt werden. Die in diesem Bereich eingesetzten proprietären Technologien verschiedener Rechnersysteme (Stellwerksrechner, Achszählrechner) sind in vielen Fällen nicht kompatibel, so dass jedes System über eigene physikalische Netze verfügt und mit Koppelschnittstellen mit anderen Systemen verbunden ist. Die Entfernungen zwischen zwei Stellwerken betragen zwischen 5 km und 10 km. Die Gesamtausdehnung kann sich über 100 km erstrecken. Aus diesem Grund werden andere Kommunikationstechniken verwendet, als sie in der Industrieautomatisierung verbreitet sind. Ein weiteres Merkmal sind höchste Anforderungen an Zuverlässigkeit und Sicherheit. Diese Eigenschaften sowie die schrittweise Modernisierung und Digitalisierung der Streckensicherungstechnik führte zu einer heterogenen Landschaft an Kommunikationssystemen, die keine gemeinsame Infrastruktur nutzen können.

Die Deutsche Bahn AG modernisiert zur Zeit die Leittechnik-Ebene. Zukünftig wird der Bahnverkehr in ganz Deutschland von sieben Betriebszentralen gesteuert und über-

wacht werden [Mur99]. Jede Betriebszentrale teilt sich in mehrere Steuerbezirke auf, die wiederum einer Unterzentrale Kommandos zur Zugführung geben. Die Unterzentrale steuert einen Streckenabschnitt, dem mehrere Stellwerke zugeordnet sind. Eine Unterzentrale (ESTW-Z) ist eine Erweiterung eines ESTW-A um einen (Not-) Bedienplatz, Zugleitsysteme und einen zentralen Stellwerksrechner. Stellwerke und Außenanlagen sind funktional sicher realisiert, so dass falsche Kommandos der Leittechnik nicht zu kritischen Zuständen in der Zugführung führen können.

Der gesamte Zugsteuerungsprozess verteilt sich auf mehrere vernetzte Systeme, die die geforderte hohe Zuverlässigkeit nur erreichen können, wenn auch die Kommunikation zwischen den Systemen eine hohe Zuverlässigkeit aufweist. Zur Vernetzung werden Lichtwellenleiter, das öffentliche oder bahneigene Telefonnetz (G.703 Schnittstelle) oder exklusive Telekommunikationskabel eingesetzt. Alle Lösungen haben entscheidende Nachteile: Exklusive Übertragungsmedien verursachen hohe Kosten, wenn diese mit einer zusätzlichen Baumaßnahme verbunden sind und die bittransparente G.703 Schnittstelle wird in der Zukunft durch paketorientierte Backbonenetze abgelöst. Anstelle eines Gateways für moderne Backbonenetze zu entwickeln und damit die Übertragungsnetze noch heterogener zu gestalten, wird eine neue Lösung gefordert: Die gesamte Kommunikationsarchitektur der Streckensicherungstechnik ist zu vereinheitlichen, so dass eine hohe Kompatibilität zwischen den Systemen gegeben ist und moderne Kommunikationstechnologien flexibel, transparent und zuverlässig genutzt werden können. Zudem ist mit standardisierten Schnittstellen eine herstellerübergreifende Kompatibilität zu schaffen.

Für diese Ziele ist es notwendig, alle Anforderungen der Systeme an die Kommunikationstechnik zu ermitteln und moderne Lösungen zu vergleichen. Sofern es möglich ist, ist ein Großteil mit kostengünstiger und im Feld erprobter COTS Technik zu realisieren. Eine zukünftige Kommunikationsarchitektur in der Streckensicherungstechnik betrifft alle hierarchischen Ebenen. Der Fokus dieser Arbeit liegt auf dem Design von Kommunikationsarchitekturen für Stellwerke und Außenanlagen. In diesen Bereichen sind die Anforderungen an Sicherheit und Zuverlässigkeit am Größten und Kosteneinsparungen mit modernen Übertragungstechnologien hoch. Der Entwicklungsprozess wird durch diese Arbeit unterstützt, indem Sicherheit und Zuverlässigkeit von elementaren Kommunikationsfunktionen analysiert und damit Architekturentscheidungen erleichtert werden. In den folgenden Kapiteln wird ein kurzer Überblick über die Merkmale der drei hierarchischen Ebenen der Streckensicherungstechnik gegeben.

2.1.1 Die Leittechnik

Die Betriebsleittechnik ist für die Disposition und das Bedienen der Stellwerke zuständig. Bedienkommandos aus der Betriebsleittechnik werden im Stellwerk von sicheren Rechnern in Stellbefehle für die Außenanlage (Weichen, Signale etc.) umgesetzt. Wesentlicher Bestandteil der Betriebsleittechnik sind Standard PCs mit herkömmlichen Betriebssystemen, so dass die Vernetzung dieser PCs mit Standardschnittstellen, wie zum Beispiel Ethernet, realisiert ist [Fen07].

2.1.2 Die Stellwerke

Es gibt verschiedene Familien von elektronischen Stellwerken. Der Fokus dieser Arbeit liegt auf elektronischen Stellwerken des Typs "sicheres Mikrocomputer System von Siemens für den Weltmarkt" (SIMIS-W). Konkrete Anforderungen, die hier herausgestellt werden, stammen von der SIMIS-W Familie, die in ähnlicher Form auch in anderen Stellwerkstypen zu finden sind. Die Stellwerke bestehen aus Stellwerksrechnern, Achszählrechner und je nach Kundenanforderungen weiteren Rechnersystemen, wie zum Beispiel Linienförmige Zugbeeinflussung (LZB), Radio Block Center (RBC) und Diagnoserechner. Stellwerksrechner übernehmen die sichere Abarbeitung von Stellwerkslogik und steuern unter anderem Weichen und Signale der Außenanlage. Achszählrechner übernehmen die Auswertung von Signalen der Achszählsensoren und bestimmen, ob ein Gleisabschnitt belegt ist. Die Kommunikation zwischen diesen Rechnern ist für die korrekte Funktion eines Stellwerks unverzichtbar. Die Anbindung zur Leittechnik erfolgt über einen Koppelrechner. Der Stellabschnitt eines Stellwerks kann sich über eine Entfernung von mehr als 100 km erstrecken, so dass die Rechnersysteme auf mehrere, vernetzte Unterzentralen verteilt sind. Die Kommunikation zwischen den Systemen ist zu einer heterogenen Landschaft von physikalischen Übertragungsverfahren und Kommunikationsprotokollen geworden.

2.1.3 Die Außenanlage

Zur Außenanlage gehören Sensor- und Aktor-Elemente, die mit der Schiene oder den Fahrzeugen interagieren. In dieser Arbeit werden jedoch nur Elemente mit digitaler Datenkommunikation berücksichtigt. Dabei kann es sich um ein modulares Stellteil (MSTT)-Signal, einen digitalen Achszählpunkt oder einen Bahnübergang handeln. Neben proprietären Übertragungstechniken ist die Integrated Services Digital Network (ISDN) Technik in diesem Bereich weit verbreitet.

2.2 Anforderungen an Kommunikationsarchitekturen

Die Anforderungen von Systemen der Streckensicherungstechnik an die Kommunikation gliedern sich in verschiedene Bereiche. Es sind technische, nicht-technische und normative Anforderungen zu erfüllen. Zur Identifikation der technischen Anforderungen sind die Anforderungen von Applikationen der Stellwerkstechnik und Außenanlagen zu betrachten. Neben dem sicheren Austausch von Prozessdaten ist Diagnose und Wartung über Kommunikationsnetze ebenfalls ein wichtiger Punkt. Grundsätzlich kann hierfür eine gemeinsame physikalische Infrastruktur verwendet werden. Diagnose und Wartungsdaten dürfen die echtzeitkritischen Prozessdaten nicht negativ beeinflussen. Es ist sicherzustellen, dass nicht-zeitkritische Daten das Kommunikationsnetz oder Kommunikationsmodule nicht überlasten, da Prozessdaten in Wartepuffern sonst veraltern oder verworfen werden. Hierbei ist die verwendete Netzwerktopologie mit zu berücksichtigen. Administrative Vorgaben legen geswitchtes Ethernet für die lokale Vernetzung von Stellwerksrechnern fest. Verbindungen zwischen Stellwerken und zwischen Stellwerken und Systemen der Außenanlage sollen zukünftig über moderne Weitverkehrstechnologien realisiert werden. Aus Kostengründen sind Backbone-Infrastrukturen, LWL Techniken und drahtgebundene Übertragungsverfahren transparent zu nutzen. Drahtlose Übertragungsverfahren rücken ebenfalls näher in das Betrachtungsfeld von Systemplanern, wobei das providerbasierte Netzwerk GSM-R für die Kommunikation zu Schienenfahrzeugen bereits eingesetzt wird. Als permanente Infrastruktur zwischen ortsfesten Systemen wird GSM-R hier nicht explizit betrachtet. Als mögliche Alternative zu bisherigen Übertragungstechnologien sind drahtlose Übertragungstechniken, wie zum Beispiel WiMAX im Gespräch.

Die Applikationen der Systeme benutzen ausschließlich logische Punkt-zu-Punkt Verbindungen zu Kommunikationspartnern. Applikationen der Stellwerksrechner tauschen vorwiegend Informationen über Veränderungen im Stellabschnitt aus. Dieses sind Zustandsänderungen im Gesamtsystemzustand. Die jeweiligen Informationen liegen in der Größenordnung von wenigen 100 Bytes. Der Ausfall einer Verbindung muss in etwa einer Sekunde sicher erkannt werden, wofür ein zyklischer Überwachungstakt im Bereich weniger 100 ms generiert wird. In diesem Bereich liegt auch die maximal tolerierbare Latenzzeit der Übertragung. Insgesamt sind die Anforderungen der echtzeitkritischen Prozessdaten an Übertragungskapazitäten im Vergleich zur Leistungsfähigkeit heutiger Technologien sehr gering. Allerdings sind die Anforderungen an eine zuverlässige und sichere Übertragung sehr hoch: Ein Safety-Protokoll muss jegliche Arten von Übertragungsfehlern aufdecken und fehlerhafte Informationen dürfen einer Applikation nicht übergeben werden. Damit setzt der Austausch von Zustandsänderungen voraus, dass keine Information verloren geht.

Zuverlässige Übertragungsmethoden sind somit unerlässlich in der Kommunikationsarchitektur von Stellwerksrechnern.

Systeme der Außenanlage (dezentrale Stellteile, Achszählpunkte etc.) kommunizieren ebenfalls über logische (und derzeit auch über physikalische) Punkt-zu-Punkt Verbindungen. Die Datenmengen und zeitlichen Anforderungen sind vergleichbar mit denen der Stellwerksrechner. Aufgrund von kleineren Systemzuständen (zum Beispiel die Belegung von I/O Punkten) ist das zyklische Übermitteln des vollständigen Systemzustands möglich. Nach dem Vorbild vieler Protokolle aus der Automatisierungsbranche (zum Beispiel PROFIsafe) ist eine Fehlertoleranz durch die mehrfache Übertragung des Systemzustands möglich. Ein fehlerhaftes Informationspaket wird von einem Safety-Layer nicht akzeptiert, wird aber innerhalb der Echtzeitanforderung die Information korrekt übertragen, besteht kein negativer Einfluss auf die Zuverlässigkeit der Übertragung. Die hier grob skizzierten Anforderungen können im Einzelfall von den nachfolgend genannten abweichen. Es wird jedoch ein ausreichender Bewegungsrahmen für zukünftige Kommunikationsarchitekturen vorgegeben.

2.2.1 Normative Anforderungen

Bei der Entwicklung von Kommunikationsarchitekturen für die Streckensicherungstechnik sind Anforderungen aus Comité Européen de Normalisation Électrotechnique (CENELEC) Normen zu berücksichtigen. Die CENELEC ist zuständig für die europäische Normung im Bereich Elektrotechnik und definiert unter andrem Vorschriften und Anforderungen an sicherheitsrelevante Kommunikationsarchitekturen. Außereuropäische Länder verwenden für diesen Bereich entweder ähnliche Normen oder orientieren sich an den CENELEC Vorschriften. Für diese Arbeit sind die EN 50126, EN 50128, EN 50129 und EN 50159 teilweise oder im Ganzen relevant. Weitere Normen (auf die von den genannten verwiesen wird) sind hier nicht aufgelistet. Im Folgenden wird ein kurzer Überblick über die einzelnen Normen gegeben sowie die Teile genannt, die das Design von sicherheitsrelevanten Kommunikationsarchitekturen und damit das Vorgehen dieser Arbeit beeinflussen. Die Tabelle 2.1 auf der nächsten Seite sowie die Abbildung 2.2 auf Seite 14 zeigen die genannten Normen und den jeweiligen Geltungsbereich.

EN 50126

Die EN 50126 definiert den Prozess zur Spezifikation und zum Nachweis von Zuverlässigkeit, Verfügbarkeit, Instandhaltbarkeit und Sicherheit (RAMS). In dieser Norm sind Prozesse und Methodiken festgelegt, die ein System, Produkt oder Anlage von einem Konzept bis zur Stilllegung und Entsorgung systematisch betrachten. Dabei muss der Nachweis

Tabelle 2.1: Ausgewählte CENELEC Standards der Eisenbahndomäne (aus [Fen07])

Norm	Titel
EN 50126:1999	Railway applications - The specification and demonstration of Reliability, Availability Maintainability and Safety (RAMS)
EN 50128:2001	Railway applications - Communications, signalling and processing systems - Software for railway control and protection systems
EN 50129:2003	Railway applications - Communication, signalling and processing systems - Safety related electronic systems for signalling
EN 50159:2010	Railway applications - Communication, signalling and processing systems

erbracht werden, dass Anforderungen bezüglich Reliability, Availability, Maintainability und Safety erfüllt werden. Der RAMS Prozess unterstützt das Aufdecken von systematischen Fehlern während der Planungsphase von Projekten. Zu dem RAMS Prozess gehören das Definieren von Systemen, das Erstellen von Risikoanalysen, das Ermitteln von Gefährdungsraten sowie detaillierte Prüfungen und das Erstellen von Sicherheitsnachweisen. Ein Teil der EN 50126 beschäftigt sich mit der Auswirkung von Ausfällen innerhalb eines Systems. Störende Einflüsse auf das Bahnsystem können sich negativ auf die Zuverlässigkeit oder Sicherheit auswirken (siehe Abbildung 2.3 auf Seite 15). Die in dieser Arbeit beschriebene Analyse von Kommunikationsarchitekturen trägt zur Ermittlung der Systemzuverlässigkeit bei.

EN 50128

In der EN 50128 ist der Software-Entwicklungsprozess für sicherheitskritische Systeme definiert. Basierend auf dem Safety Integrity Level (SIL) (Kapitel 2.2.2 auf Seite 16) sind Validations-, Verifikations- und Testmethoden in den verschiedenen Entwurfsphasen spezifiziert. Unter anderem werden formale Modellierungs- und Verifikationstechniken zur Unterstützung des Softwaredesigns und Implementierung als “Highly Recommended” für SIL 3 und SIL 4 Software vorgeschlagen. Da für jegliche Art von sicherheitsrelevanter Software die EN 50128 anzuwenden ist, müssen auch Safety-Protokolle nach diesen Richtlinien erstellt werden. Für das Design und die Analyse von Kommunikationsarchitektur-Spezifikationen ist es daher sinnvoll, formale Verifikationen von Safety-Protokoll Spezifikationen nach Forderungen der Norm durchzuführen.

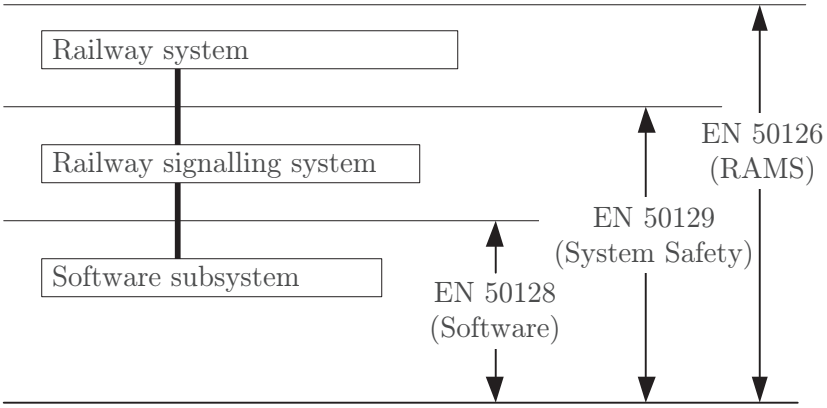


Abbildung 2.2: Geltungsbereich ausgewählter Normen (entnommen aus [BBSGS06])

EN 50129

Der Standard EN 50129 definiert das methodische Nachweisen von Sicherheitseigenschaften, um ein angemessenes Sicherheitsziel zu erreichen. Es sind unter dem Begriff Safety-Case sechs Dokumente bereitzustellen, die in strukturierter Form Aspekte eines Systems oder Subsystems dokumentieren. Eines dieser Dokumente ist der *Technical Safety Report*, in dem der Beweis von funktionaler und technischer Sicherheit dokumentiert ist.

Der Technical Safety Report beschreibt die technischen Grundsätze, welche die Safety des Designs garantieren, einschließlich der Nachweise von Berechnungen und Ergebnissen der Sicherheitsanalysen. Diese Nachweise zeigen, welchen Einfluss zufällige (Hardware) *Faults*[1] auf das System haben. Hierzu gehört die Erkennung von Single-Faults und die Auswirkung von Multiple-Faults. Single-Faults[2] müssen erkannt werden und innerhalb einer definierten zeitlichen Anforderung einen sicheren Zustand einleiten. Bezogen auf die sichere Kommunikation muss ein fail-safe-state (auch stable-safe-state genannt) innerhalb der zeitlichen Anforderungen eine *defekte* Kommunikationsverbindung erkennen. Hieraus ergeben sich Lebenstakte zur Überwachung der Verbindung sowie eine tolerierbare Übertragungszeit korrekter Daten[3].

[1] Definition im Kapitel 2.3 auf Seite 20

[2] Fehler, die zur Bedrohung der Safety führen.

[3] Im Kapitel 2.2.4 auf Seite 19 wird der Zusammenhang zwischen Übertragungsfehlern und der Dauer einer korrekten Übertragung beschrieben.

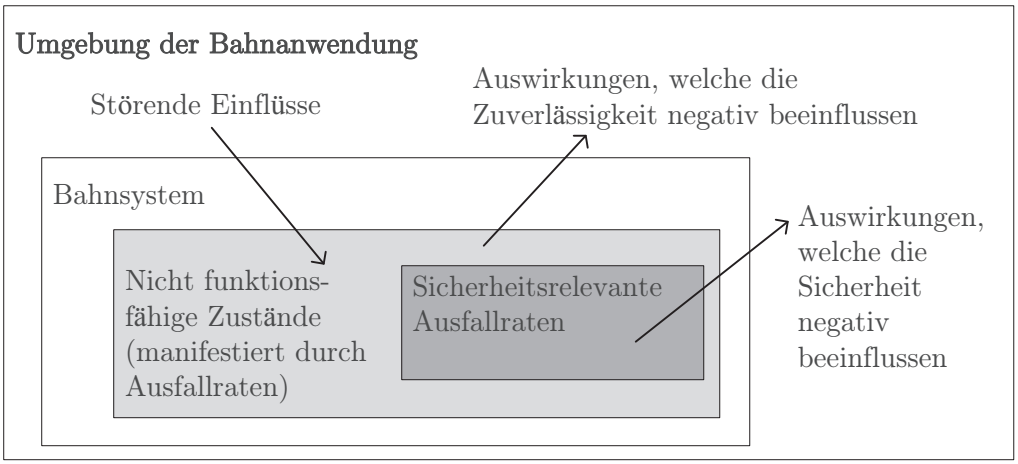


Abbildung 2.3: Auswirkungen von Ausfällen innerhalb eines Systems (aus [CEN00])

EN 50159-1:2001, EN 50159-2:2001 und EN 50159:2010

Die EN 50159-1:2001 und EN 50159-2:2001 sind Standards für die Entwicklung von sicherheitsrelevanten Kommunikationsarchitekturen über offene beziehungsweise geschlossene Übertragungssysteme. Ein Übertragungssystem gilt als geschlossen, wenn alle Eigenschaften bekannt und über die gesamte Verwendungszeit konstant sind. Damit hat der System-Safety-Designer die Kontrolle über die Safety während der Designphase. Offene Übertragungssysteme haben während ihrer Verwendungszeit teilweise oder gänzlich unbekannte Parameter. Zu diesen Parametern gehören zum Beispiel die Übertragungstechnologie und die Anzahl von Kommunikationsteilnehmern. In der EN 50159-2 sind Übertragungsnetze in sieben verschiedene Klassen unterteilt. Die Klasse 1 steht für ein geschlossenes Übertragungssystem und die EN 50159-1 kommt zur Anwendung. Für die Klasse 2 bis Klasse 7 ist die EN 50159-2 anzuwenden. In der EN 50159-2 ist auch die Verwendung von Security Mechanismen beschrieben, die ab der Klasse 6 in sicherheitsrelevanten Kommunikationsarchitekturen zur Abwehr gegen aktive Angriffe implementiert werden müssen. Diese Aufteilung auf zwei Standards wurde während der Erstellung dieser Arbeit durch die zusammengefasste Norm EN 50159:2010 abgelöst, was bereits seit längerem in Fachkreisen diskutiert wurde [AHM04].

Die EN 50159 teilt Übertragungsnetze anstelle der sieben in nur noch drei Kategorien auf: Geschlossene Netze (Kategorie 1), offene Netze ohne unautorisierten Zugriff (Kategorie 2) und offene Netze mit möglichem unautorisierten Zugriff (Kategorie 3). Für Netze der Kategorie 3 sind Security-Maßnahmen zu implementieren. Diese können in dem sicheren-

oder in dem nicht-sicheren Teil der Architektur realisiert werden. Befinden sich diese in dem nicht-sicheren Teil, ist die ordnungsgemäße Funktion von dem sicheren Teil der Architektur zu überwachen. Für diese Arbeit ist vor allem die Aufteilung der Architektur in sicherheitsrelevante und nicht-sicherheitsrelevante Bestandteile von Bedeutung (Abbildung 2.4 auf der nächsten Seite). Fehlerarten, die ein Safety-Protokoll erkennen muss, sind Ausgangspunkt für den Sicherheitsnachweis. In der Tabelle 2.2 sind geeignete Mechanismen zum Erkennen der Fehlerarten aufgelistet. Ein Safety Protokoll muss unter Anwendung der EN 50128 und EN 50129 auf sicherer Hardware implementiert werden.

Tabelle 2.2: Gefahren und Abwehrmaßnahmen der Datenübertragung

Defences Threat	Sequence number	Time stamp	Time out	Src. and Dst. ID	Feedback message	Identification procedure	Safety code
Repetition	x	x					
Deletion	x						
Insertion	x			x	x	x	
Resequence	x	x					
Corruption							x
Delay		x	x				

2.2.2 Safety

Safety steht für technische Sicherheit und bedeutet, dass ein technisches System in der Lage ist, zufällige und systematische Fehler aufzudecken und derart zu reagieren, dass weder Umwelt noch Menschen gefährdet werden. Zur Beherrschung von Gefährdungen werden Sicherheitsfunktionen definiert, die nach der EN 50126 eine Sicherheitsanforderung erfüllen müssen. Eine Sicherheitsanforderung besteht aus dem qualitativen Teil SIL und dem quantitativen Teil Tolerable Hazard Rate (THR). In [BBSGS06] ist die Sicherheitsanforderung THR als ein Maß für die Häufigkeit, mit der eine Gefährdung auftreten darf (toleriert wird) und der SIL als ein Maß für den Aufwand von fehlervermeidende Maßnahmen genannt. Das Sicherheitsziel wird in die Stufen SIL 1 bis SIL 4 klassifiziert. Bei Systemen beziehungsweise Funktionen ohne Sicherheitsverantwortung wird auch der Begriff SIL 0 verwendet. Diese Sicherheitsstufen geben bestimmte Vorgehensweisen bei der Entwicklung von Systemen vor und verlangen den qualitativen Nachweis von Reaktionen beim Auftreten von Gefährdungen, die die Sicherheit von Mensch und Umwelt in Gefahr bringen. Ein Beispiel aus dem Bereich von sicherheitsrelevanten Kommunikationssystemen ist der

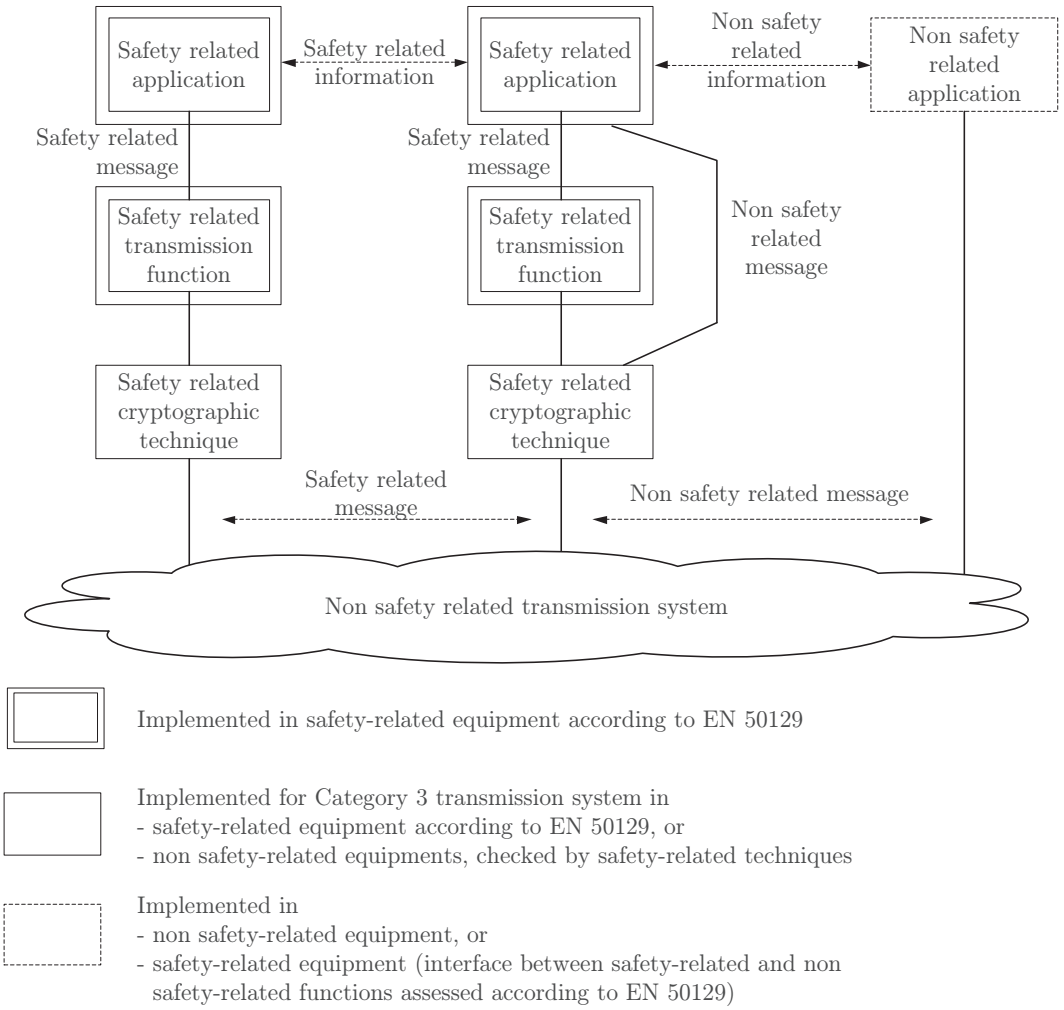


Abbildung 2.4: Referenzarchitektur nach EN 50159 (aus [CEN10])

Empfang einer durch Bitfehler verfälschten Nachricht. Damit diese Nachricht in einer sicherheitsrelevanten Applikation nicht zu falschen Reaktionen führt, muss ein Safety-Layer (Safety-Protokoll) verfälschte Nachrichten erkennen und die Weiterleitung dieser falschen Informationen verhindern. Diese Reaktion ist der qualitative Teil des Sicherheitslevels. Weiterhin gibt es noch einen quantitativen Teil, welcher als Hazard-Rate das Versagen der Methode zum Erkennen von Gefährdungen angibt. Dient zum Beispiel ein Cyclic Redundancy Check (CRC) zum Erkennen verfälschter Nachrichten, dann geschieht dieses mit einer Restfehlerwahrscheinlichkeit (siehe [Sch06]). Diese gibt abhängig von der Bitfehlerwahrscheinlichkeit die Wahrscheinlichkeit an, dass eine verfälschte Nachricht nicht mit der Prüfsumme erkannt wird. Mit mathematischen Methoden (zum Beispiel [SM10]) kann die-

se Restfehlerwahrscheinlichkeit ermittelt werden. Zusammen mit der Häufigkeit, die eine Sicherheitsfunktion pro Zeitintervall in Anspruch genommen wird, lässt sich hieraus eine Restfehlerrate bestimmen (siehe [Rei01]). Diese Restfehlerrate führt zu einer Gefährdungsrate (hazard rate) und darf eine durch das Safety-Integrity-Level vorgegebene Grenze nicht überschreiten. Ist in dieser Arbeit die Verifikation von Safety genannt, bezieht sich das ausschließlich auf den qualitativen Nachweis von Safety. Eine Restfehlerrate und die damit verbundene Gefährdungsrate wird hier nicht behandelt. Die EN 50129 definiert allgemein einen Safe-State als Reaktion auf Gefährdungen und die EN 50159 nennt diesen speziell für Safety-Protokolle Safe-Fall-Back-State (auch als *Stable-Safe-State* bezeichnet [HP02]). Die Verifikation der Safety von Kommunikationsprotokollen dient dem Auffinden von systematischen Fehlern in der Spezifikation. Ein Safety-Protokoll muss zum Beispiel eine als fehlerhaft gekennzeichnete Nachricht wiederholen oder den sichern Zustand einleiten, um gemäß der 50129 und EN 50159 zu reagieren.

2.2.3 Security

Security steht für Datensicherheit und wird in zukünftigen Kommunikationsarchitekturen eine immer größere Rolle spielen. Mit dem Verwenden von Kommunikationsinfrastrukturen, zu denen unbekannte oder nicht-vertrauenswürdige Nutzer Zugriff haben oder erlangen können, sind nach der EN 50159 geeignete Abwehrmaßnahmen in die Architektur zu integrieren. Es ist mit kryptografischen Methoden die Kommunikation von sicherheitsrelevanten Systemen so abzusichern, dass Informationen nicht unerkannt verfälscht werden können. Geeignete Mechanismen können in dem sicheren Teil (zum Beispiel dem Safety-Protokoll) oder in dem nicht-sicheren Teil der Architektur realisiert werden. Wenn diese in dem nicht-sicheren Teil realisiert sind, hat der sichere Teil die korrekte Funktion zu überwachen. Das Thema Security ist jedoch zu komplex, um es mit dieser Arbeit zu behandeln und wird daher ausgeklammert. Der Einfluss von Security Mechanismen auf Zuverlässigkeitseigenschaften wird dennoch berücksichtigt: Zusätzliche Latenzzeiten oder Security-Verfahren mit integrierten Zuverlässigkeitsmechanismen können analysiert werden, ohne die Wirksamkeit der Security gegenüber aktiven Angriffen zu berücksichtigen. Bevor eine endgültige und konkrete Kommunikationsarchitektur für die Streckensicherungstechnik festgelegt wird, muss der Einfluss von Angriffen sowie die wirksame Abwehr gegen Informationsverfälschung, Denial of Service (DoS) Attacken und unberechtigten Ressourcen-Zugriff ermittelt werden.

2.2.4 Reliability

Der Begriff Reliability (Zuverlässigkeit) ist in der Literatur nicht einheitlich definiert. Zunächst ist der Begriff Reliability für Datenübertragungsverfahren und für technische Systeme zu unterscheiden. Bezieht sich die Zuverlässigkeit auf ein Kommunuikationsverfahren oder einen Kommunikationsdienst von Übertragungsprotokollen, ist die Zuverlässigkeit auf den Austausch von Daten bezogen. Ein Kommunikationsprotokoll gilt als zuverlässig, wenn

1. alle gesendeten Daten vollständig und unverfälscht ankommen,
2. die Reihenfolge ankommender Daten mit der Sende-Reihenfolge übereinstimmt und
3. keine Duplikate oder anderweitig eingefügte Daten eintreffen.

Zuverlässige Übertragungsverfahren verwenden zum Beispiel Sequenznummern, einen Quittierungsmechanismus und ein Daten-Wiederholungsverfahren, um auftretende Fehler bei der Übertragung zu erkennen und zu korrigieren. Benutzt eine Applikation einen zuverlässigen Übertragungsdienst, dann werden sporadische Übertragungsfehler für die Applikation transparent korrigiert. Die Kommunikation ist gegenüber bestimmter Fehler und der Fehlerhäufigkeit tolerant. Es gibt jedoch eine wesentliche Einschränkung für echtzeitkritischen Datenaustausch: Mit jeder Wiederholung veraltern die Informationen. Damit gibt es eine Fehlerhäufigkeit, die trotz Zuverlässigkeitsmechanismen Applikationsdaten nicht zuverlässig (innerhalb der Echtzeitgrenze) überträgt. Die vorgestellten Safety-Mechanismen tolerieren keine veralteten Daten. Als Reaktion auf ein solches Ereignis hat ein Safety-Protokoll den sicheren Zustand einzuleiten, wobei die Kommunikation gestoppt wird. Die Zuverlässigkeit der Datenkommunikation kann somit nur anhand einer gesamten Architektur ermittelt werden. Safety- und Zuverlässigkeitsmechanismen sowie eine fehlerhafte Übertragung erlauben nur zusammen eine Aussage über die tatsächliche Zuverlässigkeit. Dieses wird im Folgenden als *Zuverlässigkeit des logischen Kanals* bezeichnet, da Safety-Protokolle den logischen Kommunikationskanal mit dem sicheren Zustand schließen.

Bezieht sich der Begriff Zuverlässigkeit auf ein technisches System, sind in den Normen folgende Definitionen zu finden:

1. The ability of an item to perform a required function under given conditions for a given time interval. (IEC 60050, 191-02-06 und EN 50128)
2. The probability that an item can perform a required function under given conditions for a given time interval. (IEC 50, 1992)

3. The capability of the software product to maintain a specified level of performance when used under specified conditions. (IEC 9126-1, 2001)

Mit dem Hinzuziehen von Zeitanforderungen zu dem Begriff Reliability im Telekommunikationsbereich lässt sich eine Verbindung zu der Definition Reliability eines technischen Systems schaffen. Gesucht ist die Zuverlässigkeit (nach EN 50128) eines technischen Systems mit einer Kommunikationsverbindung. Die gesamte Zuverlässigkeit setzt sich aus der Zuverlässigkeit der Hardware und der logischen Zuverlässigkeit zusammen. Die Zuverlässigkeit wird von Sicherheits- und Zuverlässigkeitsmechanismen sowie Fehlerarten bei der Datenübertragung bestimmt. Mit der Wahrscheinlichkeit, dass ein Protokolldienst zuverlässig arbeitet und mit der Häufigkeit pro Zeitintervall, in dem dieser Protokolldienst in Anspruch genommen wird, berechnet sich die Zuverlässigkeit des logischen Kanals. Mit dieser Information vervollständigt sich die Zuverlässigkeitsbetrachtung eines technischen Systems, das auf die Kommunikation mit anderen Systemen angewiesen ist, um insgesamt zuverlässig im Sinne der EN 50128 zu arbeiten.

Die Zuverlässigkeit von Systemen der Streckensicherungstechnik ist eine entscheidende Kenngröße und Vergleichskriterium zu Wettbewerbern, da die Zuverlässigkeit Einfluss auf Life-Cycle-Costs hat. Mit dem skizzierten Fehlerbaum (Abbildung 2.5 auf der nächsten Seite) ist zu sehen, dass die Zuverlässigkeit direkt mit der Fehlerwahrscheinlichkeit von Kommunikationsinfrastrukturen zusammenhängt. Bei dem Design von Kommunikationsarchitekturen ist daher auf die Einhaltung von Safety-Kriterien und auf Übertragungsmechanismen, die eine hohe Zuverlässigkeit garantieren, zu achten. Top-Down betrachtet, führt eine Zuverlässigkeitsanforderung zu Mindestanforderungen an Fehlerraten von Übertragungsnetzen. Damit kann während der Planung von Projekten bestimmt werden, welche Qualitätsanforderungen an Übertragungsmedien eingehalten werden müssen. In dieser Arbeit wird ein Analyse-Framework vorgestellt, dass die Zuverlässigkeit des logischen Kanals von Kommunikationsarchitekturen ermittelt.

2.3 Fault, Error, Failure

Der Ausdruck “Fehler” wird in technischen Systemen detaillierter unterschieden. Die Definitionen der Begriffe *Fault*, *Error* und *Failure* stimmen in der Literatur nicht immer exakt überein, da die Begriffe sich auf Hardware, Software oder andere Aspekte eines Systems beziehen. Gleichwohl ist der kausale Zusammenhang übereinstimmend definiert. Ein Fault kann zu einem Error führen und ein Error kann zu einem Failure werden. Ein Failure ist die sichtbare und unerwünschte Auswirkung auf einer bestimmten Systemebene. Bedingt durch die Fehlerfortpflanzung kann ein Failure gleichzeitig ein Fault auf der nächsthöheren

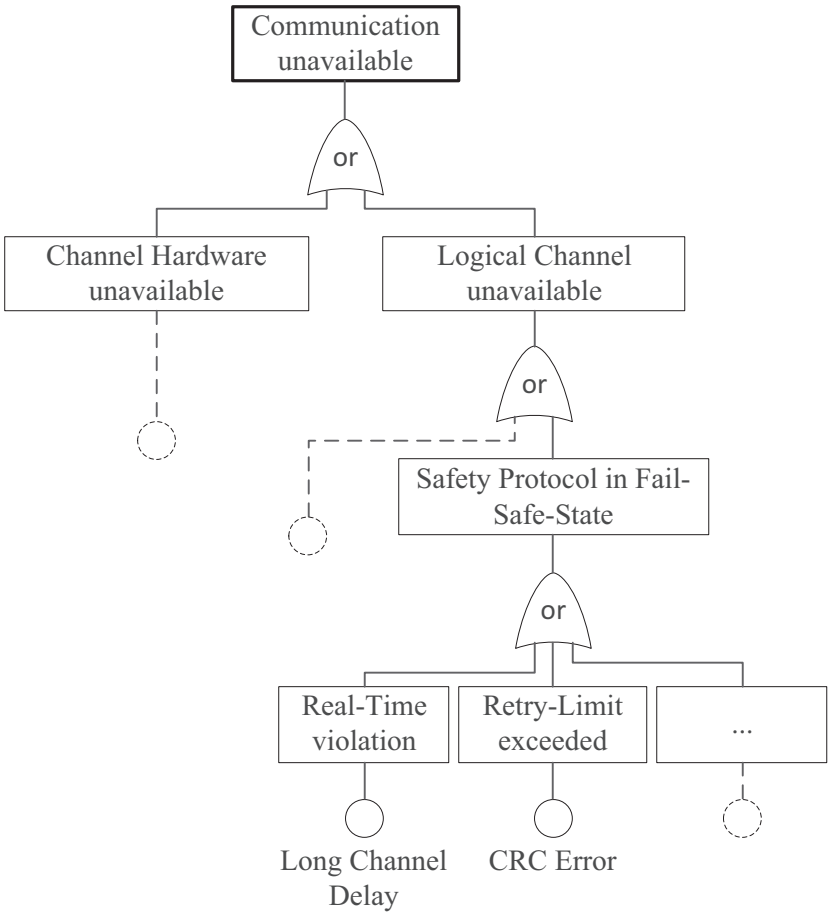


Abbildung 2.5: Zusammenhang zwischen Übertragungsfehlern und Zuverlässigkeit

Ebene sein (Abbildung 2.6 auf der nächsten Seite). Es sind zwei Arten von Fehlerfortpflanzungen zu berücksichtigen: Zum einen die Aufdeckung von Fehlern und deren Behandlung und zum anderen eine Fortpflanzung ohne Aufdeckung. Des Weiteren sind zufällige und systematische Fehler zu unterscheiden.

Die folgenden Definitionen der verschiedenen Fehlerarten sind aus den CENELEC Normen EN 50128 EN 50129 und EN 50159 entnommen und gelten für diese Arbeit. Es werden jeweils die englischen Begriffe verwendet.

Definition 2.3.1 (Fault)**.** Abnormal condition that could lead to an error or a failure in a system. A fault can be random or systematic.

Definition 2.3.2 (Error)**.** Deviation from the intended design which could result in unintended system behaviour or failure.

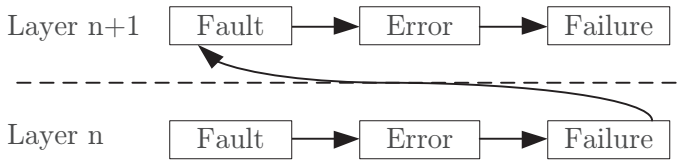


Abbildung 2.6: Fehlerfortpflanzung: Fault, Error, Failure

Definition 2.3.3 (Failure)**.** Deviation from the specified performance of a system. A failure is the consequence of a fault or error in a system.

Definition 2.3.4 (Random Failure)**.** A failure that occurs randomly in time

Definition 2.3.5 (Systematic Failure)**.** A failure that occurs repeatedly under some particular combination of inputs, or under some particular environmental condition.

Definition 2.3.6 (Random Fault)**.** The occurrence of a fault based on probability theory and previous performance.

Definition 2.3.7 (Systematic Fault)**.** An inherent fault in the specification, design, construction, installation, operation or maintenance of a system, subsystem or equipment.

Definition 2.3.8 (Fault Tolerance)**.** Built-in capability of a system to provide continued correct provision of service as specified, in the presence of a limited number of hardware or software faults.

Definition 2.3.9 (Message Errors)**.** A set of all possible message failure modes which can lead to potentially dangerous situations, or to reduction in system availability. There may be a number of causes of each type of error.

2.3.1 Verwendung der Begriffe in Bezug auf Kommunikationsarchitekturen

Kommunikationsarchitekturen bestehen aus Software und Hardware-Teilen, wodurch die genannten Definitionen der Begrifflichkeiten aus verschiedenen Normen eine Rolle spielen. In dieser Arbeit werden Kommunikationsarchitekturen im Hinblick auf *Systematic Faults* im sicherheitsrelevanten Teil der Architektur und Auswirkungen von *Random Faults* auf Zuverlässigkeitseigenschaften analysiert. Für beide Analysen sind *Message Errors* von zentraler Bedeutung. Message Errors sind aus Sicht einer Kommunikationsschicht Random

Faults. Die Annahme über die Art der Message Errors und deren Häufigkeit wird als *Fault Hypothesis* bezeichnet.

2.3.2 Message Errors - Übertragungsfehler

Message Errors sind bei der Übertragung von Nachrichten zufällig auftretende Fehler. Es können elektromagnetische Störungen, Ressourcenmangel, Pfadänderungen in Routingnetzwerken (Sequenzveränderungen von Datenpaketen) oder andere Effekte die Ursache von Übertragungsfehlern sein. Diese Fehler werden auf sechs verschiedene Fehlerarten (Tabelle 2.2 auf Seite 16) zurückgeführt. Die Abbildung 2.7 illustriert die Fehlerarten. Die Fehlerart und ihre Auftrittswahrscheinlichkeit hängt von dem Übertragungskanal und den beteiligten Komponenten ab. Das Verfälschen von Bits durch elektromagnetische Störungen ist der häufigste Fehler. Mit dem Hinzufügen einer Prüfsumme (zum Beispiel CRC) wird dieser Fehler aufgedeckt (die Restfehlerwahrscheinlichkeit wird hier nicht betrachtet).

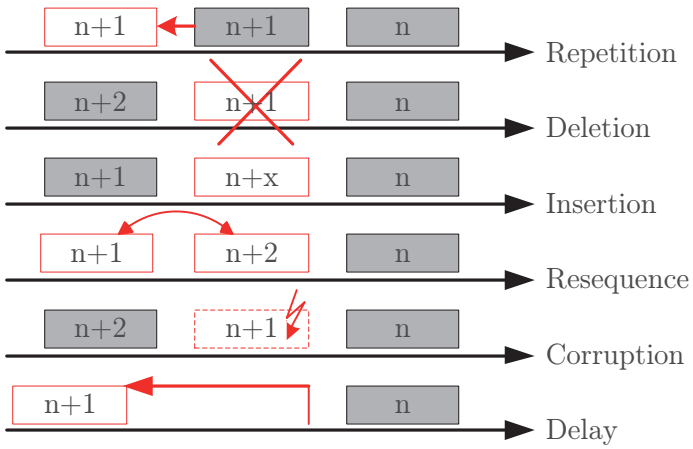


Abbildung 2.7: Fehlerarten bei der Datenübertragung

In Ethernet-basierten Kommunikationsarchitekturen führt ein CRC-Fehler zu einem Nachrichtenverlust, da nach Ethernet-Spezifikation Nachrichten mit fehlerhafter Prüfsumme verworfen werden. Ein Nachrichtenverlust entsteht auch, wenn in aktiven Netzwerkkomponenten infolge eines zu hohen Verkehrsaufkommens Ressourcen (zum Beispiel Speicher) überlastet werden. Angestrebt wird daher immer eine ausreichende Dimensionierung der Netzwerke. In der Regel sind Nachrichtenverluste durch Überlastung nicht stochastisch unabhängig, da ein aufgetretener Engpass erst nach einer gewissen Zeit abgebaut werden kann. Die Folge ist ein verstärktes Auftreten von Paketverlusten innerhalb kurzer Zeit.

Zusammenfassend betrachtet ist das Auftreten einer der in der Tabelle 2.2 auf Seite 16 definierten Fehlerarten gerade in komplexen (Backbone-) Netzwerken keine Seltenheit. Es ist Aufgabe von höheren Protokollen, eine Fehlerart zu erkennen und gegebenenfalls durch Mechanismen zu korrigieren. In der Tabelle 2.3 sind Bit Error Ratio (BER) (Bitfehlerwahrscheinlichkeit) verschiedener Übertragungstechnologien aufgeführt (siehe [Rei01, EF07, RMRHS07]). Die Zuverlässigkeit der Kommunikation gegenüber dieser Fehlerarten wird mit Zuverlässigkeitsmechanismen erhöht. Jedoch können spezifische Eigenschaften von Netzwerken, wie zum Beispiel die Rekonfigurierung eines Rings, weiterhin negative Auswirkungen auf die Zuverlässigkeit haben. Im Folgenden werden verschiedene Zuverlässigkeitsmechanismen erläutert.

Tabelle 2.3: Bitfehlerwahrscheinlichkeiten verschiedener Übertragungstechnologien

Physikalisches Medium	Technologie	BER
Lichtwellenleiter		10^{-13}
	Synchronous Digital Hierarchy (SDH)	10^{-12}
	E1/T1 leased line Service	10^{-9}
	Metro-Netze	10^{-8}
Drahtgebundene Techniken	Twisted Pair (Differential)	10^{-7}
	ADSL	10^{-7}
	Koaxial Kabel	10^{-6}
Drahtlose Techniken		$10^{-6} \ldots 10^{-2}$
	Satelliten Dienste	10^{-6}
	Mobilfunk	$10^{-3} \ldots 10^{-2}$

2.4 Zuverlässigkeitsmechanismen

Ein zuverlässiger Datentransfer übermittelt Informationen vollständig, in der richtigen Reihenfolge, fehlerfrei und ohne Duplikate. Echtzeitkritische Daten müssen zudem innerhalb einer maximal tolerierbaren Verzögerungszeit korrekt übermittelt werden. Zu den Zuverlässigkeitsmechanismen gehören Redundanz, Quittierungsverfahren, Übertragungswiederholung und Stau- beziehungsweise Flusskontrolle. Die Beschreibungen sind zum Teil entnommen aus [Tan02, MBW08].

2.4.1 Redundanz

Es wird von einer redundanten Übertragung gesprochen, wenn mehrere physikalische Verbindungen für die Übertragung genutzt werden können. Es wird dabei zwischen heißer-, warmer- und kalter Redundanz unterschieden. Bei warmer- und kalter Redundanz wird eine alternative physikalische Verbindung verwendet, sobald die Primäre als ausgefallen gilt. Bei kalter Redundanz muss die Verbindung noch aufgebaut oder installiert werden, während die warme Redundanz einen alternativen Pfad aktiv bereit hält (logisch aufgebaut und überwacht). Heiße Redundanz benutzt alle physikalischen Pfade gleichzeitig. Hierzu wird auf der Redundanzebene die Information für alle Verbindungswege kopiert und übertragen. Auf der Empfängerseite werden überflüssige Kopien wieder entfernt und das Datenpaket der nächsthöheren Schicht übergeben. Die heiße Redundanz hat gegenüber der warmen und kalten den Vorteil, dass die Übertragung an Fehlertoleranz gewinnt. Lediglich eine Nachricht muss auf einem physikalischen Pfad korrekt übertragen werden, ohne dass die Zuverlässigkeit der Übertragung negativ beeinflusst wird. Dieser Vorteil verursacht allerdings entsprechenden Overhead und Kosten durch mehrfache Systeme. Redundanz kann je nach Anforderungen und Budget auf

- Applikationsebene,
- Transport-Layer,
- Link-Layer und
- Netzwerk-Ebene

installiert sein.

2.4.2 Quittierungsverfahren

Bei der Quittierung von Daten bestätigt der Empfänger den korrekten beziehungsweise fehlerhaften Erhalt von Daten gegenüber dem Sender. Es sind drei Quittierungsverfahren zu unterscheiden:

1. Bei dem **positiv-selektivem Quittierungsverfahren** wird vom Empfänger eine Quittung (ACK) pro korrekt empfangener Nachricht gesendet. Dies hat einen zusätzlichen Nachrichtenverkehr zur Folge und blockiert weitere Telegramme, bis eine Quittung eingetroffen ist.
2. Das **positiv-kumulative Quittierungsverfahren** sendet eine Quittung für mehrere Nachrichten. Dieses reduziert die Netzlast, hat aber den Nachteil, dass Informationen über einen Datenverlust verspätet an den Sender übertragen werden.

3. Beim **negativ-selektiven Quittierungsverfahren** werden vom Empfänger selektiv verlorengegangene Nachrichten erneut vom Sender angefordert. Alle Nachrichten, zu denen keine Nachfrage kommt, gelten beim Sender nach einer bestimmten Zeit als angekommen. Nachteilig ist die Offenbarungszeit für einen Telegramverlust, die abhängig von der Zeit bis zur Folgenachricht ist. Der Verlust von Daten kann nur durch eine Lücke in den Sequenznummern festgestellt werden. Zudem muss der Verlust von negativen Quittungen behandelt werden.

2.4.3 Übertragungswiederholung

Mit der Quittierung wird der korrekte beziehungsweise fehlerhafte Empfang von Daten dem Sender mitgeteilt. Ein zuverlässiges Protokoll setzt voraus, dass fehlerhafte Übermittlungen wiederholt werden. Hiebei sind zwei Verfahren zu unterscheiden:

1. Bei der **selektiven Wiederholung** werden nur die negativ quittierten Nachrichten wiederholt. Der Empfänger puffert die nachfolgenden Nachrichten, bis die Fehlenden eingetroffen sind. Erst dann werden die Daten zur nächsthöheren Schicht weitergereicht. Die reguläre Übertragung kann während der Wiederholung fortgesetzt werden, jedoch benötigt der Empfänger eine entsprechende Pufferkapazität.

2. Beim **Go-Back-N** Verfahren werden die fehlerhaften Nachrichten sowie alle nachfolgenden Nachrichten erneut übertragen. Hierbei wird eine geringe Pufferkapazität beim Empfänger benötigt, jedoch wird die reguläre Übertragung unterbrochen.

2.4.4 Flusskontrolle

Durch die Steuerung des Datenflusses soll eine Überlastung des Empfängers vermieden werden. Traditionelle Flusskontroll-Verfahren sind:

1. **Stop-and-Wait-Verfahren**: Dies ist das einfachste Verfahren, bei dem eine Kopplung zwischen Fluss- und Fehlerkontrolle durchgeführt wird. Die nächste Nachricht wird erst nach einer erfolgreichen Quittierung gesendet (Fenstergröße 1).

2. Fensterbasierte Flusskontrolle oder **Sliding-Window** Verfahren: Der Empfänger vergibt einen Sendekredit, der eine maximale Anzahl von Nachrichten oder Bytes angibt, die unquittiert gesendet werden dürfen. Der Sendekredit reduziert sich mit jedem Senden, bis der Empfänger durch Quittungen den Erhalt bestätigt hat. Der Vorteil dieses Verfahrens ist, dass ein kontinuierlicher Datenfluss sowie ein höherer Durchsatz als bei Stop-and-Wait möglich ist.

2.4.5 Staukontrolle

Mit der Staukontrolle Congestion Control (CC) sollen Verstopfungen beziehungsweise Überlastungen im Netz vermieden werden. Die Staukontrolle reduziert den Datenverkehr durch zurückhalten von Paketen. Das Protokoll Transmission Control Protocol (TCP) verwendet zum Beispiel eine passive Staukontrolle, bei der die Fenstergröße der Flusskontrolle verkleinert wird, wenn durch häufige Paketverluste die Verbindung als überlastet gilt.

Explicit Congestion Notification (ECN).

ECN definiert in der RFC 3168 (2001) [RFB01] eine Erweiterung von IP zur aktiven Staukontrolle in IP-Netzen. Die Überlastung von Routern wird bei Transportprotokollen mit CC durch auftretenden Paketverlust festgestellt (Router verwerfen Telegramme, wenn ihre Puffer voll sind). Wenn TCP vermehrt Nachrichten wiederholen muss, wird dabei das Sendefenster (maximale Anzahl von Bytes, die unquittiert gesendet werden können) reduziert. Dieses Verfahren reagiert erst, wenn die Puffer bereits übergelaufen sind. Aus diesem Grund ist ECN eingeführt worden. Hierbei vermerken die Router des Netzwerkes mit einem Flag im IP Header die drohende Überlast. Der Datenempfänger kopiert dieses Flag dann in den Header von TCP und teilt mit der Paket-Quittierung dem Sender die drohende Überlast mit. Dieser reduziert nach den gleichen Algorithmen das Sendefenster, wie bei der passiven Staukontrolle, allerdings bevor Paketverluste aufgetreten sind. Anmerkung: CC funktioniert nur bei Verbindungsorientierten Protokollen und zum Beispiel nicht mit User Datagram Protocol (UDP). Des Weiteren kann die Verwendung von ECN zu Problemen aufgrund inkompatibler Hardware führen. Es besteht die Gefahr, dass ein nicht-ECN-fähiger Router dieses als Protokollfehler wertet und alle Pakete verwirft.

Traffic Shaping.

Neben der aktiven Staukontrolle gibt es auch die Möglichkeit eine Überflutung des Netzes durch Traffic Shaping zu verhindern. Ein möglicher Mechanismus ist der Token Bucket Filter (TBF) (Abbildung 2.8 auf der nächsten Seite). Der TBF erlaubt das Senden von Datenpaketen nur mit einem bestimmten Budget. Dieses Budget legt die maximale Bandbreite fest, die von Applikationen über das Interface genutzt werden kann. Es wird damit garantiert, dass das Senden von großen Datenmengen nicht das Netz überflutet und Daten anderer Applikationen behindert [JJ03, EF07]. Der TBF ist vor allem für die gemeinsame Übertragung von Echtzeit- und nicht-echtzeitkritischen Daten ein geeignetes Mittel, Störeinflüsse zu reduzieren. Hierbei erhält eine niederpriore Applikation nur so viel Sende-Budget, dass einer hochprioren Applikation genügend Bandbreite für echtzeitkritische Da-

ten zur Verfügung steht.

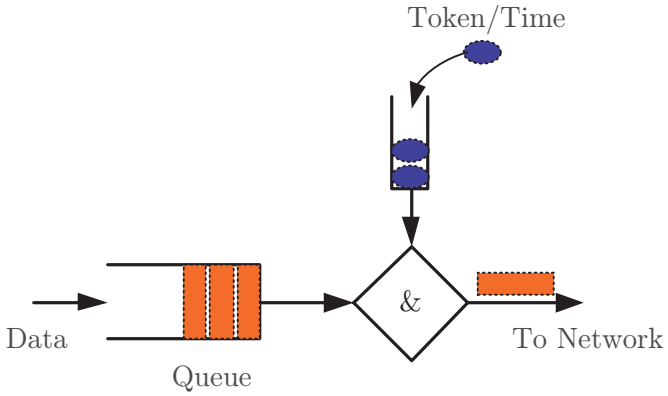


Abbildung 2.8: Token Bucket Filter

2.5 Kommunikationstechnik und Trends

In den letzen Jahren fand im Industriebereich ein Umschwung in der Kommunikationstechnik statt. Insbesondere ethernet-basierte Kommunikationstechnologien dringen in den klassischen Feldbusbereich vor. Hersteller von Netzwerktechnik liefern industrietaugliche Kommunikationskomponenten, die bisherige Feldbussysteme ablösen. Zudem dringt der Ethernet-Standard in den klassischen Backbonebereich vor, so dass auch hier der Markt auf Ethernet zugeht. Dieses ist daran zu erkennen, dass Zugangssysteme zu modernen Backbonenetzen (Network Access Unit (NAU)) oftmals den Ethernet-Standard auf der LAN-Seite unterstützen. Es gibt zum Beispiel Ethernet Modems für die Übertragung mittels Digital Subscriber Line (xDSL) oder sogenannte Multi-Service Plattformen, die backboneseitig einen Zugang zum Weitverkehrsnetz und LAN-seitig Ethernet Schnittstellen besitzen. Mit der großen und stetig wachsenden Verbreitung von Ethernet gibt es eine Vielzahl von Herstellern solcher Netzwerkkomponenten, die auch in gehärteten Industrievarianten erhältlich sind. Durch diese Herstellerdiversität und weiteren technischen Vorteilen ist Ethernet als Rechnerschnittstelle für Systeme der Streckensicherungstechnik administrativ vorgegeben. Ethernet ist jedoch nicht die einzige COTS Lösung in einer Kommunikationsarchitektur: Weitere Komponenten und Kommunikationsprotokolle müssen betrachtet werden, um eine nach allen Gesichtspunkten zukunftsfähige Kommunikationsarchitektur festzulegen. Zunächst wird daher der gesamte Bereich einer Kommunikationsarchitektur analysiert und Komponenten herausgestellt, von denen aus COTS Sicht profitiert werden kann. Für diesen Zweck sind die beteiligten Technologien in fünf Bereiche klassifiziert und im Folgenden kurz beschrieben:

1. Hardware und Software von Endgeräten
2. Lokale Netze
3. Netzzugangstechnik für Backbone Netze
4. Metronetze (MAN)
5. Weitverkehrsnetze (WAN)

Die Technologien aus dem MAN und dem WAN lassen sich nicht klar trennen und auch die Begrifflichkeiten sind nicht klar definiert. Diese Netzwerke sind daher als *Backbonetechnik* zusammengefasst.

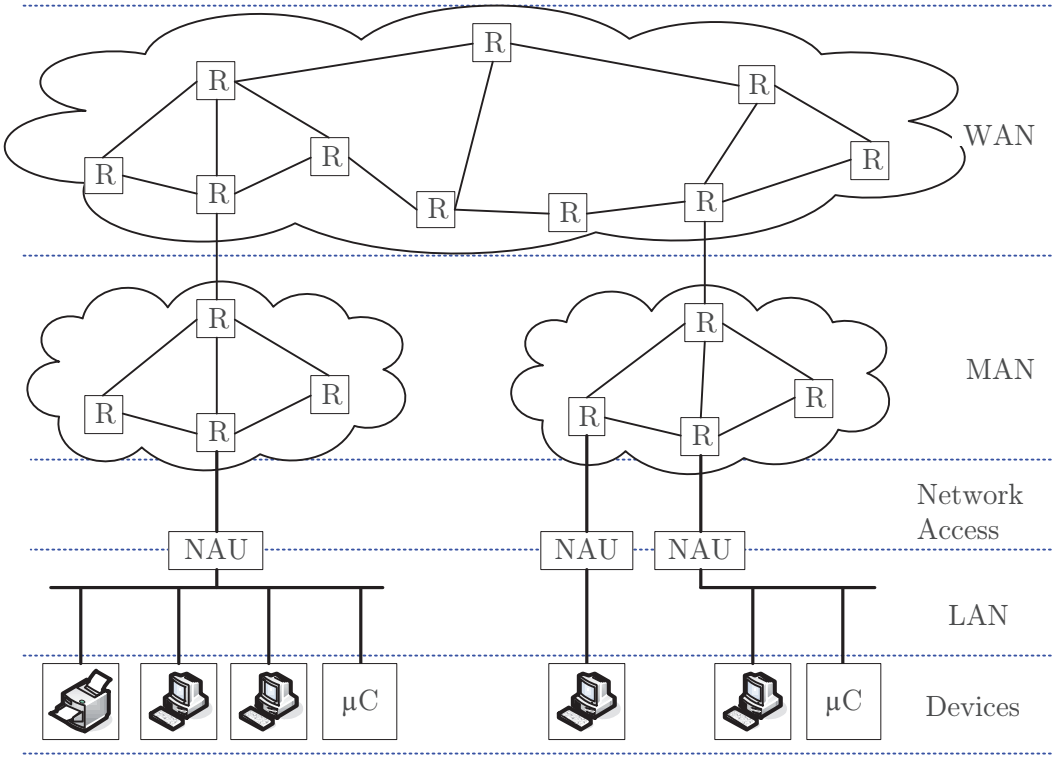


Abbildung 2.9: Klassifizierung von Kommunikationstechniken

2.5.1 Hard und Software von Endgeräten

Im Bereich von Hard- und Software (speziell von Embedded Systems) sind durch die fortschreitende Miniaturisierung und Leistungssteigerung viele Optionen hinzugekommen.

Für kleine Mikrocontroller gibt es verschiedene Standard-Betriebssysteme wie zum Beispiel VxWorks oder uC-Linux, die den TCP/IP Stack und andere Kommunikationsprotokolle bereits beinhalten. Diese Systeme werden derzeit als Kommunikationsmodule eingesetzt, die die sicheren Rechner der Streckensicherungstechnik von Kommunikationsaufgaben entlasten. Auch Field Programmable Gate Array (FPGA) Bausteine werden im Industriebereich immer beliebter. Die Hersteller bieten Bibliotheken von Kommunikationsprotokollen an, die durch Hardwarebeschreibungssprachen nicht an bestimmte Hardware gebunden sind. Für die Kommunikationsarchitektur der Streckensicherungstechnik stellen moderne Mikrocontroller und FPGA Bausteine eine Möglichkeit dar, ausgereifte Standard-Kommunikationsprotokolle zu verwenden, ohne Implementierungen von komplexen Protokollen vornehmen zu müssen.

SCTP - Stream Control Transmission Protocol.

Zukünftige Backbonenetze setzen teilweise das Internet Protocol (IP) als Netzwerkprotokoll voraus. In Verbindung mit dem IP stehen die weitestgehend bekannten Transportprotokolle UDP und TCP zur Verfügung. Ein neuer Standard dieser Familie, der bislang noch keinen großen Bekanntheitsgrad erreicht hat, jedoch durchaus als Transportprotokoll der Streckensicherungstechnik geeignet ist, ist das Stream Control Transmission Protocol (SCTP). Das SCTP [Ste07] ist ein Protokoll der Schicht 4 und kombiniert Funktionalitäten von TCP und UDP. Es ist als Nachfolger für das Common Channel Signaling System No. 7 (CSS7 oder SS7) entwickelt und im Jahr 2000 in der RFC 2960 (Übersicht in RFC 3286) veröffentlicht worden. Im September 2007 wurden in der RFC 4960 Erweiterungen definiert. Das SCTP arbeitet über IPv4 und IPv6 und wurde für hochverfügbare Systeme mit mehreren physikalischen Interfaces entwickelt. Das Protokoll stellt für die Streckensicherungstechnik eine Alternative zu TCP und UDP dar. SCTP besitzt das gleiche Portkonzept wie TCP und UDP. Eine Verbindung (Assoziation) zwischen zwei verteilten Anwendungen besteht aus m-zu-n vielen physikalischen Verbindungswegen, wobei der Datenverkehr über einen alternativen Pfad geleitet wird, falls der Primärpfad ausfällt.

Zu SCTP gehören folgende Dienste und Features:

- Verlässliche Datenübertragung von Process Data Unit (PDU)s (reihenfolgenrichtig und quittiert).
- Lose Datenübertragung (ohne Beachtung der Reihenfolge mit Quittierung).
- Vier-Wege Handshake beim Verbindungsaufbau.
- Multihoming (Redundanz, Unterstützung mehrerer Netzadressen für ein Gerät).

- Lebenstakte (Heartbeats) pro physikalischen Pfad, um Ausfälle schnell zu erkennen.

Die Tabelle 2.4 zeigt die Dienste von SCTP im Vergleich zu TCP und UDP [Car05]. Der Aufbau einer Verbindung zwischen zwei Endpunkten geschieht über ein Vier-Wege- Handshake Verfahren. Dieses wird eingesetzt, um ein Syn-Flooding (DoS Angriff) zu verhindern, wie es bei TCP möglich ist. Das SCTP bietet wie TCP eine verlässliche Datenübertragung mit einer 32 Bit Sequenznummer zum Erkennen von fehlenden oder bereits empfangenen Paketen. Wie auch bei TCP werden nach dem Ablauf eines Timers unquittierte Datenpakete wiederholt. SCTP besitzt ähnliche Fluss- und Staukontrollmechanismen wie TCP. Eine 32 Bit große CRC Prüfsumme sichert die Übertragung gegen Bitfehler ab. Ist ein Endgerät über mehrere Hardwareschnittstellen (mit unterschiedlichen IP Adressen, wobei auch IPv4 und IPv6 gleichzeitig vorhanden sein können) zu erreichen, dann ist dieses Endgerät Multihomed. Beim Aufbau einer SCTP Verbindung werden alle IP-Adressen ausgetauscht, über die das Gerät erreichbar ist. Der Datenaustausch findet über einen als Primär gekennzeichneten Pfad statt, wobei bei einem Verbindungsausfall auf einen Ersatzpfad umgeschaltet wird. Alle Pfade werden durch zyklische Heartbeat-Nachrichten permanent überwacht. Die physikalische Realisierung der Pfade spielt dabei keine Rolle. Eine Kombination von Kanälen mit unterschiedlichen Bandbreiten und Latenzzeiten ist in [Jun05] untersucht und optimiert worden. Eine übersichtliche und implementierungsnahe Beschreibung von SCTP ist in [SFR03] nachzuschlagen.

Tabelle 2.4: Vergleich von SCTP, TCP und UDP

Service	SCTP	TCP	UDP
Verbindungsorientiert	Ja	Ja	Nein
Zuverlässiger Datentransfer	Ja	Ja	Nein
Partiell zuverlässiger Datentransfer	Optional	Nein	Nein
Flusskontrolle	Ja	Ja	Nein
Staukontrolle	Ja	Ja	Nein
Selektives Quittungsverfahren	Ja	Optional	Nein
Erhalt von PDU Grenzen	Ja	Nein	Ja
PDU Fragmentierung	Ja	Ja	Nein
Zusammenfassen von PDUs	Ja	Ja	Nein
Multistreaming	Ja	Nein	Nein
Multihoming	Ja	Nein	Nein
Lebenstakte zum Kommunikationspartner	Ja	Ja	Nein

2.5.2 Lokale Netze (LAN)

Die massive Zunahme von vernetzten Rechnern in der Officedomäne machen Ethernet-basierte Kommunikationshardware günstig und für industrielle Anwendungen zunehmend interessanter. Um das Jahr 2003 wurden viele Ethernet-basierte Feldbussysteme veröffentlicht und Industrial Ethernet galt als die Ablösung der klassischen Feldbusse, die zueinander nicht kompatibel sind. Mit dem Focus auf preisgünstige und kompatible COTS Komponenten ersetzen die großen Hersteller von Automatisierungstechnik ihre klassischen Feldbusse durch Industrial Ethernet. Diese Strategie führte zu weit über 20 verschiedenen Ethernet-basierten Feldbussen, die heute am Markt erhältlich sind. Aus Applikationssicht sind die Industrial Ethernet Feldbusse nach wie vor nicht kompatibel zueinander, so dass in den meisten Fällen nur Netzwerkkomponenten wie zum Beispiel Switches, Kabel und Ethernet-Controller gemeinsam genutzt werden können. Da es im Feldbusbereich verschiedene Anwendungen mit unterschiedlichen Anforderungen gibt, hat sich Industrial Ethernet nicht zu einem Universalsystem entwickelt, sondern hat sich jeweils auf bestimmte Anwendungsbereiche spezialisiert. Vorangegangene Studien zum Einsatz von Industrial Ethernet in der Streckensicherungstechnik haben gezeigt, dass kein spezielles Feldbussystem die Anforderungen optimal erfüllt. Damit ist es nicht nötig, eine Herstellerbindung einzugehen, sondern stattdessen für die lokale Vernetzung Standard-Ethernet der Schicht 1 und Schicht 2 zu verwenden, wie es in IEEE 802.3 definiert ist. Auf spezielle Industrie-Spezifikationen oberhalb von Ethernet wird verzichtet.

2.5.3 Netzzugangstechnik

Eine wichtige Anforderung für die Kommunikationsdomäne Stellwerksbus ist die Vernetzung von Systemen über lokale Grenzen hinweg. Im klassischen Telekommunikationsbereich existieren Standardlösungen für diese Anforderungen in der Klasse Netzzugangstechnik [BMS04]. Diese Klassen sind für die Streckensicherungstechnik von Bedeutung, da einerseits Technologien wie zum Beispiel xDSL (das ‘x’ fasst hierbei ADSL, SDSL, SHDSL usw. zusammen) durch ihre Reichweite von mehreren Kilometern direkt für die Vernetzung zwischen Stellwerken und zwischen Stellwerken und der Außenanlage verwendet werden können. Provider-basierte Backbonetechnologien werden teilweise über die Netzzugangstechnik an lokale Netze angebunden. Neben einem breiten Spektrum von DSL-Techniken gehört zu dieser Klasse der Data Over Cable Service Interface Specification (DOCSIS) Standard, der Funkstandard WiMAX und die Fibre to the Home (FTTH) Technik.

2.5.4 Backbonenetze

Backbonenetze sind Infrastrukturen über große Distanzen. Diese Netze besitzen Ausdehnungen bis zu mehreren tausend Kilometern und können auch länderübergreifend installiert sein. Als moderne Übertragungstechnologien werden 10GbE und vor allem Next-Generation SONET/SDH gehandelt. Als physikalisches Übertragungsmedium kommen in der Regel Lichtwellenleiter zum Einsatz. Die Übertragung von Daten über große Distanzen ist ein Merkmal von elektronischen Stellwerken. Die Betreiber von Eisenbahn Infrastrukturen verwenden Backbone-Kommunikationsnetze für eine Vielzahl von Überwachungs-, Diagnose,- und Kommunikationssysteme. Es ist zu erwarten, dass zukünftig diese Netzwerke verstärkt für die Vernetzung von Streckensicherungssystemen verwendet und kostenintensive, exklusive Übertragungsmedien ersetzen werden. Eine Recherche nach den zu erwartenden Backbone-Infrastrukturen bei weltweiten Eisenbahngesellschaften hat ergeben, dass keine bestimmte Technologie bevorzugt wird, sondern jeder Betreiber sich für eine Technologie nach eigenen Anforderungen und Marktaspekten entscheidet [KEY08]. Die Ergebnisse der Recherche stammen aus Pressemitteilungen der jeweiligen Betreiber. Modernisiert ein Betreiber sein Backbone-Netz, dann kommen

- SDH und NG-SDH [OBB],
- MPLS [newa, newb] und
- Gigabit Ethernet [newc]

zum Einsatz. Weiterhin arbeitet das Metro Ethernet Forum (MEF) an einer Standardisierung für Backbone-Übertragungsdienste. Diese Technologie vereinigt beliebige Infrastrukturen und hat unter dem Namen Carrier Ethernet zum Ziel, Infrastruktur-Kunden über sogenannte Service-Level-Agreements Schnittstellen mit festgelegten Qualitätseigenschaften anzubieten [MEF]. Kunden bestellen darüber einen Vermittlungsdienst mit einer zuvor festgelegten Bandbreite und weiteren Qualitätseigenschaften, wie zum Beispiel, maximale Fehlerraten, Verfügbarkeit und Latenzzeiten. Die physikalische Realisierung bleibt dem Kunden verborgen. Die Anbindung erfolgt, wie aus dem Namen hervorgeht, mit Ethernet.

2.5.5 DB Systel propagiert All-over-IP Netzwerke

Nach Aussagen der DB Systel könnte sich die technologische Entwicklung in Richtung IMS (IP Multimedia Subsystem) vollziehen, bei dem das klassische Telefonnetz, das IP Netz und auch mobile Netze (zum Beispiel GSM Dienste) integriert werden. Als technologische Lösung gewinnt auch Multiprotocol Label Switching (MPLS) mehr an Bedeutung (was

für den Endkunden jedoch verborgen bleibt). Die Entwicklung vollzieht sich zu einem All-over-IP Netz, das dem Endkunden verschiedene Dienste zur Übertragung von Sprache, Video und Daten über ein gemeinsames Netzwerk anbietet. „So wird es in wenigen Jahren nur noch IP-Dosen geben und die Telefondosen werden nach und nach aussterben. Der Gedanke All-Over-IP wird sich durchsetzten und auch die Bahn wird sich dieser Idee nicht verschließen können“ (Holger Bleul, Chefarchitekt RZ-Netz, DB Systel GmbH. April 2008, per E-Mail).

2.6 Zusammenfassung

Standardisierte Kommunikationsprotokolle aus dem Industriebereich sind auf andere Anforderungen zugeschnitten, als in den Szenarien der Streckensicherungstechnik gefordert wird. Es gelten spezielle Normen, die höchste Anforderungen an Sicherheit fordern und gleichzeitig die Verwendung von Weitverkehrsnetzen eines Providers erlauben. Zur Feststellung der Zuverlässigkeit sind bislang nur Hardwarekomponenten einbezogen worden. Hierbei wurde die Zuverlässigkeit des logischen Kanals weitestgehend außer Acht gelassen. Als Argumentation galt bisher, dass die Zuverlässigkeit des logischen Kanals im Verhältnis zur Hardware bedingten Zuverlässigkeit quantitativ keine Rolle spielt. Bei den bisher verwendeten Infrastrukturen, die sehr geringe Fehlerraten aufweisen, ist dieses auch zutreffend. Es ist jedoch zu erwarten, dass durch den steigenden Kostendruck und den zunehmenden Datenverkehr (zum Beispiel Videoüberwachung von Bahnhöfen) Backbonenetze von Providern nicht an die bisherigen geringen Fehlerraten heranreichen. Des Weiteren machen Fortschritte in der drahtlosen Übertragungstechnik diese zu einer attraktiven Alternative für Kommunikationsverbindungen in Radien von etwa 10 km. Damit zukünftig von diesen Übertragungstechnologien profitiert werden kann, sind entsprechende Fehler- und Staukontroll-Mechanismen sowie Redundanztechniken erforderlich, die die Zuverlässigkeit des logischen Kanals von sicheren und echtzeitkritischen Applikationen in akzeptablen Bereichen halten. Die Zuverlässigkeit der Kommunikation beeinflusst die Life-Cycle-Costs, so dass eine scheinbar kostengünstige Infrastruktur mit ungeeigneten Kommunikationsverfahren unerwartete Folgekosten nach sich ziehen kann.

Die Referenzarchitektur der EN 50159 gibt einen Architekturrahmen vor, jedoch erlaubt dieser in einem gewissen Maße Freiheiten. Es ist nicht festgelegt, in welchem Teil der Architektur die beschriebenen Zuverlässigkeitsmechanismen implementiert sein müssen. Lediglich das Safety Protokoll muss auf der sicheren Hardware implementiert werden. Es ist erlaubt, Zuverlässigkeitsmechanismen auf nicht-sichere Hardwarekomponenten auszulagern und somit einerseits den sicheren Teil des Rechners zu entlasten und andererseits von Standardprotokollen wie zum Beispiel SCTP zu profitieren. Dieses ist sinnvoll, da

die sichere Hardware begrenzte Ressourcen zur Verfügung stellt und jede Änderung ein aufwendiges Zertifizierungsverfahren bedeutet. So können Protokolle mit Zuverlässigkeitsmechanismen unabhängig von Safety-Protokollen eingesetzt werden, wodurch das Safety-Protokoll entlastet wird. Damit ist es nötig, das Zusammenspiel zwischen einer sicheren Schicht und einer nicht-sicheren Zuverlässigkeitsschicht genau zu analysieren, um die jeweilige Toleranz gegenüber Übertragungsfehlern und Verzögerungszeiten zu ermitteln.

Safety-Protokolle für die Streckensicherungstechnik bleiben auch zukünftig spezielle Protokolle, die nicht aus dem Industriebereich stammen. Es gibt Standardisierungsbemühungen mehrerer Hersteller von Streckensicherungstechnik, um zwischen den sicheren Systemen eine einheitliche Architektur zu verwenden. Dieser Prozess ist momentan noch im Anfangsstadium und um die Standardisierung zu erleichtern ist eine Methode zum Beweis der Safety und zum Ermitteln von logischer Zuverlässigkeit erforderlich. Diese Arbeit definiert eine Methode zur Modellierung von sicheren Kommunikationsarchitekturen, deren formale Verifikation von Safety Eigenschaften und die Analyse der logischen Zuverlässigkeit. Im Kapitel 4 auf Seite 61 wird ein Framework vorgestellt, dass zum Erstellen sicherheitsrelevanter Kommunikationsarchitekturen und zum Entwurf von Safety-Protokollen benutzt werden kann. Die vorgegebenen Anforderungen aus den Normen EN 50126, EN 50128, EN 50129 und EN 50159 zum Design, Analyse, Verifikation und in Teilen auch zur Implementierung von Safety-Protokollen werden damit erfüllt.

Kapitel 3

Grundlagen zur Modellierung und formale Verifikationsmethoden

To err is human but to really foul up requires a computer.

Dan Rather, 2001

Kommunikationsarchitekturen der Streckensicherungstechnik müssen hohe Sicherheits- und Verfügbarkeitsanforderungen erfüllen. Mit dem technischen Hintergrund von Kommunikationsmechanismen, Protokollen, Fehlerarten und dem normativen Rahmen werden Methoden benötigt, die das Design sicherheitsrelevanter Kommunikationsarchitekturen unterstützen. Damit die Sicherheits- und Zuverlässigkeits-Eigenschaften zu einem frühen Entwicklungszeitpunkt nachgewiesen werden können, sind Architekturmodelle zu analysieren. Dabei werden zwei Hauptziele angestrebt: Die quantitative Verifikation von Safety-Eigenschaften und die qualitative Analyse der logischen Zuverlässigkeit der Kommunikation. Dieses geschieht mit Hilfe von Modellen, wobei auch die Korrektheit von Modellen durch den Nachweis bestimmter Eigenschaften gezeigt wird.

Die Realisierung dieser Ziele setzt Methoden aus drei Bereichen voraus: Modellierung von Kommunikationsarchitekturen, formale Verifikation von (Safety-) Anforderungen und Ermittlung von Zuverlässigkeitseigenschaften der Modelle. In diesem Kapitel werden daher State-of-the-Art Methoden zur Modellierung von Kommunikationsarchitekturen (Kommunikationsprotokolle) und automatische Beweis-Methoden (Model Checking) von Anforderungen vorgestellt.

3.1 Entwurf von Kommunikationsarchitekturen

Die Bestandteile von sicherheitsrelevanten Kommunikationsarchitekturen sind in drei Bereiche aufgeteilt: Auf der obersten Ebene befindet sich der Safety-Layer, der einem sicherheitsrelevanten Prozess einen Kommunikationsdienst zur Verfügung stellt und gemäß der CENELEC Normen Safety garantiert. Die zweite Ebene ist ein Kommunikationsdienst ohne Sicherheitsverantwortung (zum Beispiel ein COTS Kommunikationsmodul), der Protokolle der International Standards Organization (ISO)/Open Systems Interconnection (OSI) Schicht 1 bis Schicht 4 verwendet und den Datenaustausch über das Netzwerk übernimmt. Die dritte Ebene ist das physikalische Übertragungsnetzwerk. Der Entwurf von Kommunikationsarchitekturen bezieht sich auf die Festlegung von Protokollen und Funktionen der Safety-Ebene sowie der darunter liegenden Übertragungsschichten. Es können sowohl neue (Safety-) Protokolle spezifiziert, als auch existierende Protokolle in den jeweiligen Ebenen verwendet werden. Die meisten proprietären Kommunikationsprotokolle werden heutzutage empirisch entworfen. Hierbei wird selten von systematischen Entwurfsmethoden Gebrauch gemacht. Als Hauptgrund wird in [Koe03] die beschränkte Praktikabilität von existierenden Methoden genannt. Diesem Missstand kann mit einer intuitiven Modellierungssprache entgegengewirkt werden, die eine Analyse verschiedener Aspekte von sicherheitsrelevanten Kommunikationsarchitekturen erlaubt. Der zunächst zusätzliche Aufwand, der durch das Erstellen von Modellen entsteht, führt zu profitablen Ergebnissen, indem Designfehler während der Spezifikationsphase aufgedeckt werden.

Im vorangegangenen Kapitel 2 auf Seite 7 sind zwei Ziele von sicherheitsrelevanten Kommunikationsarchitekturen vorgestellt worden: Safety und Reliability. Zur Safety gehört die Fähigkeit, Fehlerarten der Übertragung zu erkennen sowie normgerecht zu reagieren. Es gilt zu beweisen, dass das Design frei von *Systematic Faults* ist (siehe Definition 2.3.7 auf Seite 22). Das Ziel Reliability kann nur mit der gesamten Architektur ermittelt werden. Zuverlässigkeitsmechanismen in der Architektur müssen in geeigneter Weise ausgewählt und platziert werden, um in Abhängigkeit von Sicherheitsanforderungen und erwarteten Kommunikationsfehlern Zuverlässigkeitsziele zu erreichen.

Für den formalen Nachweis, dass eine Spezifikation die festgelegten Anforderungen erfüllt, haben sich Model Checking Methoden etabliert. Der Entwurf von Kommunikationsarchitekturen wird von State-of-the-Art Analyse- und Beschreibungsmethoden unterstützt, um Designentscheidungen begründen zu können.

3.2 Analyse- und Beschreibungsmethoden

In der Literatur ist eine Vielzahl von Analysemethoden beschrieben, die entweder für die Verifikation von Safety-Eigenschaften oder zur Analyse von Zuverlässigkeit eingesetzt werden können. Ausgangspunkt sind informelle Spezifikationen der Architektur mit ihren Subkomponenten, die einen Katalog von Anforderungen erfüllen müssen. Das automatische Beweisen von Eigenschaften eines Modells der Architektur wird als Model Checking bezeichnet. Bereits seit den frühen 70er Jahren wird das Model Checking diskutiert und sowohl Formalismen zum Modellieren als auch die formale Beschreibung von Anforderungen erforscht. Der Begriff *formal* ist als ”genau“, “exakt” oder ”präzise“ zu verstehen [Kle09] und besagt, dass Aussagen maschinell nachgewiesen werden können.

Im Bereich von Kommunikationsprotokollen gehören die Specification and Description Language (SDL), Unified Modeling Language (UML) und Petrinetze zu den bekanntesten Modellierungssprachen. Eine Untersuchung von formalen Beschreibungsmethoden für Kommunikationsarchitekturen ist in [de 04] nachzuschlagen. Die Modellierungssprache, die ein Systemdesigner zum Beschreiben von Systemen verwendet, kann von der formalen Sprache zur Verifikation abweichen. So wird zum Beispiel die SDL zum Modellieren von Kommunikationsprotokollen benutzt. Diese Modelle werden aber dann in die Process Meta Language (Promela) übersetzt, um anschließend Verifikationen mit einem Model Checker durchzuführen. Damit wird eine intuitive (grafische) Modellierung ermöglicht und die oftmals schwer verständliche Model Checking Sprache bleibt verborgen. Auf diese Weise können Modelle einfacher erstellt werden, was die Akzeptanz bei Systemdesignern erhöht.

Seit einiger Zeit gewinnen Domänen spezifische Sprachen (DSL) an Popularität, die gegenüber einer General Purpose Language (GPL), wie zum Beispiel der UML Vorteile bieten können. Zum Beispiel kann mit einer DSL die Transformation von erstellten Modellen zu Modellen von Verifikations-Tools komfortabel definiert werden. Die Transformation ist damit automatisiert (per “Knopfdruck”), so dass ein Model Checking Tool direkt an die DSL angebunden ist. Neben der Verifikation von Safety-Eigenschaften (qualitativen Anforderungen) ist auch die Analyse der Zuverlässigkeit von Kommunikationsarchitekturmodellen als quantitative Eigenschaft nötig. Die Zuverlässigkeit ist eine probabilistische Größe, die besagt, mit welcher Wahrscheinlichkeit ein System in einem gegebenen Zeitintervall zuverlässig arbeitet. Diese Eigenschaft kann verifikationsformal oder simulativ ermittelt werden. Verifikationsformal bedeutet, dass ein Model Checker den Zustandsraum des Modells aufbaut und die Wahrscheinlichkeiten von zuverlässigen Zuständen berechnet. Hierbei wird ein exakter probabilistischer Wert ermittelt (ohne Vertrauensintervall). Das Aufbauen des Zustandsraums und die Berechnung aller Pfade zu zuverlässigen Zuständen

ist gegenüber einer Simulation sehr komplex. Simulationen haben jedoch den Nachteil, dass ein zufällig ausgewählter Pfad mit einem Vertrauensintervall behaftet ist, das sich nur mit einer großen Anzahl von Simulationsdurchläufen asymptotisch zu null verringern lässt. In [Chu08] ist der Zusammenhang zwischen der Anzahl an Simulationsdurchläufen und dem Vertrauensintervall dargestellt. Für Ergebnisse, die auf kleinen Wahrscheinlichkeiten basieren (zum Beispiel die Wahrscheinlichkeit eines fehlerhaften Informationspakets) ist der nötige Stichprobenumfang für ein akzeptables Vertrauensintervall sehr groß, so dass PMC mit exakten Ergebnissen und besserer Kontrolle über Ungenauigkeiten[1] verwendet wird.

3.3 Domänenspezifische Modellierung

Eine DSL ist im Gegensatz zu einer GPL eine an die Modellierungsdomäne angepasste Sprache. Die Frage, wann eine DSL gegenüber einer GPL Vorteile bietet, wird in Expertenkreisen viel diskutiert [Haa08, DF08]. Grundsätzlich lässt sich sagen, dass dieses nicht alleine von dem Modellierungsproblem abhängt, sondern von dem gesamten Entwicklungsvorhaben, in dem die Modellierungssprache nur einen kleinen Teil ausmacht. Eine GPL ist so gestaltet, dass sie für ein breites Spektrum von Modellierungsdomänen eingesetzt werden kann. Damit kann sie von vielen Anwendern benutzt werden und die angebundene Tool-Landschaft wächst stetig. Die bekannteste GPL ist die UML. Im industriellen Umfeld haben sich viele Entwickler an die UML "gewöhnt". Oftmals wird von administrativer Seite die Modellierungssprache UML vorgegeben, ohne einen Vergleich zu der Alternative DSL durchzuführen. Von Vorteil ist die leichte Adaptierbarkeit von domänenspezifischen Sprachen. Sofern sich neue Erkenntnisse während der Modellierung ergeben, können das Metamodell (Metasprache) und die Codegeneratoren entsprechend angepasst werden. Im Laufe der Zeit kann damit die Modellierungssprache optimal auf die Modellierungsdomäne abgestimmt werden. Zum Beispiel kann das Abstraktionsniveau erhöht oder zusätzliche Aspekte aufgenommen werden, die das gesamte Analyseframework erweitern. Vergleiche zwischen einer GPL und einer DSL [Mew09] sowie praktische Erfahrungen in der Arbeitsgruppe führten zu dem Entschluss, für diese Arbeit den DSL Ansatz zu verwenden und Anhand von Fallstudien zu bewerten.

Eine DSL muss von einem Domänenexperten zunächst erstellt werden. Auf Basis eines Meta-Meta-Modells wird die DSL (Metamodell) definiert. Dabei werden Besonderheiten aus der Modellierungsdomäne berücksichtigt: Alle Sprachelemente werden auf ein mög-

[1] Die Definition einer Fehlerhypothese beeinflusst die Zustandsraumgröße. Erlaubt der Modellierer n viele Übertragungsfehler, dann ist das Ergebnis mindestens so genau, wie die Wahrscheinlichkeit das n Fehler auftreten.

lichst hohes Abstraktionsniveau gebracht, um den Modellierungsaufwand von Systemen der Domäne gering zu halten. Zudem ist die Zielsprache zu berücksichtigen, in die die Modelle transformiert werden. Das Transformieren geschieht über Transformationsregeln (Codegeneratoren) automatisch. Codegeneratoren transformieren (grafische) Modelle einer DSL in beliebige textuelle Repräsentationen anderer Modelle. Eine DSL ist aus diesem Grund sowohl an die Modellierungsdomäne, als auch an die Tool-Sprachen zur Weiterverarbeitung angepasst.

Das Design von Kommunikationsarchitekturen wird mit Model Checking Tools auf Safety und Reliability Eigenschaften zu einer frühen Designphase analysiert (Abbildung 4.2 auf Seite 64). Neben der Transformation von Architekturmodellen zu Model Checking Modellen ist die Generierung einer (Rahmen-) Implementierung aus den verifizierten Modellen von entscheidendem Vorteil: Es steigert die Qualität durch Verringerung von Implementierungsfehlern, senkt die Entwicklungskosten und erhöht die Akzeptanz als Designframework. Der zusätzliche Modellierungsaufwand wird durch verringerte Aufwendungen bei der Implementierung kompensiert und Fehler werden durch die bewiesene Korrektheit reduziert. Insgesamt gibt es daher folgende Anforderungen an eine Modellierungssprache:

1. Modellierung von Kommunikationskanälen,
2. funktionale Beschreibung von Kommunikationsprozessen,
3. Transformation der Modelle zu Model Checking Tools,
4. Transformation von Teilen der Modelle zu Implementierungen[2].

3.4 Domänenspezifische Modellierung mit MetaEdit+

Das Tool MetaEdit+® [Met09, KT08] besteht aus einem Meta-Modellierungs-Framework zum Erstellen von Metamodellen und einer Workbench, in der auf Basis der Metamodelle Modelle erstellt werden. Das Meta-Meta-Modell von MetaEdit+ heißt Graph Object Property Port Role Relationship (GOPPRR) [Met08] und steht für die Basiselemente, aus denen Metamodelle erstellt werden. Mit der Festlegung von Objekten, ihrer grafischen Repräsentation, Properties, Ports und Beziehungen zwischen Objekten wird eine Metasprache definiert. In der abstrakten Syntax wird festgelegt, wie Objekte über Relationships, Roles und Ports miteinander verbunden werden können. Jedes dieser Elemente kann Properties von unterschiedlichen Typen besitzen. Mit diesem Konzept werden domä-

[2] Die Transformation von Analysemodellen zu Implementierungen ist hier nicht realisiert, aber eine mögliche Erweiterung der in Kapitel 4.3 auf Seite 68 vorgestellten DSL.

nenspezifische Modellierungssprachen erstellt, die nur die für die Modellierung benötigte Elemente beinhalten.

Zu jeder DSL werden generische Codegeneratoren in der Skriptsprache MetaEdit+ Reporting Language (Merl) definiert. Ein Codegenerator ist ein Model-to-Text Transformator, der aus den Modellen der Metasprache eine beliebige textuelle Repräsentation erzeugt. Diese Repräsentation kann zum Beispiel ein Modell für einen Model Checker sein. Voraussetzung dafür ist eine syntaktisch korrekte Transformation zum Zielmodell. Hierzu bietet MetaEdit+® mehrere Möglichkeiten, eine domänenspezifische Sprache mit verschiedenen Restriktionen zu versehen, so dass beim Erstellen von Modellen dieser Sprache syntaktische Bedingungen der Zielmodelle eingehalten werden. MetaEdit+® eignet sich als Modellierungs-Frontend eines Frameworks, an das durch die Codegeneratoren, Tools zur Weiterverarbeitung integriert sind.

3.5 Model Checking und Tools

Als Model Checking wird allgemein ein algorithmisches Verfahren bezeichnet, dass eine formalisierte Anforderung auf einer formalen Systembeschreibung (Systemmodell) verifiziert. Die ersten Model Checking Verfahren sind zu Beginn der 80er Jahre angewendet worden [Cla08]. Ein Modellierungsformalismus für Echtzeitsysteme ist Timed Automata. Zu den Systemmodellen wird eine formale Beschreibung von Systemeigenschaften aufgestellt, die ein System erfüllen muss. Die formale Beschreibung von Systemeigenschaften wird in Kapitel 3.6 auf Seite 44 vorgestellt. Es gibt weitere Modellierungs- und Spezifikationsformalismen, die jeweils an bestimmte Problemebereiche angepasst sind. Ein Überblick über State-of-the-Art Methoden und Tools ist in [Wan04] beschrieben.

Grundsätzlich ist das Model Checking ein komplexes Problem, so dass bei der Modellierung auf eine geeignete Darstellung des Systems zu achten ist. Wird ein Systemmodell mit zu vielen Details beschrieben, so kann dieses zu einer sogenannten Zustandsraumexplosion führen. Damit ist gemeint, dass die Anzahl von Systemzuständen zu groß ist, um in akzeptabler Zeit oder mit “normalem” Speicherbedarf Anforderungen zu verifizieren. Ein hilfreiches Mittel bei der Systemverifikation ist das Generieren von Gegenbeispielen, falls eine Anforderung von dem Systemmodell nicht erfüllt wird. In diesem Fall erzeugt der Model Checker einen Ausführungspfad des Systems, der die zu beweisende Anforderung verletzt. Edmund M. Clark hat zu diesem Feature gesagt: „It is impossible to overestimate the importance of the counterexample feature. The counterexamples are invaluable in debugging complex systems. Some people use Model Checking just for this feature.“ ([Cla08]) In Abbildung 3.1 auf der nächsten Seite ist das Model Checking Verfahren skizziert. Die Timed Automata Theorie und die Formulierung von Anforderungen in Timed

Computation Tree Logic (TCTL) sind Grundlagen dieser Arbeit. Im Folgenden ist daher ein kurzer Überblick über Timed Automata gegeben. Eine detailliertere Beschreibung zum Thema Model Checking und Timed Automata ist in [JGP99, BY04, BK08] zu finden.

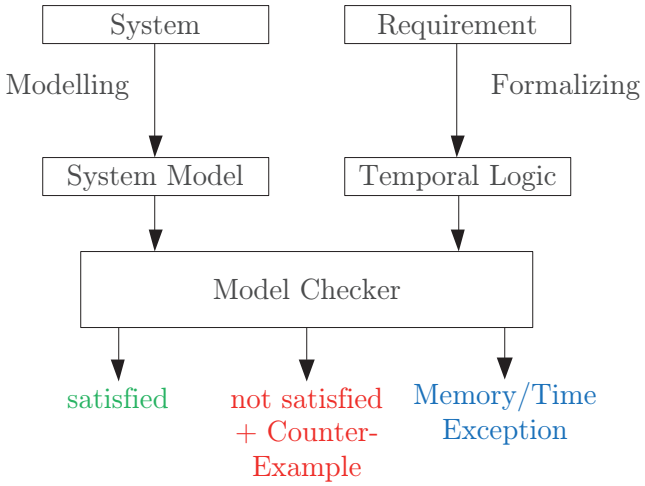


Abbildung 3.1: Prinzip des Model Checkings

Als typische Zustandsautomaten bestehen Timed Automata aus einer Menge von Locations, die über Transitionen miteinander verbunden sind. Die Timed Automata Theorie geht zurück auf Alur und Dill in [AD94] und wird zum Modellieren von Echtzeitsystemen verwendet. Timed Automata benutzen Clocks, um das Voranschreiten von Zeit zu modellieren. Henzinger et. al. führten den *Timed Safety Automata* [HNSY92] ein. Ein Timed Safety Automata definiert zwei Bedingungen auf Clocks: Guard-Bedingungen und Invarianten. Eine Guard-Bedingung an Transitionen legt fest, unter welchen Zeitbedingungen die Transition benutzt werden darf. Eine Invariante an Locations gibt an, unter welchen Zeitbedingungen sich ein Prozess in der jeweiligen Location befinden darf. Invarianten dürfen nicht verletzt werden, so dass ein Prozess gezwungen wird, nach endlicher Zeit eine Location zu verlassen. Mit Invarianten können zeitliche Eigenschaften von Systemen, realistisch abgebildet werden. In dieser Arbeit wird nur der Timed Safety Automata behandelt, wobei die übliche Bezeichnung Timed Automata benutzt wird.

Die Timed Automata Theorie ist in verschiedenen Model Checking Tools umgesetzt, zum Beispiel in Uppaal [LPY97] und Kronos [Yov97]. Seit Mitte der 90er Jahre sind Model Checking Tools in einem praxistauglichen Grad verfügbar und damit Fallstudien von industriellen Systemen durchgeführt worden. Mit dieser Arbeit wird das Verifizieren von Kommunikationsprotokollen und -Architekturen behandelt. Hierzu existieren

ebenfalls verschiedene Fallstudien in [JLS96, DY00, DKRT97, RSV10]. Bevor mit der eigentlichen Arbeit begonnen werden kann, muss zunächst ein geeignetes Model Checking Tool festgelegt werden. Das Ziel ist es, Systemdesignern ein Framework zum Verifizieren und Analysieren von Systemmodellen zur Verfügung zu stellen. Das Tool muss daher gut Dokumentiert sein, Features für das strukturierte Modellieren bereitstellen, bereits einige Jahre erprobt worden sein und durch die ständige Verbesserung von Model Checking Theorien weiterhin gepflegt werden. Auf der Grundlage dieser Auswahlkriterien und den guten Erfahrungen im universitären Umfeld ist für diese Arbeit die Entscheidung auf den Uppaal Model Checker gefallen. Das Kapitel 3.7 auf der nächsten Seite erläutert dieses Tool. Andere bekannte Model Checking Tools, wie zum Beispiel SPIN [BA08], KRONOS [BDM+98] und FDR2 [But04] kommen für diese Arbeit aus verschiedenen Gründen nicht in Frage. Das Tool KRONOS wird seit einiger Zeit nicht mehr gepflegt und ist somit nicht mehr State-of-the-Art. Die Tools SPIN und FDR2 sind keine Echtzeit Model Checker. Es können zwar auch nicht-Echtzeit Model Checker mit einer expliziten Zeitbeschreibung zeitbehaftete Systeme modellieren [Lam05], allerdings hat sich das Uppaal Tool mit vielen Fallstudien, Features und Usability als Timed Automata Model Checker fest etabliert.

3.6 Model Checking und Temporallogik

Die Temporallogik ist eine Erweiterung der Aussagenlogik um Zeitbeziehungen zwischen logischen Aussagen. Dabei wird keine kontinuierliche Zeit betrachtet, sondern nur eine Vorher-Nachher-Beziehung. Zu den wichtigsten temporalen Logiken gehören die Linear Temporal Logic (LTL) und die Computation Tree Logic (CTL). In [Pin02, BK08] sind LTL und CTL von der theoretischen Seite detailiert erläutert. Die Temporallogiken LTL und CTL besitzen eine ähnliche Ausdrucksstärke. Es hängt von dem Model Checking Tool ab, welche dieser Formalismen (oder eine Abwandlung davon) für die Formalisierung von Anforderungen verwendet wird. Mit der Temporallogik wird eine Aussage über die Entwicklung von Systemzuständen getätigt. Mit mehreren temporallogischen Aussagen über das Verhalten eines Systems werden wesentliche Eigenschaften nachgewiesen. Als Beispiel ist hier die Steuerung einer Fußgängerampel genannt. Die Steuereinheit liegt als Modell vor und beschreibt den funktionalen Ablauf des Ampelprogramms. Mit Hilfe der Temporallogik werden wesentliche Anforderungen der Ampelsteuerung formal beschrieben. Zum Beispiel darf niemals für die Fußgänger und für die Autos das grüne Signal gleichzeitig ausgegeben werden. Eine weitere Anforderung kann zur Lebendigkeit getätigt werden: Nachdem ein Fußgänger für den Überweg eine Anfrage ausgelöst hat, muss das Fußgängersignal schließlich einmal grün zeigen.

Diese informalen Anforderungen werden als temporallogische Formel Φ beschrieben. Der Model Checker prüft anhand des Systemmodells M, ob eine Aussage Φ erfüllt ist. Erfüllt das Systemmodell die Anforderung, so wird dieses als $M \models \Phi$ notiert.

Das Model Checking Tool UPPAAL benutzt als Temporallogik TCTL.

3.7 Der UPPAAL Model Checker

Der Model Checker UPPAAL ist von den Universitäten in Uppsala (Schweden) und Aalborg (Dänemark) gemeinsam entwickelt worden. Der Model Checker verifiziert oder falsifiziert temporallogische Aussagen über UPPAAL Modelle. Als Aussagenlogik verwendet UPPAAL TCTL, wobei CTL um Aussagen über Clocks erweitert ist. Die TCTL von UPPAAL unterstützt nur eine Teilmenge der CTL Formeln. Das typische Anwendungsgebiet von UPPAAL ist die Verifikation von Echtzeitsystemen. Systemmodelle bestehen aus mehreren parallelen Prozessen, die jeweils als Zustandsdiagramm beschrieben sind. Prozesse sind über Kanäle miteinander synchronisiert. Neben den Zustandsdiagrammen beinhaltet die Systembeschreibung lokale und globale Variablen. Zu den Basistypen Integer, Boolean und Clock können mit Arrays und Structs komplexe Datenstrukturen definiert werden. Des Weiteren besteht die Möglichkeit, Funktionen in einer eingeschränkten C-Syntax zu definieren. Mit diesen Features können Modelle strukturiert und übersichtlich erstellt werden. Zustandsdiagramme werden als Templates angelegt. Von einem Template können beliebig viele Prozesse mit spezifischen Parametern instanziiert werden. Für eine gute Übersicht zu dem UPPAAL Tool mit seinen Features sowie Beispiel-Modellen und eine ausführliche semantische Beschreibung sei auf [Möl02, BDL04, TS06, DILS09] verwiesen. Im Abschnitt 3.7.1 wird ein Systemmodell in UPPAAL als Beispiel vorgestellt.

Systemmodelle werden in einer grafischen Oberfläche erstellt. Diese werden als Klartext im Extensible Markup Language (XML) Format gespeichert, so dass UPPAAL auch als Backend benutzt werden kann. UPPAAL Modelle werden dabei von einem anderen Modellierungstool generiert. Für die Verifikation der Modelle werden TCTL Formeln erstellt, die jeweils eine Eigenschaft formal beschreiben. Der UPPAAL Model Checker expandiert den Zustandsraum des Systemmodells und verifiziert die TCTL Formeln. Ist eine Formel nicht erfüllt, dann gibt das UPPAAL Tool einen Ausführungspfad aus, der zur Falsifizierung der Formel führt.

3.7.1 Beispiel Ampelsteuerung

Die Abbildung 3.7.1 auf Seite 47 zeigt das Systemmodell einer Fußgängerampel, das aus einem Controller, den beiden Lichtzeichen für Fußgänger und Verkehr und dem Ampel-

knopf besteht. An den Locations ist jeweils der Name und eine optionale Invariante notiert. Die Transitionen sind mit einer Guard-Bedingung, einem Synchronisierungskanal und einer Update-Anweisung versehen. Transitionen erlauben einen Übergang, wenn die Guard-Bedingung erfüllt ist und eine optionale Synchronisation zu einem anderen Prozess möglich ist. Transitionen werden in Nullzeit ausgeführt. Dieses einfache und intuitive Beispiel gibt einen Einblick in die Modellierung mit UPPAAL.

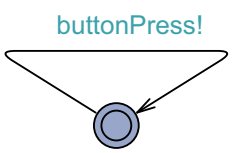
Der Prozess **Button** modelliert den Übergangswunsch eines Fußgängers. Der Knopfdruck ist über den Kanal *pressButton* mit dem Controller synchronisiert. Der **Controller** Prozess ist ein abstraktes Modell der Steuerung. Mit dem Übergangswunsch und einem Mindestaufentalt von 20 Zeiteinheiten in der Location *idle* beginnt ein Ampelzyklus. Die Urgent-Location *startCycle* muss ohne das Verstreichen von Zeit wieder verlassen werden. Die Variable c ist eine real-time Clock. Der Prozess **WalkerLight** modelliert das Fußgänger-Ampellicht. Über den Kanal *walkGreen* und *walkRed* wird das jeweilige Ampellicht dargestellt. Der Prozess **TrafficLight** modelliert das Verkehrsampel-Licht. Die jeweiligen Zwischenphasen sind zeitgesteuert integriert.

Mit dem Model Checker ist eine Safety-Eigenschaft nachzuweisen: Es darf niemals für Fußgänger und den Verkehr gleichzeitig das Ampelsignal "grün" gezeigt werden. In dem Systemmodell bedeutet die Eigenschaft $A\Box\neg(\mathit{WalkerLight.green} \wedge \mathit{TrafficLight.green})$, dass auf allen Ausführungspfaden des Modells sich niemals der Prozess *WalkerLight* und der Prozess *TrafficLight* in ihrer jeweiligen Location *green* gleichzeitig befinden dürfen. Der Model Checker expandiert alle möglichen Ausführungspfade (Sequenzen von aufeinander folgenden Zuständen) und prüft, ob diese Eigenschaft erfüllt ist[3].

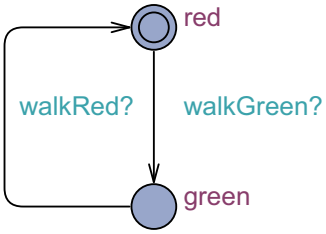
3.7.2 Formale Syntax und Semantik des UPPAAL Model Checkers

Formal bestehen UPPAAL Modelle aus parallelen Prozessen, wobei jeder Prozess als Zustandsdiagramm beschrieben wird. Prozesse können Integer und Boolesche Variablen ($Vars$) sowie Echtzeit-Uhren Clocks (C) und Synchronisationskanäle (Ch) verwenden. Integer und Boolesche Variablen haben einen festgelegten Wertebereich, der auch als Domäne $\mathcal{D}_x$ einer Variable x bezeichnet wird. Zur besseren Strukturierung können Arrays und Strukturen von Variablen angelegt werden. Eine *Guard-Bedingung* ist ein Boolescher Ausdruck über Variablen und Clocks, die Transitionen zwischen Locations aktiviert. Guards über Clocks dürfen nur in der Form $x \bowtie \mathit{int}$ oder $x - y \bowtie \mathit{int}$ angegeben werden, mit $x, y \in C$ und $\bowtie \in \{<, \leq, ==, \geq, >\}$. Eine Invariante I wird in der Form $x \lhd \mathit{int}$ oder $x - y \lhd \mathit{int}$ angegeben, mit $x, y \in C$ und $\lhd \in \{<, \leq\}$ und int evaluiert zu einer ganzen Zahl. Eine Synchronisation ist ein Handshake zwischen zwei Prozessen, wobei eine Transition mit $ch!$ eines Prozesses

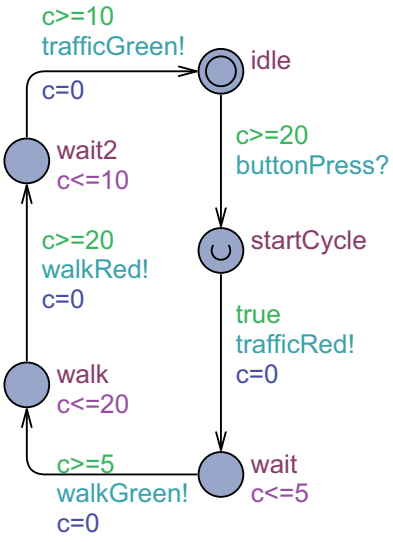
[3] Das Beispielmodell erfüllt diese Eigenschaft



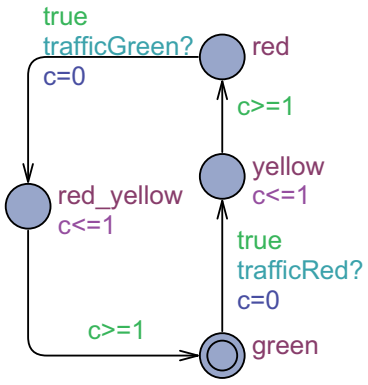
(a) Prozess Button



(b) Prozess WalkerLight



(c) Prozess Controller



(d) Prozess TrafficLight

Abbildung 3.2: Systemmodell einer Ampelsteuerung

i simultan mit der Transition $ch?$ eines Prozesses j ausgeführt wird, mit $i \neq j$. *Locations* werden als *timed*, *urgent* oder *committed* deklariert. Die als committed deklarierten Locations müssen vor urgent und timed Locations verlassen werden. Solange sich ein Prozess in einer urgent oder committed Location befindet, darf keine Zeit vergehen. Die folgende formale Beschreibung ist an [Möl02] angelehnt. Es sind nur die Teile des UPPAAL Model Checkers beschrieben, die in dieser Arbeit verwendet werden. Prioritäten von Prozessen und die Broadcast Synchronisation gehören nicht dazu.

Definition 3.7.1 (UPPAAL **Prozess**)**.** Ein UPPAAL *Prozess* ist ein Tupel $(L, l^0, Type, Inv, E)$, mit

- L ist eine Menge von Locations,
- l^0 ist die initiale Location,
- $Type : L \to \{timed, urgent, committed\}$ ist eine Funktion, die jeder Location $l \in L$ den Typ *timed*, *urgent* oder *committed* zuordnet,
- $Inv : L \to I$ ordnet einer Location eine Invariante zu (ist im Modell für l keine Invariante angegeben, dann ist $Inv(l) = true$),
- E ist eine Menge von Transitionen $l \xrightarrow{g/ch/a} l'$, mit $l, l' \in L$, g ist eine Guard-Bedingung, ch ist ein Synchronisations-Label und a eine Liste von Zuweisungen.

Definition 3.7.2 (UPPAAL **Modell**)**.** Ein UPPAAL *Modell* ist ein Tupel $(\bar{A}, Vars, C, Ch)$, mit

- $\bar{A}$ ist ein Vektor von n Prozessen $A_1, \ldots, A_n$,
- $Vars$ ist eine Menge von bounded Integer- und Booleschen-Variablen,
- C ist eine Menge von Echtzeit-Uhren,
- Ch ist eine Menge von Synchronisations-Labels.

Definition 3.7.3 (UPPAAL **Konfiguration**)**.** Es ist $s = \langle \bar{l}, e, v \rangle$ eine *Konfiguration* eines Models $(\bar{A}, Vars, C, Ch)$, mit:

- $\bar{l} = (l_1, \ldots, l_n)$, $l_i \in L_i$ ist die Location von Prozess A_i,
- $e : Vars \to \mathcal{D}$ bildet jede Variable auf einen Wert ihres Wertebereichs ab ($\mathcal{D} = \bigcup_{x \in Vars} \mathcal{D}_x$),

- $v : C \rightarrow \mathbb{R}_{\geq 0}$ bildet jede Uhr auf einen nicht-negativen reellen Wert ab. Die nachfolgend verwendete Notation $(v + \delta)$ bedeutet, dass alle Clocks um δ inkrementiert werden.

Die initiale Konfiguration ist definiert als $s_0 = \langle (l_1^0, \ldots, l_n^0), [Vars \mapsto 0], [C \mapsto 0] \rangle$: Alle UPPAAL Prozesse $A_1, \ldots A_n$ befinden sich in ihrer initialen Location, alle Variablen werten zu 0 aus und alle Clocks sind auf 0 zurückgesetzt. Beginnend mit der initialen Konfiguration entwickeln sich weitere Konfigurationen eines UPPAAL Modells nach drei Transitionsregeln: *Delay Transition*, *Action Transition* und *Synchronisation Transition*. Nachfolgend werden bestimmte Notationen verwendet, die hier kurz beschrieben sind: Die Notation $e, v \models \varphi$ bedeutet, dass eine Evaluation e und v eine Boolesche Bedingung φ erfüllt. Die Notation $\bar{l}[l'_i/l_i]$ gibt an, dass die Location l_i durch die Location l'_i im Location-Vektor $\bar{l}$ ersetzt wird. Mit der Notation $a(e)$ (und $a(v)$) wird die resultierende Evaluation e (und v) nach der Ausführung der Update-Anweisungen a angezeigt.

Definition 3.7.4 (**Delay Transition**)**.** Für die Konfiguration $s = \langle \bar{l}, e, v \rangle$ ist eine *Delay Transition* $s \xrightarrow{\delta} \langle \bar{l}, e, (v + \delta) \rangle$ aktiviert, wenn folgende Bedingungen erfüllt sind:

(i) $\forall d', 0 \leq d' \leq \delta : (e, (v + d')) \models \bigwedge_{i=1}^{n} Inv(l_i)$ und

(ii) $\forall l$ aus $\bar{l}$: $Type(l) \neq committed \wedge Type(l) \neq urgent$, das heißt, der Location-Vektor beinhaltet weder committed, noch urgent Locations.

Definition 3.7.5 (**Action Transition**)**.** Für die Konfiguration $s = \langle \bar{l}, e, v \rangle$ ist eine *Action Transition* $s \xrightarrow{a} \langle \bar{l}[l'_i/l_i], a(e), a(v) \rangle$ aktiviert, falls $l_i \xrightarrow{g/a} l'_i \in E$, l_i in $\bar{l}$ und wenn folgende Bedingungen erfüllt sind:

(i) $e, v \models g$ und

(ii) $a(e), a(v) \models Inv(l'_i)$ und

(iii) wenn $\exists l_c$ in $\bar{l}$ mit $Type(l_c) = committed$, dann ist $Type(l_i) = committed$.

Definition 3.7.6 (**Synchronisation Transition**)**.** Für die Konfiguration $s = \langle \bar{l}, e, v \rangle$ ist eine *Synchronisation Transition* $s \xrightarrow{s!?} \langle \bar{l}[l'_i/l_i, l'_j/l_j], a_j(a_i(e)), a_j(a_i(v)) \rangle$ aktiviert, wenn zwei Transitionen $l_i \xrightarrow{g_i/ch!/a_i} l'_i \in E$ und $l_j \xrightarrow{g_j/ch?/a_j} l'_j \in E$, l_i, l_j in $\bar{l}$, $i \neq j$ existieren und wenn folgende Bedingungen erfüllt sind:

(i) $e, v \models g_i \wedge g_j$ und

(ii) $a_j(a_i(e)), a_j(a_i(v)) \models Inv(l'_i) \wedge Inv(l'_j)$ und

(iii) wenn $\exists l_c$ in $\bar{l}$ mit $Type(l_c) = committed$, dann ist $Type(l_i) = committed \vee Type(l_j) = committed$.

Der Zustandsraum eines Modells besteht aus allen möglichen Folgen von Konfigurationen, die von der initialen Konfiguration nach den drei Transitionsregeln erreicht werden können. Da die Delay-Transition das reellwertige Voranschreiten von Zeit definiert, ist die Anzahl von Konfigurationen unendlich. Diesem Problem wird in [AD90] begegnet, indem die Evaluationen von Clocks zu Clock-Regions $[v]$ zusammengefasst werden, in denen die Evaluation von Variablen e und der Location-Vektor $\bar{l}$ unverändert sind. Die Clock-Evaluationen werden damit von einer unendlichen reellwertigen Menge auf eine endliche Menge von Wertebereichen abstrahiert. Eine ausführliche Beschreibung ist in [JGP99, BK08, BL08] nachzulesen. Mit dieser Abstraktion (und weiteren Restriktionen [AM04]) werden Timed Automata entscheidbar und der Zustandsraum (auch Computation Tree oder State Graph genannt) endlich.

Definition 3.7.7 (**Computation Tree CT**)**.** Der *Computation Tree CT* eines UPPAAL Modells $(\bar{A}, Vars, C, Ch)$ ist ein gerichteter Graph von erreichbaren Zuständen $s = \langle \bar{l}, e, [v] \rangle$, beginnend von dem initialen Zustand $s_0 = \langle (l_1^0, \ldots, l_n^0), [Vars \mapsto 0], [C \mapsto 0] \rangle$. Ein Zustand s_{k+1} ist von einem Zustand s_k erreichbar, wenn eine der drei Transitionsregeln

(i) $s_k \xRightarrow{\delta} s_{k+1}$,

(ii) $s_k \xRightarrow{a} s_{k+1}$,

(iii) $s_k \xRightarrow{s!?} s_{k+1}$

aktiviert ist. Die Notation $CT(M)$ beschreibt den Computation Tree von einem Modell M.

Definition 3.7.8 (**Pfad** π)**.** Ein Pfad $\pi_{s_i} = \langle s_i, s_{i+1}, s_{i+2}, \ldots \rangle$ ist eine (unendliche) Sequenz von aufeinander folgenden Zuständen in dem Computation Tree CT. Mit π_{s_i} wird ein Pfad ausgewählt, der beim Zustand s_i beginnt. Mit der Notation $\pi_{s_i}(j)$ wird der j-te Zustand aus der Sequenz $\pi_{s_i} = \langle s_i, s_{i+1}, \ldots, s_{i+j}, \ldots \rangle$ beschrieben.

Definition 3.7.9 (**Lokale Eigenschaft** φ)**.** Eine *lokale Eigenschaft* ist ein Boolescher Ausdruck φ über Variablen aus $Vars$, Clocks aus C und Locations aus dem Location-Vektor $\bar{l}$. Mit der Schreibweise $s_k \models_{loc} \varphi$ wird angegeben, dass ein Zustand $s_k = \langle \bar{l}, e, [v] \rangle$ die Eigenschaft φ erfüllt.

3.7.3 TCTL Temporallogik von UPPAAL

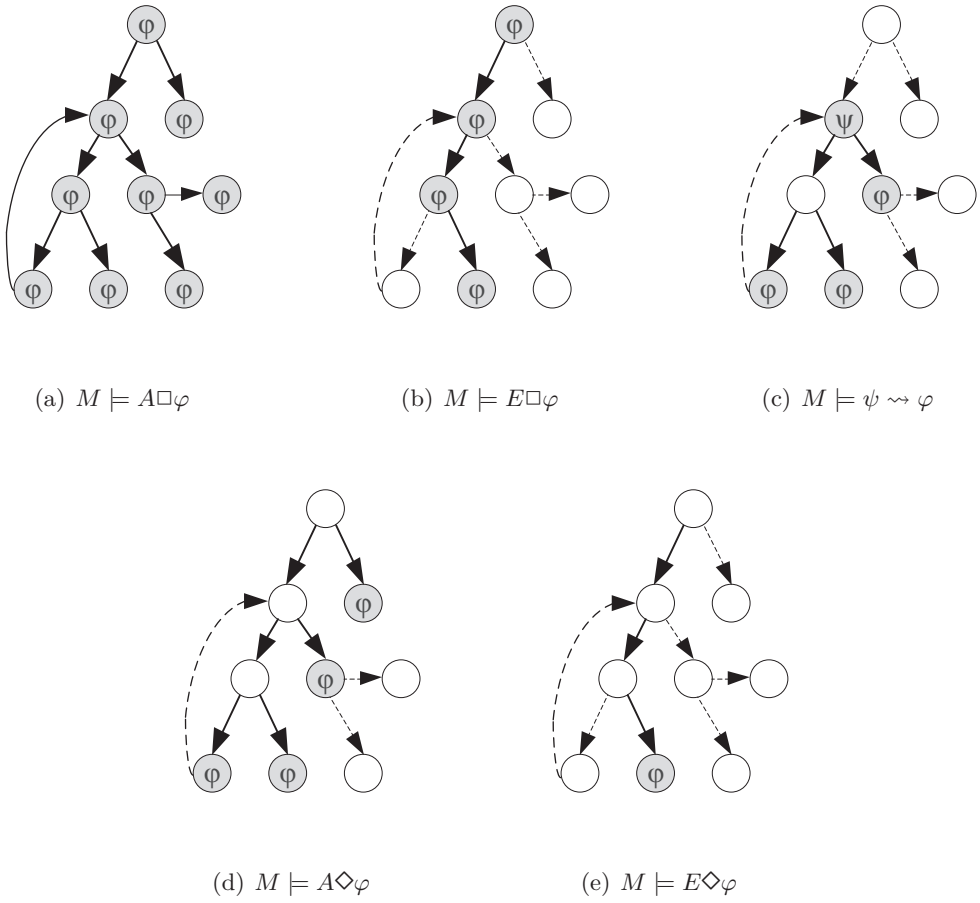
Eine Systemeigenschaft wird formalisiert, indem mit einem Pfadquantor A (Always) oder E (Exists) und einem Temporalquantor $\Diamond$ ('F', Finally) oder $\Box$ ('G', Globally) definiert wird, wie eine lokale Eigenschaft φ in einem Computation Tree gilt. In UPPAAL sind fünf verschiedene temporallogische Aussagen möglich:

1. $A\Box\varphi$: Eine Aussage φ gilt immer auf allen möglichen Ausführungspfaden.
2. $E\Box\varphi$: Es existiert ein Ausführungspfad, auf dem φ immer gilt.
3. $\psi \rightsquigarrow \varphi$, Kurzform von $A\Box(\psi \Rightarrow A\Diamond\varphi)$: Wenn einmal ψ gilt, dann gilt von dort aus auf allen Pfaden schließlich einmal φ.
4. $A\Diamond\varphi$: Auf allen Ausführungspfaden gilt schließlich einmal φ.
5. $E\Diamond\varphi$: Es existiert ein Ausführungspfad, auf dem schließlich einmal φ gilt.

Es gilt die Äquivalenz von $A\Box\varphi \equiv \neg(E\Diamond(\neg\varphi))$ und $A\Diamond\varphi \equiv \neg(E\Box(\neg\varphi))$. Es wird die Notation $M \models \Phi$ verwendet, wenn ein UPPAAL Systemmodell M die Systemeigenschaft Φ besitzt, wobei Φ eine der oben genannten temporallogischen Aussagen ist. Anhand des Systemmodells berechnet der Model Checker den Computation Tree und prüft die Systemeigenschaft. Abbildung 3.3 auf der nächsten Seite illustriert die Gültigkeit von TCTL Formeln in einem Computation Tree, wobei ψ, φ lokale Eigenschaften sind. Die Semantik der oben aufgeführten temporallogischen Formeln bezüglich eines Modells M ist definiert als:

1. $M \models A\Box\varphi \quad \Leftrightarrow \forall\pi_{s_0} \in CT(M), \forall j \geq 0 : \pi_{s_0}(j) \models_{loc} \varphi$
2. $M \models E\Box\varphi \quad \Leftrightarrow \exists\pi_{s_0} \in CT(M), \forall j \geq 0 : \pi_{s_0}(j) \models_{loc} \varphi$
3. $M \models \psi \rightsquigarrow \varphi \quad \Leftrightarrow \exists i, \pi_{s_0} \in CT(M) : \pi_{s_0}(i) \models_{loc} \psi \Rightarrow \forall\pi_{s_i} : \exists j \geq 0 : \pi_{s_i}(j) \models_{loc} \varphi$
4. $M \models A\Diamond\varphi \quad \Leftrightarrow \forall\pi_{s_0} \in CT(M), \exists j \geq 0 : \pi_{s_0}(j) \models_{loc} \varphi$
5. $M \models E\Diamond\varphi \quad \Leftrightarrow \exists\pi_{s_0} \in CT(M), \exists j \geq 0 : \pi_{s_0}(j) \models_{loc} \varphi$

Mit der vorgestellten Temporallogik TCTL werden qualitative Eigenschaften von Modellen bewiesen. Diese sind als *Reachability*, *Safety* und *Liveness* Eigenschaften klassifiziert. **Reachability** Eigenschaften ($E\Diamond\varphi$) zeigen, ob es auf dem Pfad in einem Modell einen Zustand gibt, in dem φ gilt. Reachability Eigenschaften werden zum Beispiel bei Modellen von Kommunikationsarchitekturen verwendet, um nachzuweisen, dass schließlich einmal eine Nachricht korrekt übertragen wird. **Safety** Eigenschaften besagen, dass ein unsicherer Zustand niemals erreicht wird oder positiv formuliert, dass die Sicherheit immer gewährleistet ist. Zur Klasse der Safety Aussagen gehören $A\Box\varphi$ und $E\Box\varphi$. Mit



(a) $M \models A\Box\varphi$ (b) $M \models E\Box\varphi$ (c) $M \models \psi \rightsquigarrow \varphi$

(d) $M \models A\Diamond\varphi$ (e) $M \models E\Diamond\varphi$

Abbildung 3.3: TCTL Aussagen über den Computation Tree eines Modells M

Liveness Eigenschaften wird die Lebendigkeit von Systemen überprüft. Zum Beispiel muss nach einem Knopfdruck die Fußgängerampel schließlich einmal “grün” zeigen. Zur Liveness gehören die TCTL Formeln $A\Diamond\varphi$ und $\psi \rightsquigarrow \varphi$.

Mit dem Nachweis von Safety und Liveness können wesentliche Anforderungen an Safety-Protokolle und sicherheitsrelevante Kommunikationsarchitekturen bewiesen werden. Es ist allerdings nicht möglich, probabilistische Eigenschaften zu ermitteln. Zu den Zuverlässigkeitseigenschaften von Kommunikationsprotokollen gehört zum Beispiel die Wahrscheinlichkeit, dass infolge von Übertragungsfehlerwahrscheinlichkeiten die Nachrichtenwiederholung keine Echtzeit-Anforderungen verletzt. Der Nachweis von probabilistischen Eigenschaften gelingt mit PMC. In [Old07] ist ein Vergleich zwischen verschiedenen probabilistischen Model Checking Tools zu finden. Eines dieser Tools, PRISM, ist im nächsten Kapitel vorgestellt.

3.8 Der probabilistische Model Checker PRISM

PRISM ist ein probabilistischer Model Checker und ermöglicht die formale Verifikation von probabilistischen Modellen [Pri]. Das PRISM Tool besteht aus einer textuellen Modellierungsumgebung und einem symbolischen Model Checker. Mit PRISM können drei verschiedene probabilistische Modelle erstellt werden: Continuous-Time Markov Chains (CTMC), Discrete-Time Markov Chains (DTMC) und Markov Decision Processes (MDP) [Par02]. Die CTMC Modelle werden für Systeme mit Übergangsraten zwischen zwei Zuständen verwendet [Men04]. Die Verweildauer in einem Zustand ist negativ exponentialverteilt. Mit CTMC Modellen werden zeitkontinuierliche Markov Prozesse, wie zu Beispiel Warteschlangen Prozesse, nachgebildet. DTMC Modelle sind ausschließlich probabilistisch und erlauben keinen Nichtdeterminismus [KNP02]. Zum Modellieren von nichtdeterministischen und probabilistischen Prozessen werden daher in dieser Arbeit ausschließlich MDP verwendet. Nichtdeterminismus entsteht durch die parallel ausgeführten Prozesse von Kommunikationsarchitekturen. Probabilistisches Verhalten wird in Übertragungskanälen modelliert. Jedes Datenpaket wird mit einer festgelegten Wahrscheinlichkeit korrekt oder fehlerhaft übermittelt.

Die Prozesse von MDP Modellen sind über blockierende Broadcast-Kanäle synchronisiert. Lokale oder globale Integer- und Boolesche Variablen können verwendet werden, wobei immer ein Wertebereich festzulegen ist. Die Modellierung von Uhren ist nur durch ein explizites Konzept möglich: Integer-Variablen, die in allen Prozessen synchron inkrementiert werden (über einen Broadcast Kanal) führen ein digitales Uhrenkonzept ein [RKNP04, KNPS06]. Es gibt eine Reihe von Fallstudien mit dem Model Checker PRISM, die quantitative Analysen aus verschiedenen Anwendungsdomänen beschreiben, zum Beispiel in [KNS02, KNS03]. Unter anderem existieren auch mehrere Fallstudien, in denen probabilistische Eigenschaften von Kommunikationsarchitekturen analysiert werden [Pri]. Die Besonderheit des probabilistischen Model Checkers ist, dass Transitionen mit einem Wahrscheinlichkeitswert annotiert werden. Jeder Übergang von einem Zustand zu einem Folgezustand ist damit mit einer diskreten Wahrscheinlichkeit behaftet. Die Temporallogik zum Ermitteln von Eigenschaften ist Probabilistic Computation Tree Logic (PCTL) und eine Erweiterung von CTL um probabilistische Aspekte. Anstelle eines Pfadquantors wird der probabilistische Quantor $\mathcal{P}$ verwendet. Dieser kann zusammen mit der Angabe einer Wahrscheinlichkeit $\lambda \in [0,1]$ eine Aussage φ über das Systemmodell verifizieren beziehungsweise falsifizieren ($\mathcal{P}_{\bowtie\lambda}\varphi$ mit $\bowtie \in \{<, \leq, \geq, >\}$) oder die Wahrscheinlichkeit ermitteln ($\mathcal{P}_{?}\varphi$), mit der eine Aussage φ gilt. Letzterer Fall liefert als Model Checking Resultat eine Wahrscheinlichkeit $p \in [0,1]$. Damit ist PMC beziehungsweise das Tool

PRISM der Model Checker, der die formale Analyse von Kommunikationsarchitekturen um den Aspekt der logischen Zuverlässigkeit erweitert.

3.8.1 PRISM Modelle

Die MDP Modelle bestehen aus Prozessen[4], die nichtdeterministisch ausgeführt werden. Prozesse können über Kanäle miteinander synchronisiert werden. Im Gegensatz zu UPPAAL benutzt PRISM ausschließlich blockierende Broadcast-Kanäle zur Synchronisation. Dabei führen alle synchronisierten Prozesse eine Transition simultan aus. Aus diesem Grund dürfen an synchronisierten Anweisungen keine Veränderungen von globalen Variablen durchgeführt werden. PRISM Modelle werden in einer textuellen Sprache erstellt. Die Modelle sind weniger übersichtlich als bei einer grafischen Modellierung, so dass PRISM häufig als Backend mit einer aufgesetzten grafischen Modellierungssprache benutzt wird [Gre07]. In dieser Arbeit übernimmt einen Metamodell die Aufgabe des grafischen Frontends. Das Metamodell definiert die Modellierung von Systemen als Zustandsdiagramme, die mit einem Model-to-Text Generator (Codegenerator) in die textuelle Form der PRISM Modelle übersetzt werden. Ein Systemmodell besteht aus Prozessen, die wiederum mit Anweisungszeilen das Prozessverhalten beschreiben. Eine Anweisungszeile besitzt folgenden syntaktischen Aufbau[5]:

$$[ch]\ g\ \ \rightarrow \lambda^1 : u^1 + \cdots + \lambda^k : u^k; \tag{3.1}$$

Das Label ch definiert einen Synchronisationskanal zu anderen Prozessen (optional), g ist eine Boolesche Bedingung über Variablen und $\lambda_1, \ldots, \lambda_k$ sind diskrete Wahrscheinlichkeiten, mit denen eine Variablenzuweisung (Update) $u_1, \ldots, u_k$ ausgeführt wird, mit $1 = \sum_{i=1}^{k} \lambda_i$ und $\lambda_i \in (0, 1]$. Ein Update von Variablen geschieht über temporäre Schattenvariablen: Ein neuer Wert wird den Variablen erst nach dem Ausführen aller Anweisungen zugewiesen. An der Anweisungszeile ist zu sehen, dass in PRISM keine expliziten Locations wie in klassischen Zustandsdiagrammen existieren. Zustandsdiagramme werden daher mit einer zusätzlichen Integer Variable L erstellt, wobei jeder Wert $l \in \mathbb{N}^0$ eine Location repräsentiert. Der Codegenerator des Frontends transformiert Zustandsdiagramme, indem jede Guard-Bedingung g der Anweisungszeilen um einen Location-Guard und jede Update-Anweisung $u^1 \ldots u^k$ um ein Update von L erweitert wird. Damit kann eine Anweisungszeile als probabilistische Transitionen zwischen Locations gesehen werden. Im Folgenden werden PRISM Modelle als Zustandsdiagramme beschrieben, die formal diesen

[4] In PRISM werden die als Prozesse bekannten Teile eines Systemmodells Module genannt.
[5] Das Zeichen + ist die syntaktische Trennung zwischen probabilistischen Anweisungen.

Aufbau besitzen:

$$l \xrightarrow{ch/g} \{(\lambda^1, l'^1, u^1), \ldots, (\lambda^k, l'^k, u^k)\} \tag{3.2}$$

Eine generelle Transformation dieser allgemeinen Transition ist hier als *cmd* skizziert:

$$[ch]\ g \wedge L = l\ \ \rightarrow \lambda^1 : u^1, L := l'^1 + \cdots + \lambda^k : u^k, L := l'^k; \tag{3.3}$$

Die erstellten Zustandsdiagramme im Frontend (Metamodell) werden zu entsprechenden Anweisungszeilen als PRISM Modelle transformiert. Weiterhin wird das Broadcast-Konzept im Metamodell eingeschränkt. Es dürfen jeweils nur zwei Prozesse über dasselbe Kanal-Label miteinander synchronisiert werden[6]. Anstelle des Broadcast-Konzepts sind dadurch nur binäre Synchronisationen möglich. Zudem ist ein digitales Uhrenkonzept im Metamodell definiert: Mit den digitalen Uhren können Locations *urgent* oder *timed* sein. In einer timed-Location darf Zeit in diskreten Schritten vergehen, während eine urgent-Location verlassen werden muss, ohne dass sich ein Uhrenwert verändert. Die Zeit vergeht in allen Prozessen gleich schnell. Mit anderen Worten, Clock-Variablen müssen in allen Modulen gleichzeitig inkrementiert werden. Dieses übernehmen die über den einzig erlaubten Broadcast-Kanal "tick" synchronisierten Transitionen, die alle Uhren simultan um eins inkrementiert.

3.8.2 Formale Syntax und Semantik des PRISM Model Checkers

Die formale Beschreibung der PRISM Syntax und Semantik ist an die des UPPAAL Model Checkers angelehnt, um Gemeinsamkeiten hervorzuheben. Die Menge V beinhaltet Bounded-Integer und Boolean getypten Variablen. Die Evaluation e bildet jede Variable auf einen Wert ihrer Domäne ab $e : V \rightarrow \mathcal{D}$.

Definition 3.8.1 (**PRISM-Modell**)**.** Ein PRISM-Modell ist ein Tupel $(\bar{A}, V, Ch)$, mit:

(i) $\bar{A}$ ist ein Vektor mit n PRISM Modulen $A_1, \ldots A_n$,

(ii) V ist eine endliche Menge von Booleschen- oder Integer-Variablen und

(iii) Ch ist eine Menge von Synchronisations-Labels.

Definition 3.8.2 (**PRISM Prozess**)**.** Ein PRISM Prozess ist ein Tupel (L, l_0, C, E), mit:

[6] Mit der Ausnahme des speziellen Kanal-Labels *tick*, welches für die Einführung von digitalen Uhren benötigt wird.

(i) L ist eine Location Variable,

(ii) $l_0 \in \mathbb{N}^0$ ist die initiale Location,

(iii) C ist die Menge von lokalen Uhr-Variablen und

(iv) E ist eine endliche Menge von probabilistischen Transitionen $l \xrightarrow{g/ch} \Lambda$, mit der probabilistischen Verteilung $\Lambda = \{(\lambda^1, l'^1, u^1), \ldots, (\lambda^k, l'^k, u^k)\}$ über Updates u^d und Ziel-Locations l'^d mit $d = 1, \ldots, k$, die jeweils mit der diskreten Wahrscheinlichkeit $\lambda^d \in (0, 1]$, mit $1 = \sum_{d=1}^{k} \lambda^d$ definiert sind. Die Update Funktion weist Variablen neue Werte zu.

Definition 3.8.3 (**PRISM Konfiguration**)**.** Das Tripel $s = \langle \bar{l}, e, v \rangle$ ist eine *Konfiguration* von einem PRISM Model $(\bar{A}, V, Ch)$, mit

- $\bar{l} = (l_1, \ldots, l_n)$ ist der Location-Vector, wobei l_i die aktuelle Location von Prozess A_i ist,
- $e : V \to \mathcal{D}$ bildet jede Variable auf einen Wert ihres Wertebereichs ab,
- $v : C \to \mathbb{N}^0$ bildet jede Uhr auf einen nicht-negativen ganzzahligen Wert ab. Die Notation $(v + 1)$ bedeutet, dass alle Uhren um 1 inkrementiert werden.

Der Einfachheit halber ist die initiale Konfiguration eines PRISM-Modells definiert als ein Tupel $(\langle (l_1^0, \ldots, l_n^0), [V \mapsto 0], [C \mapsto 0] \rangle$. Hierbei befinden sich alle PRISM Prozesse $A_1, \ldots A_n$ in ihrer initialen Location, alle Variablen werten zu 0 aus und alle Uhren sind auf 0 zurückgesetzt. Beginnend von der initialen Konfiguration (Zustand) entwickeln sich weitere Konfigurationen eines PRISM-Modells nach drei Transitionsregeln: *Delay Transition*, *Action Transition* und *Synchronisation Transition*.

Definition 3.8.4 (**Delay Transition**)**.** Für die Konfiguration $s = \langle \bar{l}, e, v \rangle$ wird eine *Delay Transition* $s \overset{\delta}{\Longrightarrow} \langle \bar{l}, e, (v + 1) \rangle$ ausgeführt, falls $\forall l_i$ in $\bar{l}$ es eine Transition $l_i \xrightarrow{g_i/ch_i} \{1, l_i, u_i\} \in E_i$ mit $i = 1, \ldots, n$ gibt, für die gilt:

(i) $s \models g_1 \wedge \cdots \wedge g_n$ und

(ii) $ch_1 = \cdots = ch_n = tick$

(iii) u_i ist eine Update-Anweisung von der Form $c_j := c_j + 1$, $\forall c_j \in C_i$

Definition 3.8.5 (**Action Transition**)**.** Für die Konfiguration $s = \langle \bar{l}, e, v \rangle$ wird eine *Action Transition* $s \overset{a}{\Longrightarrow} \{s'^1, \ldots, s'^k\}$ ausgeführt, falls $l_i \xrightarrow{g/a} \Lambda_i \in E_i$, l_i in $\bar{l}$ und wenn folgende Bedingungen erfüllt sind:

(i) $s \models g$ und

(ii) $ch = \emptyset$

Die Wahrscheinlichkeit von einem Zustand s auf einen Folgezustand s'^d beträgt λ^d für $d = 1, \ldots, k$. Jedes $s'^d = \langle \bar{l}[l_i'^d / l_i], a^d(e), a^d(v) \rangle$.

Definition 3.8.6 (Synchronisation Transition). Für die Konfiguration $s = \langle \bar{l}, e, v \rangle$ wird eine *Synchronisation Transition* $s \xRightarrow{sync} \{s'^1, \ldots, s'^{k_i \cdot k_j}\}$ ausgeführt, falls $l_i \xrightarrow{g_i/a_i} \Lambda_i \in E_i$, l_i in $\bar{l}$ und $l_j \xrightarrow{g_j/a_j} \Lambda_j \in E_j$, l_j in $\bar{l}$, mit $i \neq j$ und wenn folgende Bedingungen erfüllt sind.

(i) $s \models g_i \wedge g_j$ und

(ii) $ch_i = ch_j$

Es entsteht eine Menge von Folgekonfigurationen, wobei die Wahrscheinlichkeit von einem Zustand s auf einen Folgezustand s'^{d_i,d_j} gleich $\lambda_i^{d_i} \cdot \lambda_j^{d_j}$ ist.

$$\{s'^{d_i,d_j} = \langle \bar{l}'[l_i'^{d_i}/l_i, l_j'^{d_j}/l_j], (a_i{}^{d_i}(a_j{}^{d_j}(e))), (a_i{}^{d_i}(a_j{}^{d_j}(v))) \rangle \mid (\lambda_i^{d_i}, l_i'^{d_i}, a_i{}^{d_i}) \in \Lambda_i, (\lambda_j^{d_j}, l_j'^{d_j}, a_j{}^{d_j}) \in \Lambda_j\}$$

Im Anhang A auf Seite 125 ist ein gemeinsames Metamodell für Uppaal und PRISM beschrieben. Dieses definiert alle nötigen Restriktionen für die Generierung konsistenter Modelle. Die Ausführungspfade von Uppaal und PRISM Modellen besitzen die gleiche Folge von Locations und Variablen-Evaluationen. Auf Basis dieses gemeinsamen Metamodells ist eine qualitative und quantitative Verifikation möglich. Der Uppaal Model Checker übernimmt die Verifikation von Safety Eigenschaften, während der PRISM Model Checker probabilistische Aussagen zu Systemzuständen ermittelt. Der Unterschied zwischen quantitativer und qualitativer Verifikation und der daraus resultierende Aufwand, Model Checking Tools zu kombinieren, ist in Abbildung 3.4 auf der nächsten Seite angedeutet und wird im Folgenden beschrieben.

3.9 Quantitative und Qualitative Verifikation

In der Literatur sind Fallstudien aufgeführt, die sowohl qualitative, als auch quantitative Eigenschaften von Systemmodellen analysieren. Qualitative Eigenschaften werden mit 'ja' oder 'nein' beantwortet, während quantitative Eigenschaften mit konkreten Werten beziffert werden (siehe Abbildung 3.4 auf der nächsten Seite). Ein qualitatives Beispiel ist in [SB07] beschrieben, in dem der Nachweis von Safety-Eigenschaften anhand eines PROFIsafe Modells mit dem Timed Automata Tool Uppaal erbracht wird. Zu den quantitativen

Eigenschaften gehört zum Beispiel die Wahrscheinlichkeit, einen bestimmten Zustand zu erreichen. In [KNS02, KNS03] sind quantitative Analysen von Kommunikationsprotokollen mit dem probabilistischen Model Checker PRISM beschrieben. Neben den unterschiedlichen Systemmodellen von UPPAAL (Timed Automata) und PRISM (PMC) ist auch die verwendete Temporallogik der Tools verschieden. Das UPPAAL Tool verwendet TCTL (siehe Kapitel 3.7.3 auf Seite 51) und PRISM benutzt PCTL. Zwischen TCTL und PCTL gibt es äquivalente Aussagen [BK08].

$$\mathcal{P}_{>0}\left[\Diamond\varphi\right] \equiv E\Diamond\varphi \tag{3.4}$$

$$\mathcal{P}_{=1}\left[\Box\varphi\right] \equiv A\Box\varphi \tag{3.5}$$

$$\mathcal{P}_{=1}\left[\Diamond\varphi\right] \not\equiv A\Diamond\varphi \tag{3.6}$$

$$\mathcal{P}_{>0}\left[\Box\varphi\right] \not\equiv E\Box\varphi \tag{3.7}$$

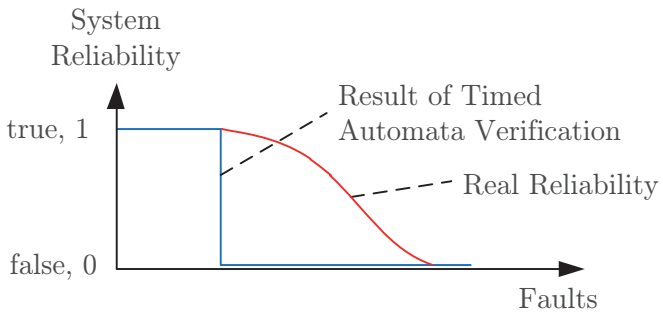


Abbildung 3.4: Qualitative versus quantitative Eigenschaften

3.9.1 Der Unterschied zwischen TCTL und PCTL

Der Unterschied zwischen TCTL und PCTL wird an einem Beispiel deutlich, dass in [Din07] anhand eines Kochs beim Zubereiten von Mettbällchen anschaulich dargestellt ist:

Einem schwedischen Koch gelingen Mettbällchen in 90% der Versuche. Wenn sie ihm gelingen, dann hört er auf und macht keine Weiteren. Wenn sie ihm allerdings nicht gelingen, dann versucht er es erneut. Mit dem Timed Automata und dem probabilistischen Model Checker wird die Aussage geprüft, ob dem Koch schließlich immer die Mettbällchen gelingen. In Abbildung 3.5 auf der nächsten Seite ist jeweils ein nichtprobabilistischer und

ein probabilistischer Automat des Koch-Algorithmus dargestellt. Das nichtprobabilistische Modell M_{np} besteht aus den Zuständen *start* (Initialzustand), *good* und *bad* mit den jeweiligen Transitionen zwischen den Zuständen. Von dem Zustand *start* aus gibt es eine nichtdeterministische Entscheidung, zum Zustand *bad* oder *good* zu wechseln. Das probabilistische Modell M_p modelliert an den Transitionen zusätzlich Übergangswahrscheinlichkeiten. Diese Aussage, ob der Koch schließlich immer gelungene Mettbällchen herstellt, wird mit einem nichtprobabilistischen und einem probabilistischen Model Checker verifiziert.

1. In dem nichtprobabilistischen Modell lautet die TCTL Aussage 'Der Zustand *good* wird schließlich immer erreicht' mit der Formel $\Phi = A\Diamond good$ ausgedrückt. Die Verifikation ergibt $M_{np} \not\models \Phi$, weil es den unendlichen Pfad $\pi = \langle start, bad, start, bad, \dots \rangle$ gibt, der niemals *good* erreicht.

2. In dem probabilistischen Modell lautet die PCTL Formel $\Psi = \mathcal{P}_{\geq 1}(\Diamond good)$. Die Verifikation ergibt $M_p \models \Psi$, da von dem Pfad $\pi = \langle start, bad, start, bad, \dots \rangle$ die Wahrscheinlichkeit gegen 0 konvergiert ($p(\pi) = 0.1 \cdot 1.0 \cdot 0.1 \cdot 1.0 \dots$) und damit die Wahrscheinlichkeit den Zustand *good* zu erreichen gegen eins konvergiert.

Dieses Beispiel zeigt den Unterschied zwischen TCTL und PCTL beziehungsweise zwischen probabilistischen und nichtprobabilistischen Modellen. Die TCTL Formel $A\Diamond\varphi$ gehört zur Klasse der Liveness Properties und ist eine wichtige Eigenschaft von reaktiven Systemen. Ein unbeschränkter Zeitraum bei der Analyse von Liveness Eigenschaften mit einem probabilistischen Model Checker ist nicht aussagekräftig, da wie bei den Mettbällchen probabilistische Werte gegen null beziehungsweise gegen eins konvergieren können. Bei der Analyse von Kommunikationsarchitekturen ist daher die Definition einer Grenze (zum Beispiel Liveness innerhalb von t Zeiteinheiten) sinnvoll.

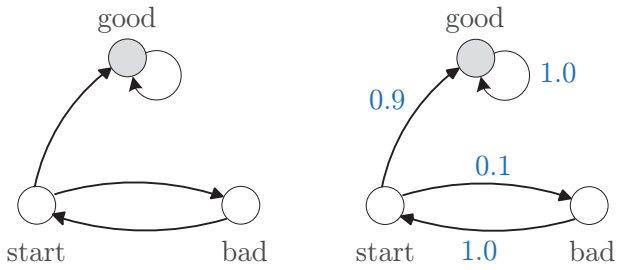


Abbildung 3.5: Der schwedische Koch und seine Mettbällchen

Im Anhang A auf Seite 125 ist ein gemeinsames Metamodell für UPPAAL und PRISM Modelle für die kombinierte Verifikation und Analyse qualitativer und quantitativer Eigenschaften beschrieben. Die Kombination von Timed Automata und PMC ist in der Lage

Safety, Reachability, Liveness und Reliability Eigenschaften von Kommunikationsarchitekturmodellen zu analysieren. Allerdings haben die zur Konsistenz nötigen Restriktionen aus Sicht der Usability Nachteile. Des Weiteren entsteht bei dem probabilistischen Model Checker bei praxisrelevanten Modellgrößen eine frühe Zustandsraumexplosion. Dieses Problem ist bereits in [Wan06] beschrieben. Diese und weitere Nachteile führten zu einem neuen Ansatz, qualitative Verifikation mit der Zuverlässigkeitsanalyse zu kombinieren. Dieser Ansatz ist im nächsten Kapitel beschrieben.

Kapitel 4

Analyse sicherheitsrelevanter Kommunikationsarchitekturen

Murphy was an Optimist: The well known Murphy's Law states that: "If anything can go wrong, it will go wrong." When requirements limit failure probabilities to one-i-a-million or less, this should be re-written as: "If anything can't go wrong, it will go wrong anyway."

Kevin R. Driscoll at SAFECOMP 2010

Aufgrund der wesentlichen Nachteile im Bereich Zustandsraumexplosion und Usability, die sich bei der Kombination von Uppaal und PRISM herausstellten, wird in diesem Kapitel ein neuer Ansatz vorgestellt. Das Ziel ist die qualitative Verifikation von Safety und die quantitative Analyse der Zuverlässigkeit von praxisbezogenen Kommunikationsarchitekturen. Hierbei bildet die domänenspezifische Modellierungssprache CAMoLa den Ausgangspunkt für Kommunikationsarchitekturmodelle, von denen beide Aspekte mit einer integrierten Tool-Kette analysiert werden. Der Begriff CAMoLa steht für ein Framework, welches die generische Modellierung von Kommunikationsarchitekturen ermöglicht und in zwei weiteren Schritten die Verifikation qualitativer- und die Ermittlung von Zuverlässigkeits-Eigenschaften mit Model-Checking-Methoden durchführt. Die frühzeitige Begleitung des Designs mit einem Modellierungs- und Verifikations-Framework nach dem CAMoLa Vorbild kann maßgeblich zur Qualitätssteigerung von sicherheitsrelevanten Kommunikationsarchitekturen eingesetzt werden. Hierzu sind drei Phasen definiert, in denen das Design erstellt und analysiert wird.

Das bereits vorgestellte Tool UPPAAL übernimmt in dem CAMoLa Framework das Model Checking, wobei CAMoLa Modelle durch Codegeneratoren automatisch in UPPAAL Modelle transformiert werden. Die Modellierungssprache CAMoLa beinhaltet ein Konzept zur Manipulation von Modellen, so dass die Analyse quantitativer Eigenschaften (Reliability) mit dem nicht-probabilistischen Model Checker UPPAAL möglich wird.

In dem ersten Teil dieses Kapitels ist das CAMoLa Framework in den Entwicklungsprozess von Safety-Protokollen und Architekturen eingeordnet. In dem zweiten Teil dieses Kapitels ist die DSL CAMoLa mit ihren einzelnen Bestandteilen definiert. Des Weiteren sind Transformationsregeln aufgestellt, welche die Übersetzung von CAMoLa Modellen zu UPPAAL Modellen übernehmen. Anschließend werden die drei Phasen des Frameworks vorgestellt: Es beginnt mit dem Erstellen von CAMoLa Modellen in der Phase 1 und ihrer qualitativen Verifikation in der Phase 2. Mit der Analyse von Zuverlässigkeit beschäftigt sich die Phase 3, in der das Konzept der Reliability-Analysen sowie ein dafür notwendiger Mutationsgenerator erläutert werden. Dieser übernimmt das Erstellen von Zuverlässigkeitsformeln auf Basis von festgelegten Entscheidungen, Nachrichten mit einer bestimmten Fehlerart zu übertragen. Die daraus entstehenden Veränderungen des Basismodells werden als Modellmutationen bezeichnet. Ein Nebeneffekt dieser Mutationen ist die Beherrschung größerer Modellkomplexität und die Möglichkeit, die Analyse durch verteiltes Model Checking in einem großen Rechner-Pool durchzuführen.

4.1 Generische Modellierung von Kommunikationsarchitekturen

Zu den Bestandteilen von Kommunikationsarchitekturen gehören Applikationen, Übertragungsprotokolle und Übertragungsnetze. Schwerpunkt der Verifikation ist das Nachweisen von Safety-Eigenschaften und die Ermittlung der Zuverlässigkeit einer Architektur. Die einzelnen Bestandteile werden dabei zu einem komplexen Systemmodell zusammengeführt. In der Praxis kann es bei Kommunikationsarchitekturen im Laufe der Systemevolution vorkommen, dass sich einzelne Bestandteile der Architektur verändern. Die generische Modellierung von Kommunikationsarchitekturen erleichtert das Erstellen von neuen Modellen durch das Anpassen von Parametern oder dem Austausch einzelner Teilmodelle. Der Modellierungsaufwand wird dadurch minimiert und verbessert die Akzeptanz des Analyse-Frameworks bei Systemdesignern. Hochzuverlässige und sicherheitsrelevante Kommunikationsarchitekturen im Bahnbereich beinhalten Fluss- und Fehlerkontroll-Mechanismen (siehe Kapitel 2.4 auf Seite 24), die eine Überflutung von Rechnerressour-

cen verhindern und vor einem Safety-Check Übertragungsfehler korrigieren. Die Entwicklung zukünftiger sicherheitsrelevanter Kommunikationsarchitekturen ermöglicht mit dem CAMoLa-Framework zu einer frühen Designphase sowohl Safety (qualitative Eigenschaft im Sinne des SIL), als auch Reliability (Zuverlässigkeit des logischen Kanals in Abhängigkeit von zufälligen Kommunikationsfehlern) zu ermitteln. Führt die Spezifikation der Architektur oder einer ihrer Subkomponenten zu unbefriedigenden Ergebnissen, ist das Design entsprechend zu ändern. In Kapitel 3 auf Seite 37 sind Methoden zur Modellierung und formalen Verifikation aufgelistet, die Teil dieses Analyse-Frameworks sind.

Die Modellierung und Analyse von Kommunikationsarchitekturen zu einem frühen Zeitpunkt des Designs (siehe Abbildung 4.2 auf der nächsten Seite) verbessert die Qualität der Architektur und verhindert einen kostspieligen Neuentwurf aufgrund unerwarteter Designschwächen. CAMoLa unterstützt das strukturierte Modellieren von Kommunikationsarchitekturen nach der Referenzarchitektur aus der EN 50159 (siehe Abbildung 2.4 auf Seite 17). Ein Hilfsmittel bei der Modell-Verifikation sind Systembeobachter oder Watchdog-Prozesse, die bestimmte Eigenschaften bei der Verifikation überwachen und die Formalisierung von Anforderungen erleichtern. In Modellen von Kommunikationsarchitekturen sind Echtzeit, Sequenzrichtigkeit und Liveness Beispiele für diese Anforderungen. Ein Systembeobachter ist so aufgebaut, dass dieser beim Verletzen einer dieser Anforderungen eine Fehler-Location einnimmt. Die formalisierte Aussage muss dann lediglich die Abwesenheit der Fehler-Location in dem Computation-Tree nachweisen, um Safety-Bedingungen zu verifizieren. Ein solcher Systembeobachter (Abbildung 4.1 und 4.3 auf Seite 65) ist daher oft Bestandteil von Systemmodellen.

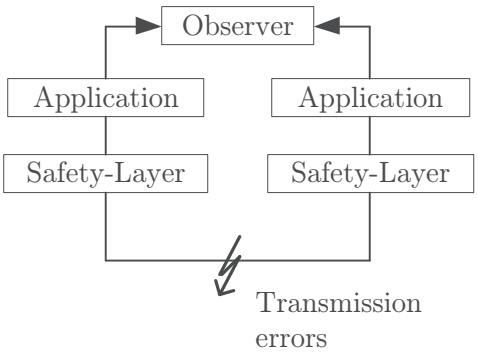


Abbildung 4.1: Modellbasierte Verifikation von Safety-Protokollen

Die Verifikation von Safety und die Analyse von Reliability einer Kommunikationsarchitektur bezieht sich auf Transaktionen, die von einem Applikationsprozess ausgelöst

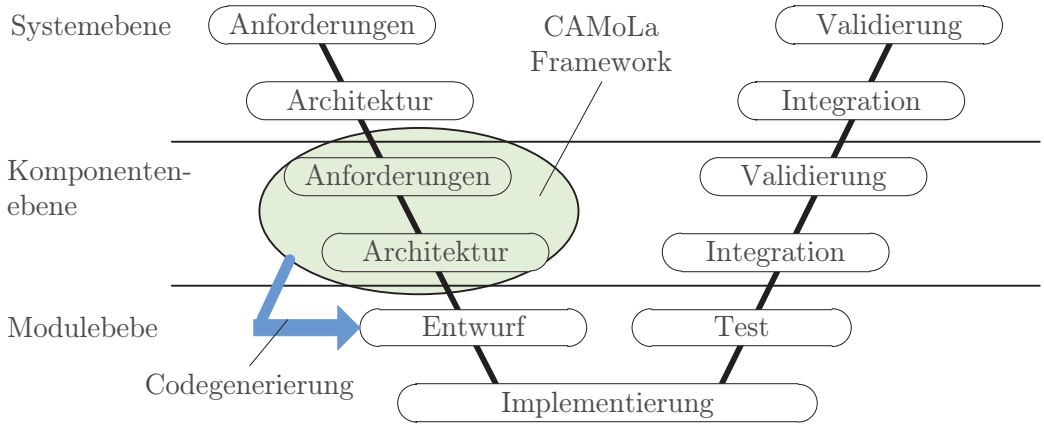


Abbildung 4.2: Einordnung der Modellierung in des V-Modell

werden. Eine Transaktion ist ein Dienst, den ein (Safety-) Protokoll einer höher liegenden Schicht zur Verfügung stellt. Mögliche Transaktionen sind zum Beispiel das Aufbauen eines logischen Kommunikationskanals oder das Senden von Daten. Eine Transaktion darf nicht zu einem unsicheren Zustand führen. Safety-Protokolle müssen daher Sicherheitsmechanismen beinhalten, die eine fehlerhafte Übertragung durch Sequenznummern, Zeitüberwachung und anderer Verfahren erkennen (siehe Tabelle 2.2 auf Seite 16). Wenn die Spezifikation eines Safety-Protokolls in diesem Sinne korrekt ist, kann es für weitere Architekturmodelle herangezogen werden, um Reliability zu analysieren. Die relevanten Systemzustände der Verifikation und Analyse sind der Abbildung 4.4 auf Seite 66 zu entnehmen. Die Zuverlässigkeit von Safety-Protokollen gegenüber Übertragungsfehlern kann mit Zuverlässigkeitsmechanismen erhöht werden. Diese können im Safety-Layer integriert sein, oder auf andere Komponenten ohne Sicherheitsanforderungen ausgelagert werden. Eine Auslagerung von Zuverlässigkeitsmechanismen ist vor allem sinnvoll, um von leistungsfähigen und erprobten COTS Techniken zu profitieren, ohne hohe Kosten mit proprietären Entwicklungen zu verursachen. Weiterhin finden durch Kundenwünsche oder anderen Bedingungen Veränderungen in der Kommunikationsarchitektur statt. Bisher musste mit zeitaufwendigen Verfahren die Kompatibilität einer neuen Architektur zu Vorgängerversionen nachgewiesen werden. Dieses Framework beinhaltet eine Bibliothek von Subsystemen, wodurch neue Architekturen aus vorhandenen Modellen zusammengestellt werden können. Die Analyse der Kompatibilität ist damit praktisch per “Knopfdruck” möglich.

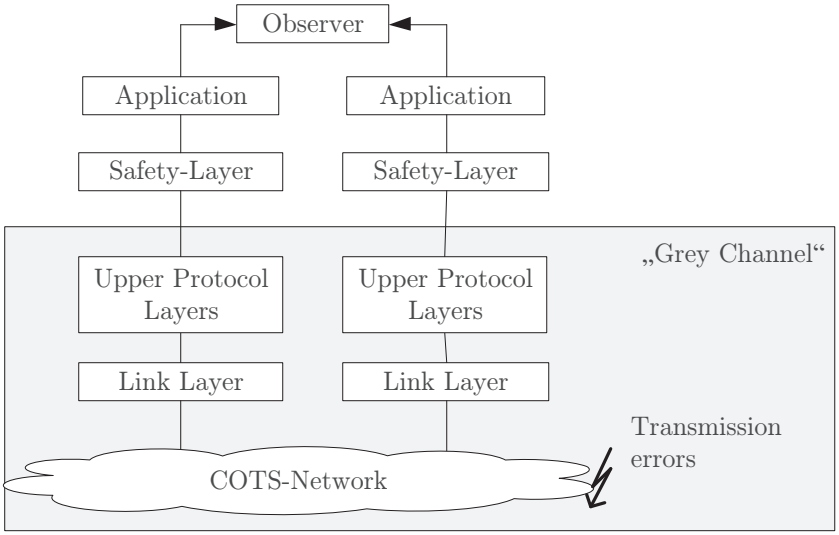


Abbildung 4.3: Verifikation und Analyse von Architekturmodellen

4.2 Analyse-Framework

Die Modellierungssprache CAMoLa ist Teil einer Tool-Kette, die das Design und die Analyse von Kommunikationsarchitekturen auf Basis formaler Modelle ermöglicht. Zu den Bestandteilen dieser Tool-Kette gehören Codegeneratoren, die CAMoLa Modelle in eine XML basierte Repräsentation von Uppaal Modellen übersetzen. Des Weiteren gehört ein *Mutationsgenerator* zum Framework, der Fehlerarten bei der Übertragung festlegt, wobei jede konkrete Fehlerkombination zu einer *Modellmutation* führt. Der Model Checker Uppaal, verifiziert formalisierte Eigenschaften von Modellen und von Modellmutationen. Hierbei werden Spezifikationsfehler aufgedeckt und in einem weiteren Schritt wird die Zuverlässigkeit gegenüber Fehlerarten und Fehlerhäufigkeiten festgestellt. Insgesamt ist die Verifikation und Analyse von Kommunikationsarchitekturen in drei Phasen gegliedert (Abbildung 4.5 auf Seite 67):

1. Phase 1: Modellieren einer Protokollspezifikation und Formalisieren von Anforderungen.

2. Phase 2: Formale Verifikation der Anforderungen und gegebenenfalls Korrektur des Modells beziehungsweise der Spezifikation.

3. Phase 3: Analyse von Zuverlässigkeitseigenschaften.

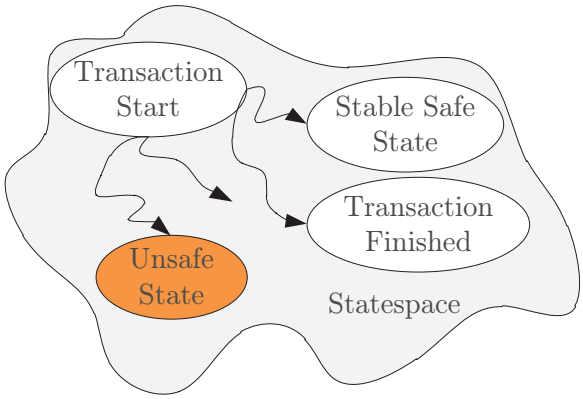


Abbildung 4.4: Mögliche Systemzustände nach dem Auslösen einer Transaktion

Das CAMoLa Framework ist als Prototyp konzipiert, um modellbasierte Entwicklung mit integrierter formaler Verifikation im Industriebereich weiter zu etablieren. Universitäre formale Methoden dringen aufgrund von speziell erforderlichen Kenntnissen der Systemdesigner nur langsam in den Industriebereich vor. Das hier präsentierte Analyse-Framework ist kein Ersatz für spätere Tests oder Validierung des Systems, es ermöglicht jedoch die Funktionalität von Kommunikationssystemen während der Spezifikationsphase zu verifizieren. Der Modellierer hat die Aufgabe, ein Modell des Systems zu erstellen und bestimmte Eigenschaften des Systems nachzuweisen. Wird eine erwartete Eigenschaft nicht erfüllt, werden automatisch Ausführungspfade (Diagnostic Trace) generiert, die zur Verletzung dieser Eigenschaft führen. Dieses liefert Hinweise, ob es sich um einen Modellierungsfehler handelt oder das System fehlerhaft spezifiziert ist. Typischerweise wird die Deadlock-Freiheit, Liveness, Erreichbarkeit bestimmter Zustände und die Einhaltung von Safety-Bedingungen geprüft. Die Formalisierung von Modelleigenschaften ist von dem verwendeten Model Checker abhängig, so dass mit dem CAMoLa-Framework und dem integrierten Uppaal Tool der Formalismus auf TCTL festgelegt ist (siehe Kapitel 3.6 auf Seite 44). Das formale Modell kann neben der Analyse auch als Ausgangspunkt für die weitere Systemrealisierung benutzt werden. Ein großer Vorteil der domänenspezifischen Modellierung sind die generischen Codegeneratoren, die Modelle in beliebige textuelle Sprachen transformieren können. Mit einem Codegenerator können Modelle von Protokollen direkt in zum Beispiel C-Code übersetzt werden. Es verringert sich der manuelle Implementierungsaufwand und Implementierungsfehler können reduziert werden, da die korrekte Funktionalität des Modells bereits mit dem Model Checker bewiesen wurde. In der bisher gängigen Praxis erfolgt die funktionale Überprüfung einer Spezifikation erst mit

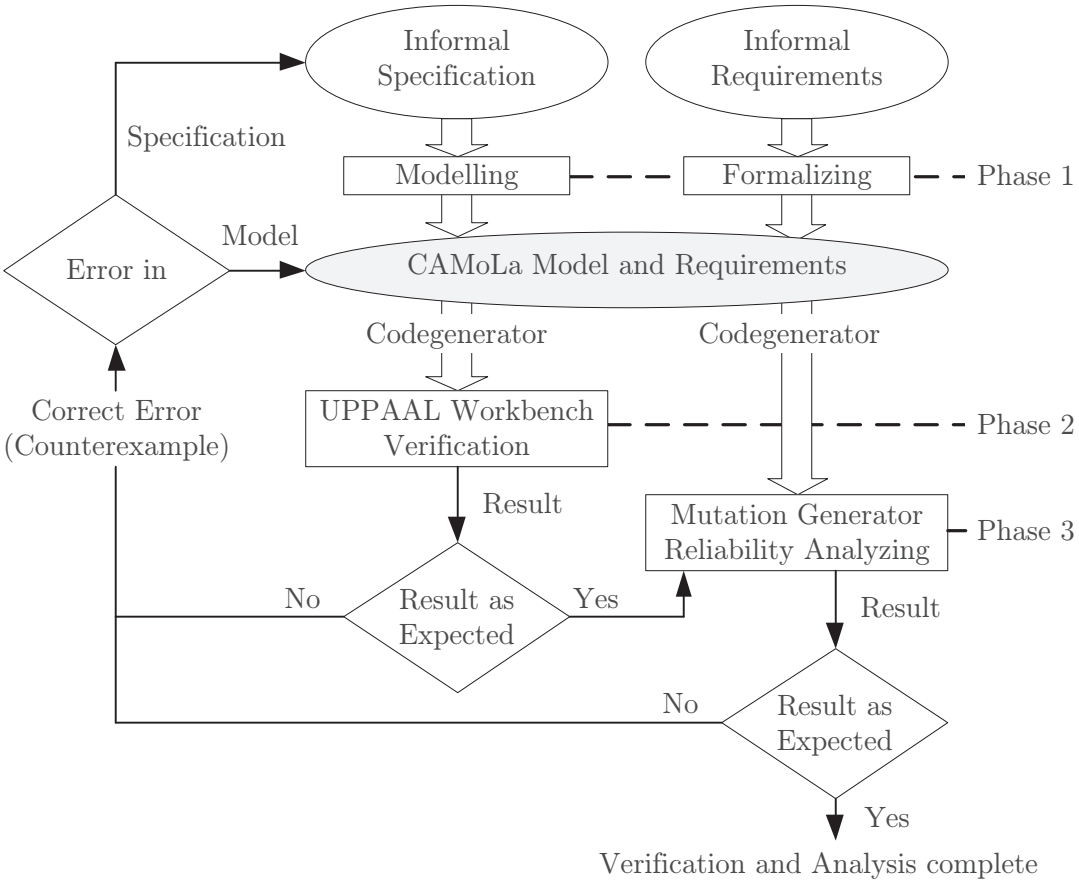


Abbildung 4.5: Modellierung, Verifikation und Analyse, Workflow

dem realen System und durch die Definition von Testfällen. Es hat sich gezeigt, dass dabei nicht immer alle Ausführungspfade zu den ungewollten Systemzuständen entdeckt wurden. Vor allem die informale Spezifikation von proprietären Kommunikationsprotokollen ist teilweise fehlerhaft oder unvollständig, so dass Freiheitsgrade bei der Implementierung entstanden. Mit der modellbasierten Entwicklung treten Unschärfen in der Spezifikation bereits bei der Modellierung auf. Zudem wird mit der angeschlossenen Verifikation das Beweisen von Systemeigenschaften automatisiert, so dass aufwendige Review-Prozesse reduziert werden können.

In den nächsten Abschnitten werden die Metasprache CAMoLa und die weiteren Bestandteile der Tool-Kette beschrieben.

4.3 Communication Architecture Modelling Language

Das Entwickeln einer DSL beginnt mit der Analyse der Domäne [Mew09]. Der Modellierungsrahmen ist in Kapitel 3.3 auf Seite 40 bereits beschrieben. Das konkrete Erstellen der DSL CAMoLa ist vor allem von der Modellierungssprache des Model Checking Tools UPPAAL geprägt. Das MetaEdit+® Tool (siehe Kapitel 3.4 auf Seite 41) ist der Ursprung der DSL CAMoLa und die Workbench von MetaEdit+® ist der grafische Modellierungs-Editor für Kommunikationsarchitekturen.

CAMoLa definiert das Modellieren von Kommunikationsarchitekturen in zwei Hierarchieebenen: *System* und *Prozess*. In der System-Ebene werden die zu einer Architektur gehörenden Prozesse und Schnittstellen (Synchronisation) zwischen den Prozessen modelliert. Mit der Synchronisation zwischen zwei Prozessen ist der Austausch von Variablenwerten möglich (*Value-Passing*). In der Prozess-Ebene wird das Verhalten der Prozesse als Zustandsdiagramm modelliert. Wiederkehrende Elemente von Kommunikationsarchitekturen (zum Beispiel Queues) sind als Objekt in CAMoLa zusammengefasst, um den Modellierungsaufwand zu reduzieren. Diese Objekte werden bei der Transformation als entsprechende Funktionen zu UPPAAL übersetzt. Die Abbildung 4.6 und die Tabelle 4.3 auf der nächsten Seite zeigen eine Beispielarchitektur.

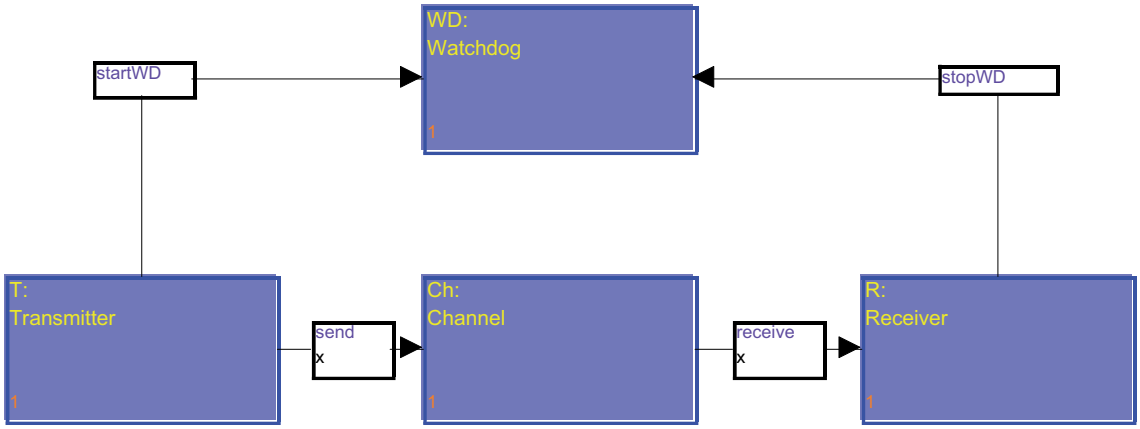
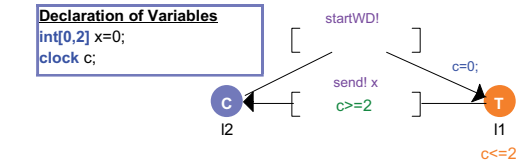
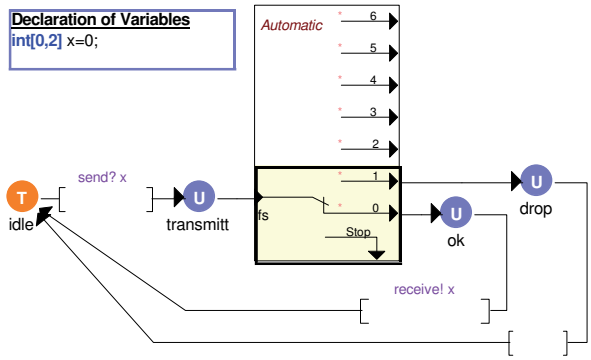
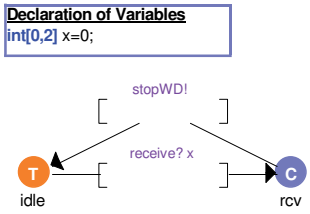
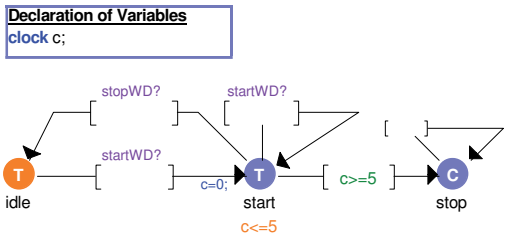


Abbildung 4.6: CAMoLa - Systemspezifikation

4.3.1 CAMoLa - System-Ebene

In der System-Ebene stehen Objekte zum Modellieren der Architektur-Bestandteile (Prozesse) sowie deren Interaktionen zur Verfügung. Des Weiteren werden globale Variablen, Konstanten, Datenstrukturen und Funktionen festgelegt, die zur Verifikation der Architektur benötigt werden. Datenstrukturen bestehen aus den Basistypen sowie Arrays und

Tabelle 4.1: Prozesse der Beispiel-Architektur

	**Transmitter.** Der Transmitter sendet in Zyklen von c=2 Zeiteinheiten die Nachricht x über den Kanal *send!* zum Prozess Channel. Unmittelbar nach dem Senden wird der Watchdog über *startWD!* gestartet.
	**Channel.** Der Channel Prozess stellt einen fehlerhaften Übertragungskanal dar, der eine Nachricht x entweder über die Location *ok* zum Receiver weiterleitet oder über die Location *drop* die Nachricht verwirft.
	**Receiver.** Der Receiver Prozess stoppt den Watchdog unmittelbar nach dem Empfangen einer Nachricht (Kanal *receive?*) über den Kanal *stopWD!*
	**Watchdog.** Der Watchdog Prozess startet in der initialen Location *idle*. Über den Kanal *startWD?* wird dieser aktiv und setzt die Clock c zurück. In der Location *start* wartet dieser, bis die Clock den Wert $c = 5$ erreicht hat, oder die Synchronisierung *stopWD?* ausgelöst wird.

Strukturen zum Bilden komplexerer Datentypen.

Typ-Definitionen Datenstrukturen und Funktionen

In Kommunikationsarchitekturen werden Strukturen von Datenpaketen der jeweiligen Protokollschichten mit verschiedenen Feldern (Sequenznummern, Flags, etc.) definiert. Zudem können Funktionen definiert werden, die komplexere Guard-Bedingungen und Update-Anweisungen zusammenfassen und an Transitionen zwischen Locations verwendet werden können. Dieses erlaubt eine übersichtliche und strukturierte Modellierung. Durch die Ähnlichkeit zur Programmiersprache C ist dieses für Systemdesigner zudem leicht verständlich. Häufig benötigte Funktionen, wie $min(x, y)$, $max(x, y)$ und $mod(x, y)$ werden vom Codegenerator automatisch erzeugt und können in allen Prozessspezifikationen verwendet werden.

Module und Batch-Module

Die Elemente *Module* und *BatchModule* beinhalten jeweils eine CAMoLa Prozessspezifikation als Dekomposition. Eine Prozessspezifikation kann dabei von mehreren Modulen instanziiert werden. Das Konzept stammt aus dem Template-Mechanismus von Uppaal und reduziert den Modellierungsaufwand von Kommunikationsarchitekturen: Ein Protokoll existiert in einer Architektur mindestens in zwei Instanzen (sender- und empfängerseitig). Mit dem Template-Konzept ist nur eine Prozessspezifikation des Protokolls erforderlich, die dann in einem Sender- und einem Empfänger-Prozess benutzt wird. Ein Modul kann spezifische Parameter der zugeordneten Prozessinstanz übergeben. Das Batch-Modul instanziiert $n + 1$ Prozessspezifikationen mit einer eindeutigen Index-Variable $0, \ldots, n$. Batch-Module dienen zum Modellieren von Systembeobachtern.

Synchronisierung und Datenfluss

Die Synchronisierung zwischen zwei Prozessen wird durch gerichtete Kommunikations-Beziehungen zwischen zwei Modulen oder Batch-Modulen festgelegt. Die Synchronisierung ist für einen Prozess blockierend, bis der Partner-Prozess für die Synchronisation bereit ist. In der Systemebene werden neben dem Synchronisierungs-Label auch die Variablen angezeigt, deren Werte zum Empfänger kopiert werden. Der Codegenerator erzeugt automatisch die Channel-Deklaration bei der Transformation zu Uppaal Modellen und legt Shared-Variables für das Value-Passing an.

4.3.2 CAMoLa - Prozessspezifikation

Die Prozessspezifikation ist die untere Modellierungsebene von CAMoLa. Eine Prozessspezifikation ist nur Bestandteil eines Architekturmodells, wenn diese in mindestens ei-

nem Modul (oder Batch-Modul) in der Systemebene instanziiert ist. Die Prozessspezifikation ähnelt der Modellierungssprache von UPPAAL. Zu den Elementen der CAMoLa Prozessspezifikation gehören *Locations*, *Transitions* und die Objekte *Queue* und *FaultSwitch*. Prozessspezifikationen werden auf der Grundlage von informalen Spezifikation von Subkomponenten einer Kommunikationsarchitektur erstellt. Zu den Subkomponenten gehört ein Safety-Layer, Übertragungsprotokolle, Übertragungskanäle und ein Systembeobachter (Abbildung 4.3 auf Seite 65). Die Prozessspezifikation wird mit einem Codegenerator automatisch zu einem UPPAAL-Template transformiert.

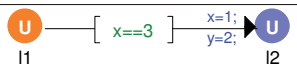
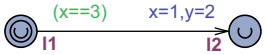
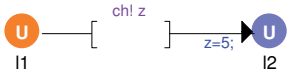
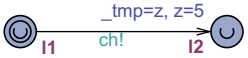
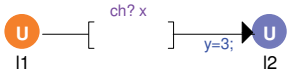
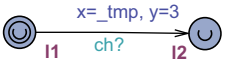
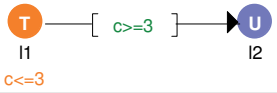
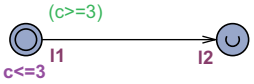
Locations.

Locations sind Bestandteil klassischer Zustandsdiagramme. Eine Location trägt in einem Prozess einen eindeutigen Namen und kann vom Typ *Timed*, *Urgent* oder *Committed* sein. Locations in CAMoLa werden direkt zu Locations von UPPAAL Prozessen übersetzt (Kapitel 3.7 auf Seite 45). Timed Locations können eine *Invariante* von der Form $x \lhd int$ oder $x - y \lhd int$ besitzen, wobei x, y Clock-Variablen, $\lhd \in \{<, \leq\}$ und $int \in \mathbb{Z}$ zu einem ganzzahligen Wert evaluiert. Genau eine Location eines Prozesses muss als *initial* gekennzeichnet sein.

Transition.

Mit Transitionen werden Übergänge zwischen Locations modelliert. Jede Transition kann mit einer *Guard-Bedingung*, einem *Synchronisation-Channel* und einer *Update-Anweisung* versehen werden. Die Guard-Bedingung ist ein Boolescher Ausdruck über Variablen und Clocks. Clocks dürfen nur als Konjunktionen verwendet werden und für Clocks x, y gilt $x \bowtie int$ oder $x - y \bowtie int$, mit $\bowtie \in \{<, \leq, ==, \geq, >\}$ und $int \in \mathbb{Z}$. Der Synchronisationskanal ist die Schnittstelle zu einem anderen Prozess. Mit der Synchronisation kann ein sendender Prozess Variablen-Werte einem empfangenden Prozess übergeben (Value-Passing). Der Synchronisation-Channel wird in der Form $ch \star x_1, \ldots, x_n$ angegeben, mit einem eindeutigen Namen ch, den lokalen Variablen $x_1, \ldots, x_n$ und $\star \in \{!, ?\}$. Ein '!' kennzeichnet den sendenden Prozess und ein '?' den empfangenen Prozess. Beim Value-Passing müssen beide Transitionen ($ch!$ und $ch?$) die gleiche Anzahl an Variablen besitzen. Diese müssen jeweils paarweise aus der gleichen Domäne (Wertebereich) sein. Die Update-Anweisung weist Variablen und Clocks mit der Ausführung der Transition nacheinander neue Werte zu. Gemäß der UPPAAL Semantik werden Transitionen atomar und zeitlos ausgeführt. Die Tabelle 4.2 auf der nächsten Seite zeigt Beispiele von Transitionen und Locations sowie deren Transformation zu einem UPPAAL Modell.

Tabelle 4.2: Transformation von Locations und Transitionen

Nr.	CAMoLa	UPPAAL	Formal
1.			$l_1 \xrightarrow{x=3/x:=1,y:=2} l_2$
2. A_i			A_i: $l_1 \xrightarrow{ch!/tmp:=z,z:=5} l_2$
2. A_j			A_j: $l_1 \xrightarrow{ch?/x:=tmp,y:=3} l_2$
3.			$l_1 \xrightarrow{c\geq 3} l_2,\ Inv(l_1) = c \leq 3$

FIFO Queue.

CAMoLa stellt das Objekt First In First Out (FIFO)-Queue bereit, um den Modellierungsaufwand beim Erstellen von Queues zu reduzieren. Mit dem Verwenden eines FIFO-Queue Objekts in einem Modell, wird bei der Transformation zu UPPAAL automatisch ein Satz von Basis-Operationen erstellt. Jedes FIFO-Queue-Objekt muss mit drei Parametern deklariert werden: Einem eindeutigen Namen, dem Variablen-Typ der gespeichert wird (hier können neben den Basistypen auch komplexe Typen verwendet werden) und der Anzahl von Speicherplätzen für Variablen. Das Benutzen der Basis-Operationen erfolgt über den Operationsnamen und dem Namen der Queue. Ein FIFO-Queue-Objekt arbeitet auf der Sequenz S mit der maximalen Länge $|S| \leq MAX$ und definiert die Operationen $enq : S \times e \to S \times f$, $deq : S \to S \times e \times f$ und $size : S \to \mathbb{N}^0$. Die Funktionen enq und deq zeigen einen Overflow beziehungsweise Underflow von S durch das Flag f an. Hat S die maximale Anzahl an Elementen aufgenommen, werden keine Weiteren eingefügt. Aus einer leeren Sequenz $S = \langle\rangle$ wird ein undefiniertes Element $\perp$ beim Aufruf von deq zurückgegeben. Beide Fälle werden bei der Modellverifikation überprüft. Hierzu wird für jedes FIFO-Queue-Objekt eine temporallogische Aussage von der Form $A\Box\neg(of \vee uf)$ generiert. Dadurch entfällt das manuelle Erstellen von TCTL Formeln für diese Objekte.

$$enq(S, e) = \begin{cases} \langle S\rangle \frown e, OK & \text{wenn } |S| < MAX, \\ \langle S\rangle, of & \text{sonst.} \end{cases} \tag{4.1}$$

$$deq(e \frown \langle S\rangle) = \langle S\rangle, e, OK \tag{4.2}$$

$$deq(\langle\rangle) = \langle\rangle, \perp, uf \tag{4.3}$$

Fault-Switch.

Die Verifikation von Safety- und Reliability-Eigenschaften eines Architekturmodells wird von dem Übertragungsverhalten des Kommunikationskanals beeinflusst. Schwerpunkt dieser Verifikation ist die Berücksichtigung von Übertragungsfehlern, wie sie in der CENELEC EN50159-2 definiert sind (siehe Tabelle 2.2 auf Seite 16). Der Fault-Switch (Abbildung 4.7 auf der nächsten Seite) modelliert verzweigte (stochastisch unabhängige) Entscheidungen in einem Prozess zwischen einer Location l auf jeweils die Locations $l'_0, \ldots, l'_n$. Dieses syntaktische Element von CAMoLa dient zum Modellieren von Fehlerhypothesen und kann in zwei verschiedenen Ausprägungen zu UPPAAL Modellen übersetzt werden: Zum einen als *Nondeterministic* und zum anderen als *Trace-Controlled*. Die Nondeterministic Variante ist nichtdeterministisch und wird zur Verifikation in der Phase-2 eingesetzt, während die Trace-Controlled Variante durch eine Trace gesteuert wird und damit deterministisch ist. Die Trace-Controlled Variante dient zur Analyse der logischen Zuverlässigkeit in der Phase-3. Ein Umschalten beziehungsweise ein manueller Eingriff zwischen diesen beiden Versionen ist nicht nötig, da diese Entscheidung vom Codegenerator bei der Transformation getroffen wird. Der Fault-Switch stellt ein zentrales Objekt bei der Analyse von Zuverlässigkeitseigenschaften mit einem nicht-probabilistischen Model Checker dar.

In der Tabelle 4.1 auf Seite 69 ist ein Beispiel eines Übertragungskanals abgebildet. Der Fault-Switch dieses Beispiels modelliert zwei verschiedene Übertragungscharakteristiken: Die korrekte Übertragung einer Nachricht (Transition 0) und der Verlust einer Nachricht (Transition 1). Zu jeder Transition wird zudem festgelegt, wie oft diese benutzt werden darf. Diese Limitierung (b_i: Grenzwert der Transition i) wird an dem entsprechenden Ausgang notiert, wobei mit der Angabe eines $*$-Zeichens die Transition beliebig oft benutzt werden darf. Der Fault-Switch modelliert damit eine Fehlerhypothese. Zum einen wird durch das Beschalten der Ausgänge festgelegt, welches fehlerhafte Übertragungsverhalten auftreten kann und zum anderen wird durch die Limitierung definiert, wie oft dieses Verhalten maximal auftreten darf. Eine solche Fehlerhypothese ist sinnvoll, da die Komplexität bei der Modellverifikation erheblich von der Kombinatorik der Fehler beeinflusst wird. Des Weiteren kann die korrekte Funktion eines Modells nur nachgewiesen werden, wenn fehlerhaftes Verhalten nicht unbeschränkt oft zugelassen wird[1]. Die Festlegung von Fehlerhypothesen ist ein adäquates Mittel für modellbasierte Verifikationen [SH93, OP06].

[1] Es ist hilfreich, ein Kommunikationsprotokoll unter verschiedenen Fehlerhypothesen zu verifizieren: Es kann damit geprüft werden, unter welcher Anzahl und welcher Art von Fehlern die Anforderungen erfüllt werden.

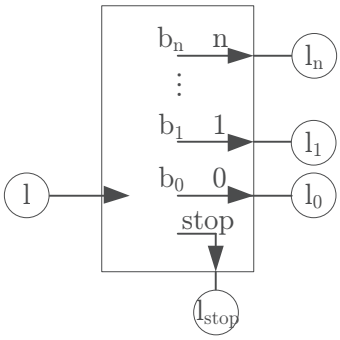


Abbildung 4.7: CAMoLa Fault-Switch

Fault-Switch: Nondeterministic.

Ist ein Fault-Switch als Nondeterministic markiert, so wird dieser bei der Transformation zu UPPAAL als nichtdeterministische Transitionen zwischen einer Eingangs-Location l und den Ausgangs-Locations $l'_0, \ldots, l'_n$ übersetzt. Es werden nur Transitionen erzeugt, wenn der entsprechende Ausgang zu einer Location führt. Die Limitierung der Übergänge über die Transition wird durch zusätzliche Zähler-Variablen $cnt_0, \ldots, cnt_n$ erreicht. Diese werden nicht für Transitionen generiert, deren Limitierung durch das $*$-Zeichen aufgehoben ist. Die Hilfsvariablen sind mit $cnt_0 := b_0, \ldots, cnt_n := b_n$ initialisiert. Syntaktisch wird ein Nondeterministic Fault-Switch folgendermaßen transformiert:

$$
\begin{aligned}
l &\xrightarrow{cnt_0>0/cnt_0:=cnt_0-1;} l'_0 \\
l &\xrightarrow{cnt_1>0/cnt_1:=cnt_1-1;} l'_1 \\
&\vdots \\
l &\xrightarrow{cnt_n>0/cnt_n:=cnt_n-1;} l'_n
\end{aligned}
$$

Fault-Switch: Trace-Controlled

Ist ein Fault-Switch als Trace-Controlled markiert, wird bei der Transformation zu UPPAAL eine deterministische Verzweigung von Transitionen generiert. Eine Trace legt bei der Modellverifikation fest, in welcher Reihenfolge die Übergänge aktiviert sind (Abbildung 4.8 auf der nächsten Seite). Eine Trace z^d der Dimension d steuert dabei die Übergänge. Eine Trace besteht nur aus Elementen, deren Transition einer zu Location führt. Jede Transi-

tion wird durch eine Zahl im Bereich $0, \ldots, n$ repräsentiert. Es ist j eine Zähl-Variable, die mit jedem Übergang über eine Transition auf $j := j + 1$ inkrementiert wird. Initial ist j mit $j := 0$ initialisiert. Sofern die Dimension d der Trace nicht ausreicht, um alle Übergänge während der Modellausführung zu definieren, wird die Transition zur Stop-Location l_{stop} aktiviert. Auf diesem Pfad stoppt die Berechnung weiterer UPPAAL Konfigurationen, so dass l_{stop} einen finalen Zustand darstellt. Für jeden Fault-Switch im Modell wird eine konstante Trace z^d festgelegt, die sich während des Model-Checking Prozesses nicht verändert. Syntaktisch wird eine Trace als Array von Integer-Konstanten im UPPAAL Modell repräsentiert[2]. Ein Trace-Controlled Fault-Switch wird wie folgt transformiert:

$$
\begin{aligned}
l &\xrightarrow{j<d \wedge z^d(j)=0 / j:=j+1;} l'_0 \\
l &\xrightarrow{j<d \wedge z^d(j)=1 / j:=j+1;} l'_1 \\
&\vdots \\
l &\xrightarrow{j<d \wedge z^d(j)=n / j:=j+1;} l'_n \\
l &\xrightarrow{j \geq d} l_{stop}
\end{aligned}
$$

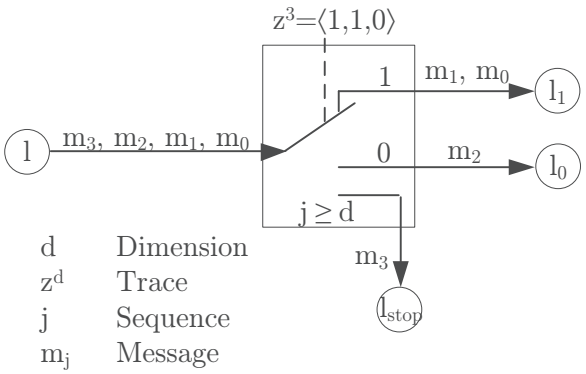


Abbildung 4.8: Trace-Controlled Fault-Switch

Die Abbildung 4.8 einen Trace-Controlled Fault-Switch schematisch. Dieser bestimmt mit der Trace z^d die Transition, die für das Passieren einer Nachricht gewählt wird.

[2]Die Traces im Modell werden vom Mutationsgenerator generiert (Kapitel 4.6 auf Seite 83).

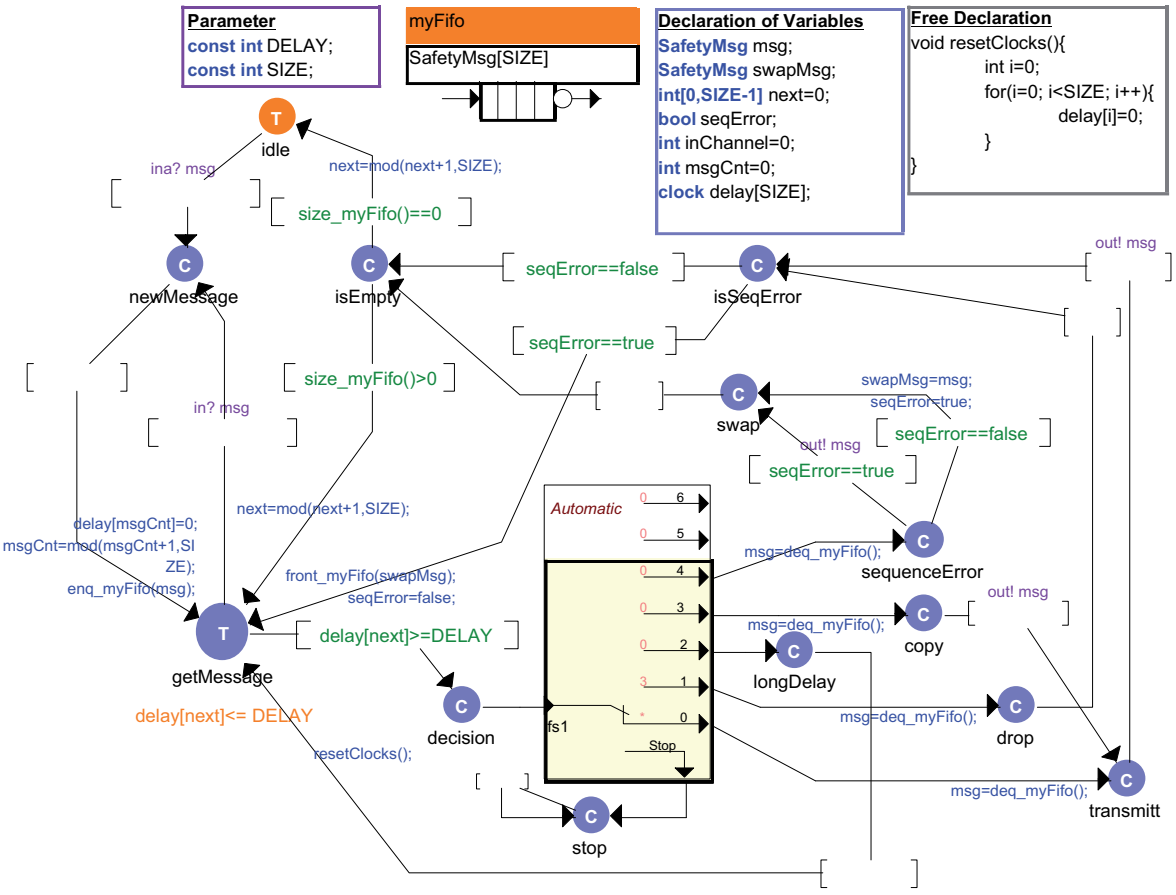


Abbildung 4.9: Übertragungskanal mit generischen Fehlerarten

4.4 Workflow Phase 1: Modellieren von Kommunikationsarchitekturen und Formalisieren von Anforderungen

In der Phase 1 des Workflows werden mit der vorgestellten Modellierungssprache Modelle von Kommunikationsarchitekturen erstellt. Die Basis der Modellierung bildet eine informale Spezifikation eines Kommunikationsprotokolls oder eines Kommunikationsdienstes. Nach dem Erstellen eines Architekturmodells aus den Protokollen werden Anforderungen formalisiert und in der Systemebene notiert. Die Anforderungen bestehen aus den drei Klassen der TCTL Formeln und beschreiben Eigenschaften, die das Systemmodell zu erfüllen hat. Ein Architekturmodell besteht somit aus der Spezifikation, den Anforderungen, die der Systemdesigner festgelegt hat und den assoziierten Anforderungen von speziellen Modellobjekten (zum Beispiel Abwesenheit von Pufferüberläufen). Mit der Generierung des UPPAAL Modells und den UPPAAL TCTL Queries[3] aus dem CAMoLa-Systemmodell

[3]Die TCTL Formeln werden in UPPAAL auch Queries genannt.

beginnt die Modellverifikation. Im Folgenden sind die drei Schritte der Phase 1, das Erstellen von Modellen, Formalisierung von Anforderungen und Generierung von Uppaal Modellen und Uppaal Queries beschrieben.

4.4.1 Modellieren von Kommunikationsarchitekturen

Eine Architektur besteht mindestens aus zwei Kommunikationspartnern und einem Übertragungskanal zwischen den Partnern. Jeder Kommunikationsteilnehmer beinhaltet ein oder mehrere Kommunikationsprotokolle. Der Entwurf einer Kommunikationsarchitektur erfordert ein sorgfältiges Planen und Zuordnen von Funktionalitäten zu den jeweiligen Protokollschichten. Spezifikationen werden in einer Bibliothek abgelegt, so dass bei dem Erstellen neuer Architekturen von bereits modellierten Systemen profitiert werden kann. Die Spezifikation der Protokolle beziehungsweise der Kommunikationsdienste werden mit der Modellierungssprache CAMoLa als Zustandsdiagramm abgebildet. Das Modell eines generischen Übertragungskanals ist fester Bestandteil des Modellierungs-Frameworks. Mit der Spezifikation von Basisparametern wird dieser auf einen abstrakten Übertragungskanal zwischen den kommunizierenden Systemen eingestellt. Der Kanal modelliert die in der EN 50159 aufgelisteten Fehlerarten (Tabelle 2.2 auf Seite 16). In der Phase 2 prüft der Model Checker auf allen Ausführungspfaden des Architekturmodells, ob die formalisierten Anforderungen eingehalten werden. Unerlässlich für das Model Checking sind Abstraktionen, die den resultierenden Zustandsraum in einer Größe halten, die den Verifikationsprozess mit verfügbaren Rechnerkapazitäten erlaubt.

4.4.2 Abstraktionen

Mit steigender Komplexität der Modelle steigt auch die Komplexität der Modellverifikation. Dieses wirkt sich nachteilig auf die Model Checking Dauer und den Speicherbedarf bei der Verifikation aus. Dieses Problem ist als Zustandsraumexplosion bekannt und ein Nachteil von Model Checking Methoden [Cla08]. Dem entgegen wirken Abstraktionstechniken, wobei zwischen automatischen Abstraktionen und manuellen Abstraktionen unterschieden werden muss. Zu den automatischen Abstraktionen gehören Techniken, die das Model Checking Tool automatisch durchführt und damit die Größe des Zustandsraums reduziert [DT98, HFV06, BK08]. Diese Methoden sind nicht Gegenstand dieser Arbeit. Bei der manuellen Abstraktion entscheidet der Modellierer selbst über die Detailtiefe einer Systemspezifikation, die in das Modell aufgenommen wird. Diese Abstraktionen können wesentlich effektiver als die automatischen Methoden sein, erfordern jedoch vom Modellierer Model Checking Erfahrung und gute Kenntnisse über das zu modellierende System [WHS06].

Die Detailtiefe eines Modells hat einen wesentlichen Einfluss auf den Zustandsraum, der bei der Verifikation expandiert wird. Von dem initialen Zustand eines Systemmodells erzeugt der Model Checker weitere Zustände, die bei einer Ausführung erreichbar sind. Interleaving, Variablen-Evaluationen und Clock-Regions tragen damit zur Zustandsraumgröße bei. Das Minimieren der Anzahl von Prozessen, Variablen und Clocks sowie die größtmögliche Einschränkung von Wertebereichen von Variablen sind die wesentlichen manuellen Abstraktionstechniken. Die Modelle von Kommunikationsarchitekturen müssen ein kleinstmögliches System repräsentieren, ohne dabei Details zu verlieren, die Safety- oder Reliability-Aussagen des Modells gegenüber dem realen System verfälschen.

Für Modelle von Kommunikationsarchitekturen sind folgende Heuristiken zur Reduktion des Zustandsraums zu beachten: Es ist die minimale Anzahl von Kommunikationspartnern zu modellieren, die für Verifikation und Analyse ausreichen. In den meisten Fällen handelt es sich um eine Peer-to-Peer-Beziehung, so dass mit zwei Teilnehmern Safety- und Reliability-Eigenschaften von Kommunikationsprotokollen nachgewiesen werden können. Zudem werden nur die Teile einer Architektur modelliert, deren Funktionalität Einfluss auf die Verifikation hat. Alle weiteren Kommunikationsschichten werden soweit wie möglich abstrahiert. Zum Beispiel ist es nicht nötig, das Ethernet-Protokoll oder einen Switch bei der Verifikation eines Safety-Protokolls zu modellieren. Der Einfluss beider Teile auf Nachrichten wird bereits im Kanalmodell abgebildet: Übertragen, Verzögern und Verlust von Nachrichten. Weiterhin ist das Adressieren von Teilnehmern Bestandteil eines Safety Protokolls, jedoch kann der Address-Check durch ein einziges Flag (Correct/Wrong) in der Protokollstruktur repräsentiert werden. Eine weitere Abstraktion ist die Einschränkung des Wertebereichs von Variablen. Insbesondere Sequenznummern beeinflussen die Größe des Zustandsraums. Anstelle des realen Wertebereichs von zum Beispiel 32 Bit wird im Systemmodell ein Bereich gewählt, der ausreicht, um Sequenzfehler zu erkennen. Wird im Systemmodell ein zu geringer Wertebereich gewählt, dann kann das Systemmodell die Verletzung einer Safety-Bedingung zeigen, die im realen System nicht vorkommt. Zum Beispiel ist der Verlust von n Nachrichten mit einer Sequenznummer von $0, \ldots, n$ festzustellen (dieses gilt analog für das Vertauschen von Nachrichten bis zur Tiefe n). Mit einem zu geringen Wertebereich ist der Verlust (oder die Vertauschung) von Nachrichten bei der Sequenzprüfung im Systemmodell nicht festzustellen, wodurch die Safety-Verifikation fehlschlägt. Hingegen werden systematische Fehler im Systemmodell (zum Beispiel ein fehlerhaft spezifizierter Sequenzcheck) mit kleinen Wertebereichen bereits aufgedeckt. Die Größe von Sequenzzählern im Modell ist damit von der verwendeten Fehlerhypothese abhängig.

Eine weitere Abstraktion ist die Reduktion der Anzahl von Prozessen im Modell. Zu den Prozessen gehören neben den realen Architekturbestandteilen auch Watchdog-Prozesse,

die als Hilfsmittel zur Verifikation dienen. Watchdog-Prozesse überwachen eine gestartete Transaktion auf das Einhalten von Echtzeitbedingungen und ermöglichen die Überprüfung weiterer Safety-Eigenschaften. Die Anzahl der Watchdog-Prozesse ist davon abhängig, wie viele Transaktionen gleichzeitig zu überwachen sind. Hierbei ist eine Abstraktion so zu wählen, dass keine wesentlichen Eigenschaften bei der Modellierung verloren gehen aber auch keine Zustandsraumexplosion entsteht.

Weiterhin haben Clocks negativen Einfluss auf den Zustandsraum. Die Komplexität hängt dabei unter anderem von der verwendeten Darstellung im Model Checker ab. In Uppaal können Clocks in Difference Bound Matrix (DBM) oder Clock Difference Diagrams (CDD) repräsentiert werden, wobei die CDD Struktur kompakter auf Kosten der Model Checking Zeit ist. Echtzeit-Uhren und ihre Repräsentation in Model Checking Methoden sind ein zentrales Forschungsthema von Timed Automata. Für weitere Informationen sei auf [LLPY97, PWY+98, MN08] verwiesen. Neben diesen wesentlichen Heuristiken zur Beherrschung des Zustandsraums gibt es weitere Techniken, die speziell bei dem Erstellen von Uppaal Modellen verwendet werden können [BDL+06, DILS09].

4.4.3 Formalisieren von Anforderungen

Anforderungen an das reale System und weiteres erwartetes Verhalten des Systemmodells werden formalisiert, indem mit der Temporallogik TCTL Aussagen über Zustände aufgestellt werden. Erfüllt das Systemmodell die formalisierten Anforderungen in erwarteter Weise, ist von einem korrekten Modell und einer korrekten Spezifikation auszugehen. Wird eine Anforderung nicht erfüllt, generiert der Model Checker ein Gegenbeispiel, das den Ausführungspfad des Modells zeigt, welcher die Anforderung verletzt. Die Temporallogik ist in drei Klassen von Aussagen aufgeteilt: *Safety*, *Liveness* und *Reachability*. Mit einer geeigneten Kombination von Aussagen dieser Eigenschaftsklassen können wesentliche Funktionen des Modells automatisch verifiziert werden [Koe03].

- **Safety-Eigenschaften** besagen, dass niemals Sicherheitsanforderungen verletzt werden. Positiv formuliert heißt das, dass auf allen Ausführungspfaden immer alle Sicherheitsbedingungen eingehalten werden. In TCTL wird eine solche Aussage formal als $A\Box\varphi$ spezifiziert, wobei φ die Sicherheitsbedingung formuliert. Die Sicherheitsbedingungen lassen sich direkt aus den in der EN 50159 definierten Fehlerarten ableiten. Sobald eine fehlerhafte Nachricht (zum Beispiel überschrittene Echtzeitanforderung, Sequenzfehler, verfälschte Nachricht) von einem Safety-Protokoll in den Applikationsprozess übergeben wird, ohne dass mit dem Einnehmen des Stable-Safe-States der Fehler offenbart wird, handelt es sich um eine Sicherheitsverletzung. Sicherheitsverletzungen können mit Hilfe eines Systembeobachters nachgewiesen werden: Dieser

ist über einen fehlerfreien Kanal mit der Sender- und Empfängerapplikation verbunden und zeigt die Verletzung von Safety-Bedingungen mit dem Wechsel in die Indikator-Location *unsafe* an. Der Nachweis von Sicherheitsbedingungen wird dann mit $A\Box\neg unsafe$ formalisiert.

- **Liveness-Eigenschaften** sind für die Zuverlässigkeit entscheidend. Ein in Anspruch genommener Protokolldienst muss schließlich einmal ausgeführt werden. Mit diesen Eigenschaften werden zum Beispiel Livelocks oder Zuverlässigkeitseigenschaften ermittelt. Liveness ist dabei immer von der Fehlerhypothese abhängig: Ein dauerhaft defekter Übertragungskanal kann das korrekte Ausführen von Transaktionen verhindern, da Fehlerzähler oder Zeitüberwachungen für Transaktionen zu einem Abbruch führen. In diesem Fall muss ein Safety-Protokoll den Stable-Safe-State einnehmen, der die Sicherheitsbedingungen erfüllt, jedoch nicht als zuverlässig zu betrachten ist. Zu den Liveness Eigenschaften gehören $A\Diamond\varphi$ und $\psi \rightsquigarrow \varphi$. Die Festlegung geeigneter Zustände φ und ψ kann ebenfalls unter Zuhilfenahme von Systembeobachtern durchgeführt werden.

- **Erreichbarkeits-Eigenschaften** dienen zum Nachweis bestimmter Systemzustände. Hierzu gehören Aussagen über Pufferüberläufe (nicht erwünscht), Variablenwerte oder Locations. Reachability wird spezifiziert mit $E\Diamond\varphi$, wobei φ der gesuchte Systemzustand ist.

- Eine weitere wichtige Eigenschaft ist die **Deadlock-Freiheit**. Diese ist zwar indirekt schon mit Liveness geprüft, es empfiehlt sich jedoch, die Deadlock-Freiheit zu Beginn der Verifikation nachzuweisen. Aus Erfahrung führen bei komplexen Modellen ungeeignete Abstraktionen und Modellierungsfehler oftmals zu Deadlocks im Modell.

Alle formalisierten Anforderungen werden in CAMoLa Modellen gemeinsam mit der Prozessspezifikation in der Prozess-Ebene definiert. Damit ist jedes Modell einer Spezifikation mit den wesentlichen Anforderungen verbunden, die beim Wiederverwenden der Spezifikation in anderen Architekturmodellen nicht erneut aufgestellt werden müssen. Neben den manuell erstellten Anforderungen besitzen FIFO Objekte vordefinierte Queries zum Nachweisen von Pufferüber- beziehungsweise Unterläufen. Diese formalisierten Anforderungen werden automatisch bei der Generierung der Query-Files für den Model Checker vom Codegenerator erzeugt.

4.4.4 Design Transformation und Modellverifikation

Die Transformation von CAMoLa Modellen zu Uppaal Modellen wird von drei Codegeneratoren übernommen. Nach den dargestellten Regeln zu den jeweiligen Objekten

von CAMoLa durchlaufen die Codegeneratoren ein Architekturmodell und transformieren daraus jeweils ein UPPAAL Modell für qualitative Verifikation (*Safety-Generator*) und qualitative Analyse (*Reliability-Generator*). Die Codegeneratoren unterscheiden sich durch die unterschiedliche Transformation von den im Modell vorhandenen Fault-Switches, die als Nondeterministic beziehungsweise als Trace-Controlled transformiert werden (Abbildung 4.10). Es werden jeweils XML-Dateien erzeugt, die von dem UPPAAL Model Checker als Eingabemodell verwendet werden. Die Codegeneratoren erzeugen automatisch die Deklaration von Synchronisierungskanälen sowie ein entsprechendes Mapping der paarweise synchronisierten Prozesse. Zudem werden für das Value-Passing temporäre Variablen erzeugt. Jede Prozessbeschreibung von CAMoLa Modellen wird zu einem UPPAAL Template transformiert. Die Codegeneratoren erzeugen automatisch eine UPPAAL-Systemdeklaration, indem die modellierten Prozesse der Architektur die Templates instanziieren. Ein dritter Codegenerator erzeugt aus den explizit definierten TCTL Formeln und den implizit gegebenen TCTL Formeln eine Query-Datei für den UPPAAL Model Checker. Der Systemdesigner muss keine Veränderungen an den generierten Modellen vornehmen und beginnt die Verifikation, indem das Modell und die Query-Datei in den Model Checker geladen werden.

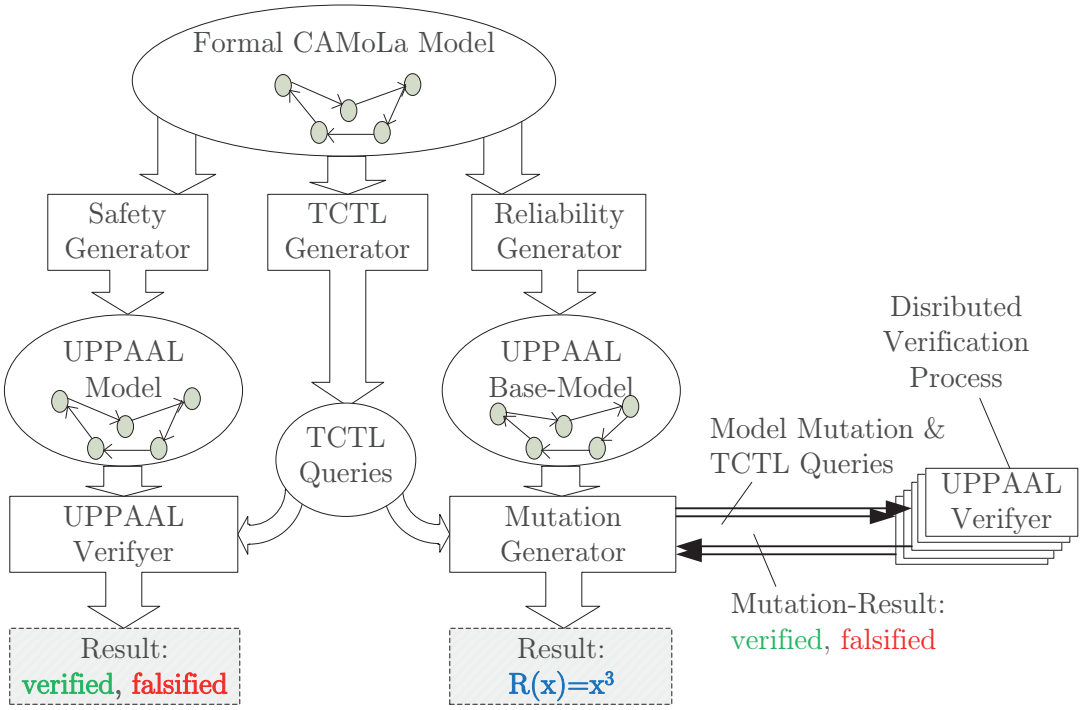


Abbildung 4.10: Transformation von CAMoLa Modellen

4.5 Phase 2: Verifikation qualitativer Eigenschaften

Mit dem Einlesen der Model- und der Query-Datei in das Model Checking UPPAAL beginnt die Phase 2 des CAMoLa Frameworks. Der UPPAAL Model Checker besitzt neben der graphischen Modellierungsoberfläche einen Simulator, der das Modell in Einzelschritten ausführt sowie einen Verifier, mit dem das Model Checking durchgeführt wird. Zunächst findet eine syntaktische Überprüfung des Modells und der Queries statt. Korrekturen werden in das CAMoLa Modell eingepflegt, bis diese Art der Fehler eliminiert sind. Nacheinander werden dann die einzelnen formalen Anforderungen verifiziert und die Ergebnisse ausgewertet. Ist eine Eigenschaft nicht erfüllt, generiert UPPAAL automatisch im Simulator den Ausführungspfad, der zur Verletzung der Eigenschaft geführt hat. Ist ein solcher Pfad gefunden, ist eine manuelle Begutachtung nötig, um festzustellen, ob es sich hierbei um einen Modellierungsfehler handelt, oder ob die Spezifikation fehlerhaft ist. Die Korrektur findet in dem CAMoLa Modell statt, welches anschließend wieder zu einem UPPAAL Modell übersetzt wird. Danach beginnt die Verifikation erneut. Dieser iterative Verbesserungsprozess wird durchgeführt, bis alle spezifizierten Eigenschaften in erwarteter Weise erfüllt werden. Insbesondere Safety-Eigenschaften dürfen mit dem Abschluss der Phase 2 nicht verletzt werden, da dieses auf Spezifikationsfehler hindeutet. Es können jedoch nur Fehler gefunden werden, die das Ergebnis einer entsprechenden Eigenschaft beeinflussen. Dem Definieren von geeigneten Queries ist daher besondere Aufmerksamkeit zu widmen. In Kapitel 4.4.3 auf Seite 79 sind Best-Praxis Methoden vorgestellt, Queries für die Modell- und Safety-Verifikation zu erstellen.

Ein nicht zu unterschätzender Nebeneffekt der qualitativen Verifikation ist das wachsende Verständnis über das modellierte System. Die informalen textuellen Systemspezifikationen definieren teilweise missverständliche Funktionen oder erlauben Freiheitsgrade, die erst mit dem schrittweisen Ausführen des Modells sinnvoll korrigiert und ergänzt werden können. Die Systemmodelle bestehen neben dem Modell der Spezifikation auch aus Modellen zur Umgebung (zum Beispiel Applikationsmodelle und Übertragungskanal-Modelle). Mit der modellbasierten Verifikation ist das Aufspüren von Spezifikationsunschärfen einfacher zu handhaben als bei der späteren Implementierungsphase.

Der zusätzliche zeitliche Aufwand bei der Modellierung und Verifikation kann jedoch zu geringerem Implementierungsaufwand führen. Mit dem Beseitigen von Spezifikationsunschärfen ist die Spezifikation bei der Implementierungsphase besser umzusetzen. Zudem kann zu dem CAMoLa Framework ein zum Beispiel C-Code Generator hinzugefügt werden, der Zustandsdiagramme von Kommunikationsarchitekturen zu einer Implementierung transformiert. Auf diese Weise kann sichergestellt werden, dass die Implementierung der

zuvor verifizierten Systemspezifikation entspricht. Mit dem CAMoLa Framework können in einer eingeschränkten C-Syntax Funktionen für Kontrollfluss-Aufgaben definiert werden, die ebenfalls mit dem Model Checker UPPAAL verifiziert werden. Implementierungsfehler, wie zum Beispiel verwechselte Boolesche Operatoren in Schleifen oder "Copy-Paste Fehler", die in der Phase 2 des Frameworks gefunden und korrigiert werden, verhindern mit der automatischen Transformation, dass diese Fehler bei der manuellen Implementierung der Spezifikation auftreten.

Ein Generator zum Erzeugen von Teil-Implementierungen aus dem CAMoLa Modell ist zurzeit nicht Bestandteil des CAMoLa Frameworks. Die Definition eines solchen Codegenerators zu einem späteren Zeitpunkt ist aufgrund des Metamodell-Konzepts und der zugehörigen Skriptsprache Merl mit wenig Aufwand zu realisieren. Die Verifikation von qualitativen Eigenschaften mit dem zugehörigen iterativen Korrektur-Prozess gliedert sich, im V-Modell betrachtet, vor der Implementierungsphase ein.

4.6 Phase 3: Analyse quantitativer Eigenschaften

Nach der Verifikation der qualitativen Eigenschaften folgt die Analyse von quantitativen Charakteristiken. Erfüllt das Architekturmodell die wesentlichen Anforderungen bezüglich Safety und Deadlock-Freiheit sowie Liveness und Reachability in erwarteter Weise, ist zunächst von einem korrekten Modell und einer korrekten Systemspezifikation auszugehen. Liveness Eigenschaften sind dabei von der Fehlerhypothese abhängig. Ist zum Beispiel der Verlust von Nachrichten unbeschränkt oft möglich, dann gibt es einen Pfad im Modell, der niemals eine Nachricht über den Kanal überträgt. Eine restriktivere Fehlerhypothese kann in Safety-Protokollen ebenfalls die Liveness-Eigenschaft negativ beeinflussen: Mit jeder Wiederholung einer fehlerhaften Nachricht veraltern die zu übermittelnden Informationen, womit schließlich der sichere Zustand (Stable-Safe-State) vom Protokoll eingenommen wird. In dem Stable-Safe-State ist die erwartete Funktion der Kommunikation aus Sicht der Applikation nicht mehr gegeben und dies führt letztendlich zum Verlust der Verfügbarkeit der Systemaufgaben. Für eine hochzuverlässige Kommunikationsarchitektur ist es erforderlich, die Wahrscheinlichkeit zu kennen, mit der dieser Stable-Safe-State eingenommen wird. Abhängig ist diese Wahrscheinlichkeit von der Fehlerart und Fehlerhäufigkeit bei der Übertragung. Im Gegensatz zu einem probabilistischen Model Checker ist ein Timed Automata Model Checker nicht in der Lage, probabilistisches Verhalten eines Übertragungskanals zu modellieren. Aus diesem Grund wird das Systemmodell mit dem nichtdeterministischen Übertragungskanal durch Modellmutationen ersetzt, die jeweils einen deterministischen Übertragungskanal beinhalten. Dieses deterministische Verhalten wird durch die *Trace-Controlled Fault-Switches* erzeugt (im Folgenden als fs ab-

gekürzt). Ein Basismodell ist der Ausgangspunkt für Modellmutationen (Abbildung 4.10 auf Seite 81) und beinhaltet k Trace-Controlled Fault-Switches, $fs_1, \ldots, fs_k$, jedoch keine Trace zum Steuern der fs. Es muss mindestens ein fs im Basismodell vorhanden sein. Jedem Fault-Switch $fs_1, \ldots, fs_k$ wird eine Trace $z_1^d, \ldots, z_k^d$ der gleichen Dimension d vom Mutationsgenerator zugeordnet, wodurch aus dem Basismodell $\mathcal{M}$ eine Modellmutation $\mathcal{M}(Z^d)$ entsteht. Mit $Z^d = (z_1^d, \ldots, z_k^d)$ wird ein Tupel von Traces abgekürzt, das jeden fs im Basismodell zu einer deterministischen Verzweigung von Transitionen macht. Die Abbildung 4.8 auf Seite 75 zeigt die schematische Funktionsweise eines Trace-Controlled Fault-Switch.

Jeder fs repräsentiert eine Menge $\mathcal{FS}$ von *wählbaren* Transitionen, die zu einer Ziel-Location $l'_0, \ldots, l'_n$ führen. Die Transition l_{stop} gehört nicht dazu, so dass $\mathcal{FS}_i = \{0, \ldots, n_i\}$ die Menge der wählbaren Transitionen von fs_i, für $i = 1, \ldots, k$ darstellt. Eine Transition gilt als wählbar,

1. wenn der entsprechende Ausgang von fs zu einer Location führt und
2. wenn z^d die jeweilige Fehlerhypothese $b_1, \ldots, b_n$ von fs erfüllt.

Eine Trace $z^d = \langle e_0, e_1, \ldots, e_{d-1} \rangle$ mit $e_0, \ldots, e_{d-1} \in \mathcal{FS}$ ist eine Sequenz von d geschalteten Transitionen (siehe Abbildung 4.8 auf Seite 75). Jede Mutation begrenzt die Anzahl möglicher Transitionen über einen fs: Ein fs schaltet permanent zur Location l_{stop}, wenn bei der Modellausführung der Transitionszähler j die Dimension der Trace $j \geq d$ erreicht. Gegenüber den nichtdeterministischen Fault-Switches muss die Schalt-Kombinatorik explizit in Z^d berücksichtigt werden. Die Anzahl von Modellmutationen ist damit von der Kombinatorik der wählbaren Transitionen und der Dimension d abhängig.

Die Zuverlässigkeit eines Systemmodells ist eine spezielle Liveness-Eigenschaft und wird mit einer temporallogischen Formel beschrieben, die als *Reliability-Query RQ* bezeichnet wird. In Kommunikationsarchitekturen bezieht sich die Zuverlässigkeit auf die korrekte Ausführung von Transaktionen, unter Einhaltung aller Anforderungen, wie zum Beispiel Real-Time- und Safety-Bedingungen. Modellmutationen legen mit Z^d die Transitions-Kombinatorik der Fault-Switches fest. Modelliert ein fs einen Übertragungskanal, dann entspricht die Transitionskombinatorik der Kombination von korrekter und fehlerhafter Übertragung von Nachrichten. Erfüllt eine Modellmutation $\mathcal{M}(Z^d)$ die Zuverlässigkeitseigenschaft RQ, dann ist die Kommunikationsarchitektur zuverlässig gegenüber der Übertragungskombinatorik in Z^d. Zuverlässigkeit von technischen Systemen wird über ein Zeitintervall $[t_0, t_1]$ angegeben. Die Größe des Zeitintervalls hat Einfluss auf die Komplexität der Analyse: Definiert ein Element e aus z^d die zeitbehaftete Übertragung einer Nachricht, dann ist die Größe des Analyse-Zeitintervalls proportional zur Dimension d. Das Voran-

schreiten von Zeit im Systemmodell wird gestoppt, wenn ein fs mit l_{stop} den Endzustand erreicht und keine weiteren Ausführungspfade von dort möglich sind. Zur Reduktion der Analyse-Komplexität ist es sinnvoll, RQ so zu definieren, dass das Ergebnis mit algebraischen Methoden von einem kleinen Zeitintervall auf ein größeres berechnet werden kann. Das kleinste Zeitintervall ist die zuverlässige Ausführung einer Transaktion. Eine korrekte algebraische Berechnung auf größere Zeitintervalle setzt voraus, dass jede Transaktion unabhängig von vorangegangenen Transaktionen ausgeführt wird. Interne Fehlerzähler, die über Transaktionen hinaus ihren Wert behalten oder zeitliche Eigenschaften infolge von Wiederholungen vorangegangener Transaktionen, können das Kriterium der Unabhängigkeit verletzen. In diesen Fällen kann von verschiedenen Startzuständen der Worst-Case-Fall ermittelt und für die Skalierung auf größere Zeitintervalle benutzt werden.

Unter der Voraussetzung, dass jede Transition des Fault-Switches stochastisch unabhängiges Verhalten modelliert, ist die Wahrscheinlichkeit, dass eine Sequenz von geschalteten Transitionen (festgelegt durch z^d) auftritt, gleich dem Produkt aller Wahrscheinlichkeiten, ein durch z^d festgelegtes Verhalten zu beobachten. Für einen stochastisch unabhängigen Kommunikationskanal heißt das, dass die Wahrscheinlichkeit eine Nachricht fehlerhaft zu übertragen, nicht von vorangegangen Nachrichten abhängig ist (Markov-Eigenschaft, Kapitel 2.3.2 auf Seite 23). Die Auftrittswahrscheinlichkeit einer konkreten Sequenz von Fehlern ist gleich dem Produkt aller Einzelfehlerwahrscheinlichkeiten. Sei zum Beispiel die Wahrscheinlichkeit eines Nachrichten-Verlusts p_{drop} und die Wahrscheinlichkeit einer korrekten Übertragung $p_{corrcet}$, dann ist die Auftrittswahrscheinlichkeit der Übertragungssequenz $z^3 = \langle corrcet, corrcet, drop \rangle$ gleich $p(z^3) = p_{corrcet} \cdot p_{corrcet} \cdot p_{drop}$. Handelt es sich mit $corrcet$ und $drop$ um die einzigen Auswahlmöglichkeiten, dann gilt $p_{corrcet} = 1 - p_{drop}$. Nach diesem Schema wird die Zuverlässigkeit von Kommunikationsarchitekturen auf Basis von Modellmutationen ermittelt. Im Gegensatz zum PMC ist es dabei nicht notwendig, konkrete Wahrscheinlichkeitswerte zum Zeitpunkt der Analyse festzulegen, da die Zuverlässigkeit durch eine symbolische Formel ermittelt wird.

Generieren von Modellmutationen.

Modellmutationen werden von dem Mutationsgenerator erzeugt. Dieser erstellt Trace-Tupel für die im Basismodell vorhandenen Trace-Controlled Fault-Switches fs und legt damit ein spezielles Systemmodell-Verhalten fest. Bei der Analyse von Transaktionen in einem Kommunikationsarchitekturmodell werden drei Systemzustände gesucht (Abbildung 4.4 auf Seite 66):

1. Ein unsicherer Zustand, in dem Safety-Eigenschaften nicht erfüllt werden.

2. Ein Stable-Safe-State (im Folgenden als SSS abgekürzt), der das System in einen sicheren Zustand versetzt, in dem üblicherweise der Kommunikationskanal geschlossen wird und einer Applikation somit nicht mehr zur Verfügung steht.

3. Ein zuverlässiger Zustand, definiert durch die Eigenschaft RQ, der unter Einhaltung aller Anforderungen eine Transaktion erfolgreich ausgeführt hat, so dass der Kommunikationsdienst für weitere Transaktionen bereit steht.

Neben diesen drei Systemzuständen gibt es weitere, die keine der genannten Zustände repräsentieren. Dieses betrifft Zustände, die sich zwischen dem Startzustand und einem der oben genannten Zustände befinden. Der unsichere Zustand wird zum Zeitpunkt der quantitativen Analyse nicht betrachtet, da seine Abwesenheit bereits in der Phase 2 bewiesen wurde (Abbildung 4.5 auf Seite 67). Der Mutationsgenerator rollt mit Hilfe von Z^d den Zustandsraum ab, bis eine Modellmutation entweder die Eigenschaft RQ (notiert als $\mathcal{M}(Z^d) \models RQ$) oder die Eigenschaft SSS (notiert als $\mathcal{M}(Z^d) \models SSS$) erfüllt. Informal bedeutet das, dass die Anzahl von Nachrichten über einen Kommunikationskanal schrittweise erhöht wird, bis eine Transaktion erfolgreich ausgeführt oder bis der Stable-Safe-State eingenommen wird. Im ersten Fall gilt die Transaktion als zuverlässig und im zweiten Fall als gescheitert, beziehungsweise unzuverlässig. Dieses Verfahren greift die Idee des Bounded Model Checkings auf [BCC+03]. Initial werden Mutationen mit Z^1 erzeugt. Die Abbildung 4.11 auf der nächsten Seite zeigt den Abroll-Algorithmus. Ein Unrolling-Step berücksichtigt die Kombinatorik, jeweils ein Element $e \in \mathcal{FS}$ zu jeder Trace z^d in Z^d hinzuzufügen und erzeugt damit eine Menge von Trace-Tupeln der Dimension $d+1$ aus dem Tupel Z^d. Diese Menge wird im Folgenden als $\mathcal{Z}^{d+1} = \{Z^{d+1}\}$ bezeichnet.

Definition 4.6.1 (**Menge von Basistupeln $\mathcal{E}$**)**.** Die Menge von Basistupeln $\mathcal{E}$ repräsentiert die Kombinatorik von wählbaren Transitionen aller im Modell vorkommenden $fs_1, \ldots, fs_k$. Eine Transition $0, \ldots, n_i$ von fs_i ist wählbar, wenn diese zu einer Location führt und für die jeweilige Limitierung $b_0, \ldots, b_{n_i} > 0$ gilt.

$$\mathcal{E} = \{\mathcal{FS}_1 \times \cdots \times \mathcal{FS}_k\} \tag{4.4}$$

Definition 4.6.2 (**Initiale Menge von Modellmutationen $\{\mathcal{M}(Z^1)\}$**)**.** Die Menge der initialen Modellmutationen entspricht der Menge der einelementigen Trace-Tupel (Basistupel).

$$\{\mathcal{M}(Z^1) | Z^1 \in \mathcal{E}\} \tag{4.5}$$

Definition 4.6.3 (**Fehlerhypothese von fs**)**.** Die Fehlerhypothese fh eines Fault-Switches fs limitiert die Anzahl der Verwendung von Transition $0, \ldots, n \in \mathcal{FS}$ auf maximal $b_0, \ldots, b_n$. Eine Trace z^d erfüllt die Fehlerhypothese, wenn die Transition $e_i \in \mathcal{FS}$

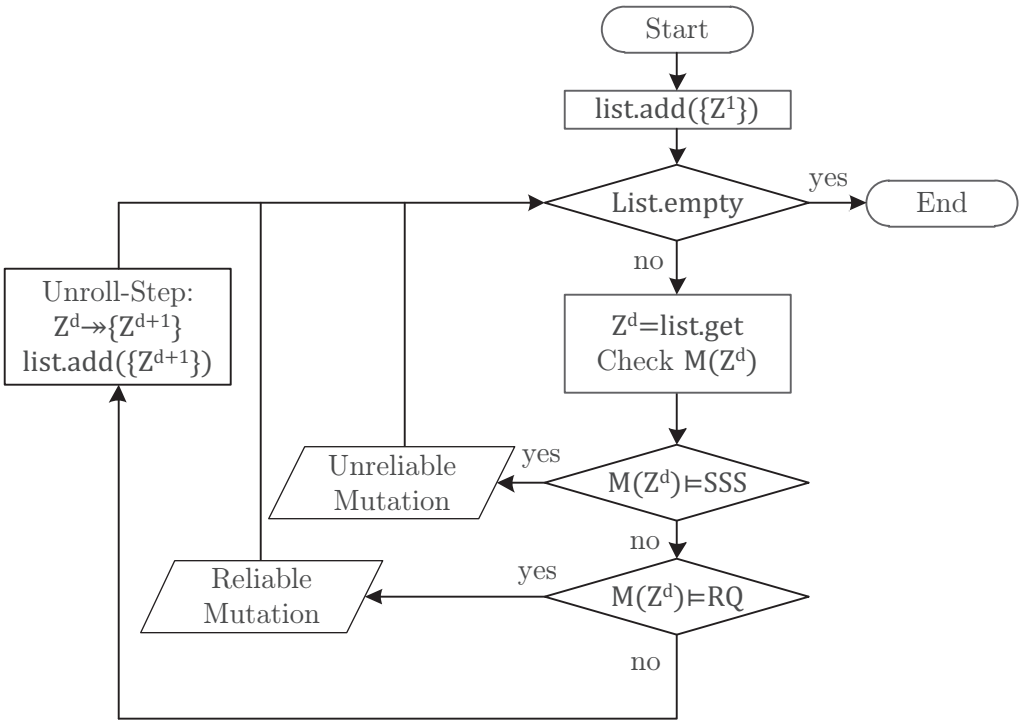


Abbildung 4.11: Generieren und Prüfen von Modell Mutationen

höchsten b_i oft in der Trace vorkommt[4].

$$z^d \models fh \Leftrightarrow \forall e_i \in \mathcal{FS} : num(e_i, z^d) \leq b_i \tag{4.6}$$

Definition 4.6.4 (**Fehlerhypothese FH einer Modellmutation $\mathcal{M}(Z^d)$**)**.** Eine Modellmutation erfüllt die Fehlerhypothese FH, wenn alle Traces $z_1^d, \ldots, z_k^d$ aus Z^d die Fehlerhypothese von $fs_1, \ldots, fs_k$ erfüllen.

$$Z^d \models FH \Leftrightarrow z_1^d \models fh_1 \wedge \cdots \wedge z_k^d \models fh_k \tag{4.7}$$

Definition 4.6.5 (**Unroll-Step $Z^d \twoheadrightarrow \mathcal{Z}^{d+1}$**)**.** Ein Unroll-Step $Z^d \twoheadrightarrow \mathcal{Z}^{d+1}$ wird vollzogen, wenn eine Mutation weder die Reliability-Query RQ erfüllt, noch einen nichtzuverlässigen Zustand SSS einnimmt. Ein Unroll-Step erzeugt eine Menge *Kind-Mutationen*, indem Z^d um jedes Tupel E aus der Menge von Basistupeln $\mathcal{E}$ und unter Berücksichtigung der Fehlerhypothese erweitert wird.

$$\mathcal{Z}^{d+1} = \{Z^d \frown E \mid Z^d \in \mathcal{Z}^d \wedge M(Z^d) \not\models RQ \wedge M(Z^d) \not\models SSS \wedge E \in \mathcal{E} \wedge (Z^d \frown E) \models FH\} \tag{4.8}$$

[4]Die Funktion *num* ermittelt die Anzahl eines bestimmten Elements e_i in einer Trace z^d

Definition 4.6.6 (**Erweiterung $Z^d \frown E$**)**.** Die Erweiterung eines Trace-Tupels ist definiert als

$$Z^d \frown E = \begin{pmatrix} z_1^d \frown e_1 \\ \dots \\ z_k^d \frown e_k \end{pmatrix}, Z^d = \begin{pmatrix} z_1^d \\ \dots \\ z_k^d \end{pmatrix}, E = \begin{pmatrix} e_1 \\ \dots \\ e_k \end{pmatrix} \tag{4.9}$$

Definition 4.6.7 (**Erweiterung $z^d \frown e$**)**.** Eine Trace $z^d = \langle e_0, \dots, e_{d-1} \rangle$ wird um ein Element $e \in \mathcal{F}S$ erweitert, indem es an das Ende der Trace angefügt wird.

$$z^{d+1} = z^d \frown e = \langle e_0, \dots, e_{d-1}, e \rangle \tag{4.10}$$

Definition 4.6.8 (**Beobachtbare Wahrscheinlichkeit $p(z^d)$**)**.** Jedes Element einer Trace steuert das Verhalten eines fs. Die Funktion $p : \mathcal{F}S \to [0, 1]$ ordnet jeder möglichen Transition (und damit dem Verhalten) $e_0, \dots, e_n$ von fs eine beobachtbare Wahrscheinlichkeit zu. Unter der Voraussetzung, dass das beobachtbare Verhalten von fs stochastisch unabhängig ist und es gilt $1 = \sum_{i=0}^{n} e_i$, ist die Wahrscheinlichkeit, dass durch eine Trace z^d bestimmte Verhalten zu beobachten, definiert als:

$$p(z^d) = \prod_{i=0}^{d-1} p(e_i) \tag{4.11}$$

Definition 4.6.9 (**Beobachtbare Wahrscheinlichkeit von $p(\mathcal{M}(Z^d))$**)**.** Die beobachtbare Wahrscheinlichkeit einer Mutation ist das Produkt aller beobachtbaren Wahrscheinlichkeit von Traces, die in einer Mutation vorkommen:

$$p(\mathcal{M}(Z^d)) = \prod_{i=1}^{k} p(z_i^d) \tag{4.12}$$

Definition 4.6.10 (**Menge zuverlässiger Mutationen $\mathcal{Z}_{RQ}$**)**.** Die Menge zuverlässiger Mutationen sind alle Mutationen, die das Zuverlässigkeitskriterium RQ erfüllen:

$$\mathcal{Z}_{RQ} = \{Z^d \mid \mathcal{M}(Z^d) \models RQ\} \tag{4.13}$$

Definition 4.6.11 (**Zuverlässigkeit einer Transaktion R_{T_ℓ}**)**.** Die Zuverlässigkeitswahrscheinlichkeit einer Transaktion ist die Summe aller beobachtbaren Wahrscheinlichkeiten von zuverlässigen Mutationen:

$$R_{T_\ell} = \sum_{Z^d \in \mathcal{Z}_{RQ}} p(M(Z^d)) \tag{4.14}$$

4.6.1 Fehlerwahrscheinlichkeit aufgrund der Fehlerhypothese

Eine Fehlerhypothese reduziert die Komplexität der Verifikation durch die Einschränkung der Fehlerkombinatorik und damit die Reduzierung zu prüfender Mutationen. Dieses geht

allerdings auf Kosten der Genauigkeit. Im Gegensatz zu einer Simulation, bei der das Ergebnis innerhalb eines Vertrauensintervalls liegt, wird bei dem in dieser Arbeit vorgestellten Ansatz eine Mindestzuverlässigkeit ermittelt, wobei der wahre Wert größer oder gleich ist. Dieser Effekt spiegelt den Worst-Case Charakter dieser Analyse wieder. Die Fehlerhypothese bestimmt die initiale Menge von Mutationen und die Menge von Mutationen, die sich mit jedem Unroll-Step entwickelt. Eine restriktivere Fehlerhypothese verringert damit die Anzahl von Mutationen, die von dem Mutationsgenerator erzeugt werden und auf denen RQ verifiziert wird. Somit kann die Menge der zuverlässigen Mutationen $\mathcal{Z}_{RQ}$ kleiner gegenüber einer unbeschränkten Fehlerhypothese (definiert als $b_0 = \cdots = b_n = \infty$ von fs) sein. Die nicht geprüften Mutationen tragen nach der Definition 4.6.11 auf der vorherigen Seite nicht zur ermittelten Zuverlässigkeit einer Transaktion bei, wodurch gegenüber der exakten Zuverlässigkeit der Transaktion $R_{T_\ell}^{exakt}$ eine untere Grenze ermittelt wird.

$$R_{T_\ell}^{exakt} = R_{T_\ell} + \varepsilon \tag{4.15}$$

Die Einschränkung der Genauigkeit aufgrund einer Fehlerhypothese ist akzeptabel, wenn der Fehler ε klein genug ist. Die maximale Größe von ε kann mit der Fehlerhypothese berechnet werden. Der Fehler kann nicht größer sein, als die Summe aller beobachtbaren Wahrscheinlichkeiten von Mutationen, bei denen Aufgrund der Fehlerhypothese mit einem Unroll-Step eingeschränkt viele Kind-Mutationen generiert werden. Die Abbildung 4.12 auf der nächsten Seite zeigt den bei der Verifikation entstehenden Baum von Mutationen, wobei jede Mutation einem Knoten entspricht. Eine Mutation ist ein Blatt (Endknoten im Baum), wenn für diese $\mathcal{M}(Z^d) \models RQ$ oder $\mathcal{M}(Z^d) \models SSS$ gilt (siehe Definition 4.6.5 auf Seite 87). Jeder Unroll-Step erzeugt nur Kind-Mutationen, welche die Fehlerhypothese erfüllen. Der Mutationsbaum kann gleichzeitig als Wahrscheinlichkeitsbaum angesehen werden, wobei jede Kante die beobachtbare Wahrscheinlichkeit einer Transition $p(e)$ darstellt (Definition 4.6.8 auf der vorherigen Seite). Die beobachtbare Wahrscheinlichkeit einer Mutation entspricht der Pfad-Wahrscheinlichkeit im Mutations-/Wahrscheinlichkeitsbaum. In Abbildung 4.12 auf der nächsten Seite ist ein Beispiel mit der Fehlerhypothese $b_0 = \infty$ und $b_1 = 3$ gezeigt. Die Grenze auf der rechten Seite der Abbildung zeigt, wann die Fehlerhypothese die Generierung aller kombinatorisch möglichen Kind-Mutationen einschränkt. In diesem Beispiel fehlen die Mutationen, bei denen in der Trace z^d die Transition $e_1 = 1$ mehr als dreimal vorkommt. Das bedeutet, dass die beobachtbare Wahrscheinlichkeit zuverlässiger Mutationen bis zum dreimaligen Vorkommen von $e_1 = 1$ korrekt ermittelt wird. Es entsteht ein Fehlerpolynom ε, welches dem Komplement aller beobachtbaren Wahrscheinlichkeiten von Blatt-Mutationen im Mutationsbaum

darstellt.

Definition 4.6.12 (Fehlerwahrscheinlichkeit ε Aufgrund einer Fehlerhypothese FH). Aus Abbildung 4.12 ist zu sehen, dass das Komplement der Summe aus R_{T_ℓ} und UR_{T_ℓ}, mit $\mathcal{Z}_{SSS} = \{Z^d \mid \mathcal{M}(Z^d) \models SSS\}$ und $UR_{T_\ell} = \sum_{Z^d \in \mathcal{Z}_{SSS}} p(M(Z^d))$ das Fehlerpolynom ε beschreiben. Es gilt:

$$\varepsilon = 1 - (R_{T_\ell} + UR_{T_\ell}) \tag{4.16}$$

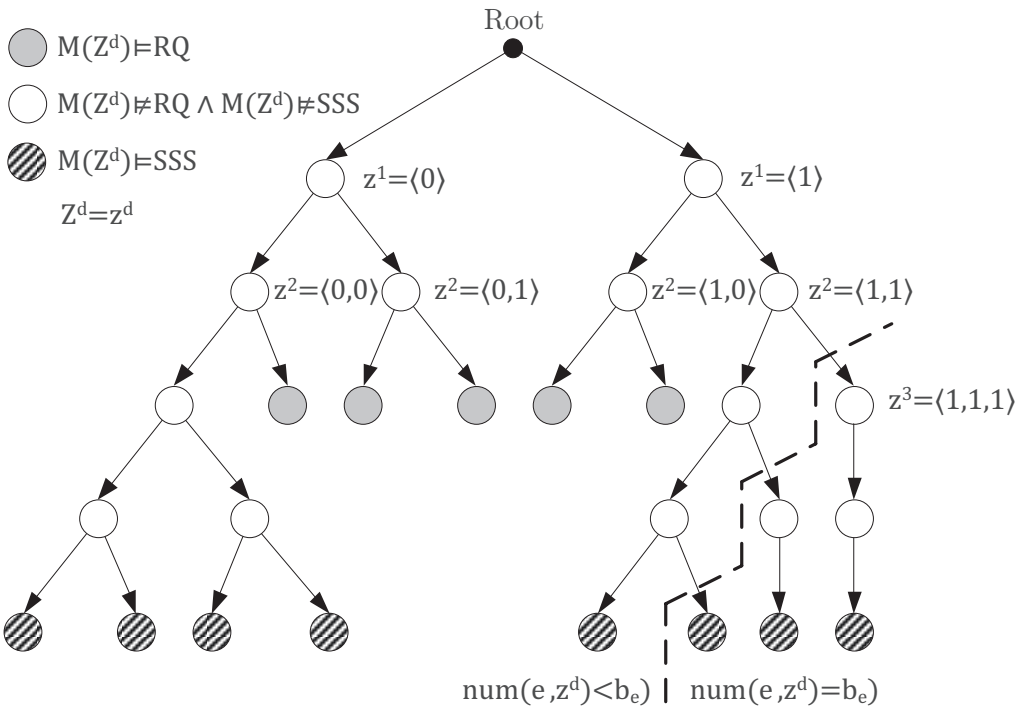


Abbildung 4.12: Beispiel eines Mutationsbaums, mit $\mathcal{FS} = \{0, 1\}, b_0 = \infty, b_1 = 3$

Zuverlässigkeit einer Architektur im Zeitintervall

Die Zuverlässigkeit des logischen Kanals in einem Zeitintervall kann zum Vergleichen von Architekturen herangezogen werden. Als Zeitintervall wird in der Literatur oftmals eine Stunde verwendet. Für die Ermittlung des Worst-Cases muss die maximale Anzahl von Transaktionen $c_\ell^{max} \in \mathbb{N}$ in dem Zeitintervall $[t_0, t_1]$ und die minimale Dauer einer Transaktion $\delta_\ell > 0$ bekannt sein.

Definition 4.6.13 (Zuverlässigkeit einer Architektur im Zeitintervall $R(t_0, t_1)$).

$$R(t_0, t_1) \geq (R_{T_\ell})^{c_\ell^{max}} \text{ mit } t_1 - t_0 \leq c_\ell^{max} \cdot \delta_\ell \tag{4.17}$$

Hier wird ersichtlich, dass die Häufigkeit der Transaktionen eine wesentliche Rolle spielt. Daher werden bei der Analyse der Zuverlässigkeit die Transaktionen (Protokolldienste) betrachtet, die "häufig" (im Dauerbetrieb) verwendet werden. Dieses sind Dienste zum Austausch von Daten. Dienste zum Aufbau einer Verbindung sind aber dennoch nicht zu vernachlässigen, da die Praxis gezeigt hat, dass Spezifikationsfehler beim Verbindungsauf- oder -Abbau zu Verklemmungen führen können. Diese Art von Spezifikationsfehlern werden nur in der Phase 2 des Frameworks ermittelt, da quantitative Bezifferungen der Zuverlässigkeit von irregulären Diensten nicht zum Vergleichen von Architekturen geeignet sind.

4.6.2 Beispiel zur quantitativen Analyse (Phase 3)

Die vorgestellte quantitative Analyse wird anhand der einfachen Beispielarchitektur aus der Abbildung 4.6 auf Seite 68 mit den zugeordneten Prozessen aus der Tabelle 4.1 auf Seite 69 durchgeführt. Es handelt sich um einen zyklischen Transmitter, der Nachrichten in festen Zyklen von zwei Zeiteinheiten sendet. Der Übertragungskanal überträgt eine Nachricht über die Transition $e_0 = 0$ von fs erfolgreich oder verwirft diese über die Transition $e_1 = 1$. Als zuverlässig gilt eine Transaktion, wenn eine Nachricht innerhalb von fünf Zeiteinheiten beim Receiver eintrifft. Andernfalls wechselt der Watchdog in die Stop-Location. Die für die Zuverlässigkeitsanalyse benötigten Queries stehen damit fest: $RQ = A\Diamond Receiver.rcv$ und $SSS = E\Diamond Watchdog.stop$. In diesen Aussagen spiegelt sich der Worst-Case Charakter der Analyse wieder: Die Reliability-Eigenschaft fordert, dass auf allen Ausführungspfaden schließlich die Nachricht ankommt, während, wenn es mindestens einen Pfad gibt, der in die Stop-Location führt, die Transaktion als gescheitert gilt. Der einzige vorhandene fs besitzt die Menge von wählbaren Transitionen $\mathcal{FS} = \{0, 1\}$. Damit ist die Menge von Basistupeln gleich $\mathcal{E} = \{\langle 0\rangle, \langle 1\rangle\}$. Als initiale Mutationen ergibt sich $\mathcal{Z}^1 = \{Z_1^1, Z_2^1\}$ mit $Z_1^1 = (\langle 1\rangle)$ und $Z_2^1 = (\langle 0\rangle)$.

Der Mutationsgenerator startet die Verifikation der Beispielarchitektur und der Model Checker liefert das Ergebnis $\mathcal{M}(Z_2^1) \models RQ$, da der Receiver bereits mit einer erfolgreichen Nachrichtenübertragung in die Location *rcv* gelangt ist. Die Mutation $\mathcal{M}(Z_1^1)$ erfüllt weder RQ noch SSS, da die erste Nachricht verworfen wird und die zweite Nachricht den Fault-Switch in die Location l_{stop} zwingt. Es wird somit ein Unroll-Step $Z_1^1 \twoheadrightarrow \mathcal{Z}^2$ durchgeführt, der weitere Trace-Tupel und damit Mutationen generiert. Nach der Unroll-Regel werden dabei $\mathcal{Z}^2 = \{Z_1^2, Z_2^2\}$ erzeugt, mit $Z_1^2 = (\langle 1, 1\rangle)$ und $Z_2^2 = (\langle 1, 0\rangle)$. Die nächste Verifikation

ergibt $Z_2^2 \models RQ$ und $Z_1^2 \not\models RQ \wedge Z_1^2 \not\models SSS$. Somit wird ein weiterer Unroll-Step $Z_1^2 \twoheadrightarrow \mathcal{Z}^3$, mit $Z_1^3 = (\langle 1,1,1 \rangle)$ und $Z_2^3 = (\langle 1,1,0 \rangle)$ durchgeführt. An dieser Stelle stoppt die Analyse, weil $\mathcal{M}(Z_2^3)$ RQ und $\mathcal{M}(Z_1^3)$ SSS erfüllt. Die während der Analyse erzeugten Trace-Tupel sind als Baum in Abbildung 4.13 dargestellt.

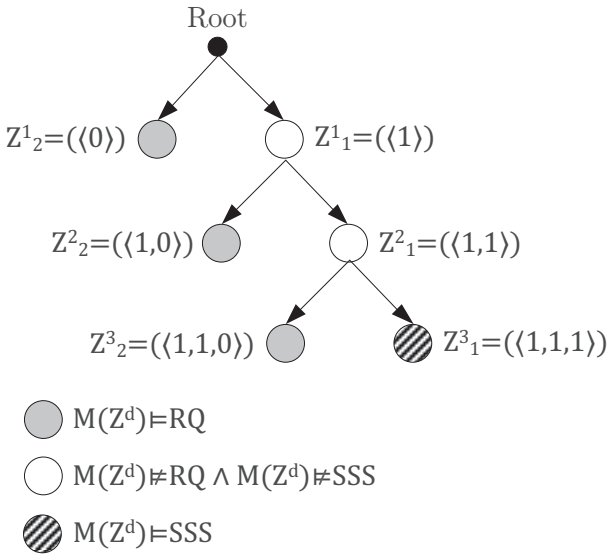


Abbildung 4.13: Mutationsbaum des Beispiels

Die Analyse hat als Menge der zuverlässigen Mutationen $\mathcal{Z}_{RQ} = \{Z_2^1, Z_2^2, Z_2^3\}$ ermittelt. Mit der Wahrscheinlichkeit $p(e_1) = x$ wird eine Nachricht über die Transition $e_1 = 1$ verworfen und mit der Wahrscheinlichkeit $p(e_0) = 1 - p(e_1) = 1 - x$ wird eine Nachricht über die Transition $e_0 = 0$ korrekt übertragen. Die Auftrittswahrscheinlichkeit der jeweiligen zuverlässigen Trace-Tupel ist:

$$p(Z_2^1) = p(0) = 1 - x \tag{4.18}$$

$$p(Z_2^2) = p(1) \cdot p(0) = x \cdot (1 - x) \tag{4.19}$$

$$p(Z_2^3) = p(1) \cdot p(1) \cdot p(0) = x \cdot x \cdot (1 - x) \tag{4.20}$$

Insgesamt ergibt sich die folgende Wahrscheinlichkeit, dass eine Transaktion in Anhängigkeit von der Paketverlustwahrscheinlichkeit x zuverlässig ist:

$$R_{T_\ell}(x) = 1 - x + x \cdot (1 - x) + x \cdot x \cdot (1 - x) = 1 - x^3 \tag{4.21}$$

Mit der ermittelten Wahrscheinlichkeit einer zuverlässigen Transaktion und der Anzahl von Transaktionen pro Stunde, kann nun die Zuverlässigkeit pro Stunde ermittelt werden.

Zudem können die Mutationen aus $\mathcal{Z}_{RQ}$ für weitere Analysen herangezogen werden. Hierzu gehört zum Beispiel die genaue Analyse von Zeitbedingungen oder ein Abschätzen des Bandbreitenbedarfs in Abhängigkeit von Fehlerwahrscheinlichkeiten. Des Weiteren erlauben mehrere Zuverlässigkeitsanalysen einer Architektur einen Vergleich bei verschiedenen Parametern und Einstellungsmöglichkeiten von Protokollen.

4.6.3 Verteiltes Model Checking mit Modell Mutationen

Mit jedem Unroll-Step $Z^d \twoheadrightarrow \mathcal{Z}^{d+1}$ entsteht eine Menge von Trace-Tupeln, womit dementsprechend mehrere Modellmutationen zur Verifikation ausstehen. Es liegt daher nahe, Modellmutationen mit parallelen Model Checking Prozessen zu verifizieren. Neben einem kleineren Zustandsraum jeder Mutation gegenüber dem nichtdeterministischen Modell kann damit die Leistung von Multicore-Systemen und Rechner-Pools genutzt werden. Klassisches paralleles Model Checking setz für einen Performance-Gewinn durch einen Rechner-Pool zwei Eigenschaften voraus: Ausgewogene Verteilung von Zuständen auf die Computer-Knoten und wenig Überschneidungen von Pfaden im Computation-Tree zwischen den Knoten [PD03]. Mit diesen Voraussetzungen kann die Parallelität von Rechen-Pools optimal genutzt werden, ohne den Verwaltungs-Overhead zu groß werden zu lassen. Algorithmen zum Balancieren und Verteilen des Zustandsraums sind Gegenstand aktueller Forschungen.

Mit dem parallelen Verifizieren von Modellmutationen ist kein gemeinsamer Zustandsraum auf alle Prozesse verteilt, sondern jede Mutation wird unabhängig von anderen Mutationen geprüft. Der Unroll- Algorithmus ist ebenfalls auf alle Model Checking Prozesse (Clients) verteilt, so dass jeder Client von einer Start-Mutation den resultierenden Mutationsbaum bis zu einer festgelegten Größe verifiziert. Ist diese erreicht, sendet der Client die Ergebnisse und die nicht geprüften Mutationen zu einem Server-Prozess, der die Verwaltung übernimmt. Eine Kommunikation ist nur zwischen dem Server und den Client-Prozessen nötig, aber nicht zwischen den Clients. Je Mutation ergibt sich aus dem Basismodell und einem Trace-Tupel. Das Basismodell kann initial auf alle Clients verteilt werden, wodurch während des Verifikationsprozesses nur Trace-Tupel zwischen dem Server und den Clients übertragen werden müssen. Diese Informationen sind wenige Bytes groß und verursachen nur geringen Kommunikations-Overhead.

Von Vorteil ist die parallele Verifikation von Mutationen, wenn mit jedem Unroll-Step mehrere Kind-Mutationen generiert werden. Dieses ist dann der Fall, wenn durch mehrere Fault-Switches oder Auswahlmöglichkeiten eine entsprechende Kombinatorik entsteht.

Definition 4.6.14 (**Maximale Anzahl zu verifizierender Mutationen $\mathcal{M}_{max}$**)**.** Die Anzahl der Mutationen wächst mit jedem Unroll-Step um die Anzahl wählbarer Tran-

sitionen der Fault-Switches $fs_1, \ldots, fs_k$ im Modell. Die Anzahl der Unroll-Steps hängt von der Länge der Traces ab, bis entweder RQ oder SSS erfüllt ist. Für die maximale Anzahl von Mutationen die verifiziert werden müssen, lässt sich eine obere Grenze bestimmen. Sind maximal d_{max} viele Transitionen möglich (entspricht der maximalen Dimension der Traces), bis RQ oder SSS erfüllt ist, dann ist die Gesamtzahl der zu verifizierenden Mutationen:

$$\mathcal{M}_{max} \leq \sum_{j=1}^{d_{max}} (\prod_{i=1}^{k} |\mathcal{FS}_i|)^j \tag{4.22}$$

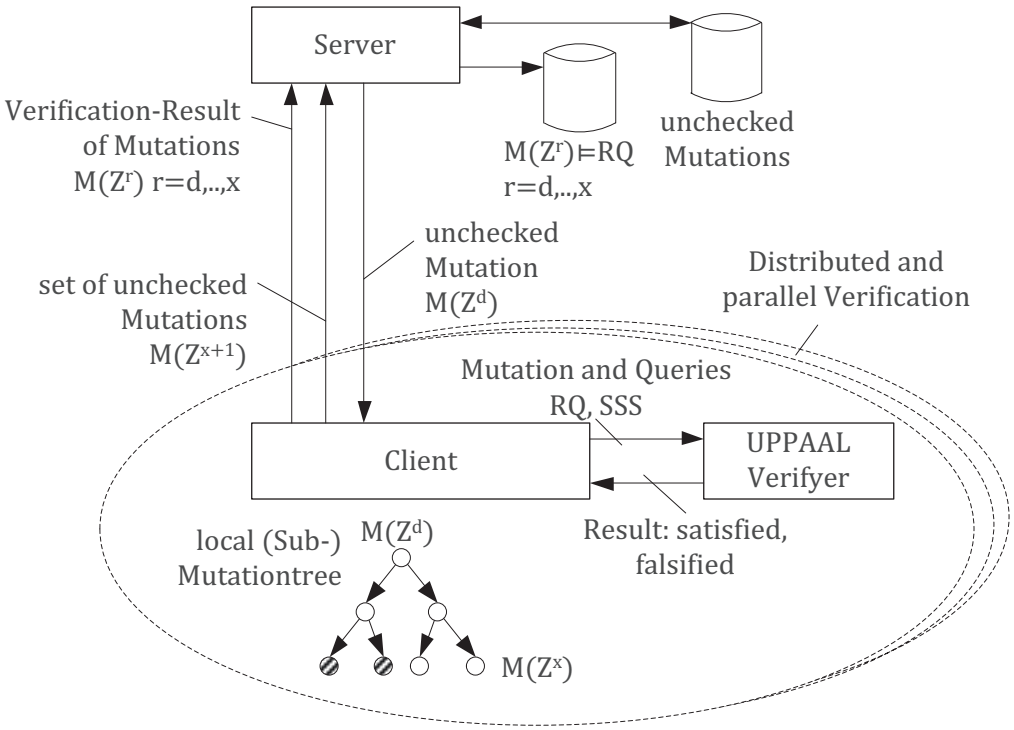


Abbildung 4.14: Verteiltes Model Checking von Mutationen

4.6.4 Auswerten der Ergebnisse

Ist die Analyse einer Kommunikationsarchitektur nach der Phase 3 abgeschlossen, so ist das Ergebnis der Zuverlässigkeit des logischen Kanals eine Zuverlässigkeitsformel, die von den jeweiligen Transitionswahrscheinlichkeiten $p(e)$ abhängig ist. Die Fault-Switches modellieren Fehlerarten eines Kommunikationskanals. Die ermittelte Worst-Case Zuverlässigkeitsformel kann verwendet werden, um in Abhängigkeit von Fehlerwahrscheinlichkeiten der Übertragung, Anforderungen an den Kommunikationskanal zu bestimmen.

Im nächsten Kapitel sind Fallstudien aufgeführt, die das Vorgehen und Auswerten von Ergebnissen nach allen drei Phasen des CAMoLa Frameworks anhand realer Kommunikationsarchitekturen beschreiben.

Kapitel 5

Design und Analyse sicherheitsrelevanter Kommunikationsarchitekturen: Fallstudien

DOS addresses only 1 Megabyte of RAM because we cannot imagine any applications needing more.

Microsoft on the development of DOS, 1980

Das vorgestellte Framework CAMoLa wird in diesem Kapitel angewendet, um sicherheitsrelevante Kommunikationsarchitekturen zu modellieren, qualitative Eigenschaften der Systemmodelle formal zu verifizieren und Zuverlässigkeitseigenschaften zu analysieren. Bestandteil der Fallstudien sind Kommunikationsprotokolle, die im Eisenbahnumfeld eingesetzt werden und somit eine angemessene Grundlage zur Evaluation des CAMoLa Frameworks darstellen. Zunächst wird das Modell des Safe Highly Available and Redundant (SAHARA) Safety-Protokolls analysiert. Die zweite Fallstudie stellt ein neues Safety-Protokoll vor, welches ein hybrides Zuverlässigkeitsverfahren beinhaltet. Die dritte Studie befasst sich mit dem in Kapitel 2.5.1 auf Seite 30 vorgestellten Protocol SCTP. Dabei wird das SAHARA Protokoll anstelle von UDP über SCTP transportiert.

Die Architekturmodelle der Fallstudien sind im Anhang B auf Seite 131 nachzuschlagen. Die Modelle der Prozesse der Architekturen sind im Anhang C auf Seite 137 zu finden.

5.1 Das SAHARA Protokoll

Das SAHARA Protokoll ist ein Kommunikationsprotokoll für den einkanalig sicheren Datenaustausch zwischen sicheren Stellwerksrechnern. Das SAHARA Protokoll erfüllt SIL 4 Anforderungen. Die Firmen Siemens und Thales bemühen sich derzeit, dieses gemeinsame Protokoll für den Bahnbereich zu standardisieren.

Das SAHARA Protokoll ist ein verbindungsorientiertes Protokoll mit Sicherheits- und Zuverlässigkeitsmechanismen. Zudem beinhaltet es eine Redundanzschicht, die eine logische Verbindung über bis zu drei physikalische Schnittstellen aufbaut. Die Redundanzschicht arbeitet nach dem Hot-Standby Prinzip und dient ausschließlich zur Steigerung der Zuverlässigkeit des logischen Kanals. Das SAHARA Protokoll ist empirisch und ohne die Unterstützung von formalen Modellierungs- und Verifikationstechniken spezifiziert worden. Während der Implementierungsphase führten Spezifikationsunschärfen zu Freiheitsgraden. Aus diesem Grund ist die Spezifikation in dieser Arbeit mit dem CAMoLa Framework verifiziert und analysiert worden, um den derzeitigen Standardisierungsprozess zu unterstützen. Schwerpunkt ist dabei die Verifikation von Safety-Eigenschaften und die Analyse der Zuverlässigkeit des logischen Übertragungskanals. Zum Zeitpunkt der Erstellung dieser Dissertation unterliegt das SAHARA Protokoll firmeninternen Regeln, welche die Veröffentlichung der Protokollspezifikation in der Version 2.2 verbieten. Eine detaillierte Funktionsbeschreibung ist daher in dem nichtöffentlichen Anhang D dieser Dissertation platziert.

5.1.1 Funktionsweise von SAHARA

Das SAHARA Protokoll verwendet gemäß der EN 50159 für die Safety Prüfung eine Empfänger- und Absenderkennung, Sequenznummern, Zeitstempel und einen Safety-Code (Prüfsumme). Die unterlagerten Kommunikationsschichten, in einem nach SIL 0 eingestuften Kommunikationsmodul, sind UDP, IP und Ethernet. Aus diesem Grund ist ein Fehlerkontroll-Mechanismus im Safety-Protokoll nötig, der die Zuverlässigkeit des logischen Kanals steigert. Dieser arbeitet nach dem NAK-Prinzip (siehe Kapitel 2.4.2 auf Seite 25) mit einer Go-Back-N Wiederholung im Fehlerfall. Die Anzahl von Wiederholungen ist durch das Alter von Daten, die mit einem Zeitstempel versehen sind, begrenzt. Übertragungsfehler, die zu einem Paketverlust führen, werden durch eine Lücke in den Sequenznummern vom Empfänger erkannt und führen zu einer Wiederholungsanfrage. Jede Verbindung wird mit Heartbeat-Nachrichten überwacht, um mit dem permanenten Strom von Sequenznummern Übertragungsfehler aufzudecken.

5.1.2 Modellierung und Formalisierung von Anforderungen

Basis für das Modell des SAHARA Protokolls ist die Spezifikation v2.2 [GBK06]. Die Verifikation und Analyse des SAHARA Protokolls fokussiert die Datenübertragung und den Fehlerkontroll-Mechanismus. Die Verbindungsauf- und Abbau-Dienste sowie die Redundanzschicht werden nicht betrachtet und sind daher nicht Teil des Modells. Ein Applikationsmodell triggert in zyklischen Abständen den Datenübertragungsdienst. Die Verletzung von Safety-Anforderungen (Fehler in der Datensequenz) werden auf Applikationsebene durch Transaktionsnummern Tn erkannt. Des Weiteren überwachen Watchdog-Prozesse anhand der Transaktionsnummer das Alter der übertragenen Informationen, um Echtzeitanforderungen zu verifizieren.

Die Spezifikation von SAHARA definiert eine Zustandsübergangsmatrix (im nicht-öffentlichen Anhang D nachzuschlagen), in der die Reaktionen auf Nachrichtenereignisse in den jeweiligen Zuständen definiert sind. Die Abbildung 5.1 auf der nächsten Seite zeigt eine vereinfachte Form der Zustandsübergangsmatrix als Zustandsmaschine. Die Zustandsübergangsmatrix ist die Basis für das Modell zur Verifikation von Safety- und Zuverlässigkeitseigenschaften. Während der Modellierung zeigten sich Schwächen in der Spezifikation: Nicht definierte aber nötige Funktionen konnten erst mit einem gewachsenen Systemverständnis sinnvoll ergänzt werden. Vor diesen Lücken steht auch ein Systemingenieur, der die Spezifikation in Form einer Implementierung umsetzt. Es trägt damit wesentlich zur Qualität der Spezifikation und zur späteren Implementierung bei, wenn mit der modellbasierte Analyse zu einem frühen Zeitpunkt das Systemverhalten anschaulich dargestellt wird und "unerwartete" Eigenschaften mit einem Ausführungspfad belegt werden.

In Abbildung 5.2 auf der nächsten Seite ist die Architektur skizziert, die im Folgenden beschrieben ist. Die Abbildung B.1 auf Seite 132 zeigt das Systemmodell dieser Skizze und im nicht-öffentlichen Anhang D ist das Modell des SAHARA Protokolls dargestellt. Zu dem Systemmodell gehören für Hin- und Rückrichtung zwei Übertragungskanäle (*ChannelSC* und *ChannelCS*), die jeweils das Kanalmodell der Abbildung 4.9 auf Seite 76 instanziieren. Diese Kanäle modellieren verschiedene Fehlerarten, die bei Datenaustausch nach der EN 50159 auftreten können. Im Systemmodell existieren demnach zwei Fault-Switches: Der Prozess *ChannelSC* enthält den Fault-Switch fs_1 und der Prozess *ChannelCS* den Fault-Switch fs_2.

5.1.3 Abstraktionen und Fehlerhypothesen

Abstraktionen sind bei der SAHARA Architektur nötig, um das Problem der Zustandsraumexplosion zu vermeiden. Es sind zwei Arten von Abstraktionen zu unterscheiden: Eine *modellbasierte Abstraktion* und eine *parameterbasierte Abstraktion*. Bei der modell-

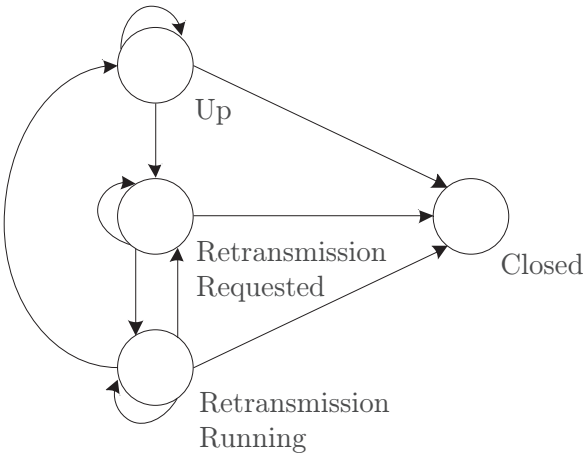


Abbildung 5.1: Vereinfachte Darstellung der SAHARA Zustandsmaschine

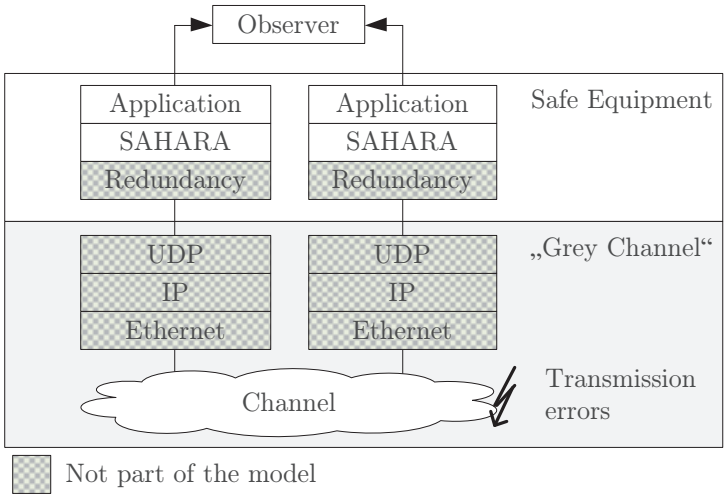


Abbildung 5.2: Skizze der SAHARA Architektur

basierten Abstraktion wird das Modell auf einem höheren Abstraktionsniveau dargestellt, um nur von bestimmten Teilen der Systemspezifikation Eigenschaften nachzuweisen. Die parameterbasierte Abstraktion bietet die Möglichkeit, durch Veränderungen einzelner Parameterwerte die Komplexität der Verifikation zu beeinflussen. Hierzu gehört zum Beispiel die Fehlerhypothese, mit welcher die Fehlerkombinatorik einer Übertragung reduziert und die Häufigkeit, mit der ein Übertragungsdienst benutzt wird. Diese Parameter können bei verschiedenen Verifikationsdurchläufen variiert werden, um einerseits ein möglichst lan-

ges Verifikationsintervall des Systems zu betrachten (ideal ist ein reaktives Systemmodell mit einem unendlich langen Analyseintervall) und andererseits mit einer weniger restriktiven Fehlerhypothese eine möglichst große Fehlerkombinatorik zu erfassen (ideal ist keine Beschränkung in der Fehlerart und Fehlerhäufigkeit). Sofern Aufgrund der Zustandsraumexplosion die ideale Analyse nicht möglich ist, wird mit der Veränderung der Parameter die Komplexität eingeschränkt. Für die Analyse wird nur der Datenübertragungsdienst von SAHARA modelliert. Der Verbindungsauf- und -Abbau sowie eine vorhandene Redundanzschicht wird nicht modelliert. Mit dem Hot-Standby Verfahren der Redundanzschicht werden Nachrichten auf mehreren physikalischen Kanälen übertragen und empfängerseitig werden überflüssige Kopien der Daten wieder entfernt. Die Redundanzschicht ist derart definiert, dass die Fehlerarten *Repetition* und *Insertion*, *Resequence* und *Corruption* (siehe Tabelle 2.2 auf Seite 16) zu einem Nachrichtenverlust werden, bevor der Safety-Check ausgeführt wird. Zur Verifikation der Safety nach EN 50159-2 werden daher nur die Fehlerarten *Deletion* und *Delay* für die SAHARA Architektur betrachtet. Im nicht-öffentlichen Anhang D sind zu den hier getätigten Aussagen detailliertere Beschreibungen nachzuschlagen.

Insgesamt wird das Architekturmodell anhand verschiedener Szenarien verifiziert, wobei hier exemplarisch die Szenarien SZ_1, SZ_2, SZ_3 und SZ_4 mit jeweils verschiedenen Abstraktionsparamerter beschrieben sind.

Szenario SZ_1.

Das erste Szenario dient zum Überprüfen der Modellkorrektheit ohne Übertragungsfehler. Der Parameter $num = -1$ der Applikationsprozesse *SApp* und *CApp* versetzen das Modell in ein reaktives System. Es werden unbegrenzt viele Aufrufe des Übertragungsdienstes von SAHARA in zyklischen Abständen getätigt. Die Fehlerhypothese erlaubt ausschließlich die korrekte Übertragung von Nachrichten, was mit der Fehlerhypothese fh_1 von fs_1 (und fh_2 von fs_2) in den generischen Übertragungskanälen[1] definiert ist. Somit gelten für SZ_1 folgende Abstraktionsparamerter:

$$num = -1$$
$$fh_1 : b_0 = \infty, b_1 = \cdots = b_6 = 0$$
$$fh_2 : b_0 = \infty, b_1 = \cdots = b_6 = 0$$

[1] Jeweils für Hin- und Rückrichtung.

Szenario SZ_2.

Das zweite Szenario erlaubt den Verlust genau einer Nachricht und prüft damit, ob der Wiederholungsmechanismus von SAHARA korrekt spezifiziert und modelliert ist. Ist der Verlust einer Nachricht vom Client zum Server erlaubt, dann zwingt dieses den Server dazu, über *RetransmissionRequested* der Zustandsmaschine, die Nachricht erneut anzufordern. Die Anzahl der Dienstaufrufe ist aus Komplexitätsgründen begrenzt.

$$num = 32$$
$$fh_1 : b_0 = \infty, b_1 = \cdots = b_6 = 0$$
$$fh_2 : b_0 = \infty, b_1 = 1, b_2 = \cdots = b_6 = 0$$

Szenario SZ_3.

Mit dem dritten Szenario wird anhand einer weniger restriktiven Fehlerhypothese die Einhaltung der Safety-Bedingungen überprüft. Es ist der Verlust von drei Nachrichten in jede Richtung möglich. Aufgrund der hohen Komplexität muss dabei die Anzahl von Dienstaufrufen mit dem Parameter num eingeschränkt werden. Insgesamt gelten für dieses Szenario folgende Parameter:

$$num = 2$$
$$fh_1 : b_0 = \infty, b_1 = 3, b_2 = \cdots = b_6 = 0$$
$$fh_2 : b_0 = \infty, b_1 = 3, b_2 = \cdots = b_6 = 0$$

Szenario SZ_4.

Das vierte Szenario verifiziert die Echtzeiteigenschaften des Architekturmodells. Die Übertragungskanäle besitzen eine auf $DELAY = 250\ ms$ festgelegte Übertragungsdauer[2]. Zudem erlaubt die Fehlerhypothese den Verlust einer Nachricht und die Verzögerung von zwei Nachrichten:

$$num = 3$$
$$fh_1 : b_0 = \infty, b_1 = 1, b_2 = 2, b_3 = \cdots = b_6 = 0$$
$$fh_2 : b_0 = \infty, b_1 = 1, b_2 = 2, b_3 = \cdots = b_6 = 0$$

[2] Es reduzieren sich Interleaving-Effekte, wenn eine minimal unterschiedliche Übertragungszeit der Kanäle festgelegt wird. Hierdurch verringert sich die Model Checking Dauer und der Zustandsraum bemerkbar. Daher sind die Delay-Zeiten der Übertragungskanäle im Modell minimal unterschiedlich gewählt: 250 ms und 249 ms.

5.1.4 Ergebnisse der Phase 2

In der Tabelle 5.1 auf der nächsten Seite sind Anforderungen an die Kommunikationsarchitektur aus den drei Klassen Safety-, Reliability und Reachability formalisiert und mit dem UPPAAL Model Checker anhand der beschriebenen Szenarien verifiziert. Informal beschreiben die aufgelisteten Queries folgende Eigenschaften:

1. Das Systemmodell gerät niemals in einen Deadlock.
2. Für alle modellierten Fifo-Queues gilt, dass niemals das Overflow- oder Underflow-Flag gesetzt wird.
3. Die Applikationsmodelle zeigen niemals die Verletzung von Safety-Eigenschaften an (Auslieferung von Daten in fehlerhafter Reihenfolge).
4. Für alle Systembeobachter gilt, dass niemals die Echtzeitanforderungen verletzt werden (jeder Systembeobachter überwacht eine Transaktion Tn).
5. Für alle Systembeobachter gilt, dass schließlich einmal eine Transaktion gestartet wird.
6. Für alle Systembeobachter gilt, dass eine gestartete Transaktion schließlich einmal erfolgreich beendet wird.
7. Es existiert ein Ausführungspfad in den sicheren Zustand (stable-safe-state).
8. Es existiert ein Ausführungspfad in den Zustand *Retransmission Requested* des SAHARA Servers.
9. Immer wenn ein Ausführungspfad in den Zustand *Retransmission Requested* existiert, gelangt der Server schließlich einmal in den Zustand *Retransmission Running*.
10. Immer wenn ein Ausführungspfad in den Zustand *Retransmission Running* existiert, gelangt der Server schließlich einmal zum Zustand *Up*.

Die Fehlerkontrolle von SAHARA ist nur in der Lage, Datennachrichten zu wiederholen. Sofern eine Steuernachricht (Retransmission-Request, Retransmission-Response) oder ein Lebenstakt (Heartbeat) bei der Übertragung infolge von Fehlern verworfen wird, leitet der Sicherheitsmechanismus von SAHARA den sicheren Zustand ein. SAHARA gewährleistet die Safety, jedoch ist die vorliegende Spezifikation nicht robust gegenüber dem Verlust der Steuernachrichten.

Tabelle 5.1: Verifikation der SAHARA-Architektur

Nr.	TCTL Query Φ	Ergebnis			
		SZ_1	SZ_2	SZ_3	SZ_4
1.	$A\Box\neg deadlock$	$\surd$	$\surd$	$\surd$	$\surd$
2.	$\forall$ Fifo-Queues: $A\Box\neg(ufFlag \vee ofFlag)$	$\surd$	$\surd$	$\surd$	$\surd$
3.	$A\Box\neg(SApp.safetyFail \vee CApp.safetyFail)$	$\surd$	$\surd$	$\surd$	$\surd$
4.	$\forall i \in Tn : A\Box\neg Watchdog(i).timeError$	$\surd$	$\surd$	$\surd$	$\surd$
5.	$\forall i \in Tn : E\Diamond Watchdog(i).start$	$\surd$	$\surd$	$\otimes$	$\otimes$
6.	$\forall i \in Tn : Watchdog(i).start \rightsquigarrow Watchdog.idle$	$\surd$	$\surd$	$\otimes$	$\otimes$
7.	$E\Diamond SApp.stableSafe \vee CApp.stableSafe$	$\otimes$	$\otimes$	$\surd$	$\surd$
8.	$E\Diamond Server.retRequested$	$\otimes$	$\surd$	$\surd$	$\surd$
9.	$Server.retRequested \rightsquigarrow Server.retRun$	$\surd$	$\surd$	$\otimes$	$\otimes$
10.	$Server.retRun \rightsquigarrow Server.up$	$\surd$	$\surd$	$\otimes$	$\otimes$

$\surd$ $M_{SZ} \models \Phi$
$\otimes$ $M_{SZ} \not\models \Phi$

5.1.5 Analyse und Ergebnisse der Phase 3

Nach dem mit der Phase 2 grundsätzliche Eigenschaften der Architektur nachgewiesen sind, ermittelt die Analyse der Phase 3 die Zuverlässigkeit des logischen Kanals in Abhängigkeit von Übertragungsfehlern. Es werden Übertragungskombinationen ermittelt, die zur erfolgreichen Ausführung von Transaktionen (Übertragen von Daten) führen. Übertragungsfehler, die wie in dem beschriebenen Beispiel zum Abbruch der Kommunikation führen, liefern keinen Beitrag zur Zuverlässigkeit. Der häufigste Fehler ist der Verlust von Informationen, so dass in diesem Schritt die Qualität des Wiederholungsmechanismus als Zuverlässigkeitsformel in Abhängigkeit von Paketverlusten quantifiziert wird. Hierzu wird ein Stable-Safe-Kriterium (SSS) und ein Zuverlässigkeitskriterium (RQ) festgelegt. Eine Transaktion gilt als zuverlässig, wenn Informationen korrekt zum Kommunikationspartner gelangt sind. Es ist RQ definiert als ein Systemzustand, bei dem die jeweils erste Transaktion (in Hin- und Rückrichtung) erfolgreich ausgeführt wird. Der Stable-Safe-State ist ein Zustand, bei dem das Safety-Protokoll den Übertragungskanal geschlossen hat. Dieser wird eingenommen, falls Übertragungsfehler nicht mehr korrigiert oder Echtzeiteigenschaften nicht mehr eingehalten werden können. Ist die Eigenschaft SSS oder RQ von einer Mutation erfüllt, stoppt das Abrollen der Traces. Zur Verifikation der Mutationen werden insgesamt 35 Model Checking Prozesse eingesetzt, die auf 18 Rechner mit jeweils 2,80 GHz Multicore-CPUs verteilt sind.

Mit den in der Tabelle 5.2 festgelegten Parametern ermittelt der Mutationsgenerator die Mutationen, die das Zuverlässigkeitskriterium RQ erfüllen. Aus den zuverlässigen Mutationen der Analyse beziehungsweise ihrer Traces, ergibt sich die Zuverlässigkeitsformel 5.2 auf der nächsten Seite. Die Variable x steht für die Wahrscheinlichkeit, eine Nachricht zu verwerfen beziehungsweise die Transition 1 des Fault-Switches in den Übertragungskanälen zu wählen. Analog hierzu ist $1 - x$ die Wahrscheinlichkeit, eine Nachricht korrekt zu übertragen, beziehungsweise die Transition 0 im Kanalmodell (Abbildung 4.9 auf Seite 76) zu wählen.

Die Zuverlässigkeitsformel 5.2 auf der nächsten Seite gilt für jeweils eine Transaktion in Hin- und Rückrichtung. Die Abbildung 5.3 auf der nächsten Seite zeigt die Formel $R_{RQ\_SAHARA}(x)$ als Diagramm. Mit den festgelegten Parametern können in einem Zeitintervall von einer Stunde $\frac{3600\ s}{300\ ms} = 12000$ Transaktions-Paare über einen Kommunikationskanal übertragen werden. Nach der Definition 4.6.13 auf Seite 91 berechnet sich die Ausfallwahrscheinlichkeit PD pro Stunde mit:

$$PD(x) = 1 - R_{RQ}(x)^{12000} \tag{5.1}$$

Die Funktion $PD(x)$ beschreibt die Ausfallwahrscheinlichkeit pro Stunde einer Client Server Beziehung mit dem SAHARA Protokoll und ist in Abhängigkeit von der Paketverlustwahrscheinlichkeit in Abbildung 5.4 auf Seite 107 dargestellt.

$$\begin{aligned}
R_{RQ_SAHARA}(x) =& (1-x)^2 + 2\cdot(1-x)^{11}\cdot x + 10\cdot(1-x)^{10}\cdot x^2 + \\
& 2\cdot(1-x)^{14}\cdot x^2 + 2\cdot(1-x)^2 2\cdot x^2 + 20\cdot(1-x)^9\cdot x^3 + \\
& 20\cdot(1-x)^{11}\cdot x^3 + 14\cdot(1-x)^{13}\cdot x^3 + 4\cdot(1-x)^{15}\cdot x^3 + \\
& 28\cdot(1-x)^{21}\cdot x^3 + 4\cdot(1-x)^{23}\cdot x^3 + 20\cdot(1-x)^8\cdot x^4 + \\
& 54\cdot(1-x)^{10}\cdot x^4 + 38\cdot(1-x)^{12}\cdot x^4 + 39\cdot(1-x)^{14}\cdot x^4 + \\
& 152\cdot(1-x)^{20}\cdot x^4 + 52\cdot(1-x)^{22}\cdot x^4 + 5\cdot(1-x)^{24}\cdot x^4 + \\
& 10\cdot(1-x)^7\cdot x^5 + 70\cdot(1-x)^9\cdot x^5 + 52\cdot(1-x)^{11}\cdot x^5 + \\
& 136\cdot(1-x)^{13}\cdot x^5 + 4\cdot(1-x)^{15}\cdot x^5 + 448\cdot(1-x)^{19}\cdot x^5 + \\
& 232\cdot(1-x)^{21}\cdot x^5 + 75\cdot(1-x)^{23}\cdot x^5 + 2\cdot(1-x)^6\cdot x^6 + \\
& 41\cdot(1-x)^8\cdot x^6 + 30\cdot(1-x)^{10}\cdot x^6 + 191\cdot(1-x)^{12}\cdot x^6 + \\
& 16\cdot(1-x)^{14}\cdot x^6 + 672\cdot(1-x)^{18}\cdot x^6 + 456\cdot(1-x)^{20}\cdot x^6 + \\
& 262\cdot(1-x)^{22}\cdot x^6 + 8\cdot(1-x)^{24}\cdot x^6 + 3\cdot(1-x)^{12}\cdot x^2
\end{aligned} \tag{5.2}$$

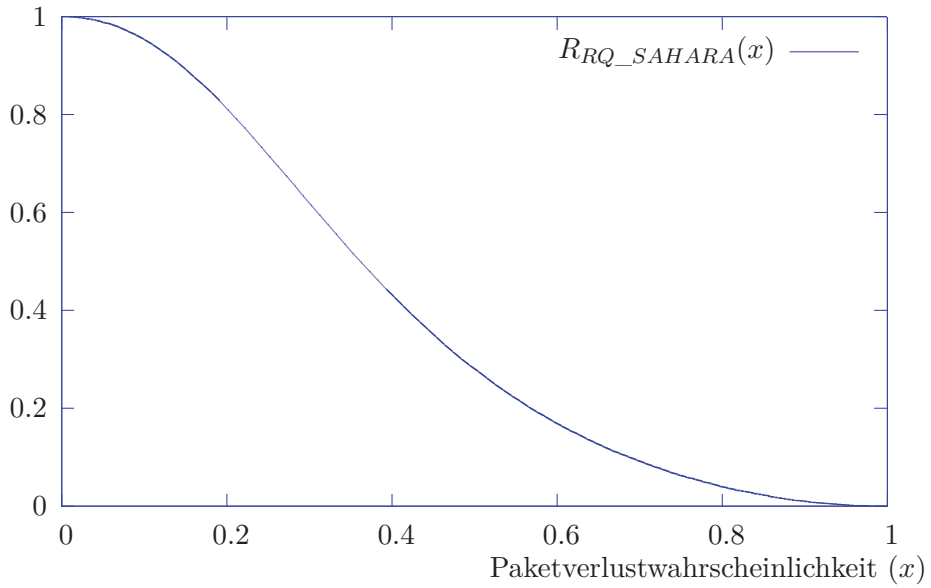


Abbildung 5.3: Ermittelte Zuverlässigkeitsfunktion von SAHARA

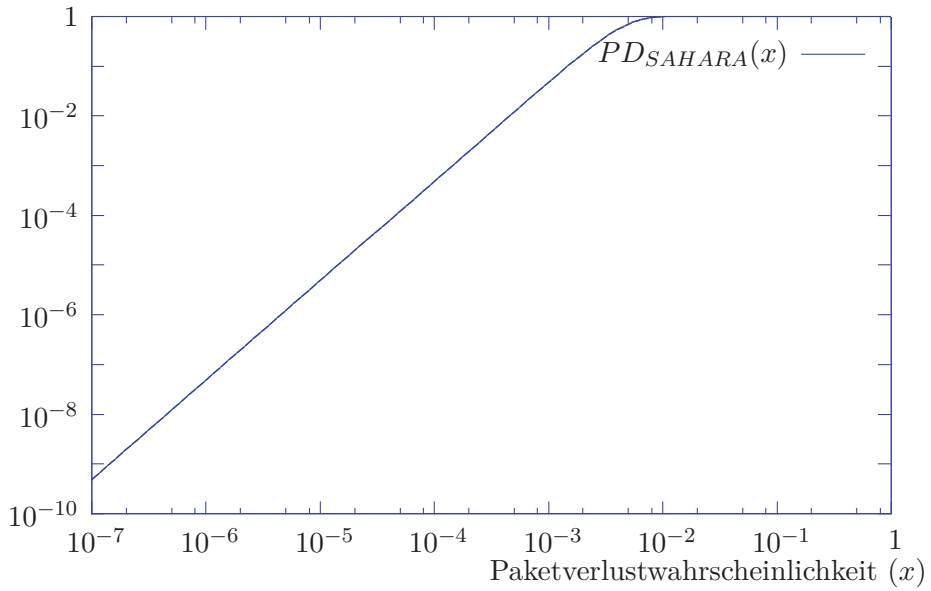


Abbildung 5.4: Ausfallwahrscheinlichkeit pro Stunde der SAHARA Kommunikation

Tabelle 5.2: Parameter der quantitativen Analyse

Eigenschaft	Parameter
Zyklisches Senden von Daten	300 ms
Anzahl der Transaktionen in $SApp$ und $CApp$ ist unbegrenzt	$num = -1$
Verzögerung durch den Übertragungskanal	$DELAY = 80\ ms$
Fehlerhypothese (beider Kanalmodelle): unbegrenzt korrekte Übertragung, maximal drei Paketverluste, keine weiteren Fehlerarten erlaubt	$b_0 = \infty$ $b_1 = 3$ $b_2 = \cdots = b_6 = 0$
Stable-Safe Kriterium SSS	$E\Diamond(SApp.stabelSafe \vee CApp.stableSafe)$
Zuverlässigkeitskriterium RQ: Erfolgreiche Transaktion $Tn = 0$ und $Tn = 1$ (jeweils die erste in Hin- und Rückrichtung)	$A\Diamond(Watchdog(0).rcv == true \wedge$ $Watchdog(1).rcv == true)$

5.1.6 Komplexität der Verifikation

Die Phase 1 und Phase 2 werden nach Abbildung 4.5 auf Seite 67 in Interationschritten durchgeführt, bis alle Anforderungen in erwarteter Weise erfüllt werden und Modellierungsfehler damit ausgeschlossen werden können. Dieser Prozess betrug mit dem CAMoLa Framework etwa eine Personenwoche bei der vorgestellten SAHARA Architektur. Die Tabelle 5.3 auf der nächsten Seite zeigt die Komplexität des Model Checkings der verschiedenen Szenarien. Die Dauer bezieht sich auf die Verifikation aller in der Tabelle 5.1 auf Seite 104 beschriebenen Eigenschaften. Die Analysekomplexität der Phase 3 profitiert aufgrund des Mutationsansatzes von der parallelen Verifikation mit einem Pool von Rechnern. Der Vergleich zwischen der Analyse mit dem Mutationsgenerator und dem Szenario SZ_3 verdeutlicht die Reduktion der Zustandsraumgröße mit deterministischen Fault-Switches[3]. In der jetzigen 32-Bit Version des Model Checkers kann der Zustandsraum nur bis maximal 4 GB anwachsen, womit das SZ_3 an der physikalischen Grenze liegt.

[3]Die Anzahl der Zustände beim Mutationsgenerator geben die größte Anzahl von Zuständen einer Mutation an. Die Fehlerkombinatorik wirkt sich mit deterministischen Fault-Switches auf die Anzahl von Mutationen wesentlich stärker aus, als auf die Größe des Zustandsraums einer Mutation. Das Problem der Zustandsraumexplosion wird in das Problem der Mutationsexplosion umgewandelt. Diesem Problem kann jedoch mit der möglichen parallelen Verifikation entgegengewirkt werden.

Mit dem Mutationsansatz beträgt die Anzahl der Zustände nur etwa 5%, bei gleicher Fehlerhypothese. Hierbei ist jedoch zu berücksichtigen, dass mit dem Mutationsgenerator die Mutationen (beziehungsweise der Zustandsraum) nur soweit expandiert werden, bis die Bedingung SSS oder RQ erfüllt ist. Die Verletzung von Safety-Eigenschaften, die nach der Eigenschaft SSS oder RQ eintreten, können mit diesem Ansatz nicht entdeckt werden, da die Mutationen nicht weiter abgerollt werden. In Kapitel 6.5.1 auf Seite 124 ist dieser Punkt noch einmal aufgegriffen.

Tabelle 5.3: Komplexität der SAHARA-Architektur

Szenario	Komplexität		
	Zustände	Speicher	Dauer
SZ_1	4.892.482	3531 MB	3 h 8 m
SZ_2	177.230	134 MB	6 m 4 s
SZ_3	7.538.440	3751 MB	4 h 17 m
SZ_4	1.626.154	860 MB	1 h 29 m
Mutationsgenerator	334.909	15895 Mutationen	1 h 57 m

5.2 Das Safety Protocol HASP

Das Hybrid Acknowledge Safety Protocol (HASP) ist eine veränderte Version des SAHARA Protokolls und wird in der vorgestellten Form nicht eingesetzt. Dieses Safety-Protokoll ist mit einem hybriden Zuverlässigkeitsmechanismus ausgestattet, um hohe Zuverlässigkeitsanforderungen bei gleichzeitig geringem Ressourcenbedarf zu kombinieren. Es ist bewusst einfach gehalten, um den Aufwand bei der Implementierung und der Begutachtung gering zu halten. Die Protokolldienste zum Verbindungsauf- und Abbau sind aus Übersichtlichkeitsgründen nicht modelliert. Es wird ausschließlich der Übertragungsdienst zum zuverlässigen Übermitteln von Safety-Nachrichten über einen paketorientierten Kanal analysiert.

Die Bezeichnung Hybrid Acknowledge stammt von dem verwendeten Mechanismus zum Quittieren von Nachrichten. In dem regulären Übertragungsmodus werden Nachrichten wie beim SAHARA Protokoll nach dem NAK-Verfahren quittiert. Wird ein Übertragungsfehler erkannt, dann fordert der Empfänger den Kommunikationspartner dazu auf, ab der fehlerhaften Sequenz die Übertragung erneut zu beginnen (Go-Back-N Verfahren). Diese Aufforderung, deren Verlust beim SAHARA Protokoll zum Verbindungsabbruch führt, wird nach dem ACK-Verfahren übermittelt. Das bedeutet, dass die Go-Back-N Aufforderung nach einer festgelegten Zeit wiederholt wird, bis diese vom Sender quittiert ist und

die Wiederholung beginnt. Hierbei besteht eine Toleranz gegenüber dem Verlust von Steuernachrichten, welches beim SAHARA Protokoll nicht gegeben ist. Die zu übermittelnden Daten werden wie bei dem SAHARA Protokoll mit einem doppelten Zeitstempelverfahen auf die Einhaltung von Echtzeitanforderungen überwacht. Dieses konservative Verfahren kann dazu führen, dass trotz einer Wiederholung die Daten aus Veralterungsgründen nicht akzeptiert werden und der Kommunikationskanal geschlossen wird. Im Gegensatz zu der Übertragung mit dem SAHARA Protokoll profitiert die Zuverlässigkeit dann jedoch von kurzen Überwachungszeiten (Heartbeats) und geringen Verzögerungszeiten des Kommunikationskanals.

Die Abbildung 5.6 auf der nächsten Seite zeigt ein Wiederholungsszenario des HASP mit einer verlorenen Nachricht und die Abbildung 5.7 auf der nächsten Seite zeigt ein Wiederholungsszenario mit einer zusätzlich verlorenen Steuerungsnachricht, was bei dem SAHARA Protokoll zwangsläufig zum Verbindungsabbruch führt.

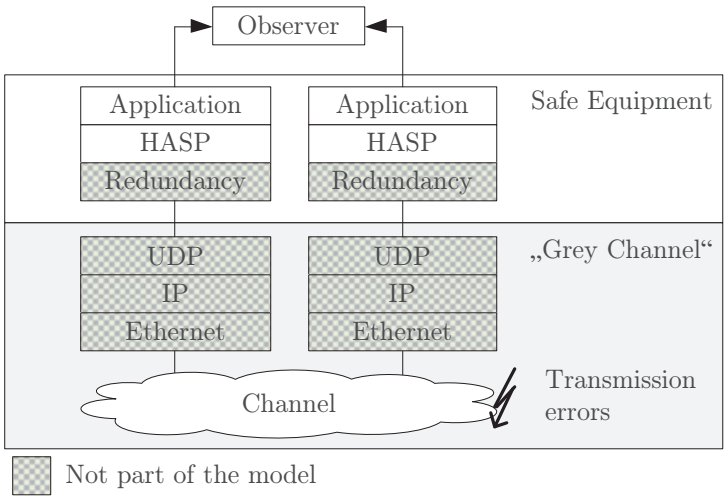


Abbildung 5.5: Skizze der HASP Architektur

Der Zustandsautomat des HASP beseitigt die Schwächen der SAHARA Spezifikation, wobei ebenfalls eine nicht-modellierte Redundanzschicht zu dem Protokoll gehört sowie unterlagerte Protokollschichten wie zum Beispiel UDP, IP und Ethernet (Abbildung 5.5). Demzufolge besteht das Systemmodell des Hybrid Acknowledge Safety Protokolls aus den gleichen Prozessen, wie das SAHARA Systemmodell (Abbildung B.2 auf Seite 133). Die qualitative Analyse von Eigenschaften wird ebenfalls anhand von vier Szenarien SZ_1, SZ_2, SZ_3 und SZ_4 durchgeführt.

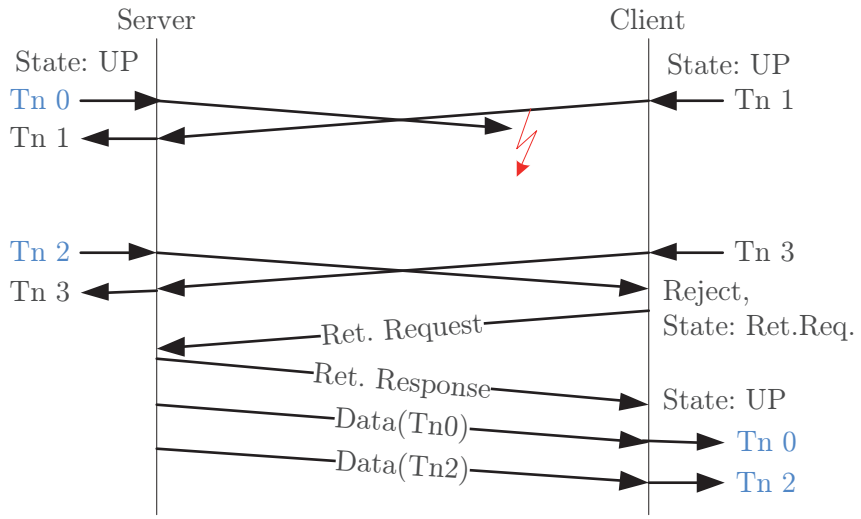


Abbildung 5.6: Erfolgreiche Übertragungswiederholung des HAS Protokolls

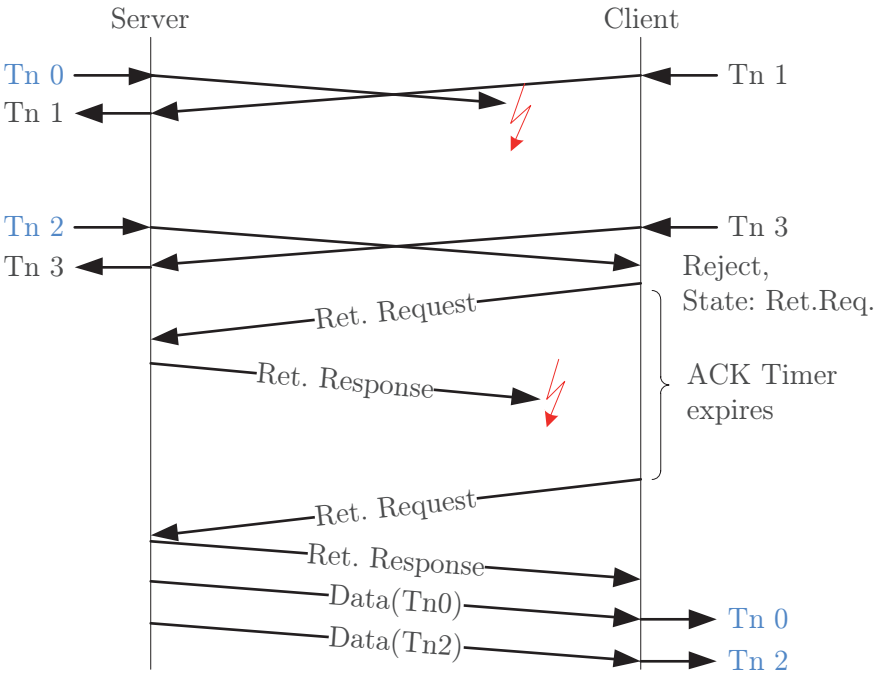


Abbildung 5.7: Übertragungswiederholung mit Verlust von zwei Nachrichten

5.2.1 Ergebnisse der Phase 2

In der Tabelle 5.4 auf Seite 111 sind die gleichen Anforderungen an die Kommunikationsarchitektur mit dem UPPAAL Model Checker anhand der beschriebenen Szenarien verifiziert, wie auch schon bei dem SAHARA Systemmodell. Die Ergebnisse sind ebenfalls identisch und entsprechen den erwarteten Ergebnissen, so dass auch hier von einem korrekten Systemmodell ausgegangen werden kann.

Tabelle 5.4: Verifikation der HASP-Architektur

Nr.	TCTL Query	Ergebnis			
		SZ_1	SZ_2	SZ_3	SZ_4
1.	$A\Box\neg deadlock$	√	√	√	√
2.	∀ Fifo-Queues: $A\Box\neg(ufFlag \vee ofFlag)$	√	√	√	√
3.	$A\Box\neg(SApp.safetyFail \vee CApp.safetyFail)$	√	√	√	√
4.	$\forall i \in Tn : A\Box\neg Watchdog(i).timeError$	√	√	√	√
5.	$\forall i \in Tn : E\Diamond Watchdog(i).start$	√	√	⊗	⊗
6.	$\forall i \in Tn : Watchdog(i).start \rightsquigarrow Watchdog.idle$	√	√	⊗	⊗
7.	$E\Diamond SApp.stableSafe \vee CApp.stableSafe$	⊗	⊗	√	√
8.	$E\Diamond Server.retRequested$	⊗	√	√	√
9.	$Server.retRequested \rightsquigarrow Server.retRun$	√	√	⊗	⊗
10.	$Server.retRun \rightsquigarrow Server.up$	√	√	⊗	⊗

5.2.2 Analyse und Ergebnisse der Phase 3

Die Parameter und die Eigenschaften SSS und RQ sind aus der Tabelle 5.2 auf Seite 105 zu entnehmen. Mit der Beseitigung der Zuverlässigkeitsschwächen, die das SAHARA Protokoll in Wiederholungszenarien aufweist, ist das Ergebnis der Zuverlässigkeitsanalyse von dem HASP gegenüber Paketverlusten besser, wie die Formel 5.3 auf der nächsten Seite belegt. Das Diagramm in Abbildung 5.8 auf Seite 113 zeigt die Zuverlässigkeitsfunktion von HASP in Abhängigkeit von der Paketverlustwahrscheinlichkeit x. Das Diagramm in Abbildung 5.9 auf Seite 113 vergleicht die Ausfallwahrscheinlichkeiten $PD_{SAHARA}(x)$ und $PD_{HASP}(x)$ pro Stunde für eine bidirektionale Kommunikation. Hierbei ist zu sehen, dass mit dem Korrigieren der Zuverlässigkeitsschwächen die Zuverlässigkeit deutlich gesteigert wird. Insgesamt sind 25813 Mutationen verifiziert worden, wovon 3343 Mutationen das Zuverlässigkeitskriterium RQ erfüllen[4]. In Tabelle 5.5 auf der nächsten Seite sind weitere

[4] Anhand der Zuverlässigkeitsformel lassen sich zuverlässige Mutationen erkennen: Jeder Term besitzt die Form $a \cdot (1-x)^b \cdot x^c$, wobei b die Anzahl von korrekten Übertragungen und c die Anzahl von fehlerhaften

Angaben zur Komplexität aufgelistet.

Die Verbesserung von SAHARA mit einem durch das HASP demonstrierten Quittierungs- und Wiederholungsmechanismus hat aus praktischer Sicht einen entscheidenden Nachteil: Die Neu-Spezifizierung eines Safety-Protokolls beziehungsweise der Zustandsmaschine bedeutet, dass ein aufwendiger Prozess zur Begutachtung und Implementierung sowie umfangreiche Tests durchgeführt werden müssen. An dieser Stelle bietet sich eine weitere Option, die zur Zuverlässigkeitssteigerung beitragen kann und gleichzeitig mit weniger Aufwand verbunden ist: Das Profitieren von zuverlässigen Übertragungsprotokollen in der nicht-sicheren Schicht. Anstelle von UDP kann zum Beispiel das zuverlässige Protokoll SCTP eingesetzt werden, was mit der nächsten Fallstudie analysiert wird.

$$
\begin{aligned}
R_{RQ\_HASP}(x) =& (1-x)^2 + 2\cdot(1-x)^9\cdot x + 8\cdot(1-x)^8\cdot x^2 + (1-x)^{10}\cdot x^2 + \\
& 4\cdot(1-x)^{12}\cdot x^2 + 4\cdot(1-x)^{18}\cdot x^2 + 12\cdot(1-x)^7\cdot x^3 + \\
& 2\cdot(1-x)^9\cdot x^3 + 24\cdot(1-x)^{11}\cdot x^3 + 8\cdot(1-x)^{13}\cdot x^3 + \\
& 44\cdot(1-x)^{17}\cdot x^3 + 6\cdot(1-x)^{19}\cdot x^3 + 2\cdot(1-x)^{25}\cdot x^3 + \\
& 8\cdot(1-x)^6\cdot x^4 + (1-x)^8\cdot x^4 + 62\cdot(1-x)^{10}\cdot x^4 + \\
& 45\cdot(1-x)^{12}\cdot x^4 + 4\cdot(1-x)^{14}\cdot x^4 + 190\cdot(1-x)^{16}\cdot x^4 + \\
& 67\cdot(1-x)^{18}\cdot x^4 + 10\cdot(1-x)^{20}\cdot x^4 + 18\cdot(1-x)^{24}\cdot x^4 + \\
& 4\cdot(1-x)^{26}\cdot x^4 + 2\cdot(1-x)^5\cdot x^5 + 84\cdot(1-x)^9\cdot x^5 + \\
& 100\cdot(1-x)^{11}\cdot x^5 + 21\cdot(1-x)^{13}\cdot x^5 + 450\cdot(1-x)^{15}\cdot x^5 + \\
& 265\cdot(1-x)^{17}\cdot x^5 + 80\cdot(1-x)^{19}\cdot x^5 + 2\cdot(1-x)^{21}\cdot x^5 + \\
& 74\cdot(1-x)^{23}\cdot x^5 + 34\cdot(1-x)^{25}\cdot x^5 + 6\cdot(1-x)^{27}\cdot x^5 + \\
& 52\cdot(1-x)^8\cdot x^6 + 90\cdot(1-x)^{10}\cdot x^6 + 34\cdot(1-x)^{12}\cdot x^6 + \\
& 528\cdot(1-x)^{14}\cdot x^6 + 436\cdot(1-x)^{16}\cdot x^6 + 185\cdot(1-x)^{18}\cdot x^6 + \\
& 14\cdot(1-x)^{20}\cdot x^6 + 184\cdot(1-x)^{22}\cdot x^6 + 130\cdot(1-x)^{24}\cdot x^6 + \\
& 42\cdot(1-x)^{26}\cdot x^6 + 3\cdot(1-x)^{28}\cdot x^6
\end{aligned}
\tag{5.3}
$$

5.3 Die SAHARA-SCTP Architektur

Das SCTP ist als zuverlässiges und paketorientiertes Übertragungsprotokoll der ISO/OSI Schicht 4 vor allem für COTS Kommunikationsmodule mit Standard Betriebssystem in si-

Übertragungen angibt. Mit a ist angegeben, wie viele Kombinationen gleicher b und c es insgesamt gibt, die RQ erfüllen. Die Summe aller a ist wiederum die Anzahl zuverlässiger Mutationen.

Tabelle 5.5: Komplexität der HASP-Architektur

Szenario	Komplexität		
	Zustände	Speicher	Dauer
SZ_1	4.892.475	3.553 MB	2 h 52 m
SZ_2	89.436	74 MB	3 m 7 s
SZ_3	399.126	212 MB	13 m 10 s
SZ_4	1.153.628	1.153 MB	1 h 35 m
Mutationsgenerator	81.854	25813 Mutationen	1 h 2 m (48 Rechenprozesse)

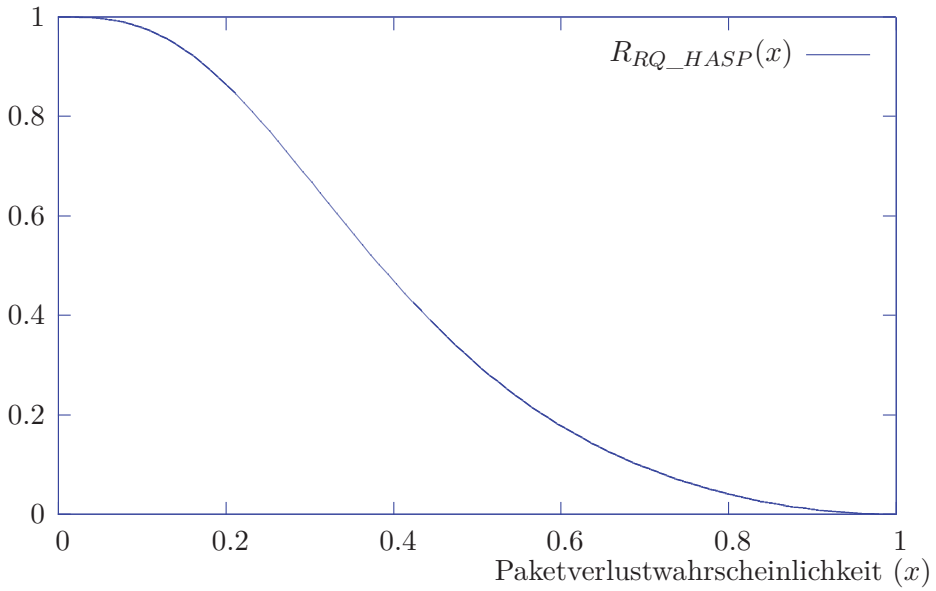


Abbildung 5.8: Ermittelte Zuverlässigkeitsfunktion des Hybrid Acknowledge Safety Protocol

cherheitsrelevanten Kommunikationsarchitekturen von Bedeutung. Dieses Protokoll arbeitet oberhalb von IP und ermöglicht damit, eine zuverlässige Verbindung über IP-basierte Backbonenetze aufzubauen. Die komplexen Zuverlässigkeitsmechanismen stammen von TCP und sind an die Eigenschaften großer, fehlerbehafteter Infrastrukturen angepasst. Die Funktionsweise von SCTP ist der RFC 4960 [Ste07] zu entnehmen. In Abbildung 5.10 auf der nächsten Seite ist die mit dieser Fallstudie analysierte Architektur skizziert. Die Spezifikation von SCTP umfasst viele Details, die das Model Checking nicht durchführbar machen. Es sind Abstraktionen nötig, die vor dem Hintergrund einer Worst-Case Analyse das mindeste Verhalten zur Zuverlässigkeit beschreiben. Das SCTP arbeitet nach dem ACK Verfahren, bei dem Quittungen kumulativ und selektiv für erhaltene Daten gesendet

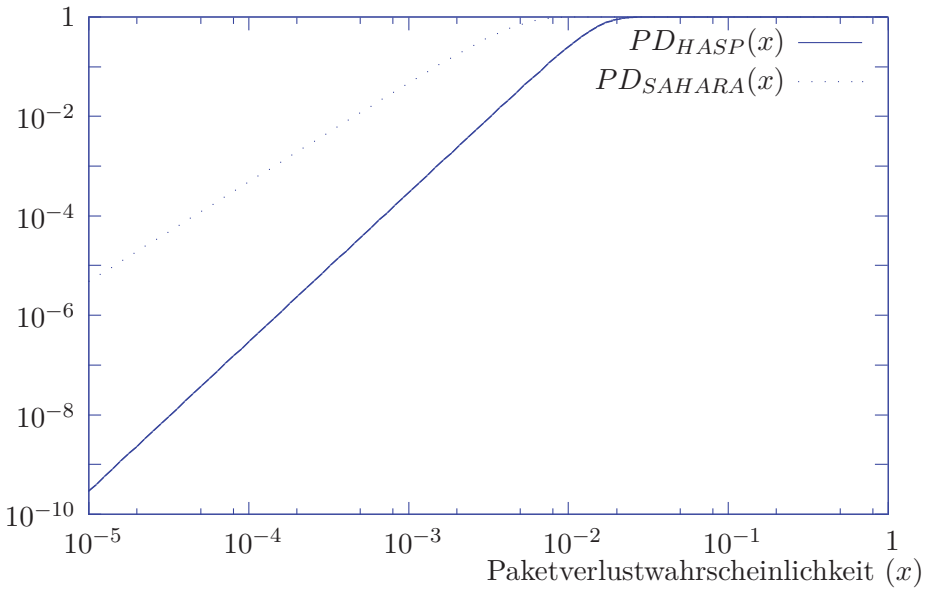


Abbildung 5.9: Ausfallwahrscheinlichkeit pro Stunde. Vergleich zwischen der HASP- und der SAHARA Architektur

werden. Der Wiederholungstimer für ausbleibende Quittungen wird wie bei TCP dynamisch an die Round-Trip-Time (RTT) angepasst. Der Anwender legt ein Intervall fest, in dem sich der Wiederholungstimer bewegt. Der konkrete Wert innerhalb des Intervalls hängt von der RTT und weiteren Parametern ab. Da bei Model Checking die RTT nicht realistisch nachgebildet werden kann und zudem die dynamische Time-Out Berechnung eine extreme Vergrößerung des Zustandsraums bedeutet, ist in dem Modell folgende Abstraktion hierzu festgelegt: Die obere Grenze des Intervalls wird als fester Time-Out Wert verwendet. Diese hat den größten negativen Einfluss auf die Echtzeitanforderungen und ist im Sinne einer Worst-Case Zuverlässigkeitsanalyse eine legitime Annahme. Es ist zu erwarten, dass in der Realität deutlich bessere Zuverlässigkeiten zu erzielen sind. Der Schwerpunkt der Analyse liegt auf dem Fehlerkontrollmechanismus. Es ist daher eine sehr vereinfachte Form des Übertragungsdienstes modelliert. Der Verbindungsauf- und -Abbau, Fluss- und Staukontrolle sowie die Kanalüberwachung mit SCTP-Heartbeats und Redundanz sind nicht berücksichtigt.

In der Tabelle 5.6 auf der nächsten Seite sind die wesentlichen Parameter des SCTP-Architekturmodells aufgelistet. Als Safety-Layer wird das SAHARA Protokoll aus der Fallstudie 5.1 auf Seite 98 eingesetzt. Die Zuverlässigkeitseigenschaft RQ und das Stable-Safe Kriterium SSS entsprechen denen der vorangegangenen Fallstudien. Die Dauer der mit dem Mutationsgenerator durchgeführte Zuverlässigkeitsanalyse betrug mit 30 Rechenprozessen 1 h 5 m. Es sind dabei 2628 Mutationen verifiziert worden, wobei 1578 Mutationen

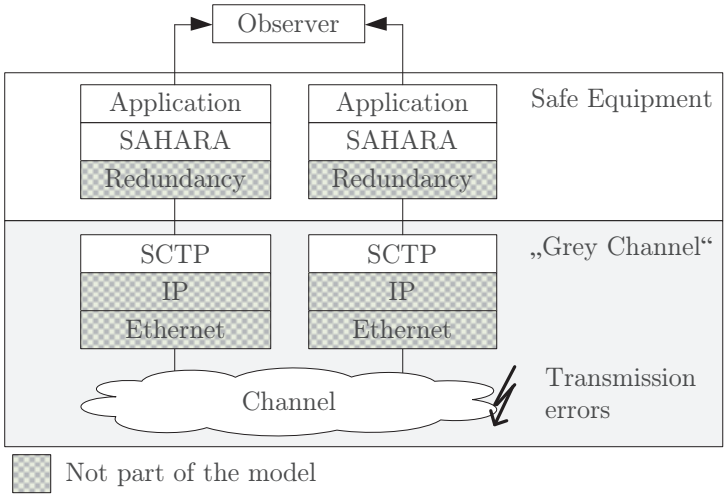


Abbildung 5.10: Skizze der SAHARA-SCTP Architektur

die Eigenschaft RQ erfüllen und damit die Formel 5.4 auf Seite 116 bilden.

Tabelle 5.6: SCTP - Spezifische Parameter

Parameter	Beschreibung
$B_{Timer} = 20\ ms$	Bundling-Timer. Wartezeit auf weitere Daten, bis das Paket gesendet wird.
$T_3 = 300\ ms$	Retransmission Time-Out. Erneute Übertragung von unquittierten Datenpaketen.
$Ack_{Timer} = 80$	Verzögertes Acknowledge.
$Delay = 80\ ms$	Latenzzeit der Übertragungskanäle (Hin- und Rückrichtung).
$fh_1 = fh_2$ $b_0 = \infty$ $b_1 = 5$ $b_2 = \cdots = b_6 = 0$	Die Fehlerhypothese erlaubt unendlich viele korrekte Übertragungen und fünf Paketverluste je Richtung. Weitere Fehlerarten sind nicht zugelassen.

Die Diagramme in den Abbildungen 5.11 auf Seite 117 und 5.12 auf Seite 117 lassen eine deutliche Steigerung der Zuverlässigkeit gegenüber den zuvor durchgeführten Fallstudien SAHARA und HASP erkennen. Obwohl mit dem Hybrid Acknowledge Safety Protocol die Zuverlässigkeitsschwächen von SAHARA beseitigt worden sind, übertrifft die Architektur

mit SAHARA und SCTP die HASP Architektur deutlich. Die Ursache hiefür liegt in dem NAK Verfahren mit dem konservativen doppelten Zeitstempel zum Erkennen von Echtzeitverletzungen. Bei SAHARA und HASP muss zunächst eine Folgenachricht eintreffen, um überhaupt einen Übertragungsfehler festzustellen. Die daraufhin initiierte Wiederholung von Daten lässt in vielen Fällen die Zeitstempelprüfung fehlschlagen, sofern ein weiterer Nachrichtenverlust auftritt. Mit einer ACK- basierten Fehlerkontrolle findet die Wiederholung mit dem Ausbleiben der erwarteten Quittung statt. Somit kann ohne hohe Folgezeiten von weiteren Daten der Verlust schneller und damit auch häufiger korrigiert werden. Die damit verbundene Erhöhung der Zuverlässigkeit macht sich vor allem in Netzwerken mit größeren Fehlerwahrscheinlichkeiten bemerkbar. In der Regel sind diese Protokolle komplexer in ihrem Aufbau und verbrauchen zudem höhere Ressourcen. Von solchen Protokollen kann profitiert werden, wenn zu den sicheren und üblicherweise leistungsschwächeren Rechner, Kommunikationsmodule eingesetzt werden. Diese Kommunikationsmodule können komplexe und erprobte Standardprotokolle zur Zuverlässigkeitssteigerung beinhalten. Im nächsten Kapitel sind die Ergebnisse der Fallstudien noch einmal zusammengefasst.

$$
\begin{aligned}
R_{RQ\_SCTP} =& (1-x)^2 + 2*(1-x)^3*x + 3*(1-x)^2*x^2 + \\
& 2*(1-x)^6*x^2 + 4*(1-x)^3*x^3 + 6*(1-x)^5*x^3 + \\
& 2*(1-x)^9*x^3 + 5*(1-x)^2*x^4 + 6*(1-x)^4*x^4 + \\
& 5*(1-x)^6*x^4 + 11*(1-x)^8*x^4 + 2*(1-x)^{12}*x^4 + \\
& 2*(1-x)^3*x^5 + 21*(1-x)^5*x^5 + 24*(1-x)^7*x^5 + \\
& 7*(1-x)^9*x^5 + 16*(1-x)^{11}*x^5 + 2*(1-x)^{15}*x^5 + \\
& 31*(1-x)^4*x^6 + 35*(1-x)^6*x^6 + 42*(1-x)^8*x^6 + \\
& 47*(1-x)^{10}*x^6 + 11*(1-x)^{12}*x^6 + 12*(1-x)^{14}*x^6 + \\
& 22*(1-x)^3*x^7 + 49*(1-x)^5*x^7 + 89*(1-x)^7*x^7 + \\
& 92*(1-x)^9*x^7 + 52*(1-x)^{11}*x^7 + 29*(1-x)^{13}*x^7 + \\
& 7*(1-x)^2*x^8 + 44*(1-x)^4*x^8 + 111*(1-x)^6*x^8 + \\
& 138*(1-x)^8*x^8 + 94*(1-x)^{10}*x^8 + 36*(1-x)^{12}*x^8 + \\
& 24*(1-x)^3*x^9 + 102*(1-x)^5*x^9 + 127*(1-x)^7*x^9 + \\
& 80*(1-x)^9*x^9 + 24*(1-x)^{11}*x^9 + 9*(1-x)^2*x^{10} + \\
& 56*(1-x)^4*x^{10} + 55*(1-x)^6*x^{10} + 31*(1-x)^8*x^{10} + \\
& 8*(1-x)^{10}*x^{10}
\end{aligned}
\tag{5.4}
$$

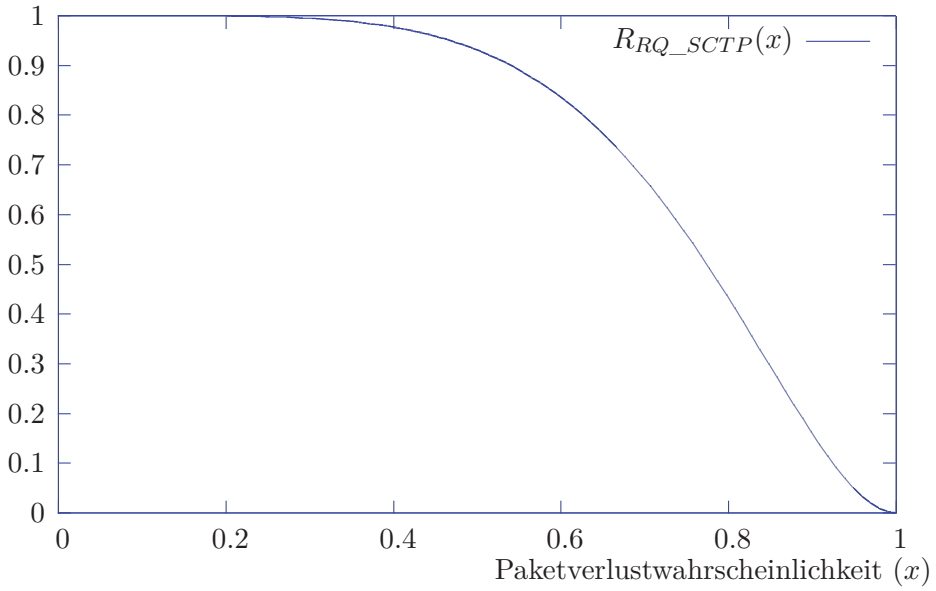


Abbildung 5.11: Ermittelte Zuverlässigkeitsfunktion der SCTP Architektur

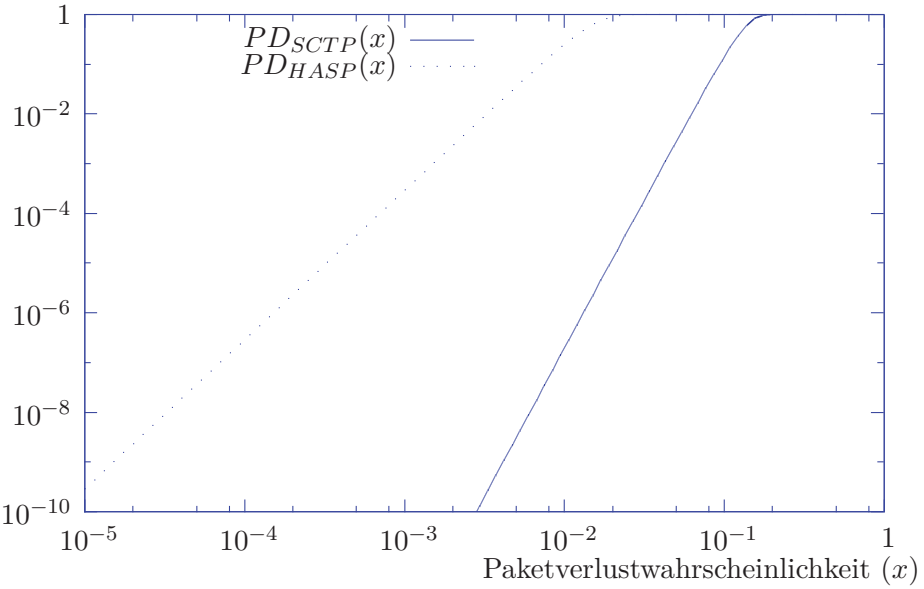


Abbildung 5.12: Ausfallwahrscheinlichkeit pro Stunde. Vergleich zwischen der SCTP und der HASP Architektur

Kapitel 6

Zusammenfassung und Ausblick

Man merkt nie, was schon getan wurde; man sieht immer nur das, was noch zu tun bleibt.

Marie Curie, 1867 - 1934

Dieses Kapitel fasst die Ergebnisse dieser Arbeit zusammen. Zunächst wird auf die in der Einleitung genannten Beiträge eingegangen. Anschließend werden die Fallstudien bewertet und die resultierenden Ergebnisse kommentiert. Im Letzten Teil dieses Kapitels ist ein Ausblick für weitere Forschungen auf Grundlage der Verifikation von Modellmutation gegeben.

6.1 Die generische Modellierung sicherheitsrelevanter Kommunikationsarchitekturen

Diese Arbeit hat mit der domänenspezifischen Sprache CAMoLa ein Framework für die Analyse von sicherheitsrelevanten Kommunikationsarchitekturen vorgestellt. Mit CAMoLa ist gezeigt, das die generische Modellierung von Architekturen und die durch Codegeneratoren integrierten Analyse-Tools, auf einfache und praxisnahe Weise mit einer DSL zu einem kompletten Framework kombiniert werden können. Die DSL CAMoLa reduziert den Modellierungsaufwand beim Erstellen von Architekturmodellen. Die Fallstudie mit dem Protokoll HASP demonstriert zum Beispiel die Generik: Die Umgebung des Architekturmodells stammt aus der zuvor analysierten Fallstudie SAHARA, so dass lediglich das Safety-Protokoll HASP für die Fallstudie ergänzt wurde. Die identische Umgebung und die gleichen formalisierten Anforderungen an die Architektur machen Vergleiche zwischen Eigenschaften der Protokolle möglich. Das CAMoLa Framework bietet mit dem integrier-

ten Model Checker die Möglichkeit, den Sicherheitsnachweis von Safety-Protokollen zu unterstützen. Das Konzept der Watchdog-Prozesse und den modellierten Fehlerarten im Übertragungskanal schaffen einen Rahmen, der gemäß der Normen EN 50128, EN 50129 und EN 50159 die Safety von sicherheitsrelevanten Kommunikationsprotokollen nachweisen kann.

6.2 Kombination von UPPAAL und PRISM

Die skizzierte Kombination zweier unterschiedlicher Model Checking Tools im Anhang A auf Seite 125 ist mit einem gemeinsamen Metamodell realisiert. Das Generieren konsistenter UPPAAL und PRISM Modelle ermöglicht das Nachweisen unterschiedlicher Modelleigenschaften, wobei gleichzeitig von den jeweiligen Stärken der Tools profitiert wird. Zum Beispiel hilft der Ausführungspfad zu einem Systemzustand, der geforderte Eigenschaften nicht erfüllt, Modellierungs- und Spezifikationsfehler zu finden. Dieses ist ein nicht zu unterschätzendes Feature von UPPAAL. Der probabilistische Model Checker PRISM ist in der Lage, probabilistische Modelleigenschaften zu ermitteln, was mit wiederum mit UPPAAL nicht möglich ist. Diese Kombination ist jedoch nicht vertieft worden, da die zur Konsistenz der Zustandsübergänge nötigen Restriktionen im Metamodell und weitere Nachteile diesen Ansatz wenig praktikabel machen.

6.3 Analyse sicherheitsrelevanter Kommunikationsarchitekturen

Das in Kapitel 4 auf Seite 61 vorgestellte Framework strukturiert die Verifikation und Analyse von sicherheitsrelevanten Kommunikationsarchitekturen in drei Phasen. Die Modellierungssprache CAMoLa verbindet dabei die bislang getrennte Verifikation von Safety und Analyse von Zuverlässigkeitseigenschaften. Safety und Reliability werden in Anwesenheit von verschiedenen Fehlerarten nachgewiesen, die bei der Übertragung von Daten auftreten können. Das Modell eines generischen Übertragungskanals wird mit Parametern auf eine Fehlerhypothese eingestellt. Safety- und weitere Modelleigenschaften werden mit dem Timed Automata Tool UPPAAL verifiziert. Wird eine Eigenschaft nicht erfüllt, so generiert UPPAAL ein Ausführungspfad, der zur Verletzung der Eigenschaft führt. Modellierungs- und Spezifikationsfehler werden auf diese Weise schnell lokalisiert. Die Zuverlässigkeit einer Kommunikationsarchitektur gegenüber Übertragungsfehlern ist eine wichtige Information, um Anforderungen an Übertragungsnetze zu ermitteln. Die in Kapitel 4.6 auf Seite 83 vorgestellte Methode zum Ermitteln der Zuverlässigkeit des logischen Kanals unterschei-

det sich von den bisher verwendeten Ansätzen. In der Literatur sind das probabilistische Model Checking und simulative Methoden genannt, um Zuverlässigkeitseigenschaften zu ermitteln. Das neue Verfahren, das mit dieser Dissertation herausgearbeitet wurde, bietet folgende entscheidende Vorteile gegenüber klassischen Methoden:

1. Die probabilistische Größe Zuverlässigkeit wird mit einem ausgereiften (nichtprobabilistischen) Model Checker ermittelt. Das Ergebnis der formalen Verifikation liegt im Gegensatz zu simulativen Methoden nicht in einem Vertrauensintervall. Gleichzeitig weist der Model Checker Echtzeiteigenschaften und weitere Sicherheitsbedingungen nach.

2. Das Ergebnis der Zuverlässigkeitsanalyse ist eine symbolische Formel, so dass im Vorfeld keine konkreten Wahrscheinlichkeiten festgelegt werden müssen.

3. Eine restriktive Fehlerhypothese verringert die Komplexität der Analyse und ermittelt dabei immer die Mindestzuverlässigkeit, die von der Architektur erbracht wird.

4. Die entstehende Kombinatorik (bedingt durch die Fehlerhypothese) wird genutzt, um mit Hilfe von Modellmutationen den Zustandsraum zu partitionieren und verteilt zu verifizieren.

Der zuletzt genannte Punkt bietet potential für weitere Forschungen. Das Verteilen der Mutationen basiert auf dem speziellen Vorwissen über Kommunikationsarchitekturen: Aus Sicht des Model Checkings ist die Art der Übertragung von Nachrichten eine wesentliche Verzweigung im Zustandsraum. Üblicherweise führt die unterschiedliche Übertragung einer Nachricht zu verschiedenen Zuständen im Modell. Mit den Modellmutationen wird die Kombinatorik im Zustandsraum verhindert und reduziert diesen damit. Es entstehen entsprechend viele Mutationen (bedingt durch die Kombinatorik), die allerdings auf mehrere Prozesse oder Rechner zur Verifikation verteilt werden können.

Zusammenfassend betrachtet ist der Ansatz der Modellmutationen, in dem speziellen Fall von Kommunikationsarchitekturen, sowohl für die Ermittlung einer Formel zur logischen Zuverlässigkeit geeignet, als auch zum Entgegenwirken der Zustandsraumexplosion beim Model Checking.

6.4 Ergebnisse der Fallstudien

Die in Kapitel 5 auf Seite 97 durchgeführten Fallstudien wenden die drei Phasen des CAMoLa Frameworks an realen Kommunikationsarchitekturen an. Zum einen ist damit die Praxistauglichkeit des Frameworks demonstriert und zum anderen zeigen die Ergebnisse,

dass die vorgestellten Safety-Protokolle SAHARA und HASP die in der EN 50159 definierten Fehlerarten aufdecken. Die Zuverlässigkeitsanalyse zeigt Schwächen in der SAHARA-UDP Architektur und liefert gleichzeitig Hinweise auf die Anforderungen an einen Übertragungskanal, um entsprechende Zuverlässigkeitsziele zu erreichen.

In der Fallstudie SAHARA-SCTP (Kapitel 5.3 auf Seite 113) ist die logischen Zuverlässigkeit gegenüber der SAHARA-UDP Fallstudie deutlich gesteigert, wie Abbildung 6.1 auf der nächsten Seite zeigt. Die Abbildung 6.2 auf der nächsten Seite vergleicht die errechnete Ausfallwahrscheinlichkeit in Anhängigkeit von der Paketverlustwahrscheinlichkeit der vorgestellten Architekturszenarien. Zur Orientierung sind die Bereiche von Paketverlustwahrscheinlichkeiten markiert, die sich aus den Bitfehlerwahrscheinlichkeiten der verschiedenen Technologien ergeben (siehe Tabelle 2.3 auf Seite 24)[1]. Die Paketverlustwahrscheinlichkeit x errechnet sich unter der Voraussetzung der stochastischen Unabhängigkeit der Bitfehlerwahrscheinlichkeit p_{Bit} und der Anzahl Bits in einem Paket (n) wie folgt:

$$x = 1 - (1 - p_{Bit})^n \tag{6.1}$$

In Netzwerken entstehen Paketverluste auch durch die Überlastung aktiver Netzwerkkomponenten, wie zum Beispiel Router. Die Paketverlustwahrscheinlichkeit hängt somit auch von der Dimensionierung beziehungsweise der Auslastung des Übertragungskanals ab. Die Ergebnisse der Fallstudien zeigen die Qualitätsmerkmale für festgelegte Zuverlässigkeitsanforderungen, die ein Übertragungskanal erfüllen muss. Mit dem Wissen über die geforderten Qualitätsmerkmale können zum Beispiel Service-Level-Agreements mit einem Netzwerkprovider vereinbart werden.

6.5 Ausblick

Mit Hilfe von Codegeneratoren können Aspekte der statischen Semantik von Modellen überprüft werden. Diese Überprüfung ist in CAMoLa derzeit nicht realisiert, aber mit der zukünftigen Erweiterung geplant. Diese Art der Modellüberprüfung liefert konkrete Hinweise auf Modellierungsfehler, sofern statische Aspekte wie zum Beispiel Wertebereiche von Variablen nicht korrekt modelliert sind. Ein wesentliches Feature für die Erweiterung von CAMoLa ist die automatische Generierung von zum Beispiel C-Code aus den Modellen von (Safety-) Protokollen. Es ist zu erwarten, dass mehr und mehr Systeme der Streckensicherungstechnik (speziell Feldelemente) zukünftig mit einer digitalen Kommunikationsschnittstelle ausgestattet werden. Sofern aufgrund von neuen Anforderungen

[1] Es ist eine Paketlänge von $n = 800$ Bit angenommen. In Ethernet-Architekturen wird eine durch Bitfehler verfälschte Nachricht verworfen.

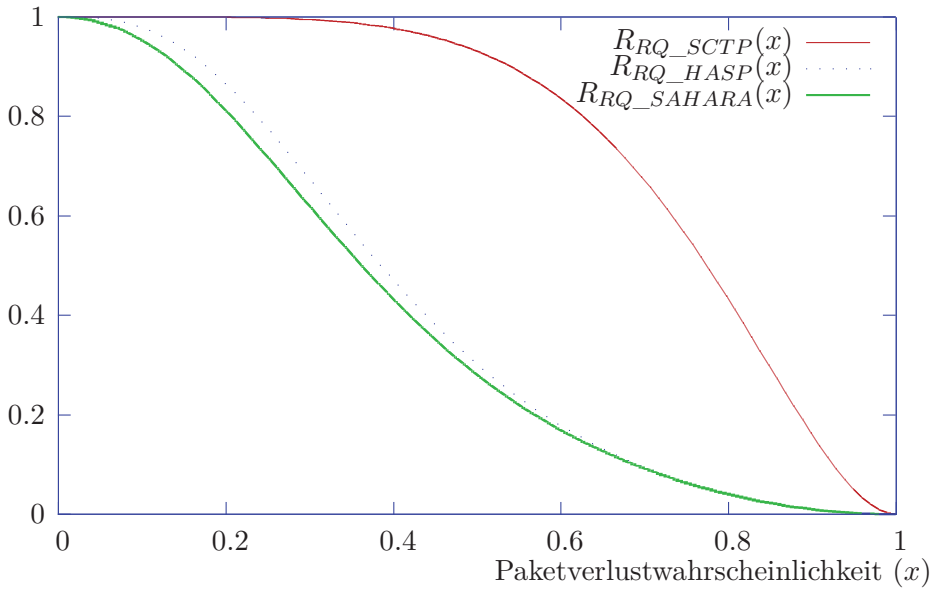


Abbildung 6.1: Vergleich der Zuverlässigkeitsfunktionen von SAHARA, HASP und SCTP

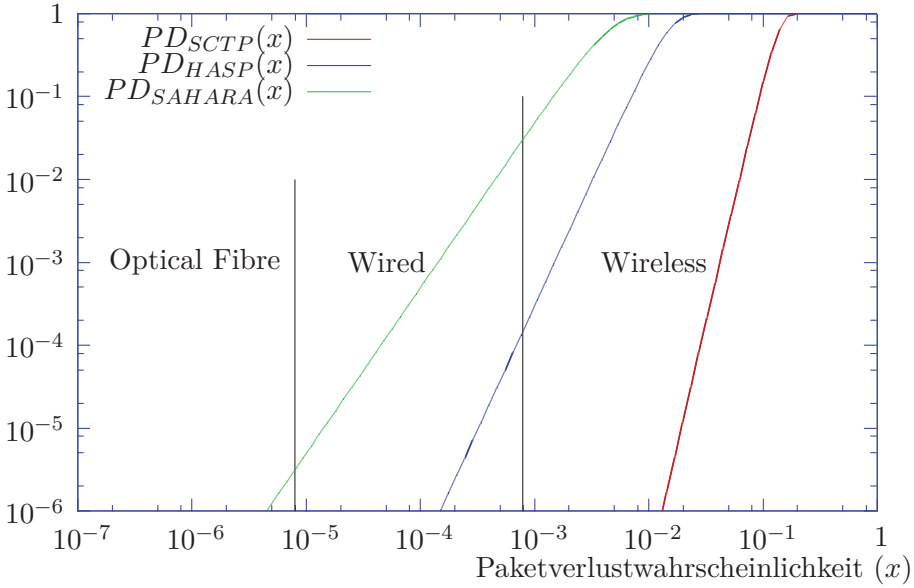


Abbildung 6.2: Ausfallwahrscheinlichkeit pro Stunde von SAHARA, HASP und SCTP

spezielle Protokolle für die Kommunikation entwickelt werden müssen, kann mit der automatischen Codegenerierung ein erhebliches Maß an manueller Arbeit gespart werden. Ist die Analyse der Architekturmodelle nach der Phase 3 zufriedenstellend angeschlossen, dann kann die Implementierung der Zustandsmaschine des neuen Protokolls automatisch aus dem Modell generiert werden. Fehler, die bei der manuellen Implementierung einer Spezifikation entstehen können, werden damit verhindert. Zudem wird der entstandene Aufwand bei der Modellierung und Verifikation kompensiert.

$$(a_0 \cdot x^{b_0} \cdot (1-x)^{c_0}) + \cdots + (a_n \cdot x^{b_n} \cdot (1-x)^{c_n})$$

6.5.1 Verifikation von Safety mit Modellmutationen

Dieser Arbeit beschreibt die Zuverlässigkeitsanalyse mit Modellmutationen. Von den genannten Vorteilen durch die Partitionierung des Zustandsraums und das Verteilen der Mutationen auf mehrere Prozesse kann auch die Verifikation von Safety-Eigenschaften profitieren. Hierzu sind weitere Forschungen nötig, da die beschriebene Weise, Mutationen "abzurollen", mit dem Bounded Model Checking (BMC) vergleichbar ist. Vom BMC ist der Begriff *Completeness Threshold* bekannt, der auch für die Verifikation von Safety nach dem Mutationsansatz eine Rolle spielt.

Die Grundsätzliche Idee von BMC ist es, die Transitionen eines Modells bis zu k Schritte weit abzurollen und mit einem SAT-Solver die Erfüllbarkeit einer Eigenschaft Φ zu prüfen. Ist die Eigenschaft erfüllt, dann wird dieses als $M \models_k \Phi$ notiert [CKOS04]. Ist mit k ein Minimum gefunden, für das gilt $M \models_k \Phi \Rightarrow M \models \Phi$, dann wird k als Completeness Threshold bezeichnet. Hierbei ist in einer begrenzen Anzahl von Schritten gezeigt, dass die Aussage Φ auch für beliebig weitere Schritte gilt.

Das Abrollen des Zustandsraums nach der in Kapitel 4.6 auf Seite 85 beschriebenen Methode ist mit dem Bounded Model Checking vergleichbar. Die Dimension d von Traces entspricht der Länge k beim BMC. Es ist das Ziel, d so klein wie möglich zu halten. Die Dimension muss allerdings so weit vergrößert werden, bis entweder das Zuverlässigkeitskriterium RQ erfüllt oder der Stable-Safe-Sate SSS erreicht wird. Die Eigenschaften RQ und SSS sind so definiert, dass sie auch gelten, wenn der Zustandsraum mit der Trace-Dimension beliebig weit abgerollt wird. Stoppt das Abrollen, dann ist der Completeness Threshold der Zuverlässigkeitsanalyse erreicht.

Für die Verifikation von Safety-Eigenschaften kann dieses jedoch nicht angewendet werden: Der Beweis, dass eine Safety-Eigenschaft bis zur Dimension d der Traces gilt, lässt nur den Schluss zu, die Safety-Eigenschaft gelte auch für $d+1$, wenn mit d der Completeness Threshold für die Safety Verifikation erreicht ist. Befindet sich in den bis d expandierten Mutationen kein unsicherer Zustand, dann muss zum Beispiel mit induktiven Ansätzen gezeigt werden, dass auch mit der beliebigen Vergrößerung von d kein unsicher Zustand existiert. Mit der Erforschung dieses Themas profitiert die Verifikation von Safety-Eigenschaften von dem Mutationsansatz, mit dem durch die Reduktion des Zustandsraums und der verteilten Verifikation komplexere Modelle verifiziert werden können.

Anhang A

Fallstudie: Kombination von UPPAAL und PRISM

Mit der Kombination der beiden Model Checking Tools UPPAAL und PRISM ist es möglich, probabilistische und nicht-probabilistische Eigenschaften eines gemeinsamen Modells zu verifizieren. Zudem werden Tool-Features kombiniert, wie zum Beispiel die Generierung von Diagnostic Traces mit dem UPPAAL Tool was in PRISM nicht realisiert ist. Auf der anderen Seite können probabilistische Eigenschaften (zum Beispiel die Zuverlässigkeit) von Modellen analysiert werden, was wiederum mit dem UPPAAL Tool nicht möglich ist (Abbildung 3.4 auf Seite 58). Für solche Analysen benutzt PRISM den probabilistischen Operator $\mathcal{P}_{=?}$, der aufgrund probabilistischer Transitionen im Modell die Wahrscheinlichkeit ermittelt, mit der eine PCTL Aussage erfüllt ist. Eine Kombination von Timed Automata und PMC ist Stand gegenwärtiger Forschung (siehe [UPP]) und bereits während der Präsentationen des Workshops *Applying Concurrency Research in Industry*, co-located mit der Konferenz *CONCUR 2007* in Lissabon, diskutiert worden: Auf die Frage: „Is there any need for stochastic and probabilistic modelling in applications? More pointedly, have you met an example that you could not model because your tool does not support stochastic or probabilistic phenomena?“ antwortet Frits Vaandrager: „Yes!! There is a great need for stochastic and probabilistic modelling and analysis techniques, and I would for instance welcome a tool that combines the functionality of UPPAAL and PRISM.“ ([ABF+08]).

Die angesprochenen Model Checking Tools UPPAAL und PRISM sind in ihrer jeweiligen Anwendungsdomäne weit verbreitet und werden kontinuierlich weiterentwickelt. Die Kombination dieser Tools kann durch eine Transformation von UPPAAL in PRISM Modelle (oder umgekehrt) realisiert werden oder durch die Definition eines gemeinsamen Metamodells, das in beide Modellrepräsentationen übersetzt werden kann. Damit die Konsistenz zwischen UPPAAL und PRISM Modellen gewährleistet ist, sind Einschränkungen

im Sprachumfang nötig. Diese Restriktionen sind mit einem gemeinsamen Metamodell am praktikabelsten umzusetzen. Das gemeinsame Metamodell definiert eine Modellierungssprache, die jeweils durch einen Codegenerator konsistente Uppaal und PRISM Modelle erzeugt (Abbildung A.1 auf der nächsten Seite) Das Common Meta Model (CMM) ist das neue Frontend für beide Tools, so dass dem Modellierer die jeweiligen Submodelle verborgen bleiben. Für die Konsistenz zwischen den Submodellen müssen die semantischen Gemeinsamkeiten identifiziert werden, die schließlich Einschränkungen für die gemeinsame Modellierungssprache definieren. Die Gemeinsamkeiten von Uppaal und PRISM liegen in der Modellierung von Transitionssystemen mit Locations, Bounded Integer und Booleschen Variablen, sowie elementare Operationen auf Variablen. Ein Systemmodell besteht in beiden Model Checking Tools aus synchronisierten Prozessen, die nichtdeterministisch ausgeführt werden. Das gemeinsame Metamodell von Uppaal und PRISM trägt den Namen CMM und ist im Folgenden skizziert.

A.1 CMM - Common Meta Model

Das CMM ist derart definiert, dass ein CMM Systemmodell aus synchronisierten Prozessen besteht, die jeweils als (probabilistisches) Zustandsdiagramm beschrieben werden. In den Prozessen können Bounded Integer und Boolesche Variablen, sowie Clocks Deklariert verwendet werden. Aufgrund der parallelen Ausführung von Prozessen werden alle CMM Modelle als MDP Modelle nach PRISM transformiert. Es sind Restriktionen definiert, um bei der Transformation konsistente Submodelle zu erzeugen. In dem Zustandsdiagramm können Variablen und Clocks als Guard-Bedingungen von Transitionen und Update-Anweisungen mit arithmetischen Operationen modelliert werden. Strukturen und Arrays, wie sie in Uppaal möglich sind, werden von PRISM nicht unterstützt und sind damit auch nicht im CMM erlaubt. Die Synchronisation der nebenläufigen Prozesse geschieht über binäre, blockierende Kanäle. Beide Model Checker erlauben zwar Broadcast-Kanäle, jedoch sind diese in Uppaal nicht-blockierend und in PRISM blockierend. Ein CMM Zustandsdiagramm besteht aus Transitionen in der Form $l \xrightarrow{g/ch} \Lambda$, mit einer diskreten Verteilung $\Lambda = \{(\lambda^1, l'^1, u^1), \ldots, (\lambda^k, l'^k, u^k)\}$ von Variablen-Updates u^d und Ziel-Locations l^d, $d = 1, \ldots, k$. Die diskrete Wahrscheinlichkeit, in die Location l'^d zu wechseln und dabei das Update u^d auszuführen, beträgt $\lambda^d \in (0, 1]$ mit $1 = \sum_{d=1}^{k} \lambda^d$. Bei der Transformation einer probabilistischen Transition in ein Uppaal Modell werden k nichtdeterministische Transitionen erzeugt. Damit geht die Wahrscheinlichkeitsinformation verloren, aber jeder Übergang von einer Quell-Location l zu den Ziel-Locations $l'^1, \ldots, l'^k$ bleibt erhalten. Zuletzt die Definition des Clock-Konzepts: In PRISM werden digitale Uhren explizit eingeführt. Dabei wird der in jedem Prozess gemeinsame Kanal

tick benutzt, um die jeweiligen lokalen Uhren simultan zu inkrementieren.

Neben den Codegeneratoren für die Transformation zu UPPAAL und PRISM Modellen gehört zum CMM ein Static-Semantic-Checker, der vor der Transformation alle Restriktionen auf den konkreten Modellen prüft. Somit sind Inkonsistenzen aufgrund von Modellierungsfehlern ausgeschlossen. Ein CMM Systemmodell definiert zwei hierarchische Ebenen: Eine Prozessebene, in der Prozesse als Probabilistic Timed Transition System modelliert werden und eine Systemebene, in der Prozesse und die Synchronisation zwischen jeweils zwei Prozessen modelliert werden. Exemplarisch werden verschiedene Transitionen und Locations in CMM dargestellt und die Übersetzung zu UPPAAL und PRISM gezeigt.

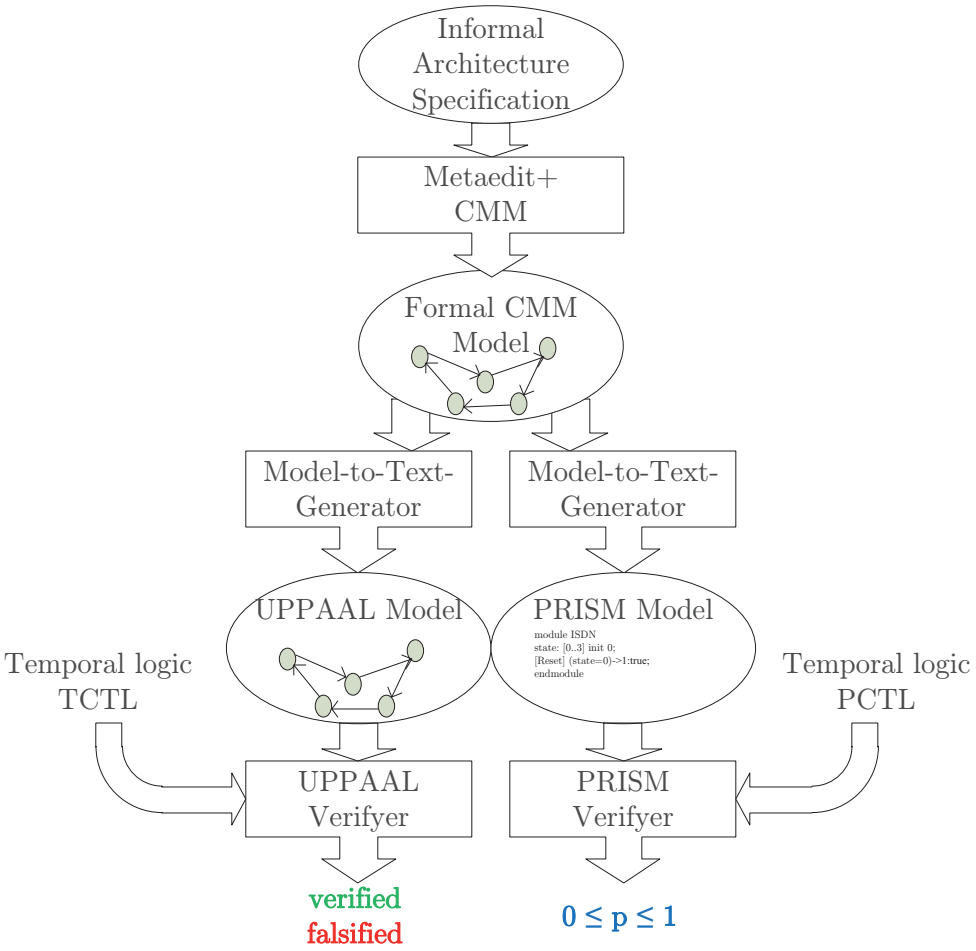


Abbildung A.1: CMM - Gemeinsames Metamodell für UPPAAL und PRISM

Definition A.1.1 (CMM Modell)**.** Ein CMM Modell ist ein Tupel $(\bar{A}, V_G, Ch)$, wobei

(i) $\bar{A}$ ein Vektor mit n CMM Prozessen $A_1, \ldots A_n$ ist,

(ii) V_G eine endliche Menge von globalen Booleschen- oder Integer-Variablen und

(iii) Ch eine Menge von Synchronisations-Labels ist.

Definition A.1.2 (CMM Prozess)**.** Ein *CMM Prozess* ist ein Tupel $(L, l_0, Type, I, E)$, wobei

- L eine Menge von Locations ist,
- l_0 ist die die initiale Location,
- $Type : L \to \{t, u\}$ ist eine Funktion, die jeder Location $l \in L$ den Typ "timed" oder "urgent" zuordnet,
- $Inv : L \to I$ ordnet einer Location eine Invariante zu,
- E ist eine Menge von Transitionen $l \xrightarrow{g/s} \{(\lambda^1, l'^1, u^1), \ldots, (\lambda^k, l'^k, u^k)\}$, mit $l, l'^1, \ldots, l'^k \in L$, g ist eine Guard-Bedingung, s ist ein Synchronisations-Label (optional), $u^1, \ldots, u^k$ sind Listen von Zuweisungen und $1 = \sum_{i=1}^{k} \lambda^i$ mit $\lambda^i \in (0, 1]$.

A.1.1 Timed Location.

Das CMM definiert die Modellierung von Timed- und Urgent Locations. Semantisch verhalten diese sich wie Timed- und Urgent-Locations in Uppaal und werden somit direkt übersetzt. Die Transformation von Timed-Locations zu PRISM erfordert einen zusätzliche Transition: Eine Transition, die alle Uhr-Variablen eines Moduls simultan mit den Uhren anderer Variablen inkrementiert. Für diesen Zweck wird in PRISM der zusätzliche Synchronisierungskanal *tick* eingeführt, über den alle PRISM Module synchronisiert sind. Somit ist sichergestellt, dass in allen Modulen die Modellzeit gleich und gleichzeitig voranschreitet.

A.1.2 Probabilistic Transitions.

Probabilistische Transitionen werden als nichtdeterministische Transitionen zu Uppaal Modellen transformiert. Dabei geht die Wahrscheinlichkeitsinformation verloren, solange aber jede Übergangswahrscheinlichkeit $\lambda^d > 0$ für $d = 1, \ldots, k$ ist, bleibt die Menge von Folgezuständen von PRISM und Uppaal konsistent.

A.1.3 Synchronisation Transitions.

Die Synchronisation zwischen Modulen beziehungsweise die simultane Ausführung von Transitionen unterscheidet sich in Uppaal und PRISM. Wahrend in Uppaal blockierende binäre Kanäle oder nicht-blockierende Broadcast Kanäle modelliert werden können,

unterstützt PRISM nur blockierende Broadcast Kanäle. Die Gemeinsamkeit und damit das Synchronisationskonzept von CMM sind blockierende binäre Kanäle. Im CMM wird nur das Modellieren von Kanal-Paaren (ein Sender und ein Empfänger) erlaubt.

A.1.4 Parallele Prozesse.

Die CMM Modelle bestehen aus parallelen Prozessen die jeweils paarweise über Kanal-Labels synchronisiert sind. Anstelle von Prozessen wird in PRISM Modellen von Modulen gesprochen. Jeder Prozess eines CMM Modells wird als Prozess zu Uppaal transformiert und zu PRISM als Modul. Das Template-Konzept von Uppaal und das Renaming-Konzept von PRISM werden nicht verwendet.

A.2 Bewertung der Kombination.

Mit dieser Kombination von Uppaal und PRISM durch ein gemeinsames Metamodell sind qualitative und quantitative Verifikationen von Modellen möglich. Es wird dabei nur ein Systemmodell benötigt, um mit verschiedenen temporallogischen Aussagen in TCTL und PCTL diese Aspekte zu verifizieren beziehungsweise zu analysieren. Zudem ist das Generieren von Diagnostic-Traces von Uppaal hilfreich, um Modellierungsfehler zu finden, wobei mit der Korrektur des CMM Modells die Veränderung nach der erneuten Transformation in Uppaal und PRISM enthalten ist. Des Weiteren ist die grafische Modellierung von Zustandsdiagrammen in vielen Fällen intuitiver als die textuelle Modellierungssprache von PRISM.

Die vorgestellte Kombination hat jedoch auch Nachteile:

Die definierten Einschränkungen in dem CMM machen die Modellierung von Systemen oftmals unnötig komplex, wie Fallstudien gezeigt haben. Besonders problematisch ist dabei das künstlich eingeführte Digital Clock Konzept. Da PRISM die internen Modelle als Multi Terminal Binary Decision Diagram (MTBDD) repräsentiert, führen große Wertebereiche von Variablen automatisch zu einer großen Modellrepräsentation. In Kommunikationsarchitekturen sind oft große Zeitbereiche anzutreffen, da zum Beispiel die Überwachungszeiten von Kommunikationsprotokollen wesentlich größer sind, als Zeiteigenschaften von Übertragunskanälen. An dieser Stelle führen realistische Werte zu großen Bereichen der Uhr-Variablen, was sich schließlich in der Größe des Zustandsraum wiederspiegelt. Des Weiteren ist die Größe des MTBDD vom Variable-Ordering abhängig, so dass die Reihenfolge der Deklarationen von Variablen und Prozessen nach bekannten Heuristiken zu

ordnen ist. Ebenfalls negativ ist die nummerische Berechnung von probabilistischen Größen beim PRISM Tool. Die Übergangswahrscheinlichkeiten von probabilistischen Transitionen werden nummerisch angegeben, so dass für die Analyse über Wertebereiche der probabilistischen Transitionen zwingend mehrere Model Checking Durchläufe nötig sind. Da bei den durchgeführten Fallstudien das Model Checking mit dem PRISM Tool deutlich mehr Ressourcen benötigte, als das Uppaal Tool ist dieser Ansatz nicht weiter verfolgt worden. Engpässe bei der Modellverifikation mit PRISM sind auch in [Wan06] genannt.

Die aus der Kombination von Uppaal und PRISM gewonnenen Erkenntnisse führten zu neuen Überlegungen, qualitative und quantitative Verifikationen miteinander zu verbinden. Zusammenfassend betrachtet ist es insbesondere der PRISM Model Checker, der die kombinierte Verifikation scheitern ließ.

Anhang B

Architekturmodelle

In diesem Teil des Anhangs sind die Architekturmodelle der in Kapitel 5 auf Seite 97 durchgeführten Fallstudien. Zu sehen sind in der Systemansicht von CAMoLa die Prozesse und die Synchronisation zwischen den Prozessen. Des Weiteren sind globale Typ-Definitionen und Konstanten aufgelistet. Die zugewiesenen Werten sind vom Szenario abhängig und können daher von denen im Code-Listing dargestellten abweichen. Die Prozessmodelle der Architekturen sind im Anhang C auf Seite 137 aufgelistet.

B.1 SAHARA Architekturmodell

Das Architekturmodell in der Abbildung B.1 auf der nächsten Seite gehört zu der ersten Fallstudie im Kapitel 5.1 auf Seite 98. Die Watchdogprozesse überwachen jeweils eine Transaktion. Zu den weiteren Bestandteilen des Architekturmodells gehören die Prozesse *Application*, *Buffer*, *ErroneousChannel* und *SAHARA_2v2*, die alle jeweils auf der Client- und Serverseite instanziiert sind.

```
1  // Global declarations
2  const int SEQMAX = 16;
3  const int MAX_Tn = 8;
4  const int TIME_OUT = 1000;
5  const int SAFETY_TIME = 1000;
6  //Typedefinitions
7  typedef struct{
8  int[0,SEQMAX] seq;
9  int[0,SEQMAX] cseq;
10 int[-1,MAX_DATA] data;
11 int[0,SEQMAX] cts;
12 int[0,SEQMAX] ts;
```

```
13 int[1,6] type;
14 }SafetyMsg;
```

Listing B.1: Globale Deklarationen der SAHARA Architektur

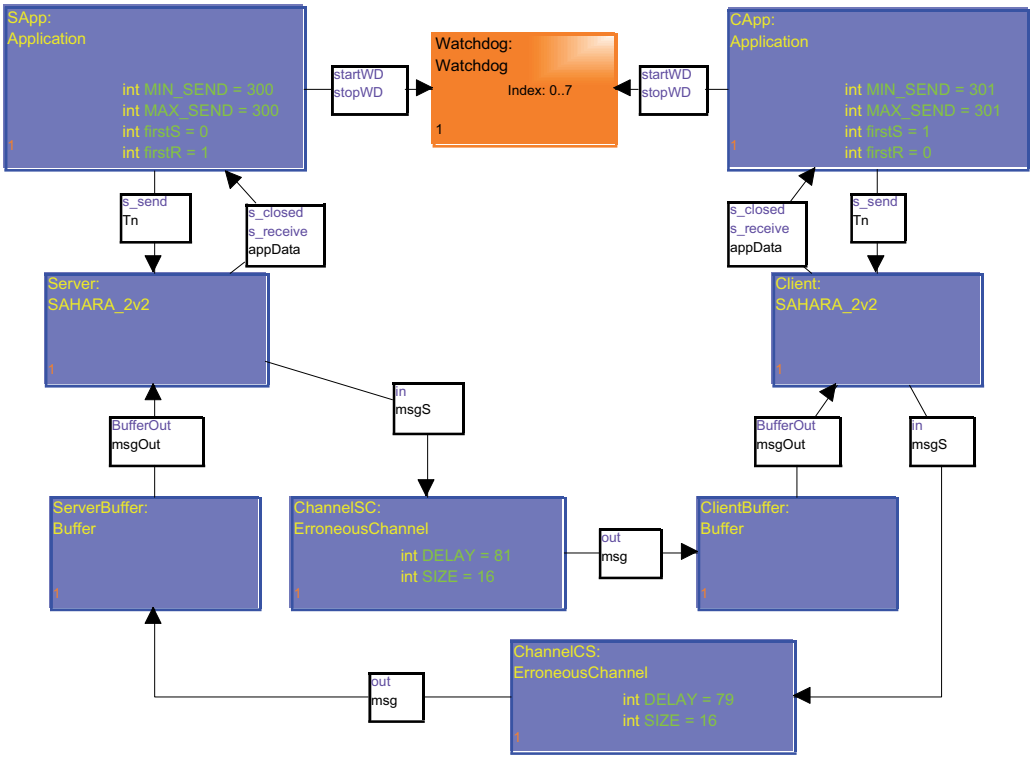


Abbildung B.1: Modell der SAHARA Architektur

B.2 HASP Architekturmodell

Das Architekturmodell mit dem HASP Protokoll ist identisch zum SAHARA Architekturmodell, wobei lediglich das SAHARA Protokoll durch das HASP Protokoll ersetzt ist. Die Fallstudie ist im Kapitel 5.2 auf Seite 108 beschrieben.

```
1 // Global declarations
2 const int SEQMAX = 16;
3 const int MAX_Tn = 8;
4 const int TIME_OUT = 1000;
5 const int SAFETY_TIME = 1000;
6 const int W_ACK = 300;
7 //Typedefinitions
8 typedef struct{
9 int[0,SEQMAX] seq;
```

```
int[0,SEQMAX] cseq;
int[-1,MAX_Tn] data;
int[0,SEQMAX] cts;
int[0,SEQMAX] ts;
int[1,6] type;
}SafetyMsg;
```

Listing B.2: Globale Deklarationen der HASP Architektur

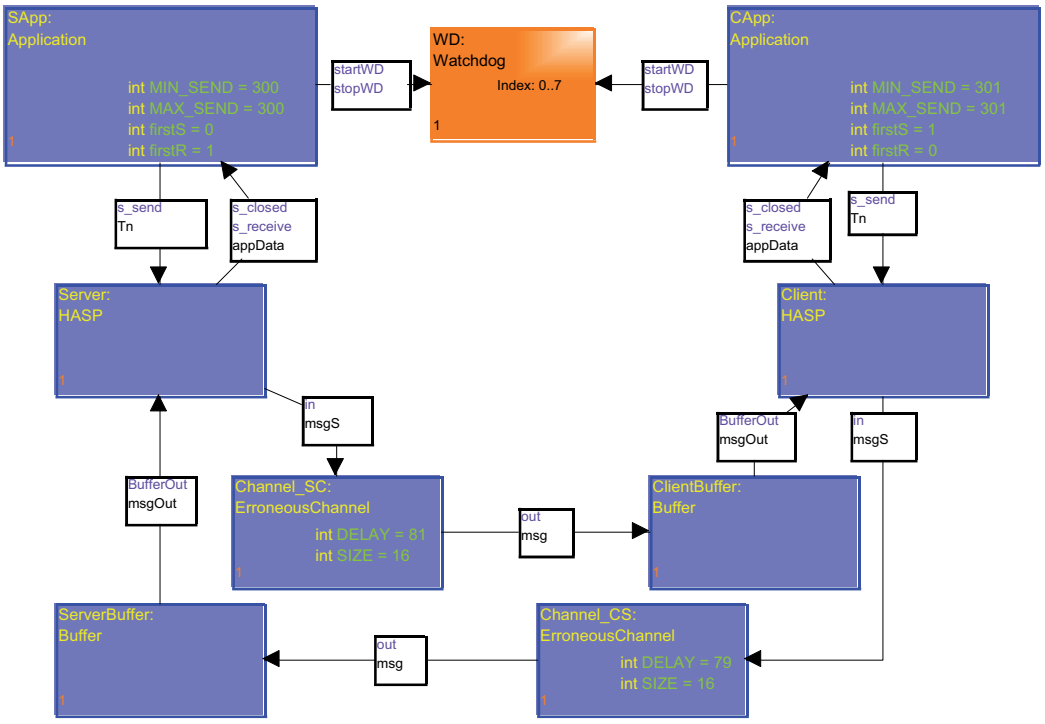


Abbildung B.2: Modell der HASP Architektur

B.3 SAHARA-SCTP Architekturmodell

Das SAHARA-SCTP Architekturmodell (Abbildung B.3 auf Seite 135) gehört zur dritten Fallstudie, die in Kapitel 5.3 auf Seite 113 aufgeführt ist.

```
// Global declarations
const int SEQMAX = 16;
const int MAX_Tn = 16;
const int TIME_OUT = 1000;
const int SAFETY_TIME = 1000;
const int DROP_SEQ = 4; //Start Seq of Drop-Border
const int MAX_DATAc = 2; //Maximum Data-Chunks in SCTP_Packet
```

```
8  const int B_TIMER = 20; //Bundling Timer
9  const int T3 = 100; //Retransmit Timeout
10 const int ACK_TIMER = 80; //Timeout for sending Ack
11 //Typedefinitions
12 typedef struct{
13 int[0,SEQMAX] seq;
14 int[0,SEQMAX] cseq;
15 int[-1,MAX_Tn] data;
16 int[0,SEQMAX] cts;
17 int[0,SEQMAX] ts;
18 int[1,6] type;
19 }SafetyMsg;
20 typedef struct{
21 bool TSNAck[SEQMAX+1];
22 bool duplicateTSN[SEQMAX+1];
23 }AckChunk;
24 typedef struct{
25 int[0,MAX_DATAc] numOfData;//Number of containing DataChunks
26 bool includedTSN[SEQMAX+1];
27 SafetyMsg dataChunk[SEQMAX+1];
28 bool usedAckChunk;//If true, then Packet contains Ack Block
29 AckChunk ackChunk;
30 }SCTPmsg;
```

Listing B.3: Globale Deklarationen der SAHARA-SCTP Architektur

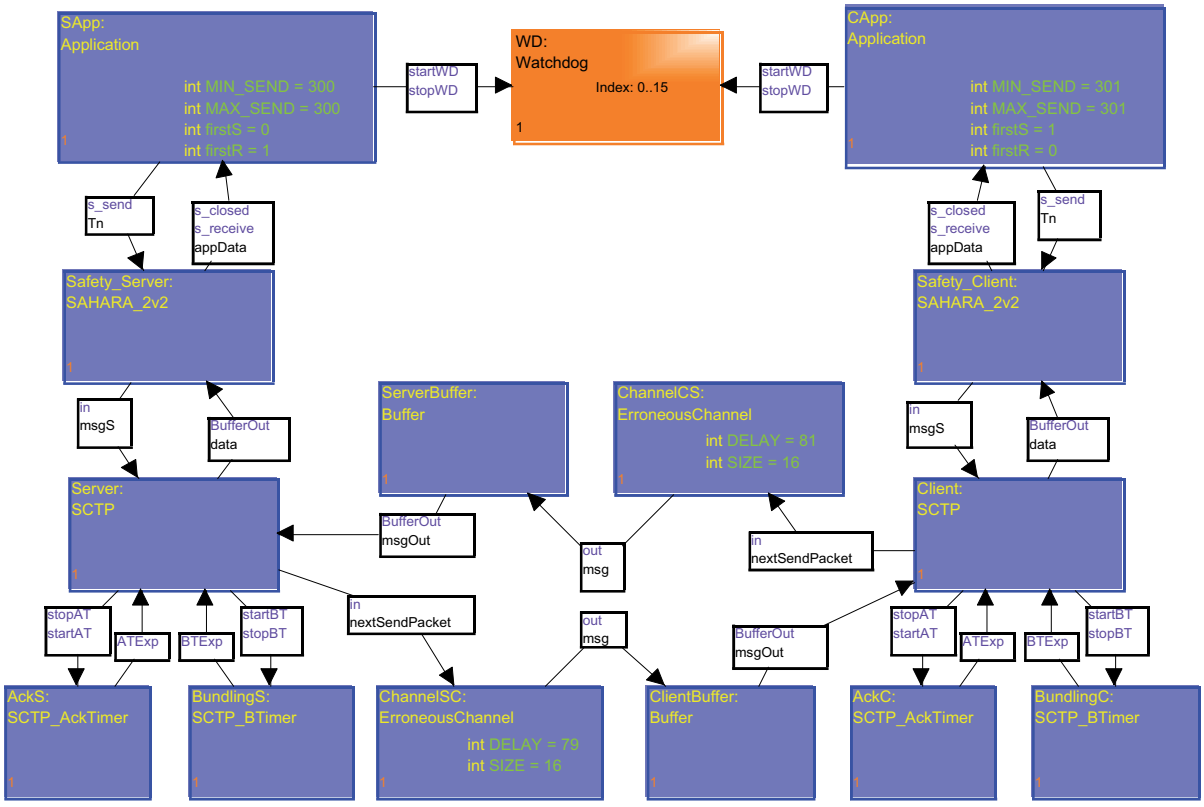


Abbildung B.3: Modell der SCTP Architektur

Anhang C

Prozessmodelle

C.1 Prozess Watchdog

Die Abbildung C.1 zeigt das Prozessmodell Watchdog. Die Konstante *Index* wird in der System-Ebene festgelegt. Jeder instanziierte Watchdog Prozess besitzt einen eindeutigen Index im Bereich $0, \ldots, n$.

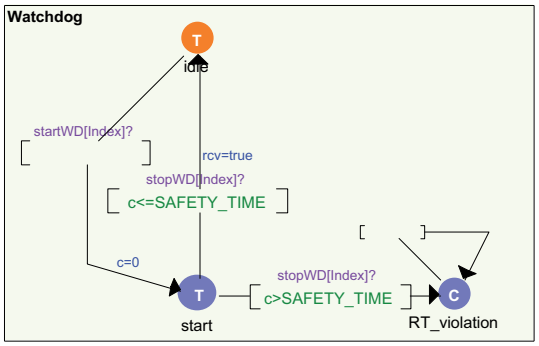


Abbildung C.1: Modell des Prozesses Application

```
1 //Variable declaration
2 clock c;
3 bool rcv=false;
```

Listing C.1: UPPAAL Code Listing des Prozesses Watchdog

C.2 Prozess Application

Die Abbildung C.2 auf der nächsten Seite zeigt das Prozessmodell Application. Die Parameter sind konstante Werte, die in der System-Ebene beim Instanziieren des Prozessen festgelegt werden.

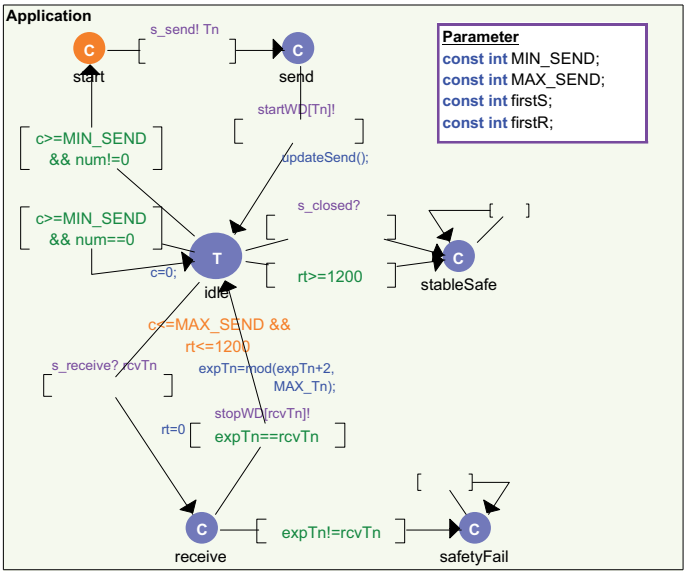


Abbildung C.2: Modell des Prozesses Application

```
1  //Variable declaration
2  int[0,MAX_Tn] Tn=firstS;
3  int[0,MAX_Tn] expTn=firstR;
4  int[0,MAX_Tn] rcvTn=0;
5  clock c;
6  clock rt;
7  int num=-1;
8  //=================
9  void updateSend(){
10     Tn=mod(Tn+2, MAX_Tn);
11     c=0;
12     if(num > 0){
13         num--;
14     }
15 }
```

Listing C.2: UPPAAL Code Listing des Prozesses Application

C.3 Prozess Buffer

Der Prozess Buffer (Abbildung C.3 auf der nächsten Seite) dient zum Speichern von Daten des Types *SafetyMsg*. Für das SAHARA-SCTP Modell ist dieser durch den Typ *SCTPmsg* ausgetauscht. Hierbei ist eine noch zu behebende Schwäche des UPPAAL Model Checkers zu beachten: Funktionen können keine Structs zurückgeben, wenn darin Arrays oder Structs

enthalten sind. Die UPPAAL Version 4.1.3 meldet dann einen Syntaxfehler, womit die Funktionen aufgelöst und die Anweisungen direkt in den Transitionen als Update definiert werden müssen.

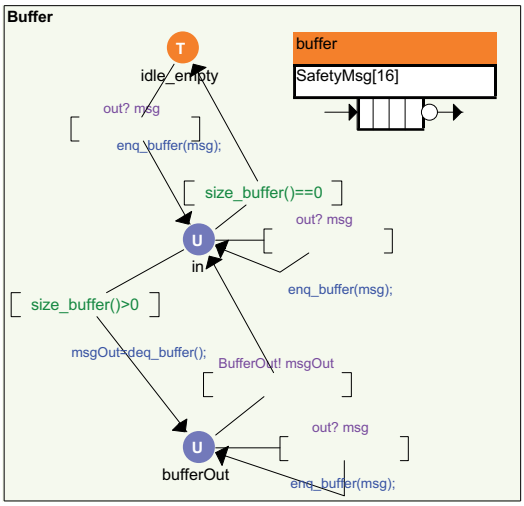


Abbildung C.3: Modell des Prozesses Buffer

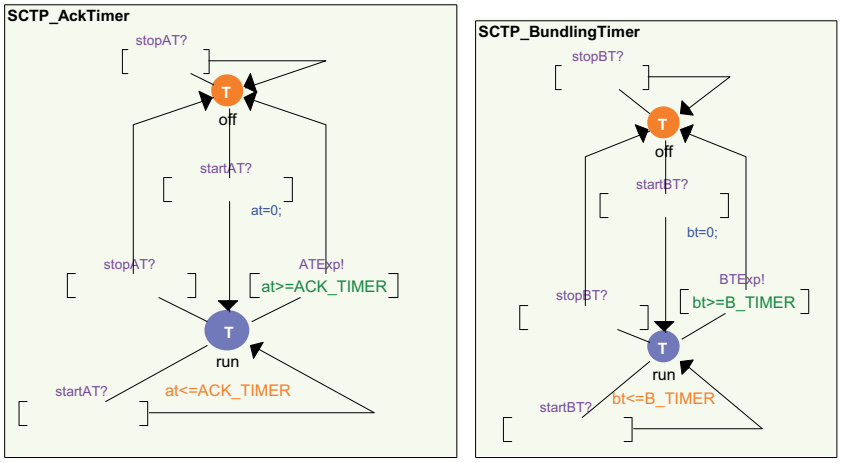
```
1  //Variable declaration
2  SafetyMsg msg;
3  SafetyMsg msgOut;
4
5  SafetyMsg buffer[16];
6  int cnt_buffer = 0; //FiFo Pointer
7  bool ofFlag_buffer = false; //Overflow Flag
8  bool ufFlag_buffer = false; //Underflow Flag
9
10 void enq_buffer(SafetyMsg e){
11     if(cnt_buffer<16){
12         buffer[cnt_buffer] = e;
13         cnt_buffer++;
14     }else{
15         ofFlag_buffer = true;
16     }
17 }
18 SafetyMsg deq_buffer(){
19     int i=0;
20     SafetyMsg ret = buffer[0];
21     if(cnt_buffer == 0){
22         ufFlag_buffer = true;
```

```
23      }else{
24          for(i=0; i<cnt_buffer-1; i++){
25              buffer[i]=buffer[i+1];
26          }       cnt_buffer = max(0, cnt_buffer-1);
27      }
28      return ret;
29  }
30  int size_buffer(){
31      return cnt_buffer;
32  }
```

Listing C.3: UPPAAL Code Listing des Prozesses Buffer

C.4 Prozess SCTP_AckTimer und SCTP_BundlingTimer

Die Abbildung C.4 zeigt die Prozessmodelle SCTP_AckTimer und SCTP_BundlingTimer. Diese Prozesse deklarieren jeweils lokal die Clock *c*.



(a) AckTimer (b) BundlingTimer

Abbildung C.4: Modell der Timer Prozesse von SCTP

C.5 Prozessmodell des HASP Protokolls

Die folgende Abbildung C.5 auf der nächsten Seite zeigt das Prozessmodell des Hybrid Acknowledge Safety Protocol aus der zweiten Fallstudie.

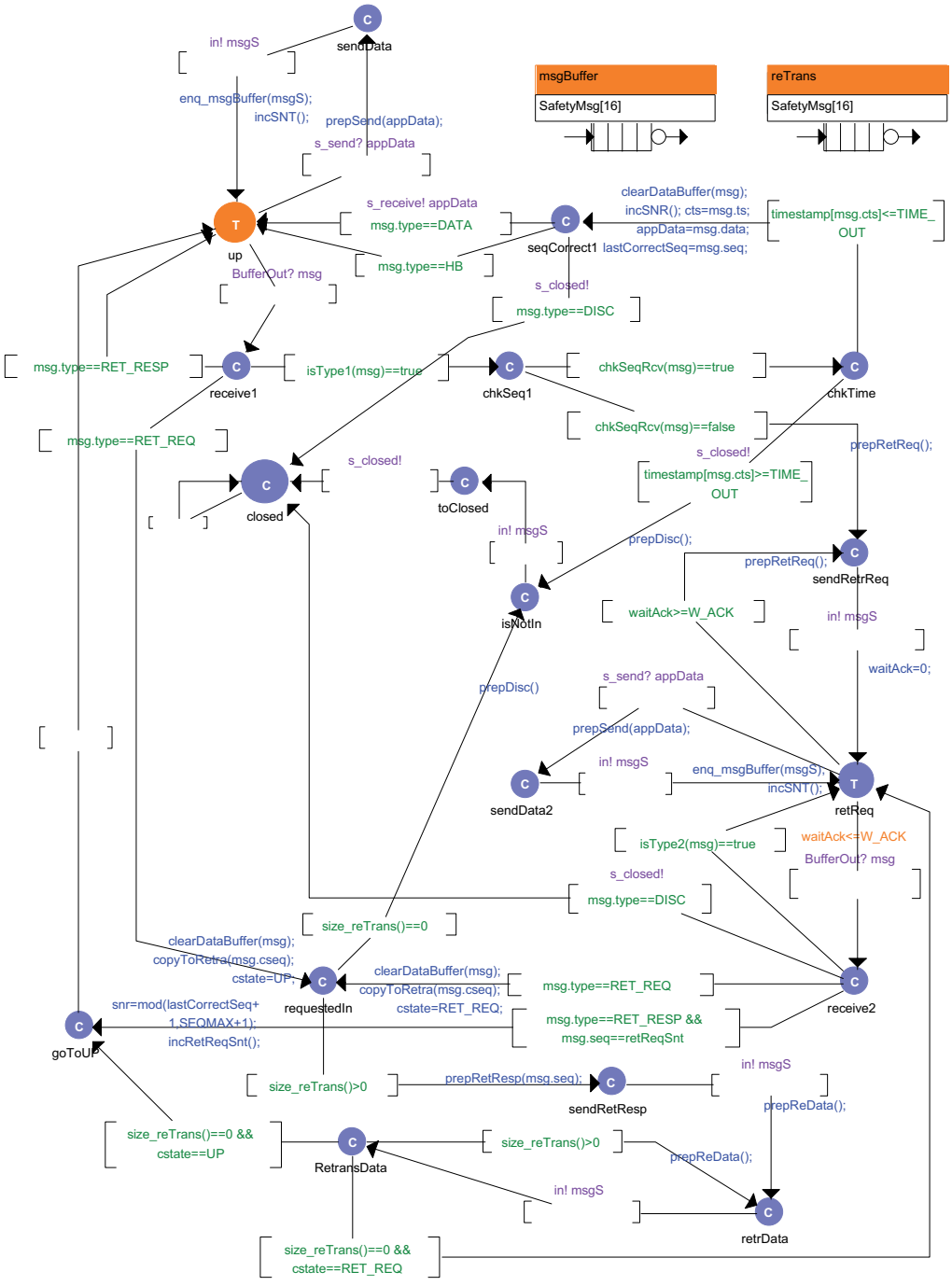


Abbildung C.5: Modell des Porokolls HASP

```
1 //Variable declaration
2 SafetyMsg msg;
3 SafetyMsg msgS;
4 int[0,SEQMAX] lastCorrectSeq=SEQMAX;
```

```
5  int[0,SEQMAX] retReqSnt=0;
6  int[0,SEQMAX] snt=0;
7  int[0,SEQMAX] snr=0;
8  clock timestamp[SEQMAX+1];
9  int[0,SEQMAX] cts=0;
10 int[0,SEQMAX] ts;
11 int cstate=0;//After Retransmission, back to UP or RetRequest
12 int appData;
13 clock waitAck;
14
15 SafetyMsg msgBuffer[16];
16 int cnt_msgBuffer = 0; //FiFo Pointer
17 bool ofFlag_msgBuffer = false; //Overflow Flag
18 bool ufFlag_msgBuffer = false; //Underflow Flag
19
20 void enq_msgBuffer(SafetyMsg e){
21    if(cnt_msgBuffer<16){
22       msgBuffer[cnt_msgBuffer] = e;
23       cnt_msgBuffer++;
24    }else{
25       ofFlag_msgBuffer = true;
26    }
27 }
28 SafetyMsg deq_msgBuffer(){
29    int i=0;
30    SafetyMsg ret = msgBuffer[0];
31    if(cnt_msgBuffer == 0){
32       ufFlag_msgBuffer = true;
33    }else{
34       for(i=0; i<cnt_msgBuffer-1; i++){
35          msgBuffer[i]=msgBuffer[i+1];
36       }       cnt_msgBuffer = max(0, cnt_msgBuffer-1);
37    }
38    return ret;
39 }
40 int size_msgBuffer(){
41    return cnt_msgBuffer;
42 }
43
44 SafetyMsg reTrans[16];
45 int cnt_reTrans = 0; //FiFo Pointer
```

```
46 bool ofFlag_reTrans = false; //Overflow Flag
47 bool ufFlag_reTrans = false; //Underflow Flag
48
49 void enq_reTrans(SafetyMsg e){
50    if(cnt_reTrans<16){
51       reTrans[cnt_reTrans] = e;
52       cnt_reTrans++;
53    }else{
54       ofFlag_reTrans = true;
55    }
56 }
57 SafetyMsg deq_reTrans(){
58    int i=0;
59    SafetyMsg ret = reTrans[0];
60    if(cnt_reTrans == 0){
61       ufFlag_reTrans = true;
62    }else{
63       for(i=0; i<cnt_reTrans-1; i++){
64          reTrans[i]=reTrans[i+1];
65       }       cnt_reTrans = max(0, cnt_reTrans-1);
66    }
67    return ret;
68 }
69 void front_reTrans(SafetyMsg e){
70    int i=0;    if(cnt_reTrans<16){
71       for(i=cnt_reTrans; i>0; i--){
72          reTrans[i]=reTrans[i-1];
73       }
74       reTrans[0] = e;
75       cnt_reTrans++;
76    }
77 }
78 SafetyMsg tail_reTrans(){
79    cnt_reTrans = max(0, cnt_reTrans-1);
80    return reTrans[cnt_reTrans];
81 }
82 int size_reTrans(){
83    return cnt_reTrans;
84 }
85 //================
86 const int UP = 0;
```

```
87  const int DATA = 1;
88  const int HB = 2;
89  const int RET_REQ = 3;
90  const int RET_RESP = 4;
91  const int RET_DATA = 5;
92  const int DISC = 6;
93
94  void prepSend(int data){
95      msgS.type = DATA;
96      msgS.seq = snt;
97      msgS.cseq=lastCorrectSeq;
98      msgS.data = data;
99      msgS.ts = ts;
100     msgS.cts = cts;
101     timestamp[ts] = 0;
102     ts = mod(ts+1,SEQMAX+1);
103 }
104 void prepDisc(){
105     msgS.type = DISC;
106     msgS.seq = snt;
107     msgS.cseq=lastCorrectSeq;
108     msgS.data = -1;
109     msgS.ts = ts;
110     msgS.cts= cts;
111     timestamp[ts] = 0;
112     ts = mod(ts+1,SEQMAX+1);
113 }
114 void prepRetReq(){
115     msgS.type = RET_REQ;
116     msgS.seq = retReqSnt;
117     msgS.cseq=lastCorrectSeq;
118     msgS.data = -1;
119     msgS.ts = ts;
120     msgS.cts= cts;
121     timestamp[ts] = 0;
122     ts = mod(ts+1,SEQMAX+1);
123 }
124 void prepRetResp(int seqReply){
125     msgS.type = RET_RESP;
126     msgS.seq = seqReply;
127     msgS.cseq=lastCorrectSeq;
```

```
128     msgS.data = -1;
129     msgS.ts = ts;
130     msgS.cts= cts;
131     timestamp[ts] = 0;
132     ts = mod(ts+1,SEQMAX+1);
133 }
134 void prepReData(){
135     msgS=deq_reTrans();
136     msgS.type = DATA;
137     msgS.cseq=lastCorrectSeq;
138     msgS.ts = ts;
139     timestamp[ts] = 0;
140 }
141 void incSNT(){
142     snt=mod(snt+1,SEQMAX+1);
143 }
144 void incSNR(){
145     snr=mod(snr+1,SEQMAX+1);
146 }
147
148 void incRetReqSnt(){
149     retReqSnt=mod(retReqSnt+1,SEQMAX+1);
150 }
151 //returns the BufferIndex of the
152 //requested Message
153 int isReqInBuffer(int sendSeq){
154     int i=0;
155     while(i<size_msgBuffer()){
156         if(sendSeq== msgBuffer[i].seq){
157             return i;
158         }
159         i++;
160     }
161     return -1;
162 }
163 //Clear all Acknowledged Data up to cseq
164 void clearDataBuffer(SafetyMsg m){
165     int i = isReqInBuffer(m.cseq);
166     while(i>=0){
167         deq_msgBuffer();
168         i--;
```

```
169     }
170 }
171 void copyToRetra(int cseq){
172     int i=0;
173     int startSeq;
174     bool copy = false;
175     startSeq = mod(cseq+1,SEQMAX+1);
176     while(i<size_msgBuffer()){
177         if(msgBuffer[i].seq==startSeq || copy==true){
178             copy=true;
179             enq_reTrans(msgBuffer[i]);
180         }
181         i++;
182     }
183 }
184 bool isType1(SafetyMsg m){
185     if(m.type==DATA || m.type==HB || m.type==DISC){
186         return true;
187     }else{
188         return false;
189     }
190 }
191 bool isType2(SafetyMsg m){
192     if(m.type==DATA || m.type==HB ||
193     (msg.type==RET_RESP && msg.seq!=retReqSnt)){
194         return true;
195     }else{
196         return false;
197     }
198 }
199 bool chkSeqRcv(SafetyMsg m) {
200     if(m.seq == snr){
201         return true;
202     }else{
203         return false;
204     }
205 }
```

Listing C.4: UPPAAL Code Listing des Protokolls HASP

C.6 Prozessmodell des SCTP Protokolls

In Abbildung C.6 ist das Modell des Protokolls SCTP dargestellt, wie es in der dritten Fallstudie beschrieben ist.

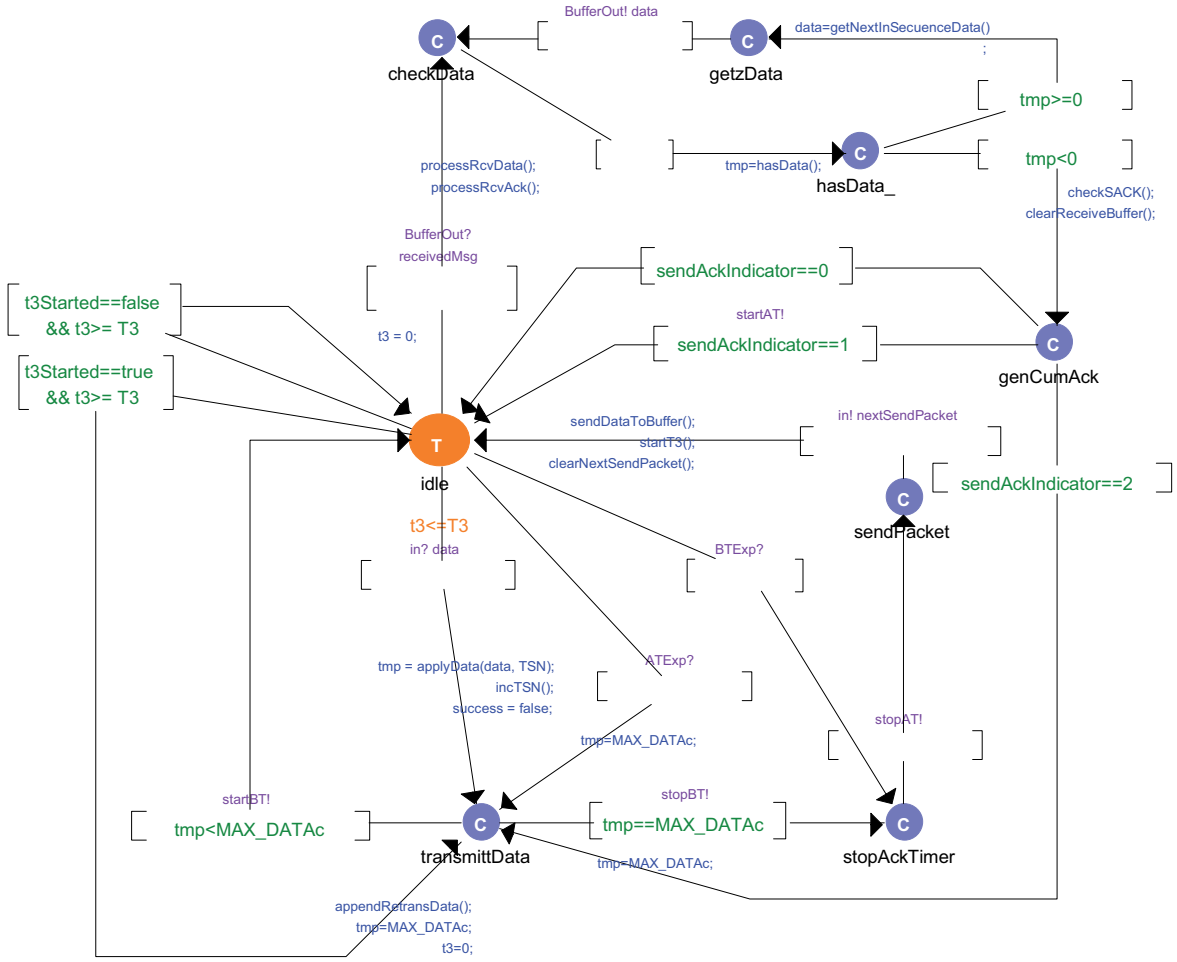


Abbildung C.6: Modell des Porokolls SCTP

```
1  //Variable declaration
2  clock t3;
3  SafetyMsg data;
4  int TSN=0;
5  int[0,SEQMAX] deliveredTSN=SEQMAX;
6  SCTPmsg receivedMsg;
7  SCTPmsg nextSendPacket;
8  int[0,SEQMAX] dupRcvBound=DROP_SEQ;
9  int[0,SEQMAX] lowUnackTSN;
10 int highRcvTSN=0;
```

```
11 bool receiveBufferFlag[SEQMAX+1];
12 bool bufferdData[SEQMAX+1];
13 bool retransMarker[SEQMAX+1];
14 bool ackList[SEQMAX+1];
15 bool success=true;
16 SafetyMsg receiveBufferData[SEQMAX+1];
17 SafetyMsg sendDataChunk[SEQMAX+1];
18 int[0,2] sendAckIndicator;
19 //1: delayed Ack
20 //2: Send Ack now
21 bool bufferOverflow=false;
22 bool t3Started=false;
23 int tmp;
24 //================
25 int applyData(SafetyMsg data, int seqNr){
26     nextSendPacket.includedTSN[seqNr] = true;
27     nextSendPacket.dataChunk[seqNr] = data;
28     nextSendPacket.numOfData++;
29     return nextSendPacket.numOfData;
30 }
31 //================
32 //Increment TSN
33 void incTSN(){
34     TSN = mod(TSN+1, SEQMAX+1);
35 }
36 //================
37 void sendDataToBuffer(){
38     int i=0;
39     for(i=0; i < SEQMAX+1; i++)
40     {
41         if(nextSendPacket.includedTSN[i] == true)
42         {
43             bufferdData[i] = true;
44             sendDataChunk[i] = nextSendPacket.dataChunk[i];
45         }
46     }
47 }
48 //================
49 void clearNextSendPacket(){
50     int i=0;
51     for(i=0; i<=SEQMAX; i++)
```

```
52      {
53          nextSendPacket.includedTSN[i] = false;
54      }
55      for(i=0; i<=SEQMAX; i++)
56      {
57          nextSendPacket.ackChunk.TSNAck[i]=false;
58          nextSendPacket.ackChunk.duplicateTSN[i] = false;
59      }
60      nextSendPacket.usedAckChunk = false;
61      nextSendPacket.numOfData = 0;
62      sendAckIndicator = 0;
63  }
64  //=================
65  void processRcvAck(){
66      int i = 0;
67      //get acked TSNs
68      if(receivedMsg.usedAckChunk==true){
69          i = lowUnackTSN;
70          //set flag from AckChunk of outstanding messages
71          while(i!=TSN){
72              ackList[i] = receivedMsg.ackChunk.TSNAck[i];
73              bufferdData[i] = false;
74              i = mod (i+1, SEQMAX+1);
75          }
76      }
77  }
78  //=================
79  void processRcvData(){
80      int i = 0;
81      int holeDetect = 0;
82  //if there is a hole in received TSN, send a SACK immediately.
83      if(receivedMsg.numOfData>0){
84          sendAckIndicator = min(sendAckIndicator+1,  2);
85          nextSendPacket.usedAckChunk = true; //send ack chunk
86          //TSNs from dupRcvBound to deliveredTSN are duplicate TSNs
87          i = dupRcvBound;
88          while(i != deliveredTSN){
89              if(receivedMsg.includedTSN[i] == true){
90              //here is a duplicate TSN
91                  nextSendPacket.ackChunk.duplicateTSN[i] = true;
92                  //mark duplicate TSN
```

```
93                  sendAckIndicator = 2; //send an ack immediately
94              }
95              i = mod(i+1, SEQMAX+1);
96          }// dont forget i==deliveredTSN
97          if(receivedMsg.includedTSN[i] == true){
98              //here is a duplicate TSN
99              nextSendPacket.ackChunk.duplicateTSN[i] = true;
100             //mark duplicate TSN
101             sendAckIndicator = 2; //send an ack immediately
102         }
103         i = mod(i+1, SEQMAX+1);
104         //now from i up to dupRcvBound are new TSNs
105         while(i != dupRcvBound){
106             if(receivedMsg.includedTSN[i] == true){
107                 nextSendPacket.ackChunk.TSNAck[i] = true;
108                 highRcvTSN = i;
109                 if (holeDetect == 1) { holeDetect = 3;}
110                 if(receiveBufferFlag[i] == false){
111                     receiveBufferData[i] = receivedMsg.dataChunk[i];
112                     receiveBufferFlag[i] = true;
113                 }else{
114                     bufferOverflow = true;
115                 }
116             }else {
117                 if (holeDetect == 0) { holeDetect = 1;}
118             }
119             i = mod(i+1, SEQMAX+1);
120         }
121     }
122     if (holeDetect == 3){
123         sendAckIndicator = 2;
124     }
125 }
126 //==================
127 int hasData(){
128     int next = mod(deliveredTSN+1, SEQMAX+1);
129     if(receiveBufferFlag[next]==true){
130         return next;
131     }else{
132         return -1;
133     }
```

```
134 }
135 //==================
136 SafetyMsg getNextInSecuenceData(){
137     int i = mod(deliveredTSN+1,SEQMAX+1);
138     if(receiveBufferFlag[i] == true){
139         receiveBufferFlag[i]=false;
140         deliveredTSN = mod (deliveredTSN+1, SEQMAX+1);
141         dupRcvBound = mod(dupRcvBound+1, SEQMAX+1);
142         return receiveBufferData[i];
143     }
144     return receiveBufferData[i];
145 }
146 //==================
147 //append all Data, which must be retransmitted
148 void appendRetransData(){
149     int i = lowUnackTSN;
150     while(i != TSN && nextSendPacket.numOfData < MAX_DATAc){
151         if(ackList[i] == false){
152             nextSendPacket.includedTSN[i] = true;
153             nextSendPacket.dataChunk[i] = sendDataChunk[i];
154             nextSendPacket.numOfData = nextSendPacket.numOfData+1;
155         }
156     }
157 }
158 //==================
159 void startT3(){
160     if(t3Started == false && nextSendPacket.numOfData > 0){
161         t3Started = true;
162         t3 = 0;
163     }
164 }
165 //==================
166 //clear the Receive Buffer
167 void clearReceiveBuffer(){
168     int i = highRcvTSN;
169     while (i != dupRcvBound){
170         receiveBufferFlag[i] = false;
171         i = mod(i+1, SEQMAX+1);
172     }
173 }
174 //==================
```

```
175 //check, if there are already unacknowledged TSNs
176 //If not, stop the t3 timer.
177 void checkSACK(){
178     int i = lowUnackTSN;
179 
180     //restart the timer, if sack acknowledge
181     //the lowest outstanding TSN
182     if(ackList[lowUnackTSN] == true){
183         t3Started = true;
184         t3 = 0;
185     }
186     while(i != TSN){
187         if(ackList[i] == true){
188             lowUnackTSN = mod(lowUnackTSN+1, SEQMAX+1);
189             i = mod(i+1, SEQMAX+1);
190         }
191         else{
192             i = TSN; //break condition.
193         }
194     }
195     if (lowUnackTSN == TSN){
196         t3Started = false;
197         success = true;
198     }
199 }
```

Listing C.5: UPPAAL Code Listing des SCTP Modells.

Literaturverzeichnis

[ABF+08] Aceto, Luca ; Baeten, Jos ; Fokkink, Wan ; Ingolfsdottir, Anna ; Nestmann, Uwe: Applying Concurrency Research in Industry Report on a Strategic Workshop. In: *Bulletin of the European Association for Theoretical Computer Science* 94 (2008), S. 113–129

[AD90] Alur, Rajeev ; Dill, David L.: Automata For Modeling Real-Time Systems. In: *ICALP*, 1990, S. 322–335

[AD94] Alur, Rajeev ; Dill, David L.: A Theory of Timed Automata. In: *Theoretical Computer Science* 126 (1994), S. 183–235

[AHM04] Alanen, Jarmo ; Hietikko, Marita ; Malm, Timo: Safety of Digital Communications in Machines / VTT Industrial Systems. 2004. – Forschungsbericht

[AM04] Alur, Rajeev ; Madhusudan, P.: Decision problems for timed automata: A survey. In: Bernardo, Marco (Hrsg.) ; Corradini, Flavio (Hrsg.): *SFM* Bd. LNCS 3185, Springer, 2004, S. 1–24

[BA08] Ben-Ari, Mordechai: *Principles of the Spin Model Checker*. Springer-Verlag London, 2008. – ISBN 1846287693, 9781846287695

[BBSGS06] Braband, Jens ; Brehmke, Bernd-E. ; Stephan Griebel, Harald P. ; Suwe, Karl-Heinz: *Die CENELEC-Normen zur Funktionalen Sicherheit*. Eurailpress Tetzlaff-Hestra GmbH & Co. KG, 2006

[BCC+03] Biere, Armin ; Cimatti, Alessandro ; Clarke, Edmund M. ; Strichman, Ofer ; Zhu, Yunshan: Bounded model checking. In: *Advances in Computers* 58 (2003), S. 118–149

[BDL04] Behrmann, Gerd ; David, Alexandre ; Larsen, Kim G.: A Tutorial on Uppaal. In: Bernardo, Marco (Hrsg.) ; Corradini, Flavio (Hrsg.): *Formal*

Methods for the Design of Real-Time Systems: 4th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM-RT 2004, Springer–Verlag, September 2004 (LNCS 3185), S. 200–236

[BDL+06] Behrmann, Gerd ; David, Alexandre ; Larsen, Kim G. ; Pettersson, Paul ; Yi, Wang ; Hendriks, Martijn: Uppaal 4.0. In: *In Quantitative Evaluation of Systems - (QEST 06)*, IEEE Computer Society, 2006, S. 125–126

[BDM+98] Bozga, Marius ; Daws, Conrado ; Maler, Oded ; Olivero, Alfredo ; Tripakis, Stavros ; Yovine, Sergio: *Kronos: A Model-Checking Tool for Real-Time Systems*. 1998

[BK08] Baier, Christel ; Katoen, Joost-Pieter: *Principles of Model Checking*. The MIT Press, 2008. – ISBN 026202649X

[BL08] Bouyer, Patricia ; Laroussinie, François: Model checking timed automata. In: Merz, Stephan (Hrsg.) ; Navet, Nicolas (Hrsg.): *Modeling and Verification of Real-Time Systems*. John Wiley & Sons, Ltd, 2008. – ISBN 9781847040244, S. 111 – 140

[BMS04] Bluschke, Andreas ; Matthews, Michael ; Schiffel, Reinhard: *Zugangsnetze für die Telekommunikation*. Carl Hanser Verlag München und Wien, 2004

[But04] Buth, Bettina: Analysing Mode Confusion: An Approach Using FDR2. In: *SAFECOMP*, 2004, S. 101–114

[BY04] Bengtsson, Johan ; Yi, Wang: Timed Automata: Semantics, Algorithms and Tools. Version: 2004. http://www.springerlink.com/content/2agtg3kjfle73j5a. In: Reisig, W. (Hrsg.) ; Rozenberg, G. (Hrsg.): *Lecture Notes on Concurrency and Petri Nets* Bd. LNCS 3098. Springer-Verlag, 2004, 87–124

[Car05] Caro, Armando L. Jr.: *End-to-end fault tolerance using transport layer multihoming*. Newark, DE, USA, University of Delaware, Diss., 2005. – Professor In Charge-Amer, Paul D.

[CEN00] CENELEC: *EN 50126. Spezifikation und Nachweis der Zuverlässigkeit, Verfügbarkeit, Instandhaltbarkeit und Sicherheit (RAMS)*. 2000

[CEN01a] CENELEC: *EN 50128. Railway applications - Communications, signalling and processing systems - Software for railway control and protection systems.* 2001

[CEN01b] CENELEC: *EN 50159-1. Railway applications -Communication, signalling and processing systems Part 1: Safety-related communication in closed transmission systems.* 2001

[CEN01c] CENELEC: *EN 50159-2. Railway applications -Communication, signalling and processing systems Part 2: Safety related communication in open transmission systems.* 2001

[CEN03] CENELEC: *EN 50129. Railway applications - Communication, signalling and processing systems - Safety related electronic systems for signalling.* 2003

[CEN10] CENELEC: *EN 50159. Railway applications -Communication, signalling and processing systems - Safety-related communication in transmission systems.* 2010

[Chu08] Chung, Pradeep Myung Jin; M. Myung Jin; Misra (Hrsg.): *Methods for Analyzing Response Times in Networked Automation Systems.* Bd. *17th IFAC World Congress.* 2008

[CKOS04] Clarke, Edmund ; Kroening, Daniel ; Ouaknine, Joël ; Strichman, Ofer: Completeness and complexity of bounded model checking. In: *In Verification, Model Checking, and Abstract Interpretation*, Springer, 2004, S. 85–96

[Cla08] Clarke, Edmund M.: The Birth of Model Checking. In: Grumberg, Orna (Hrsg.) ; Veith, Helmut (Hrsg.): *25 Years of Model Checking* Bd. 5000, Springer, 2008 (Lecture Notes in Computer Science). – ISBN 978–3–540–69849–4, S. 1–26

[de 04] de Wet, Nico: *Model Driven Communication Protocol Engineering and Simulation based Performance Analysis using UML 2.0*, University of Cape Town, Diplomarbeit, 2004

[DF08] Dalgarno, Mark ; Fowler, Matthew: UML vs. Domain-Specific Languages. In: Martinig, Franco (Hrsg.): *Methods & Tools*, 2008, S. 2–8

[DFH+04] Duflot, M. ; Fribourg, L. ; Hérault, T. ; Lassaigne, R. ; Magniette, F. ; Messika, S. ; Peyronnet, S. ; Picaronny, C.: Probabilistic Model

Checking of the CSMA/CD protocol using PRISM and APMC. In: *Proc. 4th Workshop on Automated Verification of Critical Systems (AVoCS'04)* Bd. 128(6), Elsevier Science, 2004 (Electronic Notes in Theoretical Computer Science), S. 195–214

[DILS09] David, A. ; Illum, J. ; Larsen, K.G. ; Skou, A.: Model-Based Framework for Schedulability Analysis Using UPPAAL 4.1. In: [Mos09], S. 93–119

[Din07] Dinculescu, Monica: Probabilistic Temporal Logic or: With What Probability Will the Swedish Chef Bork the Meatballs? (2007). http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.94.8074

[DKN+07] Duflot, M. ; Kwiatkowska, M. ; Norman, G. ; Parker, D. ; Peyronnet, S. ; Picaronny, C. ; Sproston, J.: Practical Applications of Probabilistic Model Checking to Communication Protocols. In: *Handbook of Formal Methods in Industrial Critical Systems*, 2007. – To appear

[DKRT97] D'Argenio, Pedro R. ; Katoen, Joost-Pieter ; Ruys, Theo C. ; Tretmans, G. J.: The Bounded Retransmission Protocol must be on time! In: *THIRD INT. WORKSHOP ON TOOLS AND ALGORITHMS FOR THE CONSTRUCTION AND ANALYSIS OF SYSTEMS (TACAS'97), LNCS 1217*, Springer-Verlag, 1997, S. 416–431

[DT98] Daws, Conrado ; Tripakis, Stavros: Model Checking of Real-Time Reachability Properties Using Abstractions. In: *Proc. of the 4th Workshop on Tools and Algorithms for the Construction and Analysis of Systems, number 1384 in Lecture Notes in Computer Science*, Springer-Verlag, 1998, S. 313–329

[DY00] David, Alexandre ; Yi, Wang: Modelling and Analysis of a Commercial Field Bus Protocol. In: *Proceedings of the 12th Euromicro Conference on Real Time Systems*, IEEE Computer Society, 2000. – ISBN 0–7695–0734–4, S. 165–172

[EF07] Evans, John W. ; Filsfils, Clarence ; wq (Hrsg.): *Deploying IP and MPLS QoS for Multiservice Networks: Theory & Practice*. San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 2007. – ISBN 0123705495

[Fen07] Fendrich, Lothar: *Handbuch Eisenbahninfrastruktur*. Springer-Verlag, Berlin/Heidelberg, 2007

[GBK06] Gröger, M. ; Bachmann, Dr. W. ; Kunz, I.: *Sicheres, hochverfügbares und redundantes SAHARA-Protokoll (Version 2.2)*. 2006

[Gre07] Greifeneder, Jürgen: *Formale Analyse des Zeitverhaltens Netzbasierter Automatisierungssystemen*, Technische Universität Kaiserslautern, Diss., 2007

[Haa08] Haan, Johan den: *DSL in the context of UML and GPL*. http://www.theenterprisearchitect.eu/archive/2008/08/20/dsl-in-the-context-of-uml-and-gpl. Version: 2008

[HFV06] Hendriks, Martijn ; Frits ; Vaandrager, W.: Model Checking Timed Automata - Techniques and Applications. In: *Faculty of Mathematics and Natural Sciences, UL*, 2006

[HNSY92] Henzinger, Thomas A. ; Nicollin, Xavier ; Sifakis, Joseph ; Yovine, Sergio: Symbolic Model Checking for Real-time Systems. In: *Information and Computation* 111 (1992), S. 394–406

[HP02] Haxthausen, A. E. ; Peleska, J.: A Domain Specific Language for Railway Control Systems. In: *Proceedings of the Sixth Biennial World Conference on Integrated Design and Process Technology (IDPT2002)*, 2002

[HP07] Haxthausen, Anne E. ; Peleska, Jan: A Domain-Oriented, Model-Based Approach for Construction and Verification of Railway Control Systems. In: *Formal Methods and Hybrid Real-Time Systems*, 2007, S. 320–348

[JGP99] Jr., Edmund M. C. ; Grumberg, Orna ; Peled, Doron A.: *Model Checking*. The MIT Press, 1999. – ISBN 0262032708

[JJ03] Jain, Madhulika ; Jain, Satish: *Data Communication and Networking*. BPB Publications, 2003. – ISBN 8176564842

[JLS96] Jensen, Henrik E. ; Larsen, Kim G. ; Skou, Arne: Modelling and Analysis of a Collision Avoidance Protocol Using SPIN and UPPAAL. In: *Rutgers University*, 1996, S. 1–20

[Jun05] Jungmaier, Andreas: *Das Transportprotokoll SCTP. Leistungsbewertung und Optimierung eines neuen Transportprotokolls*, Universität Duisburg-Essen, Diss., 2005

[KEY08] KEYMILE: BahnDatennetze. Anforderungen an höchstverfügbare Datennetze der Leit und Sicherungstechnik bei Bahnen / KEYMILE GmbH. 2008. – Forschungsbericht

[Kle09] Kleuker, Stephan: *Formale Modelle der Softwareentwicklung: Model-Checking, Verifikation, Analyse und Simulation.* Vieweg+Teubner Verlag / GWV Fachverlage GmbH, 2009

[KLR96] Kelly, Steven ; Lyytinen, Kalle ; Rossi, Matti: MetaEdit+ A fully configurable multi-user and multi-tool CASE and CAME environment. In: *Advanced Information Systems Engineering* 1080 (1996), S. 1–21

[KNP02] Kwiatkowska, M. ; Norman, G. ; Parker, D.: PRISM: Probabilistic Symbolic Model Checker. In: Field, T. (Hrsg.) ; Harrison, P. (Hrsg.) ; Bradley, J. (Hrsg.) ; Harder, U. (Hrsg.): *Proc. 12th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation (TOOLS'02)* Bd. 2324, Springer, 2002 (LNCS), S. 200–204

[KNP09] Kwiatkowska, M. ; Norman, G. ; Parker, D.: Stochastic Games for Verification of Probabilistic Timed Automata. In: Ouaknine, J. (Hrsg.) ; Vaandrager, F. (Hrsg.): *Proc. 7th International Conference on Formal Modelling and Analysis of Timed Systems (FORMATS'09)* Bd. 5813, Springer, 2009 (LNCS), S. 212–227

[KNP10] Kwiatkowska, M. ; Norman, G. ; Parker, D.: Advances and Challenges of Probabilistic Model Checking. In: *Proc. 48th Annual Allerton Conference on Communication, Control and Computing*, IEEE Press, 2010

[KNPS06] Kwiatkowska, M. ; Norman, G. ; Parker, D. ; Sproston, J.: Performance Analysis of Probabilistic Timed Automata using Digital Clocks. In: *Formal Methods in System Design* 29 (2006), S. 33–78

[KNS02] Kwiatkowska, M. ; Norman, G. ; Sproston, J.: Probabilistic Model Checking of the IEEE 802.11 Wireless Local Area Network Protocol. In: Hermanns, H. (Hrsg.) ; Segala, R. (Hrsg.): *Proc. 2nd Joint International Workshop on Process Algebra and Probabilistic Methods, Performance Modeling and Verification (PAPM/PROBMIV'02)* Bd. 2399, Springer, 2002 (LNCS), S. 169–187

[KNS03] Kwiatkowska, M. ; Norman, G. ; Sproston, J.: Probabilistic Model Checking of Deadline Properties in the IEEE 1394 FireWire Root Contention Protocol. In: *Formal Aspects of Computing* 14 (2003), Nr. 3, S. 295–318

[Koe03] Koenig, Hartmut: *Protocol Engineering. Prinzip, Beschreibung und Entwicklung von Kommunikationsprotokollen.* Teuber Verlag Wiesbaden, 2003

[KT08] Kelly, Steven ; Tolvanen, Juha-Pekka ; wq (Hrsg.): *Domain-Specific Modeling: Enabling Full Code Generation*. John Wiley & Sons, 2008. – ISBN 0470036664

[Lam05] Lamport, Leslie: Real Time is Really Simple / Microsoft Research. 2005 (MSR-TR-2005-30). – Forschungsbericht

[LLPY97] Larsen, Kim ; Larsson, Fredrik ; Pettersson, Paul ; Yi, Wang: Efficient Verification of Real-Time Systems: Compact Data Structure and State–Space Reduction. In: *In Proc. of the 18th IEEE Real-Time Systems Symposium*, IEEE Computer Society Press, 1997, S. 14–24

[LPY97] Larsen, Kim G. ; Pettersson, Paul ; Yi, Wang: Uppaal in a Nutshell. In: *Int. Journal on Software Tools for Technology Transfer* 1 (1997), Oktober, Nr. 1–2, S. 134–152

[MBW08] Mandl, Peter ; Bakomenko, Andreas ; Weiss, Johannes: *Basic course data communication. TCP/IP based communication. Foundations, concepts and standards. (Grundkurs Datenkommunikation. TCP/IP-basierte Kommunikation: Grundlagen, Konzepte und Standards.)*. Studium. Wiesbaden: Vieweg+Teubner., 2008

[MEF] http://metroethernetforum.org

[Men04] Menge, Sebastian: *Stochastische Analyse von Graphtransformationssystemen*. Dortmund, Germany, Lehrstuhl für Software-Technologie, Universität Dortmund, Diplomarbeit, Oktober 2004. – in german

[Met08] MetaCase: *Metaedit+. The Graphical Metamodeling Example. Version 4.5*, 2008

[Met09] *MetaEdit+ Workbench*. http://www.metacase.com/mwb/. Version: 2009

[Mew09] Mewes, Kirsten: *Domain-specifc Modeling of Railway Control Systems with Integrated Verifcation and Validation*, University of Bremen, Diss., 2009

[MM02] Malik, R. ; Mühlfeld, R.: *A Case Study in Verification of UML Statecharts: The Profisafe Protocol*. 2002

[MN08] Merz, Stephan (Hrsg.) ; Navet, Nicolas (Hrsg.): *Modeling and Verification of Real-Time Systems. Formalisms and Software Tools*. ISTE Ltd and John Wiley & Sons, Inc., 2008

[Möl02] Möller, M. O.: *Structure and Hierarchy in Real-Time Systems*, BRICS PhD school, University of Aarhus, Diss., Februar 2002. – available from http://www.verify-it.de/papers.html

[Mos09] Mosterman, Pieter J. (Hrsg.): *Model-Based Design for Embedded Systems*. CRC Press, Taylor & Francis Group, 2009 (Computational Analysis, Synthesis, and Design of Dynamic Models Series)

[MS87] Maxemchuk, Nicholas F. ; Sabnani, Krishan K.: Probabilistic Verification of Communication Protocols. In: *PSTV*, 1987, S. 307–320

[Mur99] Mura, Sigrud: *Einbindung vorhandener Stellwerke in das BZ-Konzept*. 1999

[newa] http://www.juniper.net/company/presscenter/pr/2005/pr-050202.html

[newb] http://www.oebb.at/bau/de/Servicebox/Telekomleistungen/index.jsp

[newc] http://www.h3c.com/portal/About_H3C/News/Corporate_News/200706/206878_1515_0.htm

[OBB] http://www.oebb.at/infrastruktur/de/Servicebox/Telekomleistungen/index.jsp

[Old07] Oldenkamp, H.A.: *Probabilistic model checking : a comparison of tools*, University of Twente (Nl), Diplomarbeit, May 2007. http://essay.utwente.nl/591/

[OP06] Obermaisser, R. ; Peti, P.: A fault hypothesis for integrated architectures. In: *In Proc. of the 4th Int. Workshop on Intelligent Solutions in Embedded Systems*, 2006

[OY02] Ong, L. ; Yoakum, J.: *An Introduction to the Stream Control Transmission Protocol (SCTP)*. RFC 3286 (Informational). http://www.ietf.org/rfc/rfc3286.txt. Version: Mai 2002 (Request for Comments)

[Par02] Parker, David A.: *Implementation of Symbolic Model Checking for Probabilistic Systems*, University of Birmingham, Diss., 2002

[PBH00] Peleska, J. ; Baer, A. ; Haxthausen, A.: Towards Domain-Specific Formal Specification Languages for Railway Control Systems. In: Schnieder,

E. (Hrsg.) ; (eds.), U. B. (Hrsg.): *9th IFAC Symposium on Control in Transportation Systems*, Technical University Braunschweig, Germany, jun 2000, S. 147–152

[PD03] Petcu, Dana ; Dubu, Diana: Parallel and Distributed Computing in Model Checking. In: *CAVIS 2003*, Institute eAustria in Timisoara, 2003

[Pin02] Pinger, R.: *Kompositionale Verifikation nebenläufiger Softwaremodelle durch Model Checking*, Technical University Braunschweig, Diss., February 2002

[Pri] http://www.prismmodelchecker.org/

[PWY+98] Pearson, Justin ; Weise, Carsten ; Yi, Wang ; Behrmann, Gerd ; Behrmann, Gerd ; Larsen, Kim G. ; Larsen, Kim G.: Efficient Timed Reachability Analysis using Clock Difference Diagrams. In: *In Proceedings of the 12th Int. Conf. on Computer Aided Veri*, Springer-Verlag, 1998, S. 341–353

[Rei01] Reinert, Dietmar ; Schaefer, Michael (Hrsg.): *Sichere Bussysteme in der Automation*. Hüthig Verlag, 2001

[RFB01] Ramakrishnan, K. ; Floyd, S. ; Black, D.: *The Addition of Explicit Congestion Notification (ECN) to IP*. RFC 3168 (Proposed Standard). http://www.ietf.org/rfc/rfc3168.txt. Version: September 2001 (Request for Comments)

[RKNP04] Rutten, J. ; Kwiatkowska, M. ; Norman, G. ; Parker, D.: *CRM Monograph Series*. Bd. 23: *Mathematical Techniques for Analyzing Concurrent and Probabilistic Systems,* P. Panangaden and F. van Breugel (eds.). American Mathematical Society, 2004

[RMRHS07] Rahmani, Mehrnoush ; Müller-Rathgeber, Bernd ; Hintermaier, Wolfgang ; Steinbach, Eckehard: Error Detection Capabilities of Automotive Network Technologies and Ethernet - A Comparative Study -. In: *Proceedings of the IEEE Intelligent Vehicles Symposium*. Istanbul, Turkey, Jun 2007

[RSV10] Ravn, A.P. ; Srba, J. ; Vighio, S.: A Formal Analysis of the Web Services Atomic Transaction Protocol with UPPAAL. In: *Proceedings of the 4th International Symposium On Leveraging Applications of Formal Methods, Verification and Validation (ISOLA'10)*, Springer-Verlag, 2010 (LNCS). – To appear.

[SB07] Schlingloff, Friske ; Barthel, Herbert: *Verifikation und Test des PROFIsafe-Sicherheitsprofils*. 2007

[Sch06] Schnell, Gehard ; Schnell, Gerhard (Hrsg.) ; Wiedemann, Bernhard (Hrsg.): *Bussysteme in der Automatisierungs- und Prozesstechnik*. Friedr. Vieweg Sohn Verlag GWV Fachverlage GmbH, Wiesbaden, 2006

[SFR03] Stevens, W. R. ; Fenner, Bill ; Rudoff, Andrew M.: *UNIX Network Programming, Vol. 1*. Pearson Education, 2003. – ISBN 0131411551

[SH93] Schepers, Henk ; Hooman, Jozef: *A Trace-Based Compositional Proof Theory for Fault Tolerant Distributed Systems*. 1993

[Sho02] Shooman, Martin L.: *Reliability of Computer Systems and Networks. Fault Tolerance, Analysis and Design*. John Wiley & Sons, 2002

[SM10] Schiller, Frank ; Mattes, Tina: Residual Error Probability of Embedded CRC by Stochastic Automata. In: Schoitsch, Erwin (Hrsg.): *SAFECOMP* Bd. 6351, Springer, 2010 (Lecture Notes in Computer Science). – ISBN 978–3–642–15650–2, S. 155–168

[SP10] Schulz, Oliver ; Peleska, Jan: Reliability Analysis of Safety-Related Communication Architectures. In: *SAFECOMP*, 2010, S. 1–14

[Ste07] Stewart, R.: *Stream Control Transmission Protocol*. RFC 4960 (Proposed Standard). http://www.ietf.org/rfc/rfc4960.txt. Version: September 2007 (Request for Comments)

[SV04] Sokolova, A. ; Vink, E. P. D.: Probabilistic Automata: System Types, Parallel Composition and Comparison. In: *In Validation of Stochastic Systems: A Guide to Current Research*, 2004, S. 1–43

[SXM+00] Stewart, R. ; Xie, Q. ; Morneault, K. ; Sharp, C. ; Schwarzbauer, H. ; Taylor, T. ; Rytina, I. ; Kalla, M. ; Zhang, L. ; Paxson, V.: *Stream Control Transmission Protocol*. RFC 2960 (Proposed Standard). http://www.ietf.org/rfc/rfc2960.txt. Version: Oktober 2000 (Request for Comments). – Obsoleted by RFC 4960, updated by RFC 3309

[Tan02] Tanenbaum, Andrew S.: *Computer Networks (4th Edition)*. 4. Prentice Hall PTR, 2002. – ISBN 0130661023

[TS06] Thrane, Claus ; Sørensen, Uffe: Slicing and Predicate Abstraction for the Uppaal Modeling Language / Aalborg University, Department of Computer Science. 2006. – Forschungsbericht

[UPP] http://www.cs.aau.dk/~arild/uppaal-probabilistic/

[Wan04] Wang, Farn: Formal verification of timed systems: A survey and perspective. In: *Proceedings of the IEEE*, 2004, S. 2004

[Wan06] Wang, Fuzhi: *Symbolic Implementation of Model-Checking Probabilistic Timed Automata*, University of Birmingham, Diss., 2006

[WHS06] Wang, Chao ; Hachtel, Gary D. ; Somenzi, Fabio: *Abstraction Refinement for Large Scale Model Checking (Series on Integrated Circuits and Systems)*. Secaucus, NJ, USA : Springer-Verlag New York, Inc., 2006. – ISBN 0387341552

[Yov97] Yovine, Sergio: Kronos: A Verification Tool for Real-Time Systems. (Kronos User's Manual Release 2.2). In: *International Journal on Software Tools for Technology Transfer* 1 (1997), S. 123–133