TECHNISCHE
UNIVERSITÄT
DARMSTADT

Entwicklung einer Spezifikationssprache zur modellbasierten Generierung von Security-/ Safety-Monitoren zur Absicherung von (Eingebetteten) Systemen

VOM FACHBEREICH ELEKTRO- UND INFORMATIONSTECHNIK
DER TECHNISCHEN UNIVERSITÄT DARMSTADT
ZUR ERLANGUNG DES AKADEMISCHEN GRADES
EINES DOKTOR-INGENIEURS (DR.-ING.)
GENEHMIGTE DISSERTATION

VON

DIPL.-ING. SVEN PATZINA

GEBOREN AM

01. SEPTEMBER 1981 IN BAD SODEN AM TAUNUS

REFERENT: PROF. DR. RER. NAT. ANDY SCHÜRR
KOREFERENT: PROF. DR. JAN JÜRJENS

TAG DER EINREICHUNG: 31. JANUAR 2014
TAG DER MÜNDLICHEN PRÜFUNG: 10. JULI 2014

D17
DARMSTADT 2014

ERKLÄRUNG

Ich versichere hiermit, dass ich die vorliegende Dissertation allein und nur unter Verwendung der angegebenen Literatur verfasst habe. Die Arbeit hat bisher noch nicht zu Prüfungszwecken gedient.

Darmstadt, 31. Januar 2014

Sven Patzina

If our basic tool,
the language in which we design and code our programs,
is also complicated,
the language itself becomes part of the problem
rather than part of its solution.

— Charles A. R Hoare. [Hoa81]

DANKSAGUNG

Besonderer Dank gilt meinem Doktorvater Herrn Prof. Dr. rer. nat. Andy Schürr, der mich durch das Promotionsverfahren begleitete und durch kritische und inspirierende Fachdiskussionen auf neue Ideen brachte. Herrn Prof. Dr. Jan Jürjens danke ich für die Übernahme des Zweitgutachtens. Meinem Bruder Lars Patzina danke ich insbesondere für die konstruktiven, fachlichen Diskussionen und für das Engagement bei der Zusammenführung unserer beiden Promotionsthemen. Des Weiteren danke ich Thorsten Piper für die Hilfe bei der Anpassung des in dieser Arbeit vorgestellten Entwicklungsprozesses an die AUTOSAR-Methodik. Den Kolleginnen und Kollegen am Fachgebiet Echtzeitsysteme danke für die fachliche, technische und freundschaftliche Unterstützung.

ABSTRACT

Driven by technical innovation, embedded systems are becoming increasingly interconnected. Nowadays, not only PCs communicate by local networks or the Internet, but also mobile devices, such as smart phones and tablets, capture the market. These mobile devices offer a new target for attacks because their limited processing power prevents the direct adaptation of protection mechanisms used within the PC domain. By linking these mobile devices with vehicles and by introducing new connectivity between vehicles and external services, even embedded in-vehicular systems that perform safety-critical tasks are no longer completely isolated from the outside world. Often, little attention has been paid to security mechanisms of these embedded systems, such as encryption and safe component design for protection against attacks. Caused by this development, the embedded systems in vehicles become vulnerable for active and passive attacks from the outside world. Even if safety and security issues are considered during the development of an embedded system, in the majority of cases it is impossible to eliminate all security vulnerabilities and to foresee all possible attacks. Considering complex heterogeneous systems or components, it is often economically or technically infeasible to secure them against external adversaries retroactively. Therefore, systems cannot be considered as safe, either due to unknown vulnerabilities, or due to the required integration of legacy components. To secure such systems, monitoring the system at runtime is proven able to detect attacks that exploit previously unknown errors and security vulnerabilities.

Such a protection is reached by the *Model Based Security and Safety Monitor* (MBSecMon) development process presented in this PhD thesis and in [Pat14]. The MBSecMon-Process integrates seamlessly into existing system development processes and generates automatically efficient safety and security monitors for highly concurrent communication processes based on specifications from the requirement phase.

This PhD thesis focuses on two steps of this development process. The first part of this thesis presents a new graphical, model-based signature description language that is based on the scenario driven approach – the *MBSecMon Specification Language*. This language combines the advantages of existing formalisms by (1) providing all essential concepts for modeling behavioral signatures for highly concurrent communication processes and enables their compact representation. It (2) takes advantage of existing development artifacts in the modeling of the signatures, (3) is located on a higher abstraction level as commonly used languages and distinguishes explicitly between standard behavior and attack patterns and classes. By this explicit distinction and its resemblance to the UML, the language achieves (4) a high comprehensibility for software engineers as well as for non-experts.

The second part of this thesis presents the transformation of the compact specification, which is modeled in the MBSecMon specification language, into the

formal intermediate language Monitor Petri nets. This transformation formalizes, on the one hand, the semantics of the MBSecMon specification language and provides, on the other hand, the automatic transformation in the MBSecMon development process to a more suitable representation for the generation of efficient monitors.

ZUSAMMENFASSUNG

Getrieben durch technische Innovationen gewinnt die Kommunikation zwischen eingebetteten Systemen immer mehr an Bedeutung. So kommunizieren heutzutage nicht nur PCs über lokale Netzwerke oder das Internet, sondern auch mobile Geräte wie Smartphones und Tablets erobern den Markt. Diese bieten aufgrund ihrer geringeren Rechenleistung neue Angriffsflächen, da Sicherungsmaßnahmen der PC-Domäne nicht ohne Anpassung anwendbar sind. Durch die Vernetzung dieser mobilen Geräte mit Fahrzeugen und die Anbindung der Fahrzeuge an externe Dienstleistungen sind selbst eingebettete Systeme im Fahrzeug, die sicherheitskritische Aufgaben erfüllen, nicht mehr vollständig von der Außenwelt abgeschirmt. Bei ihrer Entwicklung wurde jedoch wenig Aufmerksamkeit auf Sicherheitsmechanismen, wie Verschlüsselung und sicheres Komponentendesign, zur Abwehr von Angriffen aus der Außenwelt gelegt. Solche Fahrzeuge sind hierdurch von außen für aktive und passive Angriffe anfällig. Selbst wenn bei einer Neuentwicklung eines eingebetteten Systems großer Wert auf die Absicherung gelegt wird, ist es meist nicht möglich, alle Sicherheitslücken zu eliminieren und jeden möglichen Angriff vorherzusehen. Betrachtet man komplexe heterogene Systeme oder Komponenten und will diese nachträglich absichern, ist dies meistens ökonomisch oder technisch nicht zu realisieren. Resultierend daraus kann bei keinem System davon ausgegangen werden, dass es sicher ist – sei es durch unbekannte Schwachstellen oder der Verwendung von Legacy-Komponenten. Um Systeme dennoch gegen Angriffe, die vorher unbekannte Fehler und Sicherheitslücken ausnutzen, absichern zu können, hat sich die Überwachung eines Systems während der Laufzeit als geeignet herausgestellt.

Eine solche Absicherung wird durch den in dieser Dissertationsschrift und in [Pat14] vorgestellten verständlichen *Model Based Security/Safety Monitor*-Entwicklungsprozess (MBSecMon-Entwicklungsprozess) erreicht, der sich in bestehende modellbasierte Systementwicklungsprozesse nahtlos eingliedert. Dieser MBSecMon-Entwicklungsprozess generiert aus einer in der Anforderungsphase entstandenen Spezifikation automatisch effiziente Sicherheitsmonitore für hoch nebenläufige Kommunikationen.

Diese Arbeit betrachtet zwei Schritte dieses Entwicklungsprozesses. Der erste Teil der Arbeit stellt eine neue auf dem szenariobasierten Design aufbauende graphische, modellbasierte Signaturbeschreibungssprache vor – die *MBSecMon-Spezifikationssprache*. Diese Sprache vereinigt die Vorteile bestehender Formalismen, indem sie (1) alle wichtigen Konzepte zur Modellierung von verhaltensbeschreibenden Signaturen für hoch nebenläufige Kommunikationsabläufe unterstützt und diese kompakt repräsentieren kann. Sie bezieht (2) bestehende Entwicklungsartefakte des Systementwicklungsprozesses in die Modellierung ein, (3) befindet sich auf einer höheren Abstraktionsebene als üblicherweise zur Spezifikation eingesetzte Sprachen und unterscheidet explizit zwischen Normalverhalten und bekannten Angriffsmustern und -klassen. Durch diese Unterscheidung und durch

Nähe der Sprache zur UML wird (4) eine hohe Verständlichkeit der Spezifikation sowohl für den Softwaretechniker als auch für Nicht-Experten erreicht.

Den zweiten Teil dieser Arbeit bildet die Abbildung der sehr kompakten in der MBSecMon-Spezifikationssprache verfassten Spezifikationen in die formale Zwischensprache Monitor-Petrinetze [Pat14]. Hierdurch wird zum einen die Semantik der MBSecMon-Spezifikationssprache formalisiert und zum anderen der im MBSecMon-Entwicklungsprozess eingesetzte automatische Übergang in eine für die Generierung effizienter Monitore besser geeignete Repräsentation realisiert.

VERÖFFENTLICHUNGEN

Einige in dieser Arbeit vorgestellten Ansätze wurden bereits in folgenden Veröffentlichungen publiziert:

[PPPS10] Patzina, Lars; Patzina, Sven; Piper, Thorsten; Schürr, Andy: Monitor Petri Nets for Security Monitoring. In: *Proceedings of the International Workshop on Security and Dependability for Resource Constrained Embedded Systems (S&D4RCES 2010)*, ACM, 2010 (S&D4RCES). – ISBN 978–1–4503–0368–2, S. 3:1–3:6

[BPP+11] Biedermann, Alexander; Piper, Thorsten; Patzina, Lars; Patzina, Sven; Huss, Sorin A.; Schürr, Andy ; Suri, Neeraj: Enhancing FPGA Robustness via Generic Monitoring Cores. In: *1st International Conference on Pervasive and Embedded Computing and Communication Systems (PECCS 2011)*, 2011, S. 379–386

[ALPS11] Anjorin, Anthony; Lauder, Marius; Patzina, Sven; Schürr, Andy: eMoflon: Leveraging EMF and Professional CASE Tools. In: *3. Workshop Methodische Entwicklung von Modellierungswerkzeugen (MEMWe 2011)*. Bonn : Gesellschaft für Informatik, 2011 (Lecture Notes in Informatics) (Zitiert auf Seite 180.)

[PPS11] Patzina, Sven; Patzina, Lars; Schürr, Andy: Extending LSCs for Behavioral Signature Modeling. In: Camenisch, Jan; Fischer-Hübner, Simone; Murayama, Yuko; Portmann, Armand; Rieder, Carlos (Hrsg.): *Future Challenges in Security and Privacy for Academia and Industry (IFIP SEC 2011)* Bd. 354 Springer, Springer, June 2011 (IFIP Advances in Information and Communication Technology). – ISBN 978–3–642–21423–3, S. 293–304 (Zitiert auf Seite 74 und 97.)

[PP12] Patzina, Sven; Patzina, Lars: A Case Study Based Comparison of ATL and SDM. In: Schürr, Andy; Varró, Dániel; Varró, Gergely (Hrsg.): *Applications of Graph Transformations with Industrial Relevance (AGTIVE 2012)* Bd. 7233, Springer, 2012 (LNCS). – ISBN 978–3–642–34175–5, S. 210–221 (Zitiert auf Seite 175.)

[PPPM13] Patzina, Lars; Patzina, Sven; Piper, Thorsten; Manns, Paul: Model-Based Generation of Run-Time Monitors for AUTOSAR. Version: 2013. http://dx.doi.org/10.1007/978-3-642-39013-5_6. In: Gorp, Pieter; Ritter, Tom; Rose, Louis M. (Hrsg.): *Modelling Foundations and Applications (ECMFA 2013)* Bd. 7949. DOI. – 10.1007/978–3–642–39013– 5_6. – ISBN 978–3–642–39012–8, S. 70–85. – Best Paper Arward (Zitiert auf Seite 195 und 196.)

INHALTSVERZEICHNIS

I Einleitung 1

1 Einleitung 3
- 1.1 Motivation . . . 4
- 1.2 MBSecMon-Entwicklungsprozess . . . 7
  - 1.2.1 Der MBSecMon-Prozess . . . 9
  - 1.2.2 Abgrenzung zu vorhandenen Ansätzen . . . 10
  - 1.2.3 Sicherheitsmodell und Gültigkeit des Ansatzes . . . 12
  - 1.2.4 Security im Rahmen der Arbeit . . . 13
- 1.3 Beitrag . . . 15
- 1.4 Überblick . . . 18
- 1.5 Durchgängiges Beispiel: Car2X-Mautbrückenszenario . . . 18
  - 1.5.1 Intention des CARDME-Protokolls . . . 19
  - 1.5.2 Kommunikationsphasen des CARDME-Protokolls . . . 20

II Allgemeine Grundlagen 23

2 Modellierungssprachen 25
- 2.1 Strukturdiagramme . . . 25
  - 2.1.1 UML2-Klassendiagramme . . . 26
  - 2.1.2 UML2-Komponentendiagramme . . . 28
- 2.2 Metamodellierung . . . 30
  - 2.2.1 Die Meta Object Facility der OMG . . . 30
  - 2.2.2 UML-Profile . . . 32
  - 2.2.3 EMF und das Ecore-Metamodell . . . 33
- 2.3 Transitionssysteme . . . 34
  - 2.3.1 Petrinetze . . . 35
  - 2.3.2 Monitor-Petrinetze . . . 37

III MBSecMon-Spezifikationssprache 45

3 Grundlagen und Verwandte Arbeiten 47
- 3.1 Anwendungsfalldiagramme . . . 48
  - 3.1.1 Use-Case-Diagramme . . . 48
  - 3.1.2 Misuse-Case-Diagramme . . . 52
- 3.2 Sequenzdiagramme . . . 54
  - 3.2.1 Message Sequence Charts . . . 55
  - 3.2.2 Eignung der MSCs zur Signaturbeschreibung . . . 60
  - 3.2.3 UML2-Sequenzdiagramme . . . 60
  - 3.2.4 Eignung der UML2-SD zur Signaturbeschreibung . . . 64
- 3.3 Verwandte Arbeiten (Signaturspezifikationssprachen) . . . 66
  - 3.3.1 Verhaltensbeschreibende Spezifikationssprachen . . . 66
  - 3.3.2 Eignung der Formalismen zur Signaturbeschreibung . . . 71
  - 3.3.3 Strukturierung der Spezifikation . . . 72
  - 3.3.4 Eignung der Ansätze zur Signaturstrukturierung . . . 73

4 Einsetzbarkeit der LSCs als verhaltensbeschreibende Signatursprache 75
4.1 Live Sequence Charts . . . 75
4.2 Anforderungen an den MBSecMon-Entwicklungsprozess . . . 85
4.3 LSCs als verhaltensbeschreibende Signatursprache . . . 87
5 Die MBSecMon-Spezifikationssprache 97
5.1 Erweiterte Live Sequence Charts . . . 97
5.1.1 Anpassungen und Erweiterungen der Syntax und Semantik 98
5.1.2 Die abstrakte Syntax der eLSCs . . . 104
5.2 Use-/Misuse-Case-Sprache (MUC-Sprache) . . . 105
5.2.1 Die konkrete Syntax und Semantik der MUC-Sprache . . . 107
5.2.2 Die abstrakte Syntax der MUC-Sprache . . . 115
5.2.3 Zusammenfassung: Einbettung von eLSCs in die MUC-Sprache . . . 116
6 Der Editor (Enterprise Architect Add-In) 119
6.1 MBSecMonSL-Editor . . . 119
6.1.1 MBSecMon-Profile . . . 119
6.1.2 Das MBSecMon-EA-Add-in . . . 124
6.2 Export in ein auf EMF basierendes XMI-Format . . . 126

IV MBSecMonSL zu Monitor-Petrinetze 129
7 Transformationen von Sequenzdiagrammen zu Automaten 131
7.1 Transformationssprachen . . . 132
7.1.1 Atlas Transformation Language . . . 132
7.1.2 Story Driven Modeling . . . 135
7.2 Generierung von Monitoren aus Spezifikationssprachen . . . 138
8 MBSecMonSL zu Monitor-Petrinetze 143
8.1 Abbildung der eLSC-Sprache in konkreter Syntax . . . 143
8.1.1 Grundlegende Elemente . . . 143
8.1.2 Kombinationen von heißen und kalten Nachrichten . . . 148
8.1.3 Komplexe Elemente zur Kontrollflusssteuerung . . . 154
8.1.4 Chartplätze für Verbotene und Ignorierte Elemente . . . 159
8.2 Abbildung der MUC-Sprache in konkreter Syntax . . . 162
8.2.1 «include»-Beziehung . . . 162
8.2.2 «extend»-Beziehung . . . 164
8.2.3 «threaten»-Beziehung . . . 166
8.2.4 «mitigate»-Beziehung . . . 168
8.3 Kaskadierung der Beziehungen der MUC-Sprache in konkreter Syntax . . . 170
8.4 Modellierungsrichtlinien der MBSecMonSL . . . 171
9 Konkrete Realisierung der Transformation 175
9.1 Auswahl der Transformationssprache . . . 175
9.1.1 Anforderungen an die Transformationssprache . . . 176
9.1.2 Vergleich der Transformationssprachen . . . 179
9.2 Konkrete Realisierung der Transformation mit SDM und ATL . . . 180
9.2.1 Ergebnisse für die Implementierung der Transformation . . . 188

9.2.2 Implementierung der MBSecMonSL-zu-MPN-Transformation . . . 189

V Evaluation 191

10 Implementierung und Einbindung in einen Entwicklungsprozess 193
10.1 MBSecMon-Toolsuite . . . 193
10.2 AUTOSAR-Integration der MBSecMon-Toolsuite . . . 195
10.2.1 AUTOSAR . . . 196
10.2.2 Der angepasste Monitorgenerierungsprozess für AUTOSAR 196
10.2.3 Beispiel: Automatische Getriebesteuerung . . . 198
10.2.4 Einsetzbarkeit des MBSecMon-Entwicklungsprozesses . . . 199
10.2.5 Zusammenfassung . . . 203

11 Evaluation der Transformation 205
11.1 Vergleich syntaktischer Elemente der beiden Repräsentationen . . . 205
11.2 Vergleich der Laufzeit der Transformationen in ATL und SDM . . . 209

VI Zusammenfassung 215

12 Zusammenfassung 217
12.1 Erreichte Ziele der Arbeit . . . 218
12.2 Zukünftige Arbeiten . . . 220

Literaturverzeichnis 223

Curriculum Vitae 239

ABBILDUNGSVERZEICHNIS

Abbildung 1.1 Der MBSecMon-Prozess im V-Modell . . . 8
Abbildung 1.2 Der MBSecMon-Prozess . . . 10
Abbildung 1.3 Das Sicherheitsmodell des MBSecMon-Prozesses . . . 13
Abbildung 1.4 Einordnung der Beiträge . . . 16
Abbildung 1.5 Aufbau der Thesis . . . 18
Abbildung 1.6 Domänen des CARDME-Konzepts . . . 19
Abbildung 1.7 Instanzen des CARDME-Konzepts . . . 20
Abbildung 2.1 Transformationsmetamodell eLSC-zu-MPN (Ausschnitt) . . . 26
Abbildung 2.2 CARDME-Beispiel als UML2-Komponentendiagramm . . . 29
Abbildung 2.3 Die Vierschichtenarchitektur der UML . . . 31
Abbildung 2.4 Stereotypen für eLSC-Lebenslinien . . . 33
Abbildung 2.5 Netzstruktur und der Vor- und Nachbereich der Transition 35
Abbildung 2.6 Schalten und Markierung der Petrinetze . . . 36
Abbildung 2.7 Gewichtete Kanten in Petrinetzen . . . 36
Abbildung 2.8 Konflikte in Petrinetzen . . . 37
Abbildung 2.9 Elemente der Monitor-Petrinetze . . . 38
Abbildung 2.10 Makroschritte der MPNs . . . 39
Abbildung 2.11 Eingabegenerationen der MPNs . . . 40
Abbildung 2.12 Subgenerationen der MPNs . . . 41
Abbildung 3.1 MBSecMon-Prozess: Die Monitorspezifikationssprache . . . 47
Abbildung 3.2 CARDME-Protokoll als UML2-Use-Case-Diagramm . . . 49
Abbildung 3.3 CARDME-Protokoll als Misuse-Case-Diagramm . . . 53
Abbildung 3.4 CARDME-Protokoll als Message Sequence Chart . . . 56
Abbildung 3.5 Vereinfachtes CARDME-Protokoll als HMSC . . . 58
Abbildung 3.6 Vereinfachtes MSC-Dokument des CARDME-Protokolls . . . 59
Abbildung 3.7 CARDME-Protokoll als UML2-Sequenzdiagramm . . . 61
Abbildung 3.8 Kategorien der Ereignissequenzen in der MSC- und UML2-Semantik . . . 65
Abbildung 4.1 CARDME-Protokoll als LSC-Diagramm . . . 76
Abbildung 4.2 Anti-Szenarien des CARDME-Protokolls als LSC-Diagramm 81
Abbildung 4.3 Car2X-Beispielszenario – Systemarchitektur . . . 89
Abbildung 4.4 Muster zur Signaturbeschreibung als LSCs (Typ und Reihenfolge) . . . 90
Abbildung 4.5 Muster zur Signaturbeschreibung als LSCs (Kontrollfluss) . . . 92
Abbildung 5.1 Muster zur Signaturbeschreibung als eLSCs . . . 98
Abbildung 5.2 Metamodell der eLSC-Sprache . . . 104
Abbildung 5.3 eLSC-Signatur zum Einsatz des CARDME-Protokolls zum Parkraummanagement . . . 106
Abbildung 5.4 CARDME-Protokoll strukturiert mit MUC-Sprache . . . 108
Abbildung 5.5 eLSC-Signatur des CARDME-Protokolls zur Gebührenerfassung durch die Mautbrücke . . . 109

Abbildung 5.6 Erlaubte Beziehungen zwischen Use- und Misuse-Cases in konkreter Syntax . . . 111
Abbildung 5.7 Angriffssignaturen des CARDME-Protokolls zur Gebührenerfassung durch die Mautbrücke als eLSC . . . 114
Abbildung 5.8 Metamodell der MUC-Sprache . . . 115
Abbildung 6.1 Struktur der UML-Profile der MBSecMonSL . . . 120
Abbildung 6.2 Das Profil der speziellen eLSC-Elemente . . . 121
Abbildung 6.3 Das Profil des eLSC-Diagramms . . . 122
Abbildung 6.4 Das Profil der eLSC-Toolbox . . . 122
Abbildung 6.5 Das Profil der speziellen MUC-Elemente . . . 123
Abbildung 6.6 Das Profil des MUC-Diagramms . . . 123
Abbildung 6.7 Das Profil der MUC-Toolbox . . . 124
Abbildung 6.8 Export der MBSecMonSL-Signaturen aus EA . . . 126
Abbildung 7.1 MBSecMon-Prozess: Übersetzung in die Zwischensprache . 131
Abbildung 7.2 Transformationsmetamodell der SDM-Transformation . . . 136
Abbildung 7.3 Teilausschnitt aus der SDM-Transformation eLSC zu MPN . 137
Abbildung 8.1 MPN-Repräsentation der eLSC-Lebenslinien . . . 144
Abbildung 8.2 MPN-Repräsentation einer eLSC-Nachricht . . . 144
Abbildung 8.3 MPN-Repräsentation einer eLSC-Bedingung . . . 145
Abbildung 8.4 MPN-Repräsentation einer eLSC-Zuweisung . . . 146
Abbildung 8.5 Offene Plätze in der Transformation in die MPN-Sprache . . 147
Abbildung 8.6 MPN-Repräsentation eines Existential eLSCs . . . 148
Abbildung 8.7 MPN-Repräsentation eines Universal eLSCs . . . 149
Abbildung 8.8 MPN-Repräsentation eines eLSCs mit einer kalten Nachricht (Fall 5) . . . 150
Abbildung 8.9 MPN-Repräsentation eines eLSCs mit einer Nachricht, die empfangen werden muss, wenn sie gesendet wurde (Fall 2) 150
Abbildung 8.10 MPN-Repräsentation eines eLSCs mit einer Nachricht, die gesendet, aber nicht empfangen werden muss (Fall 7) . . . 152
Abbildung 8.11 MPN-Repräsentation eines eLSCs mit heißen und kalten synchronen Nachrichten (Fall 1) . . . 153
Abbildung 8.12 MPN-Repräsentation eines eLSCs mit heißen und kalten synchronen Nachrichten (Fall 2) . . . 153
Abbildung 8.13 MPN-Repräsentation eines eLSCs mit kalter Location als letzte Location einer Lebenslinie . . . 154
Abbildung 8.14 MPN-Repräsentation eines *If-Then-Else*-Fragments mit einer enthaltenen unbeschränkten Schleife . . . 155
Abbildung 8.15 MPN-Repräsentationen der *Loop*-Fragmente . . . 156
Abbildung 8.16 MPN-Repräsentation eines *Par*-Fragments . . . 157
Abbildung 8.17 MPN-Repräsentation eines *Alt*-Fragments . . . 159
Abbildung 8.18 MPN-Repräsentation eines eLSCs mit verbotenen und zu ignorierenden Nachrichten . . . 161
Abbildung 8.19 Übersetzung der «include»-Beziehung in das Referenzsystem der MPNs . . . 163
Abbildung 8.20 Übersetzung der «extend»-Beziehung in das Referenzsystem der MPNs . . . 165

Abbildung 8.21 Übersetzung der «threaten»-Beziehung in das Referenzsystem der MPNs . . . 167
Abbildung 8.22 Übersetzung der «mitigate»-Beziehung in das Referenzsystem der MPNs . . . 169
Abbildung 8.23 Kaskadierung von «include» und «extend»-Beziehung im Referenzsystem der MPNs . . . 170
Abbildung 9.1 Metamodell der MPN-Sprache . . . 176
Abbildung 9.2 Konkrete Transformation asynchroner Nachrichten . . . 177
Abbildung 9.3 Initiale Regeln der Transformation (T1) . . . 181
Abbildung 9.4 Korrespondenzmetamodell der SDM-Transformation . . . 183
Abbildung 9.5 Erstellung der Initialplätze im MPN (T2) . . . 183
Abbildung 9.6 SDM-Regeln für asynchrone Nachrichten (T3) . . . 185
Abbildung 9.7 ATL-Regeln für asynchrone Nachrichten (T3) . . . 186
Abbildung 9.8 Synchronisation der offenen Plätze am Ende des MPNs (T4) 187
Abbildung 10.1 Die Struktur der MBSecMon-Toolsuite . . . 194
Abbildung 10.2 MBSecMon-Spezifikationssprache – Das EA-Add-In . . . 195
Abbildung 10.3 Übersicht über die AUTOSAR-Schichtenarchitektur . . . 195
Abbildung 10.4 Das in den AUTOSAR-Entwicklungsprozess eingebettete MBSecMon-Rahmenwerk . . . 197
Abbildung 10.5 AUTOSAR-Komponentensicht eines automatischen Getriebes . . . 199
Abbildung 10.6 UML2-Komponentendiagramm des AUTOSAR-Systemmodells . . . 200
Abbildung 10.7 Angepasster Editor für den MBSecMon-Entwicklungsprozess für AUTOSAR . . . 202
Abbildung 10.8 Signatur des Use-Cases „AllowedSpeedChanges“ . . . 203
Abbildung 11.1 CARDME-eLSCs als MPN-Repräsentation . . . 206
Abbildung 11.2 Wachstum der Transitionen durch kalte Nachrichten . . . 209
Abbildung 11.3 eLSCs zur Evaluation der Transformationszeiten . . . 210
Abbildung 11.4 ATL-Regel zur Erstellung von Übersprungstransitionen im MPN . . . 211
Abbildung 11.5 Laufzeiten der Transformationen für heiße Nachrichten . . . 212
Abbildung 11.6 Laufzeiten der Transformationen für kalte Nachrichten . . . 213
Abbildung 12.1 MBSecMon-Prozess: Übersetzung in die Zwischensprache . 217

TABELLENVERZEICHNIS

Tabelle 1.1 Angriffe und ihre Erkennung durch den MBSecMon-Ansatz 14
Tabelle 3.1 Empfohlene Notationen zur Use-Case-Beschreibung . . . 52
Tabelle 3.2 Vergleich der Sprachkonstrukte von MSC-2000 und UML2-Sequenzdiagrammen . . . 63
Tabelle 3.3 Vergleich der MSC-2000-Inline- und UML2-Interaktionsoperatoren . . . 64

Tabelle 3.4 Abbildung von deontischer Logik auf modale Logik 70
Tabelle 4.1 Konflikte in der Interpretation von asynchronen Nachrichten und ihren Locations 78
Tabelle 4.2 Definition von Bedingungen in LSCs 80
Tabelle 4.3 Anforderungen an eine verhaltensbeschreibende Signaturbeschreibungssprache 88
Tabelle 5.1 Definition von Bedingungen in eLSCs zur Modellierung von Monitorsignaturen 103
Tabelle 5.2 Erlaubte Beziehungen zwischen Use- und Misuse-Cases in der MUC-Sprache 110
Tabelle 5.3 Erweiterungs- und Inklusionspunkte in der MBSecMonSL . 110
Tabelle 9.1 Unterstützung der Anforderungen durch die Transformationssprache 180
Tabelle 11.1 Vergleich der syntaktischen Elemente 207

AUFLISTUNGSVERZEICHNIS

Auflistung 2.1 Abstrakte Operationssignatur 28
Auflistung 6.1 Ausschnitt der EA_OnPostNewElement-Methode für eLSCs 125
Auflistung 7.1 Import und Kopf einer ATL-Transformation 133
Auflistung 7.2 OCL-Helper isAsyncSendingLocation 133
Auflistung 7.3 Standard-Regel für asynchrone Nachrichten 134
Auflistung 7.4 Vererbung 135
Auflistung 10.1 Typsicherheit in AUTOSAR (ARXML-Datei) 201
Auflistung 10.2 eLSC-spezifische Informationen der PSI-Datei 203

Teil I

EINLEITUNG

1 EINLEITUNG

Die Kommunikation zwischen Systemen ist ein grundlegendes Konzept der Informatik. So findet eine Vernetzung zwischen Computern und anderen Geräten sowohl lokal als Intranet als auch global über das Internet statt. Damit diese verteilten Systeme untereinander kommunizieren können, müssen sie eine gemeinsame Sprache sprechen. Diese Sprache wird durch Protokolle spezifiziert, die für eine Kontaktaufnahme der einzelnen Kommunikationspartner kompatibel sein müssen.

Zur Realisierung dieser Kommunikationen zwischen den verschiedenen Kommunikationspartnern stehen verschiedene Netzwerkarchitekturen zur Verfügung. Zwei Architekturen, die weit verbreitet sind, sind die Client-Server- und die Peer-to-Peer-Architektur. Während in einer Client-Server-Architektur zwischen Anbieter einer Dienstleistung (Server) und Benutzer der Dienstleistung über ein Anwendungsprogramm (Client) unterschieden wird, sind in einem Peer-to-Peer-Netzwerk alle Kommunikationspartner gleichberechtigt. Die Client-Server-Architektur bildet die Grundlage für eine Vielzahl an Internet-Protokollen. So werden Server u. a. zur Speicherung und Bereitstellung von Internetseiten und von E-Mail-Diensten genutzt.

Diese allgemeine Interaktion zwischen Systemen über das Internet birgt jedoch große Risiken. Alleine für den *Consumer*-Markt beziffert der *2013 Norton Security Report* [Sym13] den Schaden auf 113 Milliarden US-Dollar pro Jahr und über eine Millionen Opfer am Tag. Zum Schutz vor unerlaubten Zugriffen aus dem Internet auf Computer und internen Netzwerke werden Firewalls, die den Datenverkehr überwachen und einschränken, eingesetzt. Um dennoch auf Systeme im Intranet von extern zugreifen zu können, muss die Firewall nach außen geöffnet werden, wodurch ihre Schutzwirkung verringert wird. Hierdurch ist ein Angriff auf interne Systeme, Anwendungen und Dienste möglich und eine weitere Absicherung notwendig um Schaden abzuwenden. Einen erweiterten Schutz bietet Virenschutzsoftware oder allgemein Intrusion-Detection-Systeme, die die Integrität von Netzwerken und einzelner Systeme überwachen und Gegenmaßnahmen gegen erkannte Bedrohungen ausführen.

Allerdings schreitet die Vernetzung sowohl im geschäftlichen und privaten Bereich immer weiter voran. So sind nicht nur PCs in Privathaushalten mit dem Internet verbunden, sondern eine wachsende Anzahl an mobilen Geräten wie Smartphones und Tablets erobern den Markt. Diese bieten neue Angriffsflächen auf die Daten der Nutzer, da diese häufig weniger leistungsfähig als PCs sind und so die Sicherungsmaßnahmen der PCs nicht ohne Anpassung verwendbar sind. Zusätzlich zeigt der *2013 Norton Report* [Sym13], dass die Hälfte der Nut-

zer von Smartphones und Tablets auf grundlegende Sicherheitsvorkehrungen wie Passwörter, Sicherheitssoftware und Backup ihrer Daten verzichten.

Diese Entwicklung macht jedoch nicht vor weiteren Bereichen des täglichen Lebens halt. So hält auch das Internet und die Nutzung von Diensten Einzug ins Auto – sei es z. B. durch die Anbindung des Navigationsgeräts an das Internet um aktuelle Verkehrsdaten abzufragen oder auch dem Fahrer Zugriff auf Informationen des Internets zu bieten. Zusätzlich entstehen neue Bedrohungsszenarien durch die direkte Anbindung des Smartphones oder mobilen Navigationssystems an das Fahrzeug, da diese Geräte kompromittiert sein können. Selbst ohne physischen Zugang zum Fahrzeug können Sicherheitslücken in den Steuergeräten, z. B. in der Implementierung des Bluetooth-Protokolls der Freisprecheinrichtung, ausgenutzt werden, um Kontrolle über Funktionen des Fahrzeugs zu erhalten [CMK+11].

Diese Entwicklung der verstärkten Vernetzung von Systemen in Fahrzeugen schreitet unter dem Schlagwort Car2X in Zukunft noch weiter voran. Ziel ist es, die Verkehrssicherheit und Verkehrseffizienz zu erhöhen und neue Infotainmentangebote im Fahrzeug zu etablieren. Hierfür werden Adhoc-Verbindungen über eine angepasste WLAN-Technologie oder auch über den Mobilfunk aufgebaut. Im Fahrzeug-zu-Infrastruktur-Szenario (*engl. Car2Infrastructure, C2I*) verbinden sich Steuergeräte des Fahrzeugs mit sogenannten Roadside-Units (RSUs), die Dienste anbieten. Beispiele hierfür sind die Mautgebührenerfassung für LKWs über Mautbrücken als auch zukünftige Szenarien, wie Ampelassistenten oder Parkleitsysteme. Im Fahrzeug-zu-Fahrzeug-Szenario (*engl. Car2Car, C2C*) kommunizieren die Fahrzeuge untereinander und können so z. B. Warnungen vor Gefahrenstellen an andere Verkehrsteilnehmer verschicken.

Doch schon heutzutage sind Fahrzeuge des Premiumsegments teilweise mit Telematiksteuergeräten ausgestattet, die über das Mobilfunknetz Verbindung zu Internetdiensten und im einem Notfall Verbindung zu einem Callcenter aufbauen. Neben diesen Diensten können Funktionen des Fahrzeugs wie das Öffnen und Verschießen der Türen ferngesteuert werden. Wie Checkoway et al. [CMK+11] gezeigt haben, kann selbst über dieses Telematiksteuergerät, das eine abgesicherte Verbindung aufbaut, Daten in das Fahrzeug eingeschleust und bösartiger Programmcode zur Ausführung gebracht werden.

1.1 Motivation

Getrieben durch diese technischen Innovationen werden eingebettete Systeme immer mehr untereinander vernetzt. Hierdurch können diese nicht mehr als von der Außenwelt abgeschlossene Einheiten angesehen werden, obwohl sie oftmals ursprünglich als solche entwickelt wurden. Hiervon sind auch die internen Netzwerke und miteinander kommunizierenden Steuergeräte der Fahrzeuge nicht ausgeschlossen. Bei ihnen wurde wenig Aufmerksamkeit auf Sicherheitsmechanismen, wie Verschlüsselung und sicheres Komponentendesign zur Abwehr von Angriffen von der Außenwelt, gelegt. So existieren zwar verschiedene Bussysteme und Busse in modernen Fahrzeugen, diese sind jedoch über bidirektionale Gateways miteinander verbunden. Diese Gateways übersetzen die Nachrichten auf

das andere Protokoll des Netzwerks und filtern Nachrichten, die nicht einem bestimmten Typ entsprechen. Diese Gefährdung des Systems wird durch das häufig eingesetzte CAN-Protokoll weiter verschärft. Die Nachrichtenpakete des CAN-Protokolls werden grundsätzlich an alle Steuergeräte des Netzwerks versendet, die dann eigenständig entscheiden, ob das Paket für sie relevant ist. Diese Pakete enthalten jedoch in ihrem Header keine Informationen über den Sender der Nachricht, sondern nur Informationen über den Typ des Pakets. Somit reicht ein kompromittiertes Steuergerät oder integriertes externes Gerät aus, um gefälschte Nachrichten selbst über verschiedene Bussysteme hinweg zu verschicken, die durch die Filter der Gateways nicht erkannt werden. Die hierdurch entstehenden Schwächen für passive und aktive Angriffe zeigen Groll und Ruland [GR09] für moderne Netzwerke in Autos und Checkoway et al. [CMK+11] für die drahtlosen Schnittstellen der Autos auf und fordern zusätzliche Sicherheitsmaßnahmen. Selbst wenn bei einer Neuentwicklung eines eingebetteten Systems alle vorgeschlagenen Verfahren durchgeführt werden, ist es jedoch meist nicht möglich, alle Sicherheitslücken zu eliminieren und jeden möglichen Angriff vorherzusehen.

Ein weiterer Faktor für die Anfälligkeit des Netzwerks und der eingebetteten Systeme (Steuergeräte) des Fahrzeugs gegen passive und aktive Angriffe ist die wachsende Komplexität der Software der Steuergeräte und der Interaktion der Steuergeräte im Fahrzeug. Aktuelle Fahrzeuge des Premiumsegments besitzen nach [BMT11] bis zu 100 Steuergeräte auf denen bis zu 4000 Softwarefunktionen als Anwendungen einer Schichtenarchitektur realisiert sind. Diese Funktionen erstrecken sich von technischen Funktionen des Motormanagements, der Steuerung des Antriebsstrangs, der Bremsen und der Stabilität des Fahrzeugs bis hin zu Komfortfunktionen und Infotainment. Zur Unterstützung der immer komplexeren Softwarefunktionen werden zum Teil Steuergeräte eingesetzt, die als Mehrprozessor-Systeme realisiert sind. Hierdurch steigen die Anforderungen an die Softwareentwicklung weiter und die Gefahr einer fehlerhaften Implementierung der Schnittstellen oder der Funktionen.

Um mit dieser Komplexität besser umgehen zu können, hält auch in der Automobilindustrie die modellbasierte Softwareentwicklung Einzug. Hierbei werden problemspezifische Modelle zur Beschreibung relevanter Entwicklungsartefakte durch die Entwickler einsetzt. So wurde von der Automobilindustrie der AUTOSAR-Standard (Automotive Open System Architecture) [KF09] entwickelt, der eine Schichtenarchitektur vorschreibt, und so die Anwendungssoftware von der Hardware der einzelnen Steuergeräte abtrennt. In der AUTOSAR-Architektur werden Softwarekomponenten (SW-C) mit eindeutigen Schnittstellen zur Basissoftware (BSW), die auf die Hardware angepasst ist, spezifiziert. Durch eine Generierung einer Laufzeitumgebung (RTE) werden diese Schnittstellen der ebenfalls generierten SW-Cs an die Dienste der BSW angebunden. Diese Dienste umfassen u. a. die Mechanismen zur Kommunikation mit anderen SW-Cs, System- und Diagnosedienste und Speicherverwaltung. Im Gegensatz zur manuellen Implementierung der Anwendung wird durch die Generierung auf Basis der spezifizierten Schnittstellen eine weniger fehleranfällige Integration und Verteilung der Anwendungen auf den Steuergeräten erreicht. Zusätzlich werden weitere Fehlerquellen durch die modellbasierte Entwicklung von SW-Cs, die Kontrollfunktionen imple-

mentieren, in MATLAB/Simulink und die Generierung ihrer Implementierungen reduziert.

Betrachtet man jedoch große stark nebenläufige heterogene Systeme oder Komponenten oder will man vorhandene Komponenten nachträglich absichern, ist die Aufdeckung und Behebung aller Sicherheitslücken und Fehler meistens ökonomisch oder technisch nicht zu realisieren. Resultierend daraus kann bei keinem System davon ausgegangen werden, dass es sicher ist – sei es durch unbekannte Schwachstellen oder der Verwendung von *Legacy*-Komponenten. Um solche Systeme dennoch absichern zu können, ist die Überwachung eines Systems während der Laufzeit (Def. 1) dazu geeignet Angriffe, die vorher unbekannte Fehler und Sicherheitslücken ausnutzen, zu erkennen.

Definition 1 (*Überwachung zur Laufzeit (engl. Monitoring)*)**.** Die Überwachung eines Systems bzw. einer Implementierung zur Laufzeit überprüft, ob sich das System entsprechend seiner Spezifikation korrekt und sicher verhält. Diese Überwachung kann durch Ad-hoc-Methoden bis hin zu formalen Methoden (Laufzeitverifikation), die auf Inferenz und Logiken basieren, realisiert werden. In der formalen Laufzeitverifikation werden zu einem Referenzverhalten, meist in Temporaler Logik spezifiziert, *Monitore* generiert, die das überwachbare Systemverhalten gegen die Spezifikation überprüfen. Monitore laufen parallel zum System und sollten dieses nicht beeinflussen. (basierend auf [BJ10])

Annahme: Heterogene Systeme und Komponenten, die durch Laufzeitüberwachung abgesichert werden sollen, können nicht als sicher gegen Angriffe und Fehlfunktionen angesehen werden. Diese Annahme ist gültig, unabhängig davon, ob vorhandene Systeme nachträglich abgesichert werden, oder diese neu entwickelt werden. Für den Einsatz der Laufzeitüberwachung muss davon ausgegangen werden, dass die Ereignisse, die überwacht werden, nativ als Nachrichten oder Methodenaufrufe nach außen sichtbar sind, oder durch Instrumentierung des Systems zur Verfügung gestellt werden können. Die Monitore sollen zum einen zentral, als auch zum anderen verteilt über die einzelne Software bzw. Hardware-Komponenten einsetzbar sein. Ein zentraler Monitor überwacht als separate Komponente in einem Netzwerk die Kommunikation. Im Gegensatz hierzu wird der Monitor in einem verteilten Szenario als Wrapper um eine *Legacy*-Komponenten (Software oder Hardware) realisiert oder läuft auf den eingebetteten Systemen selbst.

Damit ein System überwacht werden kann, muss eine Definition seines Verhaltens vorliegen. Zur Beschreibung des Normalverhaltens, optionalen Verhaltens oder fehlerhaften Verhaltens eines Systems in frühen Entwicklungsphasen, wird häufig die Technik des *szenariobasierten Designs* [JCJ92, RQJZ07] eingesetzt. Hierfür werden *Szenarien* des Systems definiert, die sich auf einzelne Funktionen konzentrieren. Hierdurch muss weder der Entwickler noch der Nicht-Experte (*engl. Stakeholder*) die gesamte Funktionalität des Systems auf einmal erfassen, wodurch das Verständnis erleichtert wird. Ein Szenario wird als Use-Case verfasst und

beschreibt die Funktionalität des Systems durch Sequenzen von Interaktionsereignissen zwischen den Systemelementen und der Umgebung des Systems. Allerdings finden bei dem szenariobasierten Ansatz zur Erfassung des Verhaltens des Systems nicht-funktionale Eigenschaften wie Security und Safety keine oder wenig Beachtung [SO05, Ale03a]. Ihre Beschreibbarkeit ist jedoch für die Überwachung eines Systems essentiell und muss für die Laufzeitüberwachung zur Verfügung stehen. Da sich diese Eigenschaften häufig als nicht erlaubte Interaktionssequenzen beschreiben lassen, sind Szenarien zu ihrer Darstellung geeignet. Der szenariobasierte Ansatz bietet hierfür jedoch selbst keine Möglichkeit, diese negativen Szenarien zu markieren und sie in Beziehung zu positiven Szenarien zu setzen. Hierfür ist eine Erweiterung der szenariobasierten Beschreibungssprache um eine Variante, die zwischen erlaubtem und verbotenem Verhalten unterscheidet, notwendig. Eine solche Erweiterung sind die Misuse-Cases [SO05, Ale02, Ale03a], die an die Use-Cases der UML angelehnt sind. Sie bieten eine explizite Beschreibung verbotener Sequenzen und ihren Beziehungen zu den positiven Sequenzen eines Systems, die die Funktionalität beschreiben. Aus diesem Grund bildet dieses Konzept die Grundlage der Signaturspezifikationssprache (Def. 2), die im *Model Based Security/Safety Monitor*-Entwicklungsprozess zur Laufzeitmonitorspezifikation genutzt wird.

Definition 2 (*Signatur / Signaturspezifikationssprache*)**.** Ein *Kommunikationsprotokoll* beschreibt die erlaubte Kommunikation zwischen Systemelementen umfassend bis auf die Implementierungsebene. Hierbei beschreibt das Konzept der *Dienste* das Ergebnis der Kommunikation und das der *Protokolle* wie dieses Ergebnis erreicht wird. Im Gegensatz hierzu steht in dieser Arbeit der Begriff der *Signatur* für die Beschreibung erlaubter und verbotener Teilabläufe einer Kommunikation. Mehrere dieser Signaturen zusammen bilden die Spezifikation für einen Laufzeitmonitor, der diese Abläufe überwacht. Diese Signaturen werden in einer *Signaturspezifikationssprache* verfasst, die im Gegensatz zu Protokollspezifikationssprachen auch zur expliziten Beschreibung verbotener Abläufe geeignet ist.

1.2 MBSecMon-Entwicklungsprozess

Ziel des in dieser Arbeit und in [Pat14] vorgestellten *Model Based Security/Safety Monitor*-Entwicklungsprozesses (MBSecMon-Entwicklungsprozesses) für Sicherheitsmonitore ist die einfache Einbindung in einen bestehenden Entwicklungsprozess, wie er z. B. vom V-Modell beschrieben wird. Ein Entwicklungsstandard, der auf dem V-Modell aufbaut, ist das V-Modell XT [BTZ05, Bun12]. Es ist ein aktueller Vorgehensstandard für IT-Projekte im Verteidigungsbereich und Projekte der öffentlichen Hand des Bundes. Er findet jedoch durch seine große Anpassbarkeit vermehrt Anwendung in der Privatwirtschaft. Abbildung 1.1 zeigt grau hinterlegt die Schritte eines solchen an das V-Modell XT angelegten Entwicklungsprozesses.

Das V-Modell teilt den Entwicklungsprozess in drei Phasen auf – die Zerlegung, die Implementierung und die Integration. Die *Zerlegungsphase* besteht aus mehreren Schritten, in denen das zu entwickelnde System immer detaillierter be-

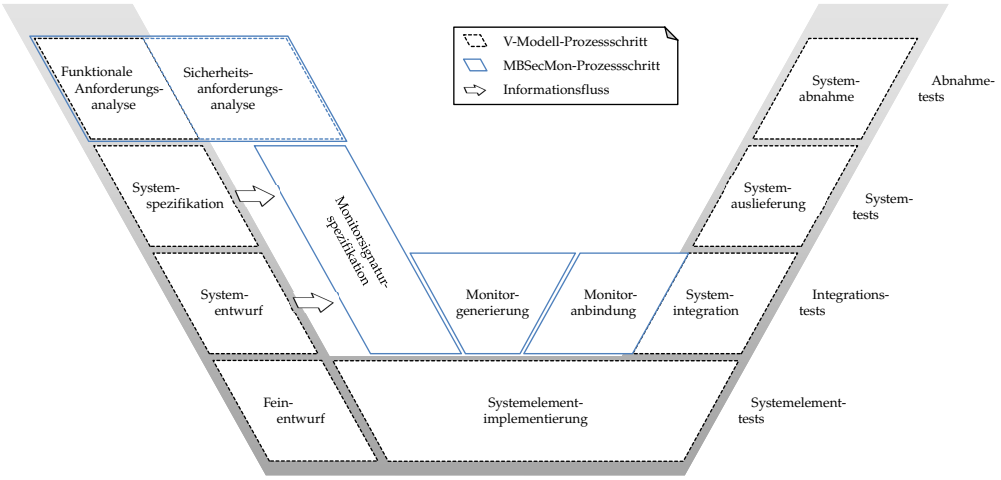


Abbildung 1.1: Der MBSecMon-Prozess im V-Modell

schrieben wird. Der erste Vorgehensschritt – die *Anforderungsanalyse* – dient der Erstellung eines Lastenhefts. In ihm wird die Ausgangssituation und somit die Einsatzumgebung des zu entwickelnden Systems und seine Zielsetzung beschrieben. Basierend auf diesen Beschreibungen werden funktionale und nicht-funktionale Anforderungen an das System festgelegt und parallel zu diesem Festlegungsprozess eine Skizze der Gesamtarchitektur erstellt. Zur Beschreibung der funktionalen Anforderungen kommen u. a. objektorientierte Ansätze zum Einsatz, die in Form von Anwendungsfällen erfasst und als Szenarien beschrieben werden können. Dieser Ansatz entspricht dem bereits vorgestellten szenariobasierten Design. Nicht-funktionale Anforderungen wie Benutzerfreundlichkeit oder Zuverlässigkeit werden meist textuell erfasst, sollten jedoch messbar sein.

Im zweiten Vorgehensschritt *Systemspezifikation* wird eine grobe Gesamtsystemarchitektur anhand der im vorherigen Schritt ermittelten funktionalen und nicht-funktionalen Anforderungen erstellt und diese Anforderungen werden der Architektur zugeordnet. Zusätzlich werden in ihr die Schnittstellen zwischen den Systemen und der Umgebung identifiziert und spezifiziert. Zusammen mit dem in diesem Schritt festgesetzten Lieferumfang und den Abnahmekriterien bildet diese Erweiterung des Lastenhefts die Gesamtsystemspezifikation – das Pflichtenheft.

Im *Systementwurf* werden die in der Gesamtsystemarchitektur enthaltenen Systemelemente weiter verfeinert, indem ihr Verhalten genauer spezifiziert wird. Ziel des *Feinentwurfs* ist die konsistente Definition aller Konzepte, die zur Implementierung, Integration und Prüfung der Systemelemente notwendig sind, festzulegen.

Der Feinentwurf bildet gleichzeitig mit dem Schritt der *Systemelementimplementierung* die Phase der *Implementierung*. Ergebnis dieser Phase sind die implementierten Systemelemente, die einzeln gegen die im Feinentwurf festgelegten Spezifikationen getestet werden. In der Phase der *Integration*, werden die im vorherigen Schritt realisierten Systemelemente zu einem Gesamtsystem integriert (*Systemintegration*). Hierbei finden Integrationstests zur Absicherung der Kompatibilität der einzelnen Systemelemente untereinander statt. Anschließend wird in der Pha-

se *Systemauslieferung* vom Auftragnehmer überprüft, ob Anforderungen des Auftraggebers an das System erfüllt sind. Basis für diese Tests bildet das Pflichtenheft, das u. a. die Schnittstellen zur Umgebung des Systems und die Anforderungen an das System beschreibt. Entspricht das integrierte System dem Pflichtenheft, wird das System an den Auftraggeber ausgeliefert und eine *Systemabnahme* durch den Auftraggeber durchgeführt.

Der MBSecMon-Prozess besteht aus vier Vorgehensbausteinen, die jedoch nicht in das V-Modell eingebettet werden, sondern parallel zum V-Modell ablaufen. Hierdurch wird die Unabhängigkeit des MBSecMon-Prozesses vom gewählten Entwicklungsprozess erreicht. Zusätzlich zu der funktionalen Anforderungsanalyse des V-Modells findet für die Laufzeitmonitorgenerierung eine *Sicherheitsanforderungsanalyse* statt, in der mögliche Angriffe auf das System und nicht erlaubte Kommunikationssequenzen identifiziert werden. Diese Szenarien werden durch Misuse-Cases beschrieben. Im Schritt der *Monitorsignaturspezifikation* werden diese Szenarien der funktionalen Anforderungsanalyse und der Sicherheitsanalyse verwendet um die Monitorsignaturen zu spezifizieren. Je nachdem, welche Kommunikation überwacht werden soll, fließen zusätzlich Informationen über die Systemelemente und Schnittstellen des Systems aus der Systemspezifikation und dem Systementwurf in den Spezifikationsprozess ein. Basierend auf diesen Signaturen, die Use-Cases und Misuse-Cases des Systems beschreiben, werden im Schritt der *Monitorgenerierung* Laufzeitmonitore generiert und diese im Schritt der Systemintegration an das Gesamtsystem angebunden (*Monitoranbindung*).

1.2.1 *Der MBSecMon-Prozess*

Für die Generierung solcher Sicherheitsmonitore wurde ein verständlicher, modellbasierter Entwicklungsprozess entworfen. Die vier Vorgehensschritte der Monitorgenerierung, die in Abbildung 1.1 vorgestellt wurden, sind im MBSecMon-Prozess in Abbildung 1.2 als Prozess dargestellt. Auf der linken Seite ist ein stark vereinfachter *System-Entwicklungsprozess* dargestellt, der basierend auf *Entwicklungsartefakten* den Programmcode erzeugt, der später auf einer Ausführungsplattform eingesetzt wird.

In der Anforderungsphase des modellbasierten MBSecMon-Prozesses für Sicherheitsmonitore wird das erwartete Verhalten des Systems als Use-Cases spezifiziert. Diese können aus dem normalen System-Entwicklungsprozess des zu überwachenden Systems extrahiert oder z. B. bei Legacy-Software neu erstellt werden. Neben den Use-Cases, die zur Anomalieerkennung (Abweichungserkennung) genutzt werden können, wird das Konzept der Misuse-Cases genutzt, um auch bekanntes Fehlverhalten und Angriffe auf das System direkt zu beschreiben. Diese Misuse-Cases entsprechen somit dem Konzept der signaturbasierten Erkennung spezifischer Angriffsmuster, die in Computernetzen eingesetzt wird. Die Use- und Misuse-Cases werden mithilfe der Beziehungen «include» und «extend» bzw. «threaten» und «mitigate» zueinander in Beziehung gesetzt. Für die Beschreibung möglicher Abläufe der (Mis-)Use-Cases werden aufgrund ihrer erweiterten Ausdruckskraft (Unterscheidung zwischen möglichem und notwendigem Verhalten) eine erweiterte Variante der Live Sequence Charts (LSCs) [DH01,

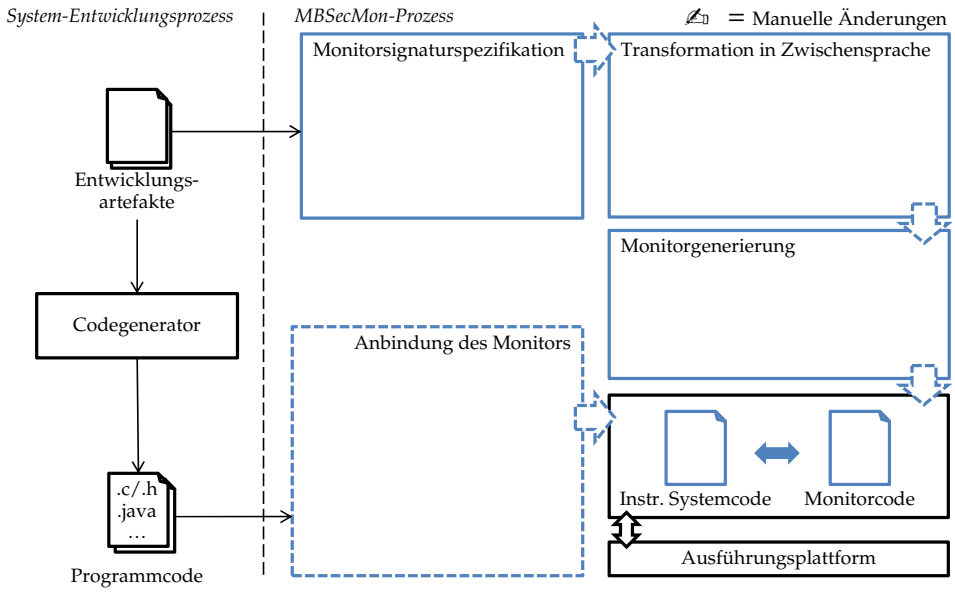


Abbildung 1.2: Der MBSecMon-Prozess

HM03], statt UML-Sequenzdiagrammen oder Message Sequence Charts, verwendet. Die Spezifikation des Systems als Use-Cases und des nicht erlaubten Verhaltens als Misuse-Cases dient als Grundlage für die automatische Generierung eines Sicherheitsmonitors. Diese in dieser Arbeit entwickelte Kombination aus Use- und Misuse-Cases und der erweiterten Variante der LSCs bilden die MBSecMon-Spezifikationssprache (MBSecMonSL).

Aufgrund der Ausdrucksstärke der LSCs, in denen viele verschiedene Abläufe implizit beschrieben sind, ist deren Interpretation, die zur direkten Codegenerierung notwendig ist, sehr komplex. Um diesen Schritt zu vereinfachen und die Nutzung einer großen Bandbreite anderer Spezifikationssprachen zu ermöglichen, wurde eine Transformation der LSCs in ein petrinetz-ähnliches Zwischenformat – die Monitor-Petrinetze (MPNs) – realisiert (*Transformation in Zwischensprache*). In diesem Zwischenformat werden die möglichen Abläufe der LSCs explizit repräsentiert. Durch die an die Laufzeitüberwachung angepasste Semantik der MPNs und die Parallelität der Sprache ist diese Zwischenrepräsentation kompakt und einfach in Monitoringcode umzusetzen. Im Schritt der *Monitorgenerierung* werden die Signaturen in der Zwischensprache mit zusätzlichen systemspezifischen Informationen angereichert und aus ihnen Software- und Hardware-Sicherheitsmonitore generiert. Diese werden, falls die zu überwachenden Ereignisse nicht vom System direkt zur Verfügung gestellt werden, an das System angebunden (*Anbindung des Monitors*).

1.2.2 *Abgrenzung zu vorhandenen Ansätzen*

Im Sicherheitsumfeld werden zur formalen Beschreibung funktionaler und nichtfunktionaler Anforderungen als Signaturen für Intrusion Detection Systeme (IDS), die Angriffe auf Computersysteme erkennen, spezielle Policysprachen oder temporale Logiken eingesetzt. Diese haben jedoch, wie in Abschnitt 3.3 aufgezeigt

wird, verschiedene Nachteile, die durch die in dieser Arbeit entwickelten MBSecMonSL aufgehoben werden.

Viele der verwendeten Spezifikationssprachen stellen aufgrund ihres Formalismus große Herausforderungen an den Modellierer der Signaturen und sind für Nicht-Experten nur schwer verständlich. So sind *Expertensysteme*, in denen deklarative Inferenzregeln und temporale Regeln spezifiziert werden, die auf Fakten arbeiten und diese gleichzeitig manipulieren, durch ihre impliziten Zusammenhänge in der Spezifikation nur schwer zu überblicken. Zudem sind temporale Zusammenhänge, die ebenfalls auf Basis der Faktenlage spezifiziert werden, nur schwer modellierbar. Im Gegensatz hierzu eignen sich verschiedene Varianten der häufig zur Spezifikation von Protokollen eingesetzten *Temporalen Logiken* besser zur Modellierung von temporalen Zusammenhängen als Expertensysteme. Die in dieser Sprache verfassten Spezifikationen sind jedoch aufgrund ihrer abstrakten Notation und ihres oftmals großen Umfangs nur schwer zu spezifizieren und zu verstehen.

Die Verständlichkeit für Nicht-Experten ist jedoch eine grundlegende Anforderung an die Signaturspezifikationssprache des MBSecMon-Prozesses. Hierfür eignen sich Ansätze, die auf dem bekannten De-facto-Standard der UML2 basieren. Die prominentesten Modellierungssprachen, deren Annotationen an UML-Modellen zur Verifikation von Sicherheitsanforderungen dieser Spezifikation eingesetzt werden, sind die UMLsec [Jür05] und SecureUML [LBD02, BDL06]. Diese Erweiterungen der UML werden zwar auch zur Laufzeitüberwachung der hauptsächlich statischen Systemeigenschaften eingesetzt [BJ08, BJ10, BJY11], sehen jedoch keine explizite Modellierung verhaltensbeschreibender Signaturen für den Einsatz in IDS bzw. Monitore vor.

Eine weitere Kategorie der Spezifikationssprachen für Signaturen sind zustandsbasierte Ansätze wie STAT [VEK00]. Die Modellierung der Signaturen führt in diesen Sprachen jedoch häufig zu einem sehr starken Anwachsen der zu modellierenden Zustände, da der Sprache Konzepte zur Beschreibung nebenläufigen Verhaltens (z. B. Hierarchie der Zustände oder Fork bzw. Join-Operatoren) fehlen. In Sprachen, die auf gefärbten Petrinetzen basieren, lassen sich Signaturen kompakter modellieren. Hierbei tragen Plätze, Transitionen oder auch die Marken der Petrinetze zusätzliche Merkmale (Färbung), die Eigenschaften kodieren. Dies führt jedoch, wie die Ansätze IDOT [Kum95] und EDL [Sch04, MS05, FM12] zeigen, durch die starke Färbung der Netze zu für Nicht-Experten schwer verständlichen Spezifikationen.

Durch eine Anhebung der Abstraktionsebene der Modellierungssprache kann sowohl die Modellierung der Signaturen als auch die einfachere Verständlichkeit durch Nicht-Experten erreicht werden. So werden in [MN08] und [SES07] Sequenzdiagramme der UML zur Modellierung der Signaturen verwendet; hierbei wird jedoch eine von der UML abweichende Semantik verwendet. Somit besteht die Gefahr, dass Modellierer mit vorhandenem UML-Wissen fehlerhafte Signaturen modellieren. Der Einsatz von Live Sequence Charts (LSCs) als Spezifikationssprache [BG01, BGS05, KMB09], die selbst die Unterscheidung von notwendigem, möglichem und verbotenem Verhalten unterstützt, führt zu klareren einfacher verständlicheren Signaturbeschreibungen.

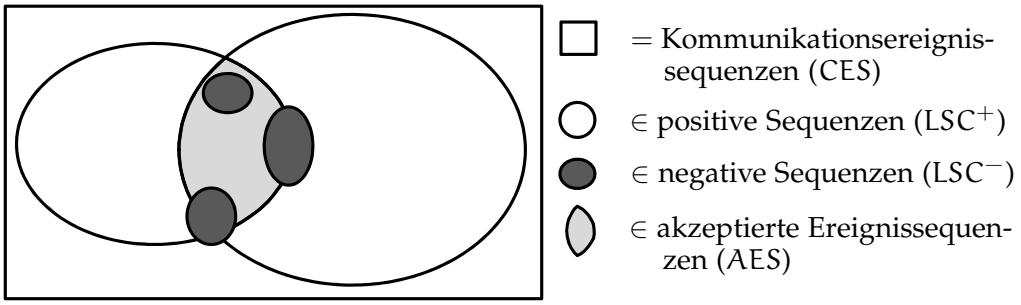
Die MBSecMonSL vereinigt die Vorteile der verschiedenen Formalismen in einer grafischen, modellbasierten Signaturbeschreibungssprache. So findet die Kommunikationssequenzbeschreibung als LSCs auf einer *höheren Abstraktionsebene der Modellierung* als die der Expertensysteme und Temporalen Logiken statt. Die grafische Notation der LSC ist Systementwicklern durch die Sequenzdiagramme des UML-Standards größtenteils bekannt und, wie der szenariobasierte Ansatz der Anforderungsanalyse zeigt, sind diese Spezifikationen auch für Nicht-Experten verständlich. Dieses hohe Abstraktionsniveau wird durch die Misuse-Case-Sprache, die zur Strukturierung der Szenarien eingesetzt wird, weiter erhöht. Durch die *explizite Unterscheidung zwischen Use- und Misuse-Cases* der MUC-Sprache wird ein Strukturierungsmechanismus angeboten, der schon in frühen Entwicklungsphasen eingesetzt werden kann und durch schrittweise Verfeinerung zu überwachbaren und übersichtlichen Signaturbeschreibungen führt. Zusätzlich wird durch den Referenzierungsmechanismus der MBSecMonSL eine *bessere Skalierbarkeit* der Spezifikationen erreicht, indem Teilsignaturen aus LSCs ausgelagert und in verschiedenen Signaturen wiederverwendet werden können. Dieser Mechanismus wird zur Steigerung der Übersichtlichkeit und zur Visualisierung der Zusammenhänge auch auf Ebene der MUC-Sprache eingesetzt und als «include»-Beziehung modelliert.

1.2.3 *Sicherheitsmodell und Gültigkeit des Ansatzes*

Die durch den MBSecMon-Prozess generierten Monitore werden zur Laufzeitüberwachung von reaktiven rechnergestützten und eingebetteten Systemen eingesetzt. Hierbei stehen die Laufzeitüberwachung der Interaktionen dieser Systeme mit der Umgebung und die Kommunikation zwischen den Komponenten eines Systems im Vordergrund. Der Fokus der zu spezifizierenden Signaturen liegt auf stark nebenläufigen Kommunikationsabläufen. Die generierten Laufzeitmonitore sollen sowohl zentral auf einem Kommunikationsmedium als auch verteilt eingebunden in dem zu überwachenden System eingesetzt werden. Zusätzlich zur Überwachung der Kommunikation, die für den Monitor direkt zugänglich ist, lässt sich durch Instrumentierung des zu überwachenden Systems auch der Kontrollfluss von Teilfunktionalitäten der Komponenten überwachen.

Die Nachvollziehung komplexer Berechnungen der Systeme und Komponenten und die Überprüfung komplexer Daten sind in dieser Arbeit Out-of-Scope des Modellierungsansatzes. Diese können jedoch durch manuell implementierte Methoden, die in der Modellierungssprache durch Aktionen und die Überprüfung des Ergebnisses durch Bedingungen ausgedrückt werden, spezifiziert werden.

Für die Beschreibung des Verhaltens reaktiver Systeme eignet sich in frühen Entwicklungsphasen der szenariobasierte Ansatz. Durch diese Technik lässt sich das Verhalten eines Systems einfacher erfassen als durch die vollständige zustandsbasierte Modellierung der einzelnen Komponenten und des gesamten Systems [HM03]. Das komplette Verhalten eines Systems kann jedoch häufig nicht durch einen szenariobasierten Ansatz, dessen Szenarienbeschreibungen auf einfachen Sequenzdiagrammen basieren, beschrieben werden. In einigen Fällen ist diese Beschreibung zwar möglich, es entstehen hierbei jedoch anscheinend un-



$$AES = \bigcap LSC^{+} \setminus \bigcup LSC^{-} \subseteq CES$$

Abbildung 1.3: Das Sicherheitsmodell des MBSecMon-Prozesses

korrelierte Zusammenhänge, die nicht nachvollziehbar sind. Diese Lücke des szenariobasierten Ansatzes wird im MBSecMon-Prozess durch den Einsatz einer erweiterten Variante der LSCs überbrückt, indem sie die Unterscheidung zwischen notwendigen und möglichen Szenarien und Teilabläufen anbietet.

Die im MBSecMon-Prozess spezifizierten verhaltensbeschreibenden Signaturen in der MBSecMon-Spezifikationssprache (MBSecMonSL), bestehend aus positiven und negativen Szenarien, definieren, welche Ereignissequenzen vom generierten Laufzeitmonitor akzeptiert werden. Abbildung 1.3 zeigt die Menge aller möglichen `Kommunikationsereignissequenzen` (CES). Die durch die positiven Szenarien beschriebenen Sequenzen (LSC^{+}) schränken diese Ereignissequenzen ein. Da keine dieser positiven Sequenzen verletzt werden darf, bildet ihre Schnittmenge die durch den Monitor akzeptierten Ereignissequenzen (AES). Durch die als Misuse-Cases modellierten Szenarien, die negative Sequenzen (LSC^{-}) beschreiben, werden diese AES weiter eingeschränkt. Die vereinigte Menge der negativen Sequenzen darf somit nicht auftreten. Wird die Menge der AES während der Überwachung des Systems verlassen, muss dies der Laufzeitmonitor erkennen und Gegenmaßnahmen einleiten. Hierbei wird nicht eine illegale durch eine legale Sequenz ersetzt, sondern das System nachträglich durch eine ausführbare Sequenz von Ereignissen wieder in einen Zustand der AES versetzt. Ist diese Gegenmaßnahme erfolgreich, fährt der Laufzeitmonitor mit der Überwachung des Systems auf Basis der Szenarien fort.

1.2.4 *Security im Rahmen der Arbeit*

Ziel der generierten Monitore ist die Überwachung der Kommunikation zwischen verschiedenen Kommunikationspartnern der eingebetteten Systeme. Hierfür muss die Signaturbeschreibungssprache die Formulierung verbotener und erlaubter Kommunikationssequenzen zwischen den verschiedenen Kommunikationspartnern unterstützen. Die möglichen Manipulationen der Kommunikation und somit die möglichen Angriffe auf das System hängen maßgeblich vom verwendeten Kommunikationskanal und der Absicherung der Kommunikation (z. B. durch Verschlüsselung der Nachrichten oder Manipulationsschutz) ab. Ebenso

kann die Manipulation von Nachrichten an verschiedenen Stellen des verteilten Systems stattfinden. Neben der bereits erwähnten Manipulation der Nachrichten auf dem Kommunikationskanal, die z. B. aus einem Man-In-the-Middle-Angriff resultieren kann, ist ebenso eine Änderung der Nachrichten auf der Empfänger- oder Senderseite möglich. Dies kann durch die Übernahme des entsprechenden Kommunikationspartners erfolgen.
Orte der Manipulation in verteilten Systemen sind somit:

- Änderung der Nachricht auf dem Kommunikationsmedium
- Änderung auf der Senderseite
- Änderung auf der Empfängerseite

Eine mögliche Gegenmaßnahme ist die bereits erwähnte Verschlüsselung der Kommunikation. Dies ist jedoch insbesondere auf kommunizierenden Eingebetteten Systemen durch begrenzte Rechenleistung und begrenzte Kapazität des Kommunikationsmediums häufig nicht umsetzbar. Zusätzlich kann nicht von einer sicheren, fehlerfreien Implementierung der Verschlüsselung ausgegangen werden, wie u. a. der aktuelle Heartbleed-Angriff [ZCL+14] auf SSL-Verschlüsselungen zeigt. Ohne eine Verschlüsselung bzw. einen Manipulationsschutz sind die in Tabelle 1.1 aufgeführten Angriffe auf den Kommunikationskanal und auf die Kommunikationspartner möglich. Die zweite Spalte der Tabelle beschreibt Möglichkeiten des MBSecMon-Ansatzes diese Angriffe zu erkennen.

Der Angreifer kann	**Erkennbar durch**
1) Nachrichten auf dem Kommunikationskanal manipulieren	• Überwachung des Nachrichteninhalts
2) Nachrichten löschen	• Überwachung der Nachrichtenreihenfolge • Zeitabweichungen zwischen den Nachrichten
3) Nachrichten hinzufügen	• Falsche Nachrichtenreihenfolge • Zu hohe Nachrichtenfrequenz
4) Nachricht durch eine beliebige andere Nachricht ersetzen	• Falsche Nachrichtenreihenfolge • Nicht erlaubte Nachricht auf dem Kommunikationskanal • Abweichung des Nachrichteninhalts
5) Sender/Empfänger der Nachricht ändern (aber keinen weiteren Inhalt verändern)	• Nicht erkennbar, wenn legitime Nachricht
6) Nachrichten verzögern	• Verspätete oder ausbleibende Nachrichten
7) Sender oder Empfänger übernehmen	
• Nachricht nur auf Senderseite oder Empfangsseite direkt manipulieren	• siehe 1)
• Deaktivierung eines Kommunikationspartners und Senden von Nachrichten in seinem Namen	• Nicht erkennbar, wenn legitime Nachricht

Tabelle 1.1: Angriffe und ihre Erkennung durch den MBSecMon-Ansatz

Die Monitore des vorgestellten MBSecMon-Prozesses können nur Angriffe erkennen, die sich als Abweichung von den modellierten positiven und negativen

Ereignissequenzen manifestieren. Diese überwachbaren Abweichungen werden durch folgende Eigenschaften der Kommunikation erkannt:

- Typ der Nachrichten
- Inhalt der Nachrichten
- Timing der Nachrichten
- Reihenfolge der Nachrichten

Angriffe auf das System, die die Anforderungen an die Kommunikation, bestehend aus den modellierten positiven und negativen Szenarien, erfüllen, sind nicht erkennbar. Hierfür könnte eine statistische Anomalieüberwachung eingesetzt werden, die jedoch aufwendig zu konfigurieren ist und viele Fehlerkennungen erzeugen kann, die Nutzerinteraktionen erfordern.

Somit erlaubt der MBSecMon-Ansatz, der in dieser Arbeit beschrieben wird, im Bereich der Angriffssicherheit (*engl. Security*) hauptsächlich die Überwachung der *Datenintegrität* der Nachrichten auf dem Kommunikationskanal. Zusätzlich kann die *Verfügbarkeit* des Kommunikationskanals sichergestellt werden, indem die Frequenz der Nachrichten basierend auf einem Schwellwert überwacht wird und Gegenmaßnahmen eingeleitet werden. Der Fokus der Arbeit liegt jedoch hauptsächlich auf der Überwachung von Safety-Sicherheitszielen.

1.3 Beitrag

Der in Abschnitt 1.2 vorgestellte modellbasierte Entwicklungsprozess für Security- und Safety-Monitore (MBSecMon-Prozess) wurde im Rahmen von zwei Promotionen konzipiert und realisiert. Die Arbeit von Lars Patzina [Pat14] greift den zweiten Teil des MBSecMon-Prozesses in Abbildung 1.2 auf – die Entwicklung und Formalisierung einer für die Laufzeitmonitorgenerierung optimierten Zwischensprache (MPN-Sprache) und die *Monitorgenerierung* aus dieser Zwischensprache. Ziel dieser Zwischensprache ist zum einen die kompakte Repräsentation stark nebenläufiger Signaturen und zum anderen die Generierung effizienter Laufzeitmonitoren in Java, C und VHDL aus der MPN-Sprache. Hierbei steht die universelle Einsetzbarkeit der Monitore auf verschiedenen Plattformen wie PC, eingebettete Systeme und als Hardwareimplementierung im Fokus.

Diese Arbeit geht auf den ersten Teil des MBSecMon-Entwicklungsprozesses ein – die *Monitorsignaturspezifikation* und die *Transformation in die Zwischensprache* der MPNs. Der Fokus liegt insbesondere auf der Entwicklung einer verhaltensbeschreibenden Signatursprache für Laufzeitmonitore, die vorhandene Entwicklungsartefakte des Systementwicklungsprozesses einbezieht, und auf einer Formalisierung dieser Sprache durch die Transformation in die formalisierte Zwischensprache der MPNs. Hierbei sind folgende Anforderungen an diesen Prozess und seine Spezifikationssprache in Abbildung 1.4 zu stellen:

① **Eingliederung in bestehende System-Entwicklungsprozesse:** Der MBSecMon-Prozess muss sich in bestehende Vorgehensmodelle der modellbasierten Systementwicklung eingliedern lassen. Hierfür wurde ein zum Systementwicklungsprozess paralleler Ansatz verwendet, der auf Entwicklungsartefakte der Anforderungsanalyse und der Systemspezifikation zurückgreift und erst in der Pha-

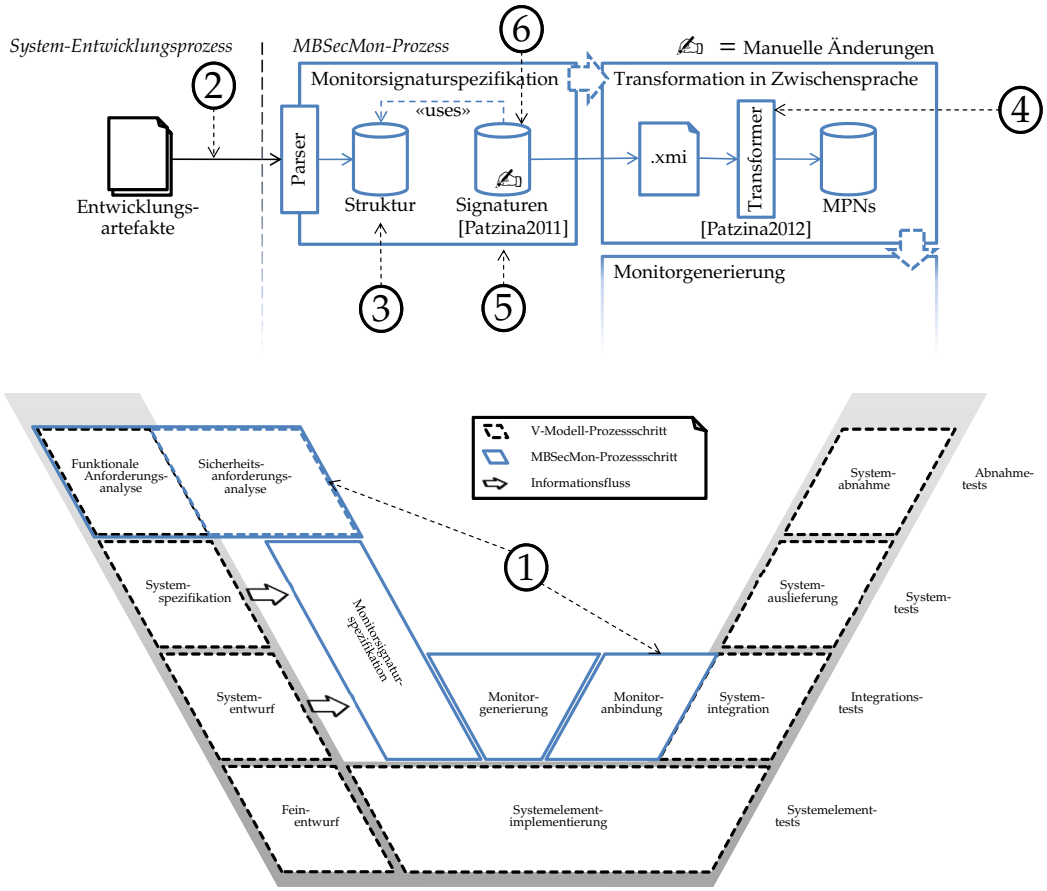


Abbildung 1.4: Einordnung der Beiträge

se der Systemintegration den Monitor an das zu überwachende System anbindet. Beispielhaft wurde diese Einbettung in Abschnitt 1.2 anhand eines auf dem V-Modell basierenden Entwicklungsprozesses vorgestellt und für einen AUTOSAR-Entwicklungsprozess in Abschnitt 10 implementiert und evaluiert.

② **Einbeziehung bestehender Entwicklungsartefakte:** Die Spezifikation der Monitore erfolgt basierend auf vorhandenen Entwicklungsartefakten des Entwicklungsprozesses des zu überwachenden Systems. So können Strukturinformationen wie Systemelemente, ihre Schnittstellen, deren Datentypen oder sogar ganze Signaturen, die als Szenarien der Anforderungsanalyse modelliert sind, aus den vorhandenen Artefakten extrahiert werden. Diese Informationen werden zur Unterstützung des Entwicklers bei der Spezifikation der Signaturen eingesetzt.

③ **Unterstützung verschiedenster Zielplattformen:** Ein Hauptziel des gesamten MBSecMon-Prozesses ist die Generierung von Monitoren für verschiedenste Zielplattformen ohne die Spezifikationssprache für die Signaturen anzupassen. Die in dieser Arbeit entwickelten MBSecMon-Spezifikationssprache ist eine universelle verhaltensbeschreibende Signatursprache, die zur Spezifikation von Kommunikationsverhalten eingesetzt wird. Um die Generierung von Monitoren zur Überwachung der komplexen hoch nebenläufigen Kommunikationen zu unterstützen, werden die MBSecMon-Spezifikationen in eine formale Zwischenspra-

che – die MPN-Sprache [Pat14] – übersetzt. Hierdurch wird zusätzlich auch eine Unabhängigkeit der Codegenerierung von der gewählten Spezifikationssprache erreicht.

④ **Hohe Abstraktionsebene der Modellierung:** Die Spezifikation findet auf derselben Abstraktionsebene wie die Spezifikation des zu überwachenden Systems im modellbasierten Entwicklungsprozess statt. Hierfür wird das Konzept des szenariobasierten Designs eingesetzt, in der die MBSecMonSL, die Szenarien in der eLSC-Sprache beschreibt und diese über Use- und Misuse-Cases der MUC-Sprache in Beziehung setzt. Basierend auf den Kommunikationsschnittstellen der Systemelemente werden die Signaturen modelliert. Wird eine feingranularere Überwachung bis hin zur Methodenebene benötigt, können für die Signaturmodellierung weitere Ereignisse durch Instrumentierung der Systemimplementierung bereitgestellt werden.

⑤ **Unterstützung aller wichtigen Konzepte zur Beschreibung komplexer Kommunikationssignaturen:** Die Spezifikationssprache muss alle Muster beschreiben können, die zur Spezifikation von Signaturen notwendig sind. Hierfür wurde die Ausdruckstärke der Live Sequence Charts in Abschnitt 4.3 untersucht, die Schwächen in Bezug auf die Modellierung verhaltensbeschreibender Signaturen als Szenarien aufgezeigt und um benötigte Modellierungskonzepte erweitert. In Kombination mit der MUC-Sprache lassen sich mit der MBSecMonSL Signaturen kompakt und lesbar modellieren.

⑥ **Hohe Verständlichkeit der Spezifikation:** Die Spezifikation muss sowohl für Softwaretechniker als auch für Nicht-Experten verständlich sein, um eine schnelle Modellierung der Signaturen zu gewährleisten und den Kunden die Absicherung ihres Systems zu vermitteln. Die MBSecMon-Spezifikationssprache (MBSecMonSL) erfüllt diese Anforderung, indem sie auf einer grafischen Notation basiert, die stark an den UML2-Standard angelehnt ist, und somit vielen Softwaretechnikern bekannt ist. Zusätzlich sind die zu überwachenden Szenarien, die in der MBSecMonSL spezifiziert sind, auch für Nicht-Experten klar verständlich.

Zusammenfassend wird in dieser Arbeit der MBSecMon-Prozess zur automatischen Generierung von Sicherheitsmonitoren vorgestellt. Der erste Fokus liegt hierbei auf der *MBSecMon-Spezifikationssprache*, die in Teil III vorgestellt wird.

- Sie unterstützt alle wichtigen Konzepte zur Modellierung verhaltensbeschreibender Signaturen als Szenarien.
- Sie unterstützt die abstrakte Modellierung der erlaubten und verbotenen Kommunikation auf Systemebene bzw. Systemelementebene bis hin zur Spezifikation des Kontrollflusses durch Instrumentierung des implementierten zu überwachenden Systems.
- Sie unterstützt die kompakte Modellierung hoch nebenläufiger Kommunikationsabläufe.
- Sie ist dadurch, dass sie auf der UML aufbaut und diese nur um einige Elemente erweitert, vielen Softwaretechnikern schnell vertraut.
- Sie ist durch ihre grafische Notation, die auf Sequenzdiagrammen basieren, auch für Nicht-Experten verständlich.

Abbildung 1.5: Aufbau der Thesis

Der zweite Fokus liegt auf der *Übersetzung* der durch ihre Ausdrucksstärke sehr kompakten Spezifikationen in der MBSecMonSL in die expliziteren aber dennoch kompakten MPNs. Diese Übersetzung wird in Teil IV vorgestellt.

- Sie bildet die MBSecMonSL auf die formalisierten MPNs ab und legt somit die von den LSCs abweichende Semantik der eLSCs und der Beziehungen zwischen den Use- und Misuse-Cases fest.
- Sie definiert einen Übergang zwischen der Spezifikation in der MBSecMonSL und der zur Codegenerierung besser geeigneten MPN-Sprache.

Diese beiden Prozessschritte der Spezifikation und Übersetzung werden im Kontext des gesamten MBSecMon-Generierungsprozesses in Teil V eingebettet und evaluiert.

1.4 Überblick

Diese Arbeit ist in insgesamt sechs Teile (Abb. 1.5) gegliedert, wobei Teil III und Teil IV den Hauptbeitrag der Arbeit enthalten. Teil I und Teil VI bilden den Rahmen der Arbeit. Während Teil I in das Thema einführt, die Problemdomäne beschreibt, den Beitrag erläutert und einen groben Überblick über den MBSecMon-Generierungsprozess und den Aufbau der Arbeit gibt, werden in Teil VI die Arbeit und die Ergebnisse zusammengefasst, bewertet und offene Punkte aufgezeigt. Teil II gibt einen Überblick über die im MBSecMon-Prozess eingesetzten Techniken und über die in [Pat14] vorgestellten MPNs. Die zwei Hauptteile setzen sich jeweils aus Grundlagen, dem Beitrag und einer Implementierung zusammen. Teil V betrachtet die Einsetzbarkeit des MBSecMon-Prozesses und somit der MBSecMonSL aus Teil III in einem existierenden Entwicklungsprozess und evaluiert die Implementierungen der in Teil IV beschriebenen Transformation.

1.5 Durchgängiges Beispiel: Car2X-Mautbrückenszenario

Die in der Einleitung angesprochene Vernetzung der Fahrzeuge untereinander (*Car2Car*-Kommunikation) und der Fahrzeuge mit Roadside-Units (*Car2Infrastruc-*

ture-Kommunikation), die Informationen an das Fahrzeug liefern, schreitet immer weiter voran. So wird in Deutschland für die Mauterfassung auf Autobahnen für LKWs ein satellitengestütztes Mautsystem mit dem Namen Toll Collect eingesetzt. Die Verwendung mautpflichtiger Strecken durch On-Board-Units (OBEs), die den Fahrtweg über GPS aufzeichnen, ermittelt und die erfassten Daten über Mobilfunk mit dem Rechenzentrum der Abrechnungsstelle austauschen. Die Mautbrücken über den Autobahnen dienen in diesem System nur zur Bilderfassung der Fahrzeuge und zum Abgleich mit den Daten die durch die OBEs gemeldet werden, um Betrug aufzudecken. Die Spezifikationen dieses System sind jedoch nicht öffentlich verfügbar. Aus diesem Grund dient als durchgängiges Beispiel in dieser Arbeit das öffentlich zugängliche CARDME-4-Protokoll [Oeh02].

1.5.1 *Intention des CARDME-Protokolls*

Ein aktuelles Konzept eines Gebührenerfassungssystems im Car2X-Umfeld (u. a. für Mautbrückenszenarien) ist das CARDME-4-Protokoll [Oeh02], das ein anbieterunabhängiges Bezahlsystem zwischen allen europäischen teilnehmenden Partnern in ihrem Konzessionsgebiet ermöglicht. Die Unabhängigkeit des CARDME-Systems von Hardware und die Einsetzbarkeit für sowohl offene als auch geschlossene Systeme soll die Einbindung in bestehende Systeme vereinfachen. Durch die Verwendung eines zentralen Kontensystems erhält der Kunde eine kombinierte monatliche Abrechnung für alle Transaktionen, sei es im *Heim*gebiet seines Anbieters oder im Konzessionsgebiet anderer Anbieter (z. B. im Ausland). Somit kann CARDME zu jedem bestehenden elektronischen Gebührenerfassungssystem (EFC-System) ohne großen Aufwand hinzugefügt werden.

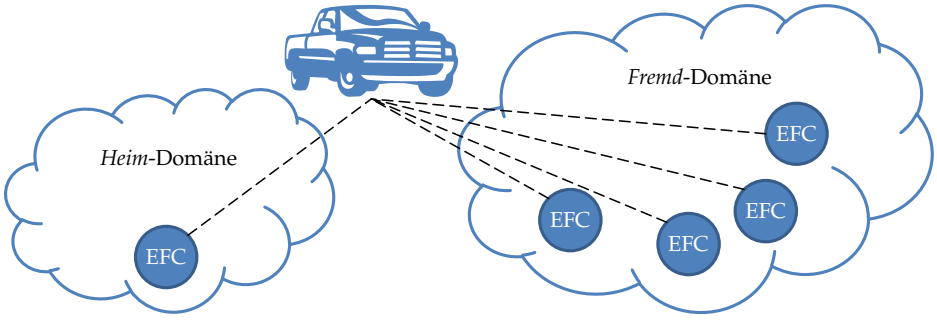


Abbildung 1.6: Domänen des CARDME-Konzepts nach [Oeh02]

Grundlegend unterscheidet das CARDME-Konzept zwischen zwei Domänen (Abb. 1.6) – der *Heim-* und der *Fremd*-Domäne, in denen verschiedenen Betreiber unterschiedliche EFC-Systeme einsetzen können. Die Interaktion der Teilnehmer des CARDME-Konzepts ist in Abbildung 1.7 dargestellt. Den Vertrag für die *Fremd*-Domäne stellt der Vertragsaussteller der *Heim*-Domäne aus. Wenn der Kunde ein EFC-System in der *Fremd*-Domäne passiert, übermittelt er die Vertrags- bzw. Zahlungsinformationen seines Vertragsausstellers. Die Abrechnung des Betreibers des EFC-Systems in der *Fremd*-Domäne erfolgt durch die Übermittlung einer Forderung mit den Zahlungsinformationen des Kunden an den Vertrags-

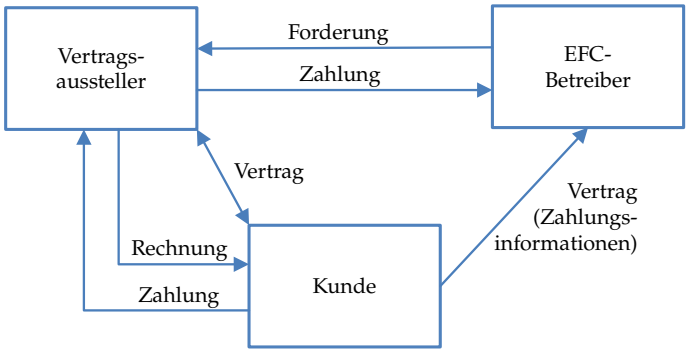


Abbildung 1.7: Instanzen des CARDME-Konzepts nach [Oeh02]

aussteller. Dieser überprüft die Forderung auf Gültigkeit, erfüllt sie und rechnet dann mit dem Kunden über die vereinbarte Bezahlmethode ab.

Damit ein Kunde dieses System nutzen kann, bekommt er von seinem Vertragsaussteller ein *On-Board Equipment (OBE)* gestellt, in der die Vertragsinformationen und die Eigenschaften des Fahrzeugs eingespeichert sind. Die OBE übernimmt dann die Kommunikation mit allen EFC-Systemen, den *Road-Side Equipment (RSE)*.

1.5.2 *Kommunikationsphasen des CARDME-Protokolls*

Die Transaktion mit dem RSE, in diesem Beispiel eine Mautbrücke, ist in vier Hauptphasen unterteilt – die *Initialisierungs-*, die *Präsentations-*, die *Quittungs-* und die *Verfolgungs- und Schließenphase*.

Die *Initialisierungsphase* dient zur Aushandlung des zu verwendenden EFC-Vertrags. Hierzu sendet die Mautbrücke kontinuierlich ein *Beacon*-Signal aus, um Kontakt mit sich nähernden Fahrzeugen aufzunehmen. Dieses periodische Signal übermittelt die *Vehicle Service Table (VST)*, der die OBE des Fahrzeugs darüber informiert, dass das Fahrzeug in einen mautpflichtigen Bereich eintritt. Empfängt die OBE des Fahrzeugs diese Nachricht, antwortet diese auf das Signal der Mautbrücke mit der *Vehicle Service Table (VST)*, die eine Liste mit allen EFC-Verträgen der OBE enthält. Basierend auf den Informationen, die in der VST abgelegt sind, und den Diensten, die die Mautbrücke anbietet, wählt die Mautbrücke ein passendes Protokoll aus und geht in die nächste Phase über.

In der *Präsentationsphase* liest die Mautbrücke die benötigten Informationen der OBE über eine DSRC-Verbindung, eine semi-passive Transpondertechnik mit kleinem Kommunikationsradius, aus. Die Entscheidung der Mautbrücke, welche Daten benötigt werden, hängen von dem in der vorherigen Phase gewählten EFC-Vertrag ab. Hierzu gehören unter anderem die persönliche *Kontoidentifikationsnummer* mit einer *optionalen Signatur*, die vorher erhaltenen *Quittungen* und Details zur *Klassifikation des Fahrzeugs*. Die *Kontoidentifikationsnummer* identifiziert den Nutzer und seinen Vertrag mit seinem Vertragsaussteller und dient zur Abrechnung mit einem lokalen Nutzer oder zum Einzug von Geld eines fremden Vertragsausstellers. Die zwei aktuellsten *Quittungen*, die durch EFC-CARDME-Stationen ausgestellt wurden, werden aus dem Speicher der OBE ausgelesen.

Diese Transaktion wird immer durchgeführt um eine einheitliche Kommunikation zu gewährleisten. Jedoch werden die Quittungsinformationen abhängig vom Szenario entweder zur Erfassung der gefahrenen Strecke in einem geschlossenen Mautbrückenszenario ausgewertet, oder ignoriert. Die *Klassifikation des Fahrzeugs* erfolgt in diesem Protokoll durch die Übermittlung der hinterlegten Daten in der OBE. Mautbrücken, die die Klasse des Fahrzeugs über Sensoren bestimmen, ignorieren diese Daten. Die mit der Identifikationsnummer übermittelte Signatur repräsentiert einen Platzhalter für einen Sicherheitsmechanismus, der in diesem Beispiel als eine zweite Präsentationsphase für Fahrzeuge, deren Vertragsaussteller nicht dem lokalen Betreiber der Mautbrücke entspricht, realisiert wird.

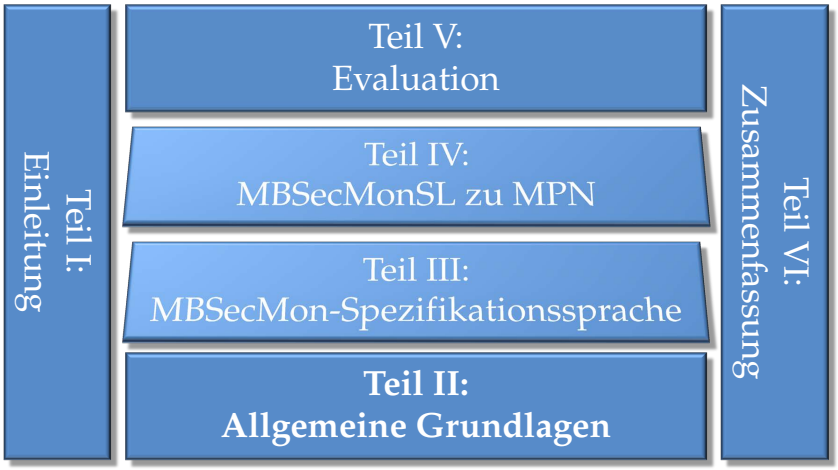
Die optionale *zweite Präsentationsphase* ist eine Anfrage der Mautbrücke an die OBE, um eine zusätzliche Authentifizierung der Zahlungsinformationen durchzuführen. Hierbei wird ein Schlüssel übermittelt, der nur dem Vertragsaussteller bekannt ist. Dieser ermöglicht dem Betreiber der Mautbrücke zu beweisen, dass das Fahrzeug tatsächlich die Mautbrücke passiert hat.

In der *Quittungsphase* schreibt die Mautbrücke die neue Quittung in den Speicher der OBE und erhöht den Transaktionszähler. Ist dies geschehen, sendet die OBE eine Bestätigung für den Erhalt der Quittung. Während der Verarbeitung der Transaktion durch die Mautbrücke findet eine *Verfolgung* des Fahrzeuges statt. Hierbei wird an die OBE ein Echo-Signal gesendet, auf die die OBE antworten muss. Erst wenn die Transaktion verarbeitet ist, wird von der Mautbrücke die *Schließenphase* mit einem Schließen-Signal durchgeführt.

Dieses Kommunikationsprotokoll wird im Folgenden als durchgängiges Beispiel verwendet. Hierbei wird in den meisten Fällen von der OBE abstrahiert, und in den modellierten Signaturen stattdessen allgemein vom Fahrzeug (*engl. vehicle*) gesprochen.

Teil II

ALLGEMEINE GRUNDLAGEN



2

MODELLIERUNGSSPRACHEN

In der Softwareentwicklung werden Modellierungssprachen eingesetzt, um die Struktur und das Verhalten von Softwaresystemen auf einer abstrakteren Ebene als die Implementierung selbst zu beschreiben. Diese Modellierung der Eigenschaften eines Systems in konkreter Syntax kann sowohl textuell als auch grafisch erfolgen. Die grafische Spezifikation der Anforderungen findet in der Anforderungsphase eines Entwicklungsprozesses statt. Durch diese Abstraktion der Sicht auf das System soll die Funktionalität und Struktur selbst für Nicht-Experten verständlich dargestellt werden. Um dies zu erreichen, ist neben der Wahl des richtigen Abstraktionsgrades eine klar definierte Syntax und statische (deklarative) Semantik der Modellierungssprachen zwingend erforderlich [PM06]. Die Syntax der Modelle wird entweder über eine Grammatik oder über Metamodelle spezifiziert, wobei in dieser Arbeit die zweite Variante in einem Modellbasierten Entwicklungsprozess (MDE) angewendet wird.

In Abschnitt 2.1 werden zunächst die in dieser Arbeit verwendeten Strukturdiagramme (Klassen- und Komponentendiagramme) der Unified Modeling Language (UML) vorgestellt. Folgend beschreibt Abschnitt 2.2 die Funktion der Metamodellierung und seine Rolle in der MDE bzw. der Model Driven Architecture (MDA) der OMG. Abschnitt 2.3 stellt verschiedene Zustands-/Transitionssysteme vor und setzt hierbei den Fokus auf die Monitor-Petrinetze (MPNs) des MBSecMon-Entwicklungsprozesses. Weitere verhaltensbeschreibende Diagramme – Anwendungsfalldiagramme und Sequenzdiagramme –, die die Grundlage der MBSecMon-Spezifikationssprache (MBSecMonSL) bilden, werden in Kapitel 3 ausführlich vorgestellt.

2.1 Strukturdiagramme

Strukturdiagramme beschreiben statisch den technischen Aufbau eines Systems und visualisieren die Beziehungen bzw. den Informationsfluss zwischen dieses Strukturen. Zu diesem Zweck werden in dieser Arbeit zwei Varianten der Strukturdiagramme aus der UML2 verwendet: Zum einen die *Klassendiagramme*, die hauptsächlich zur Beschreibung von Datenstrukturen bzw. Teilsystemen eingesetzt werden und zum anderen die *Komponentendiagramme*, die auf einer höheren Abstraktionsebene die Kommunikationsschnittstellen zwischen Teilsystemen beschreiben.

2.1.1 *UML2-Klassendiagramme*

Klassendiagramme [RQJZ07] werden zur Darstellung der inneren Struktur des zu repräsentierenden oder zu entwickelnden Systems verwendet. Sie beschreiben die wichtigsten statischen Eigenschaften eines Systems, die Datentypen und das Zusammenspiel der Klassen untereinander. Die Notation der Klassendiagramme repräsentiert den Kern der UML und wird ebenfalls zur Metamodellierung (Abschn. 2.2) der UML eingesetzt.

Beispiel *Metamodell mit Klassendiagrammnotation*

Abbildung 2.1 zeigt einen Ausschnitt eines Metamodells, das zur Spezifikation der Transformation von LSCs zu MPNs eingesetzt wird. Die Notation der Elemente entspricht den UML-Klassendiagrammen. Es beschreibt die Struktur eines Modells, das während der Transformation – eine automatische Übersetzung von einem Modell in ein anderes Modell – Buch über die Zusammenhänge zwischen den Elementen des Quell- und Zielmodells führt. In diesem Abschnitt wird die Syntax des Metamodells bzw. Klassendiagramms betrachtet und später in Abschnitt 7.1 auf den Einsatz in einer Transformationsspezifikation eingegangen.

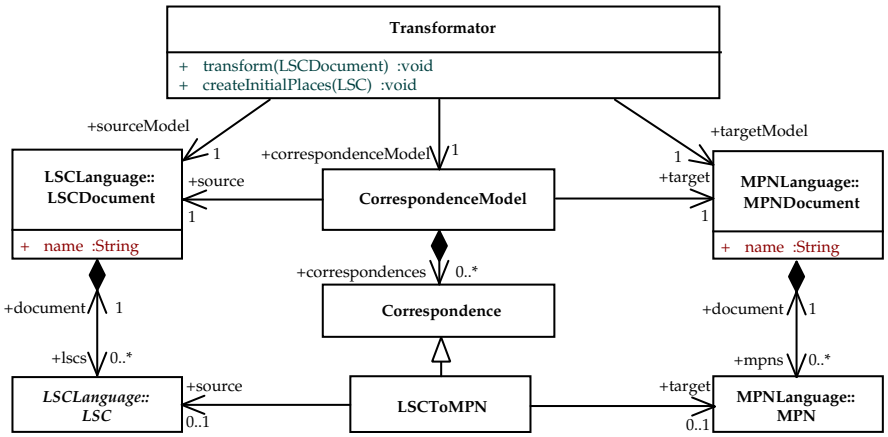


Abbildung 2.1: Transformationsmetamodell eLSC-zu-MPN (Ausschnitt)

Zur Abbildung statischer Daten werden in Klassendiagrammen Klassen und deren Attribute eingesetzt. *Klassen* beschreiben in der UML eine Menge von Objekten, die über gemeinsamen Eigenschaften, Semantik und dasselbe Verhalten verfügen, und sind somit eine Abstraktion ihrer Ausprägungen, den Objekten. Klassen werden als Rechteck dargestellt, und ihr eindeutiger Name wird im oberen Abschnitt des Rechtecks notiert. Weitere Eigenschaften wie Attribute und Operationen werden in weiteren Abschnitten, abgetrennt durch eine horizontale Linie, annotiert. Attribute werden üblicherweise im zweiten und Operationen im dritten Abschnitt dargestellt. Die Notation der *Attribute* besteht aus einer Sichtbarkeitsangabe, dem eindeutigen Namen des Attributs und einem Typ, der ein Skalar (z. B. `int`, `boolean`) oder eine Klasse sein kann.

Beispiel *Klassen und Attribute*

Das Klassendiagramm in Abbildung 2.1 besteht aus acht Klassen. Die Klasse `Transformator` besitzt zwei Abschnitte, wobei im oberen der Name der Klasse und im unteren zwei Operationen stehen. `LSCDocument` und `MPNDocument` besitzen jeweils ein Attribut mit dem Namen `name` und dem UML-Datentyp `String`, jedoch keine Operationen. Alle anderen Klassen, die die Struktur eines Korrespondenzmodells der Transformation beschreiben, besitzen weder Attribute noch Operationen. Auf der linken Seite des Diagramms sind zwei Klassen aus dem Klassendiagramm `LSCLanguage` abgebildet – `LSCDocument` und `LSC`. Diesen Klassen ist ein eindeutiger Namensraum zugeordnet, der den Namen der Behälter (z. B. Pakete) getrennt durch `::` entspricht (`LSCLanguage::LSCDocument`). Durch die Verwendung des voll qualifizierten Namens der Klasse wird bei Kombination mehrere Klassen aus verschiedenen Diagrammen der Ursprung der Klassen angegeben und die Eindeutigkeit der Namen der Klassen gewährleistet. Die Klassen des MPN-Modells auf der rechten Seite sind dem Namensraum `MPNLanguage` zugeordnet.

Die Beziehungen zwischen den Klassen werden durch Generalisierungsbeziehungen und Assoziationen mit ihren Spezialfällen der Aggregation und Komposition modelliert. *Generalisierungsbeziehungen* werden durch einen Pfeil mit einer nicht ausgefüllten dreieckigen Spitze repräsentiert. Der Pfeil zeigt von der spezialisierten *Kindklasse* zu der allgemeineren *Elternklasse*, von der Attribute, Operationen und Beziehungen zu anderen Klassen, die durch Assoziationen modelliert sind, geerbt werden, wenn es die Sichtbarkeit erlaubt. *Assoziationen* dienen zur Beschreibung von Beziehungen zwischen Klassen und werden durch durchgezogene Linien repräsentiert. Neben diesen binären Assoziationen existieren in der UML auch n-äre Assoziationen, die hier jedoch nicht genauer betrachtet werden. An den Enden der Assoziationen können optional die Navigationsrichtung, unidirektional oder bidirektional, angegeben werden und den Enden Rollennamen und Multiplizitäten zugeordnet werden. Der Rollenname beschreibt die genauere Bedeutung der Klasse für die Klasse am anderen Ende der Assoziation. Die Kompositionsbeziehung ist eine Teil-Ganzes-Beziehung. Sie wird durch eine ausgefüllte Raute an der Assoziation auf der Seite der besitzenden Klasse (Ganzes) repräsentiert. Die Aggregationsbeziehung ist eine abgeschwächte Variante der Kompositionsbeziehung und wird hier nicht weiter betrachtet.

Beispiel *Beziehungen zwischen Klassen*

Im Klassendiagramm in Abbildung 2.1 werden verschiedene Varianten der Assoziationen eingesetzt. Die Klasse `CorrespondenceModel` kennt die Klasse `LSCDocument` unter dem Rollennamen `source` und die Multiplizität `1` am navigierbaren Ende der Assoziation legt fest, dass nur eine Instanz der Klasse `LSCDocument` der Instanz der Klasse `CorrespondenceModel` zugeordnet werden darf. Das `CorrespondenceModel` besitzt durch die Kompositionsbeziehung zu der abstrakten Klasse `Corresponcence` null bis beliebig viele Instanzen dieser Klasse. Durch eine Generalisierungsbeziehung zwischen der abstrakten Klasse `Correspondence` und `LSCToMPN` ist `LSCToMPN` eine Ausprägung der Klasse `Corespondence`. Somit dient die Kompositionsbeziehung als Sammelbehälter,

in dem alle Ausprägungen von `Correspondence` enthalten sind. Eine beidseitig navigierbare Assoziation definiert die Kompositionsbeziehung zwischen den Klassen `LSCDocument` und `LSC`.

Verhalten bzw. Zustandsänderungen der Instanzen einer Klassen (Objekte) wird auf Klassenebene durch Operationen definiert. Diese Operationen lösen ein Verhalten aus, das durch verhaltensbeschreibende Diagramme der UML beschrieben werden kann. Bei der Implementierung des Systems werden Operationen als Methoden repräsentiert. Operationen besitzen in ihrer Signatur (Auflistung 2.1) immer ihren Namen, der die Funktionalität beschreibt, die Sichtbarkeit, die vor dem Namen annotiert wird und die Zugreifbarkeit festlegen. Optional können in Klammern beliebig viele Parameter kommasepariert angegeben werden, die mindestens aus Name und Datentyp des Parameters bestehen. Beendet wird die Operationssignatur mit dem Rückgabetyp, der ein Skalar oder eine Klasse ist, und am Ende der Ausführung der Operation zurückgeliefert wird.

Auflistung 2.1: Abstrakte Operationssignatur

```
[Sichtbarkeit] Name (Param_Name : Typ, ...) : Rückgabetyp
```

Beispiel *Operationen*

Das Klassendiagramm in Abbildung 2.1 enthält, wie vorher erwähnt, nur die Klasse `Transformator`, die Operationen besitzt. Beide Methoden haben die Sichtbarkeit *public* (+) und einen Parameter, der hier nur mit dem Typ `LSCDocument` bzw. `LSC` dargestellt ist. Der Rückgabetyp `void` kennzeichnet, dass die Operation keinen Rückgabewert zurückliefert.

Während Klassendiagramme das System auf Ebene der Klassen beschreiben, stellen Komponentendiagramme die Beziehungen zwischen Teilsystemen, bestehend aus mehreren Klassen, auf einer abstrakteren Ebene dar.

2.1.2 *UML2-Komponentendiagramme*

Komponentendiagramme beschreiben Bestandteile eines Systems (Komponenten) und liegen somit eine Abstraktionsebene über den Klassendiagrammen, da Klassen und somit Verhalten in Komponenten zusammengefasst werden können. Der Fokus dieser Diagramme liegt auf der Festlegung klar definierter Schnittstellen, über die auf das Verhalten der Komponenten zugegriffen werden kann. Für diese Beschreibung unterstützen Komponentendiagramme alle Elemente der Klassen- und Objektdiagramme und bieten zusätzlich einige neue Modellelemente an. Die zum Verständnis dieser Arbeit notwendigen Elemente sind die Komponenten selbst, Schnittstellen und Ports, die im Folgenden vorgestellt werden.

Komponenten werden wie Klassen als Rechteck dargestellt und zusätzlich mit dem Stereotyp «component» versehen. Dieser Stereotyp kann alternativ als grafisches Element oben rechts in der Komponente als Rechteck mit zwei kleineren Rechtecken auf der linken Seite dargestellt werden. Den Komponenten werden über *Ports*, kleine Rechtecke, die den Rand der Komponente überlappen, externe

Schnittstellen zugeordnet werden. Diese Schnittstellen können durch die Lollipop-Notation, als bereitgestellte Schnittstelle und durch die Socket-Notation, als benötigte Schnittstelle markiert werden.

Beispiel *Komponenten und Ports*

Als Beispiel für ein Komponentendiagramm dient die Kommunikation zwischen Mautbrücke und Fahrzeug in Abbildung 2.2. Hierbei ist zum einen das gesamte Fahrzeug (`Vehicle`) und die gesamte Mautbrücke (`Tollbridge`) als jeweils eine Komponente modelliert. Zusätzlich besteht das Fahrzeug aus zwei weiteren Komponenten: dem *Tollbridge Module* (`TBM`), das zur Kommunikation mit Mautbrücken verantwortlich ist und dem *Driver Information Center* (`DIC`), das Informationen des TBM dem Fahrer präsentiert. In diesem Beispiel werden

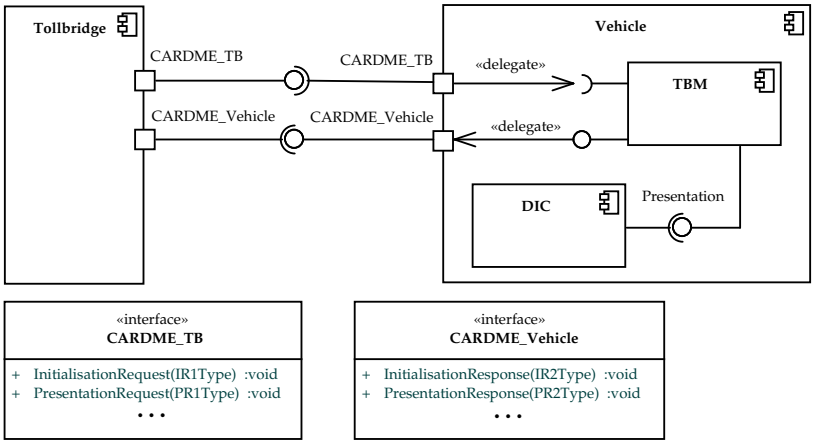


Abbildung 2.2: CARDME-Beispiel als UML2-Komponentendiagramm

Ports zur Repräsentation der Schnittstellen der Komponente `Tollbridge` und der Komponente `Vehicle` nach außen verwendet. Die jeweiligen Ports sind mit dem entsprechenden Protokoll `CARDME_TB` und `CARDME_Vehicle` beschriftet. Eine weitere Variante der Schnittstellenrepräsentation sind die in der Lollipop-Notation und der Socket-Notation repräsentierten Schnittstellen des TBM. Das Protokoll der Schnittstellen bzw. Ports wird durch Klassen mit dem Stereotyp «interface», deren Name am Port annotiert ist, genauer beschrieben.

Ein *Konnektor*, eine durchgezogene Linie zwischen einer Schnittstelle (Port) und einer anderen kompatiblen Schnittstellen, spezifiziert eine Kommunikationsverbindung. Die direkte Verbindung zwischen Ports oder auch zwischen Komponenten kann auch durch einen *Kompositionskonnektor*, der die Lollipop-Notation beinhaltet, hergestellt werden. Sollen Ports mit Schnittstellen in der Lollipop- bzw. Socket-Notation verbunden werden, kann ein *Delegationskonnektor*, ein offener Pfeil mit dem Stereotyp «delegate» verwendet werden.

Beispiel *Konnektoren*

Im Komponentendiagramm wurden zur Repräsentation der Kommunikationsverbindungen hauptsächlich Kompositionskonnektoren verwendet. Während die Komponenten `Tollbridge` und `Vehicle` mit einem Kompositionskonnektor

über die Ports `CARDME_TB` und `CARDME_Vehicle` miteinander verbunden sind, zeigt der Konnektor mit dem Namen `Presentation`, dass auch Komponenten direkt ohne Ports verbunden werden dürfen. Die Schnittstellen des `TBM` werden als weitere Alternative über eine Delegationsbeziehung mit den nach außen sichtbaren Ports der Komponente `Vehicle` angebunden.

In diesem Abschnitt wurde nur auf die für die Arbeit relevanten Modellierungselemente der Komponentendiagramme eingegangen. Nähere Informationen zu den UML-Strukturdiagrammen und ihre Anwendung kann in [RQJZ07] nachgelesen werden.

2.2 METAMODELLIERUNG

Definition 3 (*Modell*)**.** „Ein *Modell* stellt ein *Abbild* eines realen oder abstrakten Systems [...] dar, das Analogien aufweist und durch ein oder mehrere Aspekte eine Abstraktionsbeziehung zum Ausgangsobjekt besitzt. [...] Das Ziel von Modellen ist i. d. R. die Reduktion der Komplexität durch Weglassen von Eigenschaften und Berücksichtigung nur eines oder einiger weniger Aspekte."
[PM06, S. 44]

In der Modellbasierten Softwareentwicklung (MDE) dienen Modelle, wie die in den vorherigen Abschnitten vorgestellten strukturbeschreibenden Modelle und die verhaltensbeschreibenden Modelle, als Abstraktion vom realen zu entwickelnden System. Verschiedene Modelle können genutzt werden, um unterschiedliche Perspektiven abzubilden. Neben der konkreten Syntax der Modellierungssprache, in der der Modellierer die Modelle erstellt, wird zur maschinellen Verarbeitung, Speicherung und Austausch der Modelle eine eindeutige Spezifikation der Modellierungssprache benötigt. Diese Spezifikation der abstrakten Syntax erfolgt durch ein weiteres Modell (Metamodell), repräsentiert in einer Metasprache. Dieses Metamodell muss ebenfalls eindeutig spezifiziert sein, um u. a. eine Speicherung verschiedener Modelle, die auf unterschiedlichen Metamodellen basieren, in einem gemeinsamen Modelldepot (Repository) zu ermöglichen. Ein weiteres Modell, das Meta-Metamodell, beschreibt die Instanzen des Metamodells in einer Meta-Metasprache. Um diese Rekursion der Beschreibung von Instanzmodellen durch Metamodelle zu beenden, beschreibt das Meta-Metamodell sich selbst mit eigenen Sprachmitteln.

2.2.1 *Die Meta Object Facility der OMG*

Eine spezielle Variante einer solchen Metadaten-Architektur stellt die Object Management Group (OMG) mit der Model-Driven-Architecture (MDA) [PM06] und der darin verwendeten Meta Object Facility (MOF) 2.0 [OMG06] als De-Facto-Standard zur Verfügung. Der Standard definiert mit der MOF-Sprache ein gemeinsames Meta-Metamodell (MOF-Modell), auf dem alle Metamodelle der OMG-

Modellierungssprachen basieren. Eine der bedeutendsten Modellierungssprachensammlungen der OMG ist die Unified Modeling Language (UML), die struktur- und verhaltensbeschreibende Modellierungssprachen zusammenfasst. Die MOF-Metadaten-Architektur (Abb. 2.3) definiert vier Abstraktionsebenen M_0 bis M_3. Jede Ebene M_x instanziiert die im Abstraktionsgrad höher liegende Ebene M_{x+1} und ist somit lokal betrachtet als Modell zu sehen, das die Beschreibungsmittel des Metamodells auf Ebene M_{x+1} nutzt.

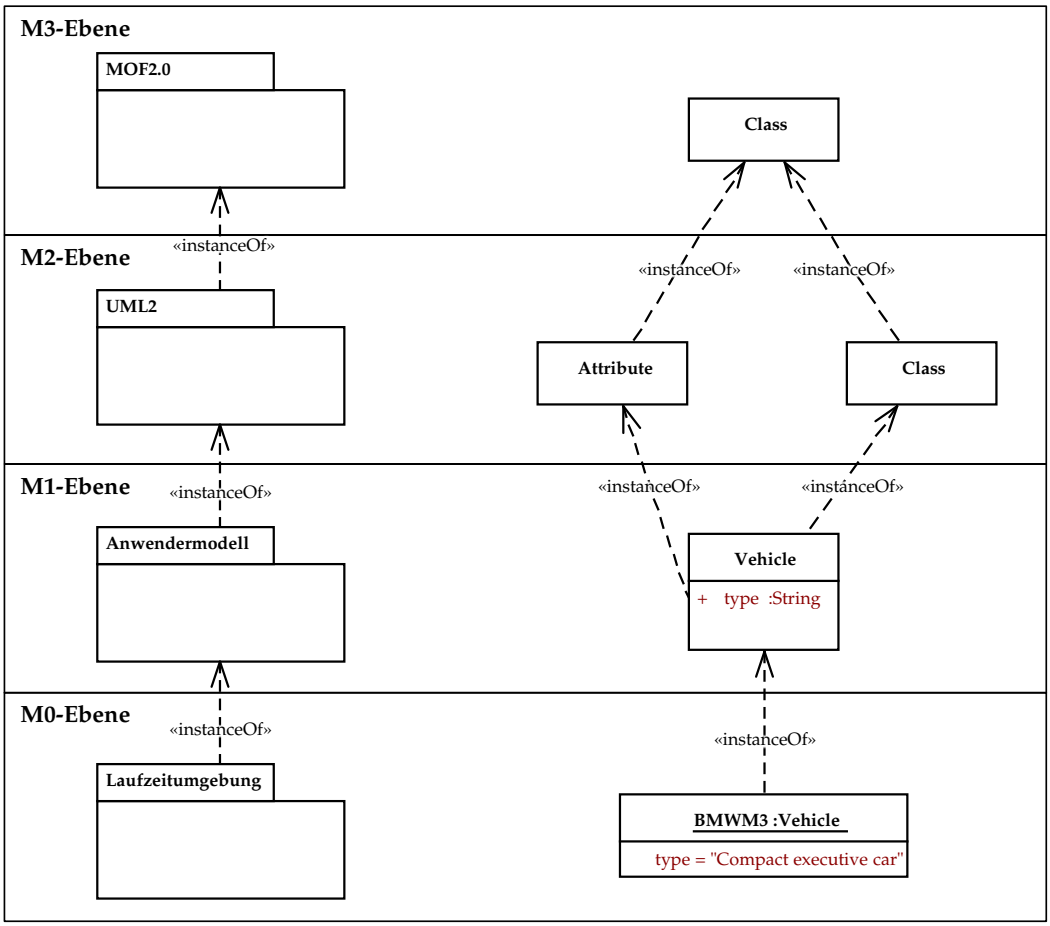


Abbildung 2.3: Die Vierschichtenarchitektur der UML

Die oberste und sich selbst beschreibende Ebene der Vier-Schichten-Architektur bildet die M_3-Ebene, die Meta-Metamodellierungsebene, die dem MOF-Modell entspricht. Diese Beschreibung der eigenen Sprachkonstrukte erfolgt durch die Definition eines *Reflection*-Pakets [OMG06, OMG11b], das jedes Modell um diese Eigenschaft erweitern kann. In der M_2-Ebene, der Metamodellebene, ist jedes Element eine Instanz des Meta-Metamodells der M_3-Ebene. Die OMG definiert auf M_2 das UML-Metamodell, das die Syntax der UML beschreibt. Hierzu zählen u. a. die Klassen-, Komponenten- und Zustandsautomaten der UML, die in Abschnitt 2.1 vorgestellt wurden und Abschnitt 2.3 beschrieben werden. Diese Metamodellelemente werden auf der Modellebene (M_1) vom Modellierer instanziiert, um das reale Softwaresystem auf seine statischen und dynamischen Eigenschaf-

ten reduziert zu modellieren. Die unterste Ebene bildet die M_0-Ebene, die die instanziierten Modelle während der Laufzeit des Softwaresystems beschreibt.

Die MOF 2.0 ist modular aufgebaut und besteht aus zwei Hauptteilen – der Complete MOF (CMOF) und der Essential MOF (EMOF). Die EMOF ist der Kern der Sprache und enthält eine minimale Menge an Elementen, die zur Metamodellierung objektorientierter Systeme benötigt werden. Zusätzlich zu den grundlegenden Mechanismen ergänzt die CMOF den EMOF-Kern um komplexe Sprachkonstrukte zur Modellierung von Metamodellen, wie *Assoziationen*, *PackageMerge* und *PackageImport* [OMG06].

2.2.2 *UML-Profile*

UML-Profile [PM06, OMG11c] stellen neben einer schwergewichtigen Änderung des Metamodells auf der M_2-Ebene, eine leichtgewichtige Erweiterung der bestehenden Sprachen zur Verfügung. Mechanismen für die leichtgewichtigen Änderungen werden durch die UML, basierend auf Mechanismen der MOF, bereitgestellt. Diese Mechanismen werden zur Erweiterung des UML-Metamodells um neue spezialisierte Typen (Stereotypen) mit zusätzlichen Eigenschaften eingesetzt. Diese und die Eigenschaften der Metaklasse können durch Regeln weiter eingeschränkt werden. Durch Annotation dieser Stereotypen in einem UML-Modell, werden diese neuen Eigenschaften auf das annotierte UML-Element angewendet. So kann die UML auf Plattformen und Domänen angepasst werden, ohne die bestehenden Konstrukte des UML-Metamodells zu verlieren, wodurch die Formalisierung der domänenspezifischen Modelle in der MDA vereinfacht wird. Um dies zu erreichen, unterliegt der Profilmechanismus jedoch den Einschränkungen, dass das UML-Metamodell als schreibgeschützt angesehen wird. Hieraus resultiert, dass keine neuen Metaklassen dem Metamodell hinzugefügt oder die Klassen des UML-Standards selbst verändert werden dürfen.

Profile bestehen aus Paketen, die mit dem Stereotyp «profile» annotiert sind, und gruppieren Stereotypen und einschränkende Randbedingungen (*engl. constraints*). Zusätzliche Eigenschaften der Stereotypen werden als Attribute der Stereotypen (*Tagged Values*) hinterlegt, die einen Namen und einen Wert besitzen. Die Randbedingungen werden in UML-Profilen durch OCL-Constraints [OMG10] oder natürlichsprachlich textuell spezifiziert.

Beispiel *Stereotypen in der MBSecMonSL*

In dieser Arbeit werden UML-Profile zur Anpassung des Werkzeuges Enterprise Architect (EA) verwendet, um basierend auf den UML-Sequenzdiagrammen von EA die Modellierung von Live Sequence Charts zu unterstützen. Hierzu müssen zusätzlich zu den Lebenslinien der UML2 u. a. zwei weitere Varianten der Lebenslinien im Werkzeug modelliert werden können – *Symbolic Instance* und *Environment*. Der Ausschnitt des UML-Profils in Abbildung 2.4 zeigt die Metaklasse `Object`, die mit dem Stereotyp «metaclass» annotiert ist. Die zwei neu definierten Stereotypen werden als Klasse modelliert, deren Name den Namen des Stereotyps repräsentiert. Zusätzlich sind diese Stereotypklassen mit «stereotype» bzw. seiner grafischen Repräsentation – dem blauen Stereotyp-

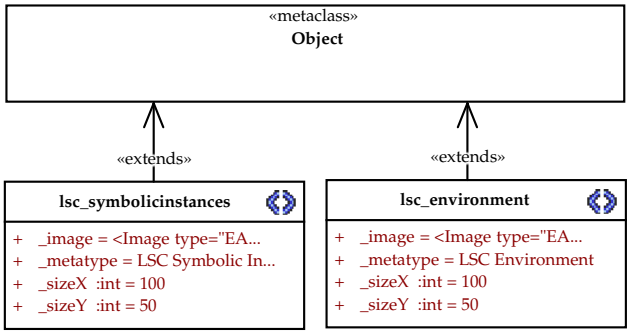


Abbildung 2.4: Stereotypen für eLSC-Lebenslinien

Symbol – markiert und ihre Attribute definieren *Tagged Values*. In diesem Fall sind alle diese *Tagged Values* EA-spezifisch und werden erst in Abschnitt 6 genauer betrachtet. Die Zuordnung des Stereotyps zu der zu erweiternden Metaklasse erfolgt durch eine spezielle Abhängigkeit, die in EA als durchgezogener offener Pfeil, der mit dem Stereotyp «extends» annotiert ist, modelliert wird.

Neben dem MOF-Standard der OMG und den UML-Profilen zur Anpassung der Metamodelle hat sich in der Industrie aufgrund der zu hohen Komplexität des MOF-Standards für Implementierungen der Ecore-Standard des Eclipse Modeling Frameworks (EMF) durchgesetzt. Dieser wird in den meisten Metamodellierungswerkzeugen unterstützt und in dieser Arbeit zur Visualisierung der Metamodelle verwendet.

2.2.3 *EMF und das Ecore-Metamodell*

Der Ecore-Standard ist im Rahmen der Entwicklung des Eclipse Modeling Framework (EMF) [SBPM08] entstanden. EMF ist ein Eclipse-Plugin, das die Erstellung von Modellen und Generierung von JAVA-Quelltext aus diesen Modellen für Java-Entwickler bereitstellt. Diese Integration zwischen der Modellierung und der Programmierung reduziert die Trennung zwischen dem High-Level-Engineering und der Low-Level-Programmierung, wodurch eine Produktivitätssteigerung erreicht wird. Das ursprüngliche Vorhaben des Initiators IBM, die vollständige MOF als Kern des Werkzeuges Rational Rose zu implementieren, stellte sich jedoch als zu komplex für den Einsatz zur Toolintegration heraus. Aus diesem Grund wurde ein eigenes Metamodell *Ecore* spezifiziert. Seit der Weiterentwicklung des MOF-Standards in die Version 2.0 kann Ecore als eine Implementierung des EMOF-Metamodells angesehen werden, die jedoch Unterschiede und Vereinfachungen aufweist. Ecore ist eine vereinfachte Untermenge der vollständigen UML und unterstützt ausschließlich die klassenbasierte Modellierung. Ecore ähnelt der MOF bezüglich der Spezifikation von Klassen, deren strukturellen und verhaltensbeschreibenden Eigenschaften, der Vererbung zwischen ihnen, Paketstrukturen und den reflektiven Eigenschaften der Sprache. Auf komplexe Mechanismen wie das Gebiet der Lebenszyklusbeschreibung, Datentypstrukturen, Paketbeziehungen und komplexe Konzepte der Assoziationen wurden im

Ecore-Metamodell zur effizienteren Codegenerierung verzichtet. Die Assoziationen der UML-Klassendiagramme werden in Ecore-Diagrammen durch ein einfaches Referenzsystem ersetzt. Referenzen sind immer nur einseitig navigierbar, wobei beidseitig navigierbare Assoziationen der UML durch zwei als zusammengehörig markierte Referenzen nachgebildet werden können.

In dieser Arbeit wird trotz des MDA-ähnlichen Ansatzes des Entwicklungsprozesses zur Definition der MBSecMonSL auf das Ecore-Metamodell statt der MOF 2.0 zurückgegriffen. Auch die Implementierung der MBSecMon-Toolsuite basiert auf Mechanismen des EMF-Rahmenwerks.

2.3 Transitionssysteme

Eine der grundlegendsten Modellierungssprachen zur Beschreibung des Verhaltens eines zustandsbasierten Systems sind Transitionssysteme. Sie bestehen aus einer Menge an *Zuständen*, die ein System einnehmen kann, und Übergängen zwischen diesen Zuständen, die durch *Transitionen* modelliert werden. Die verbreitetste Darstellung eines solchen verhaltensbeschreibenden Systemmodells ist ein beschrifteter Graph. Einen Spezialfall der Transitionssysteme sind endliche Automaten, deren Menge der Zustände endlich ist und spezielle Start- und Endzustände besitzt. Hierdurch können diese Automaten als gerichtete, potentiell zyklische Graphen mit beschrifteten Kanten dargestellt werden.

Diese einfachen Varianten der Transitionssysteme erlauben zwar die Beschreibung der Zustände, die ein System einnehmen kann, und die möglichen Übergänge, können jedoch komplexe Systeme nicht überschaubar und vollständig auf einer detaillierten Ebene beschreiben [RQJZ07]. Aus diesem Grund definiert die UML *Zustandsautomaten*, die die Modularisierung der Verhaltensbeschreibung des Systems ermöglicht. Zustandsautomaten basieren auf einer Kombination von Mealy- und Mooreautomaten, die mit neuen Elementen angereichert wurden [Har87]. Zur Vereinfachung der Modellierung trifft die UML für die Zustandsautomaten zwei Einschränkungen:

- Das System befindet sich immer in einem einzigen definierten Zustand.
- Die Übergänge zwischen den Zuständen benötigen keine Zeit.

Durch die Erweiterungen der Automaten um *zusammengesetzte Zustände*, *Unterzustandsautomaten* und weiteren Pseudozuständen, wie z. B. Kreuzungspunkte, Entscheidungen, Gabelung und Vereinigung, kann das Gesamtsystemverhalten in UML-Zustandsautomaten in Teile zerlegt werden, um so eine größere Übersichtlichkeit der Diagramme zu erreichen. Hierbei wird die Einschränkung auf einen einzigen aktiven Zustand durch die Einteilung der Zustände in Regionen jedoch aufgeweicht. Zwar ist immer nur der in Regionen aufgeteilte Zustand (orthogonaler zusammengesetzter Zustand) aktiv, jedoch lassen sich in den einzelnen Regionen durch weitere eingebettete Zustandsautomaten parallele Abläufe modellieren. Ein Verlassen dieses zusammengesetzten Zustands schließt jedoch das Beenden aller Regionen und damit aller parallelen Abläufe ein. Ein direktes Verlassen eines einzelnen Unterzustandes über eine Transition aus dem zusammengesetzten Zustand führt zu einem Beenden aller anderen Regionen. Dies erschwert die Model-

lierung paralleler Abläufe von stark verschränkt ablaufenden Protokollen. Hierfür eignen sich die aus endlichen Automaten entwickelten Petrinetze.

2.3.1 *Petrinetze*

Petrinetze [DR98] wurden von Carl Adam Petri in den 60er Jahren zur Beschreibung nebenläufiger Schaltvorgänge entwickelt. Sie bestehen in der Regel aus einer endlichen Menge an Plätzen (Stellen) und einer endlichen Menge an Transitionen, die jeweils einen Vor- und einen Nachbereich besitzen. Plätze werden durch einen Kreis und Transitionen durch ein Rechteck wie in Abbildung 2.5 repräsentiert. Diese Elemente werden mit gerichteten Kanten von Plätzen zu Transitionen und von Transitionen zu Plätzen zu einem Netz verbunden. Den Vorbereich einer Transition bilden alle Plätze (Vorplätze), die durch ausgehende Kanten mit einer Transition verbunden sind. Den Nachbereich einer Transition bilden alle Plätze (Nachplätze), die mit der Transition durch ausgehende Verbindungen verbunden sind.

Beispiel *Netzstruktur und der Vor- und Nachbereich*

Abbildung 2.5 zeigt eine einfache Netzstruktur eines Petrinetzes. Es setzt sich aus zwei Plätzen im Vorbereich (Vorplätze) der Transition und zwei Plätzen im Nachbereich (Nachplätze) der Transition zusammen. Die Plätze des Vorbereichs sind mit Kanten auf die Transition verbunden, während die Transition mit Kanten zu den Plätzen des Nachbereichs verbunden sind.

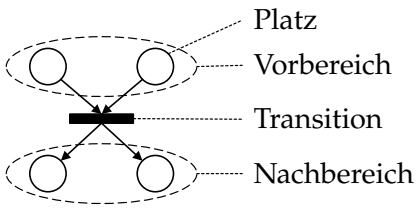


Abbildung 2.5: Netzstruktur und der Vor- und Nachbereich der Transition

Der Zustand eines Petrinetzes wird nicht wie bei den UML-Zustandsautomaten durch einen einzelnen aktiven Zustand repräsentiert, sondern durch eine Menge von aktiven Plätzen. Diese Menge aktiver Plätze wird in Petrinetzen als Markierung des Netzes bezeichnet und die Belegung der Plätze wird durch Punkte dargestellt. Für jedes Petrinetz gibt es eine Anfangsmarkierung, die durch das Schalten der Transitionen verändert wird. Eine Transition kann nur dann schalten, wenn ihre Vorplätze belegt sind und die Transition ausgelöst wird. Dieses Auslösen einer Transition kann durch Beschriftung der Transitionen, die zu einem Wahrheitswert ausgewertet werden kann, beeinflusst werden. Wird dieser Wahrheitswert `wahr` und alle Vorplätze sind belegt, werden die Marken (Token) im Vorbereich der Transition gelöscht und in ihren Nachbereich gelegt.

Beispiel *Markierung und Schalten*

Das Petrinetz auf der linken Seite der Abbildung 2.6 repräsentiert das Petrinetz aus Abbildung 2.5 mit einer Anfangsmarkierung. In den beiden Vorplätzen der

Transition liegt jeweils eine Marke, wodurch die Transition schalten könnte. Die Transition dieses Netzes schaltet jedoch nur, wenn die annotierte Bedingung [x>2] zu wahr evaluiert. Der Schaltvorgang entfernt dann jeweils eine Marke aus den beiden Vorplätzen und erzeugt in den Nachplätzen jeweils eine neue Marke.

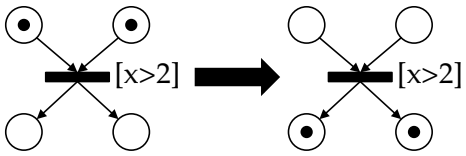


Abbildung 2.6: Schalten und Markierung der Petrinetze

Zusätzlich können die Kanten eines Petrinetzes gewichtet sein und so die zum Schalten der Transition benötigte Anzahl an Markierungen in den Vorplätzen und die erstellten Markierungen in den Nachplätzen der Transition beeinflusst werden.

Beispiel *Gewichtete Kanten*

Abbildung 2.7 zeigt das Petrinetz aus Abbildung 2.5 mit zusätzlich annotierten Kantengewichten. Die Gewichte der in die Transition einlaufenden Kanten beschreiben, wie viele Marken im jeweiligen Vorplatz liegen müssen, damit die Transition schalten kann. Im Petrinetz auf der linken Seite der Abbildung liegt im Vorbereich auf dem linken Vorplatz des Petrinetzes nur eine Markierung, die Kante fordert aber zwei. Hierdurch schaltet die Transition nicht. Auf der rechten Seite der Abbildung befindet sich eine passende Anzahl an Marken im Vorbereich der Transition und diese schaltet. Im Nachbereich werden hierdurch die an den in die Plätze einlaufenden Kanten annotierte Anzahl an Marken erzeugt.

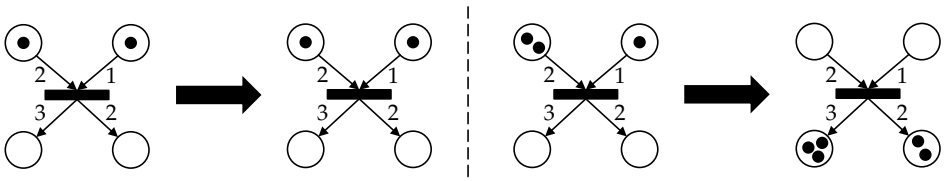


Abbildung 2.7: Gewichtete Kanten in Petrinetzen

In Petrinetzen kann es durch das Konzept, dass Transitionen nacheinander schalten und Marken hierdurch aus einem gemeinsamen Vorbereich entfernt werden, zu Verzweigungskonflikt führen. Hierdurch wird in Petrinetzen ein nichtdeterministisches Verhalten modelliert.

Beispiel *Konflikte*

Ein solcher Verzweigungskonflikt ist in Abbildung 2.8 dargestellt. Der Vorplatz ist mit zwei Transitionen verbunden, die mit derselben Annotation `a` versehen sind. Somit ist zunächst die Vorbedingung zum Schalten beider Transitionen

gegeben. Die Semantik der Petrinetze wählt jedoch eine der beiden Transition zufällig aus, entfernt die Markierung aus dem gemeinsamen Vorplatz und die zweite Transition kann nicht mehr schalten.

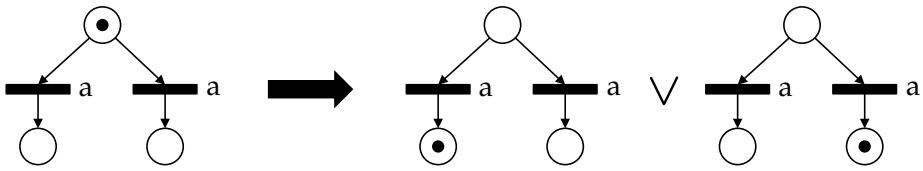


Abbildung 2.8: Konflikte in Petrinetzen

Dieser Nichtdeterminismus wird zur Modellierung von Signaturen, die zur Überwachung eines Systems genutzt werden, nicht benötigt. Da eine unbeabsichtigte Modellierung solcher Konflikte zu fehlerhaften Laufzeitmonitoren im MBSecMon-Prozess führen kann, wurde als Zwischenrepräsentation vor der Monitorgenerierung die Sprache der Monitor-Petrinetze (MPNs) [Pat14] entwickelt.

2.3.2 *Monitor-Petrinetze*

Monitor-Petrinetze (MPNs) [Pat14] sind eine Variante der Petrinetze, die speziell zur Abbildung von komplexen hoch nebenläufigen Monitorsignaturen und deren Auswertung entwickelt wurden. Die Syntax der MPNs ist im Gegensatz zu den in Abschnitt 2.3.1 vorgestellten Standard-Petrinetzen um zusätzliche Platztypen, Annotation der Plätze des MPNs als Vorbedingung und Hauptteil und der Färbung von Marken mit einer Generation erweitert.

Syntax und Netzstruktur: Plätze werden wie in Petrinetzen als Kreise und Transitionen als Rechtecke dargestellt und die Plätze mit Transitionen bzw. die Transitionen mit Plätzen über gerichtete Kanten verbunden. Zusätzlich zu den einfachen Plätzen des Petrinetzes, existieren in MPNs spezielle Initialplätze, und zwei Varianten der Terminalplätze – End- und Fehlerplätze. *Initialplätze* sind die einzigen Plätze des MPNs, die in der Anfangsmarkierung mit Marken belegt sind, und werden durch einen Kreis mit einem Punkt in der Mitte dargestellt. Der *Endplatz* stellt einen positiven Abschluss des MPNs dar, während der *Fehlerplatz* ein negatives Beenden des MPNs repräsentiert. In ihrer konkreten Syntax werden sie wie ein Standardplatz ergänzt durch ein kleines `e` für einen Endplatz und ein kleines `f` für einen Fehlerplatz dargestellt. *Transitionen* der MPNs können mit einem Ereignis, Bedingung und Aktionen wie bei Zustandsautomaten annotiert werden. Transitionen, an denen kein Ereignis annotiert ist, werden als *Epsilon-Transitionen* bezeichnet, da sie auch ohne Auftreten eines Ereignisses schalten können.

Beispiel *Netzstruktur des MPNs*

Abbildung 2.9 zeigt die Netzstruktur eines MPNs, dass aus einem Initialplatz, einem Standardplatz und zwei Terminalplätzen (Endplatz und Fehlerplatz) besteht. Verbunden werden diese über insgesamt drei Transitionen. An der ersten Transition mit dem Initialplatz als Vorplatz ist ein Ereignis `event1` mit einem Parameter `x` des Typs `int` und eine Aktion `action1` annotiert. Die anderen

beiden Transitionen sind mit zwei Bedingungen annotiert, die sich in diesem Beispiel ausschließen, aber nicht müssen. Die aus diesen beiden Epsilon-Transitionen auslaufenden Kanten führen zum einen auf einen Endplatz für `[x>0]` und zum anderen auf einen Fehlerplatz für `[x<=0]`.

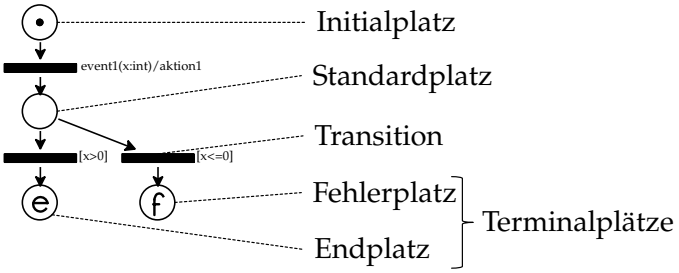


Abbildung 2.9: Elemente der Monitor-Petrinetze

Makroschrittsemantik: MPNs besitzen eine deterministische Semantik, die implizite Konzepte zur Auswertung enthält, und so eine kompaktere Modellierung ermöglicht. Tritt ein annotiertes Ereignis ein und die Bedingung einer Transition ist erfüllt, kann die Transition schalten, wenn ihr Vorbereich belegt ist. Das Schalten der MPNs ist über Makroschritte definiert, die alle Transitionen des MPNs in einem Schritt betrachten, und sich in vier Teilschritte einteilen lassen:

1. Erstelle eine Kopie der aktuellen Markierung, auf der die Entscheidung über das Schalten der Transitionen getroffen wird.

2. Lösche aus der aktuellen Markierung alle Marken im Vorbereich der aktiven Transitionen (Vorbereich in Kopie der Markierung belegt) und führe voneinander unabhängige Aktionen aus, die an den aktiven Transitionen annotiert sind.

3. Erstelle in der aktuellen Markierung Marken auf alle Plätzen des Nachbereichs der als aktiv ermittelten Transitionen und erstelle eine neue Kopie der Markierung.

4. Löse solange Epsilon-Transitionen aus (wie in Schritten 2 und 3 beschrieben), bis das MPN seine Markierung nicht mehr verändert.

Durch diese globale Betrachtung aller Transitionen eines MPNs, basierend auf einer Kopie der Markierung, wird das deterministische Verhalten der MPNs erreicht. Im Gegensatz zu den vorgestellten Petrinetzen darf in einem MPN auf einem Platz immer nur eine Marke liegen. Würde eine zweite Marke durch die Makroschrittsemantik erzeugt werden, wird diese mit der vorhandenen Marke zusammengefasst.

Beispiel *Keine Verzweigungskonflikte durch Makroschrittsemantik*

Abbildung 2.10 zeigt ein MPN, in dem Marken durch die Ziffer `1` repräsentiert werden. Die beiden Transitionen, die mit dem Initialplatz verbunden sind, besitzen dasselbe Ereignis `event1`, während es sich bei der dritten Transition um

eine Epsilon-Transition handelt, die sofort schaltet, wenn ihre Vorplätze belegt sind. Der Makroschritt beginnt mit der Übergabe des Ereignisses `event1` an das MPN. Im ersten Schritt des Makroschritts wird eine Kopie der aktuellen Markierung erstellt, die hier durch eine Kopie der Netzstruktur inklusive der Marken dargestellt ist. Zusätzlich werden alle Transitionen, die schalten können ermittelt – in diesem Beispiel die beiden Transition, die `event1` annotiert haben. Im zweiten Schritt werden die Marken in den Vorplätzen dieser beiden Transitionen gelöscht. Im dritten Schritt werden zunächst in den beiden Nachplätzen der Transitionen jeweils eine Marke erstellt und dann eine neue Kopie des Netzes angelegt. Auf dessen Basis wird im vierten iterativen Schritt die Epsilon-Transitionen geschaltet, deren Vorplätze belegt sind. Hierbei wird wie im

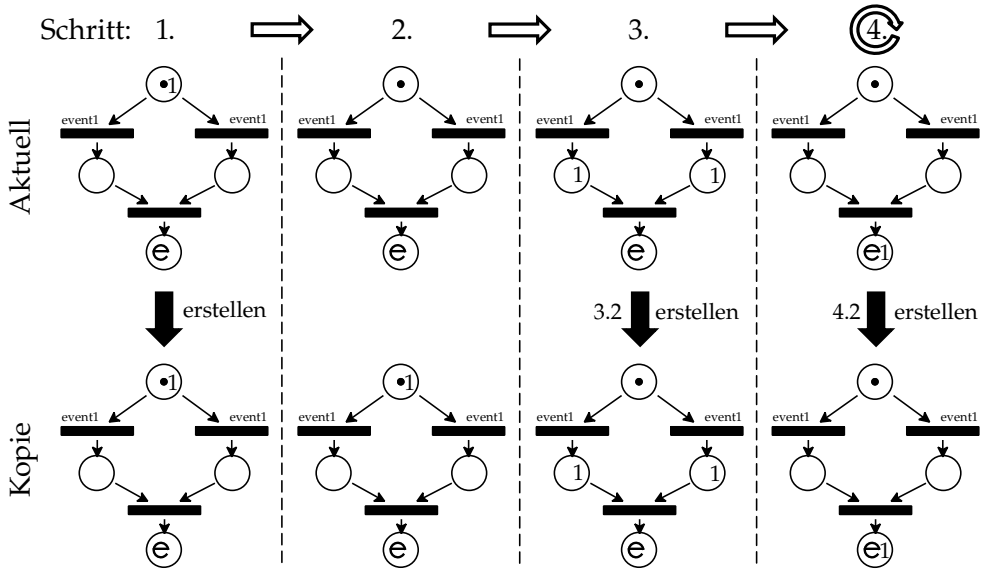


Abbildung 2.10: Makroschritte der MPNs

zweiten und dritten Schritt vorgegangen und es werden auf Basis der neuen Kopie wiederum die aktivierten Epsilon-Transitionen ermittelt. Da in diesem Beispiel keine Epsilon-Transition mehr schalten kann, wird der Makroschritt beendet und die Kopie verworfen. Dieses Beispiel zeigt, dass es sich um eine deterministische Semantik handelt, da immer alle aktiven Transitionen schalten und keine Konkurrenz um Marken besteht.

Eingabegenerationen: MPNs werden zur Spezifikation hoch nebenläufiger Prozesse als Monitorsignaturen eingesetzt. Hierzu unterstützen MPNs *Eingabegenerationen*, die einen eindeutigen Identifizierer eines zu überwachenden Objekts oder einer Menge von zu überwachenden Objekten darstellen – einen Kommunikationsstrang. Die Marken des MPNs werden diesen Eingabegenerationen zugeordnet (Färbung der Marken), wodurch eine Überwachung mehrerer paralleler Kommunikationsstränge in einer MPN-Instanz ermöglicht wird. Marken verschiedener Eingabegenerationen werden unabhängig voneinander behandelt, als ob das MPN mehrfach instanziiert wäre, und können hierdurch auch gleichzeitig auf einem Platz liegen. Wird dem MPN eine neue Eingabegeneration übergeben, zu

der noch keine Marken existieren, wird in jedem Initialplatz eine Marke mit dieser Eingabegeneration erzeugt.

Beispiel *Eingabegenerationen*

Abbildung 2.11 zeigt eine leicht abgeänderte Variante des MPNs aus Abbildung 2.10, dessen Epsilon-Transition mit einem weiteren Ereignis `event2` annotiert wurde, um so ein direktes Beenden der MPN-Überwachung in einem Makroschritt zu verhindern. In der Anfangsmarkierung ist das MPN mit einer

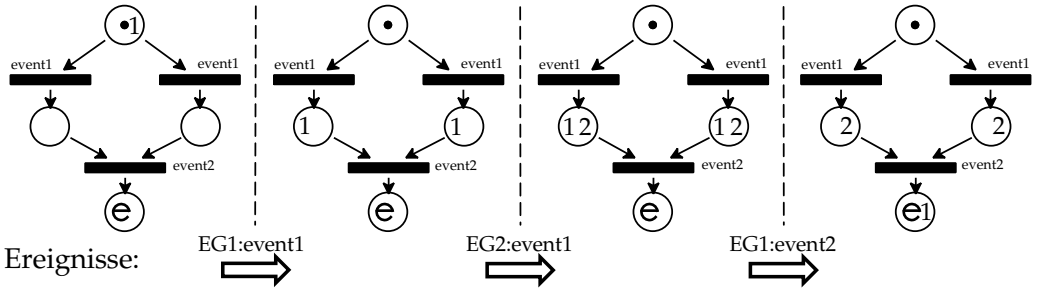


Abbildung 2.11: Eingabegenerationen der MPNs

Marke der Eingabegeneration `EG1` im Initialplatz belegt. Im ersten Makroschritt wird dem MPN das Ereignis `event1` übergeben, das an die Eingabegeneration `EG1` gebunden ist. Hierdurch schalten die beiden mit `event1` annotierten Transitionen und die Marken der `EG1` werden auf den Nachplätzen der Transitionen erzeugt. Als Zweites wird dem MPN wiederum das Ereignis `event1` übergeben, das jedoch in diesem Fall aus einem zweiten Kommunikationsstrang (`EG2`) stammt. Es wird somit eine Marke der Eingabegeneration `EG2` im Initialplatz erzeugt und die entsprechenden Transitionen schalten. Wird im nächsten Makroschritt das Ereignis `event2` der `EG1` übergeben, wird das MPN für den Kommunikationsstrang `EG1` beendet und für `EG2` weiter überwacht.

Subgenerationen: Die Überwachung einer als MPN modellierten Signatur beginnt mit dem ersten modellierten Ereignis, das dem MPN übergeben wird. Da jedoch Ereignisse in einer Signatur auch mehrfach vorkommen können, ist nicht sichergestellt, dass das später auftretende Ereignis mit der gleichen Signatur nicht der Anfang der eigentlich gesuchten Ereignisabfolge ist. Somit muss das MPN für jedes Auftreten des ersten Ereignisses in der Signatur von Neuem überwacht werden. Diese *verschränkte Überwachung* wird in MPNs durch *Subgenerationen* der Eingabegenerationen realisiert, die in den Initialplätzen neu erzeugt werden, sobald das erste Ereignis eingetreten ist.

Neben diesen Signaturen, die explizit einen endlichen Ausschnitt eines Kommunikationsstrangs beschreiben, existiert ein weiterer Fall, in dem es sinnvoll ist, Signaturen verschränkt (concurrent) zu überwachen. Ein solcher Fall ist die durchgängige Überwachung eines Wertes eines wiederkehrenden Ereignisses. Beim ersten Auftreten des Wertes wird dieser zwischengespeichert und beim nächsten Auftreten erst mit dem neu übermittelten Wert verglichen, um dann für den nächsten Vergleich zwischengespeichert zu werden. Zur Abbildung einer solchen

Signatur wird in der MPN-Sprache ebenfalls das Konzept der *Subgenerationen* eingesetzt.

Beispiel *Überwachung eines wiederkehrenden Ereignisses*

Abbildung 2.12 zeigt eine Überwachungssignatur, die überprüft, das die Quittungsnummern streng monoton ansteigen. Hierzu wird das Konzept der Subgenerationen (SG) verwendet und die Belegung eines Platzes mit einer Marke im Format `<EG>.<SG>` unten rechts neben dem entsprechenden Platz annotiert. In der Anfangsmarkierung liegt eine Marke `1.1` im Initialplatz. Sobald das Ereignis `ReceiptRequest(1)` der `EG1` an das MPN übergeben wird, schaltet nur die erste Transition, da nur ihre Vorplätze belegt sind, und führt die Aktion `lastRN=n` aus. Hierdurch wird die aktuell übergebene Quittungsnummer `n` auf der Variablen `lastRN` gespeichert. Da am Ende dieses Makroschritts keine Marke der höchsten Subgeneration mehr im Initialplatz liegt, wird eine neue Subgeneration `1.2` erstellt. Wird im nächsten Schritt wiederum ein Ereignis `ReceiptRequest(2)` der `EG1` übergeben, werden die Subgenerationen aufsteigend abgearbeitet. Zunächst schaltet die auf den Endplatz führen-

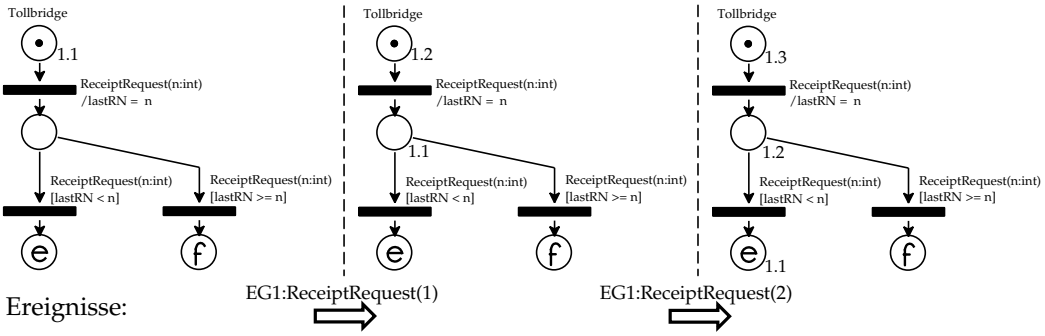


Abbildung 2.12: Subgenerationen der MPNs

de Transition für `1.1`, da die Bedingung `[lastRN<n]` erfüllt ist und die Marke `1.1` erreicht den Endplatz. Als Zweites wird ein Makroschritt für die neue Subgeneration `1.2` durchgeführt und wie für 1.1 die Variable `lastRN` mit der neuen Quittungsnummer `n` überspeichert. Nun wird eine Marke mit der Subgeneration `1.3` im Initialplatz erzeugt. Diese verschränkte Überwachung des `ReceiptRequest`-Ereignisses und dessen Parameter wird fortgesetzt, bis die Bedingung `lastRN>=n` zutrifft und eine Marke auf dem Fehlerplatz erzeugt wird.

Ohne dieses gesonderte Konzept der MPNs müsste diese automatische Instanziierung der MPNs und die Verwaltung der Abarbeitungsreihenfolge von außerhalb des Monitors über Eingabegenerationen erfolgen.

Implizite Konzepte der MPNs: Zur kompakteren Modellierung von Signaturen und einer übersichtlicheren Darstellung unterstützen MPNs zusätzliche Konzepte zur Auswertung des Zustandes einer Signatur. Bei der Konstruktion eines MPNs müssen nicht explizit alle Plätze beim positiven oder negativen Beenden der Signatur abgeräumt werden. Die MPN-Semantik löscht automatisch alle Marken der entsprechenden Eingabe- oder Subgeneration aus dem MPN, wenn dieses durch das Erreichen eines Teminalplatzes beendet wird.

Zusätzlich beendet sich die Überwachung einer Eingabe- oder Subgeneration in einem MPN automatisch, sobald das MPN ein Ereignis übergeben bekommt, das zu keinem Schaltvorgang führt. Durch den Verzicht auf eine explizite Modellierung können die MPN-Signaturen viel kompakter modelliert und aus ihnen speichereffizientere Monitore generiert werden. Um die Auswertung der beendeten MPNs trotz teilweisem Verzicht auf explizite End- und Fehlerplätze durchführen zu können, ist eine Einteilung der MPNs in die *Vorbedingungssignatur* (*engl. antecedent signature*) und *Hauptsignatur* (*engl. consequent siganture*) notwendig. Beendet sich das MPN in der Vorbedingungssignatur eines MPNs implizit, entspricht dies dem Erreichen eines Endplatzes. Beendet sich das MPN jedoch implizit in der Hauptsignatur, entspricht dies dem Erreichen eines Fehlerplatzes.

Positive und negative MPNs: Signaturen beschreiben sowohl positives, erlaubtes Systemverhalten als auch negatives, verbotenes Systemverhalten. Um auch verbotenes Verhalten explizit modellieren zu können, werden MPNs in *positive MPNs* (eine Art Use-Case-Beschreibung) und *negative MPNs* (eine Art Misuse-Case-Beschreibung) eingeteilt. Negative MPNs sind hierdurch intuitiv entgegengesetzt zu positiven MPNs modellierbar, indem diese auf Fehlerplätzen enden und im Gegensatz zu Abbildung 2.9 nicht erfüllte Bedingungen auf Endplätze statt Fehlerplätze führen. Durch die Einordnung als negatives MPN führt ein implizites Beenden der Überwachung des MPNs nicht wie bei positiven MPNs zu einer Fehlererkennung, sondern zu einem einfachen Abbruch der Überwachung.

Referenzsystem der MPNs: Ein übergeordnetes Konzept zur Modellierung von Beziehungen zwischen MPNs stellt das Referenzsystem der MPNs zur Verfügung. Erreicht das referenzierende MPN einen bestimmten Zustand codiert durch seine Markierung, wird die Überwachung eines referenzierten MPNs aktiviert. Dies ermöglicht die Auslagerung von Teilsignaturen, die als MPN modelliert sind, und die parallele Überwachung in bestimmten Zuständen des Systems.

Modelliert werden diese Referenzen durch *Erweiterungspunkte* (EP) der MPNs, die zum referenzierenden MPN gehören. Die referenzierten MPNs (refMPNs) werden dem EP zugeordnet und basierend auf der Belegung von Aktivierungsplätzen (S_{act}) wird die parallele Überwachung der refMPNs gestartet. Zusätzlich definiert jeder Erweiterungspunkt eine boolesche Variable, die zur Blockierung von Epsilon-Transitionen im referenzierenden MPN genutzt werden kann. Wenn Teilsignaturen ausgelagert werden, werden durch Setzen dieser Variablen auf `false` bestimmte Transitionen des MPNs solange blockiert, bis eines der ausgelagerten MPNs erfolgreich erkannt wurde.

Soll das referenzierende MPN während der Ausführung oder Überwachung der referenzierten MPNs vollständig blockiert werden, kann die Eigenschaft `concurrent` des EPs auf `false` gesetzt werden. Zusätzlich zur Auswertung der Schaltergebnisse der einzelnen MPNs müssen die Zusammenhänge über die EPs beachtet werden. Hierzu werden den Erweiterungspunkten alle grundlegenden Signaturen (dependentMPNs) zugeordnet, die direkt oder indirekt von dem Schalten der referenzierten MPNs bei der Auswertung abhängen.

Dieses Konzept wird in Abschnitt 8.2 zur Abbildung der Beziehungen zwischen Use-Cases und Misuse-Cases der MBSecMonSL in die MPN-Sprache eingesetzt. In diesem Abschnitt wird genauer auf die Einsetzbarkeit und Semantik des

Erweiterungspunktes eingegangen. Tiefergehende Erläuterungen zur Syntax und Semantik der MPN-Sprache können in [Pat14] nachgelesen werden.

Teil III

MBSECMON-SPEZIFIKATIONSSPRACHE

3

GRUNDLAGEN UND VERWANDTE ARBEITEN

Im ersten Schritt des MBSecMon-Prozesses (*Monitorsignaturspezifikation*) müssen, wie in Abbildung 3.1 dargestellt, Signaturen auf Basis von vorhandenen Entwicklungsartefakten spezifiziert werden. Hierzu wird im fertigen MBSecMon-Prozess die in dieser Arbeit entwickelte verhaltensbeschreibende Signatursprache verwendet – die MBSecMon-Spezifikationssprache (MBSecMonSL). Die MBSecMonSL

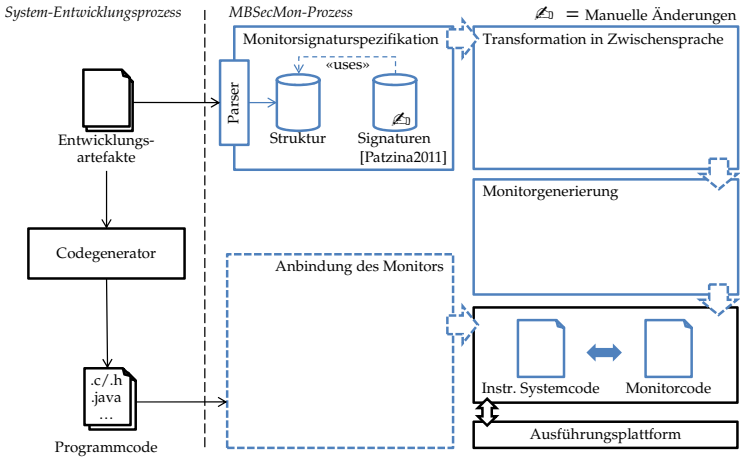


Abbildung 3.1: MBSecMon-Prozess: Die Monitorspezifikationssprache

besteht zum einen aus der MUC-Sprache, die zur Strukturierung der Spezifikationen eingesetzt wird, und zum andern aus einer erweiterten Variante der LSC-Sprache, die die Kommunikationssequenzen als Szenarien beschreibt.

Dieses Kapitel stellt die Grundlagen für diese beiden Teilsprachen des MBSecMon-Prozesses vor und zeigt anhand existierender Formalismen, warum die Misuse-Case-Modellierung und die Spezifikation der Sequenzen in einer Variante der LSCs vielversprechend sind. Hierzu werden in Abschnitt 3.1 die Grundlagen der Anwendungsfallmodellierung und Erweiterungen um negative Anwendungsfälle betrachtet, auf der die MUC-Sprache basiert. In Abschnitt 3.2 werden verschiedene Varianten der Sequenzdiagramme, die Message Sequence Charts (MSCs) und die UML2-Sequenzdiagramme (UML2-SD), ihre Nachteile für den Einsatz als verhaltensbeschreibende Signaturspezifikationssprache aufgezeigt. In Abschnitt 3.3 werden abschließend existierende Formalismen zur Spezifikation von Signaturen und Strukturierungsmechanismen für Sequenzdiagramme vorgestellt und auf ihre Eignung als verhaltensbeschreibende Signaturspezifikationssprache für den MBSecMon-Prozess untersucht.

3.1 Anwendungsfalldiagramme

Anforderungen werden in den frühen Entwicklungsphasen häufig durch szenariobasierte Beschreibungen verfasst. Hierzu dienen die sogenannten Anwendungsfallnotationen, die häufig auf einfachen textuellen Schablonen (*engl. templates*) [KG03, Coc01, Kru04] basieren. Eine grafische Notation der Use-Cases floss 1992 in den ersten UML-Standard als Anwendungsfalldiagramm (*engl. use case diagram*) ein. Use-Cases der UML eignen sich ausschließlich zur Beschreibung funktionaler Anforderungen und lässt nicht-funktionale Anforderungen, wie die Modellierung von Fehlbedienung oder Sicherheitslücken des Systems, außer Acht. Sindre und Opdahl [SO05] und Alexander et al. [Ale02][Ale03b][Ale03a] führten zur Erweiterung der Use-Cases um eine Beschreibungsmöglichkeit für nicht-funktionale Anforderungen Misuse-Cases ein.

3.1.1 *Use-Case-Diagramme*

Use-Cases beschreiben das funktionale Verhalten eines Systems, das für den Nutzer beobachtbar ist. Diese Idee das nach außen sichtbare Verhalten eines Systems zu beschreiben, entstand in den späten 70er und frühen 80er Jahren [MP84] unter dem Begriff *Essential Modelling*. Neben vielen verschiedenen formalen und informellen Notationen für die Definition von Anwendungsfällen setzten sich zunächst einfache textuelle Schablonen [KG03, Coc01, Kru04] durch, die durch verschiedene Diagrammvarianten ergänzt wurden. Der von Jacobson [JCJ92] präsentierte anwendungsfallgetriebene Ansatz für die objektorientierte Software-Entwicklung setzt zum ersten Mal Anwendungsfälle als Haupttechnik zur Systemanalyse ein und steigert dessen Popularität. Die Aufnahme der szenariobasierten Beschreibung durch Anwendungsfälle (*engl. use cases*) in die erste Version der UML mit den Anwendungsfalldiagrammen, führte zu einer ersten Standardisierung der Anwendungsfallbeschreibungen. Diese Anwendungsfalldiagramme [RQJZ07] änderten sich durch die Evolution des UML-Standards zur Version 2 nur noch geringfügig. Es wurde explizit im UML2-Standard herausgearbeitet, dass beliebige *Classifier* der strukturbeschreibenden Diagrammtypen als Akteure einsetzbar sind – eine Freiheit, die vorher kaum genutzt wurde. Da der Fokus der Diagramme auf dem nach außen sichtbaren Verhalten und nicht auf dem Auslöser des Verhaltens liegt, wurden die Anwendungsfalldiagramme den verhaltensbeschreibenden Diagrammen der UML2 zugeordnet.

UML2-Anwendungsfalldiagramme bestehen aus *Systemgrenzen*, *Anwendungsfällen*, *Akteuren* außerhalb der Systemgrenze oder in einem anderen System und *Beziehungen* zwischen Akteuren, zwischen Anwendungsfällen und untereinander.

Beispiel *Das CARDME-Protokoll als Use-Case-Diagramm*

Abbildung 3.2 zeigt ein Use-Case-Diagramm der UML2 für das CARDME-Protokoll. Dieses stellt die erlaubten Aktivitäten des Protokolls und die Interaktion der Akteure mit diesen Aktivitäten dar.

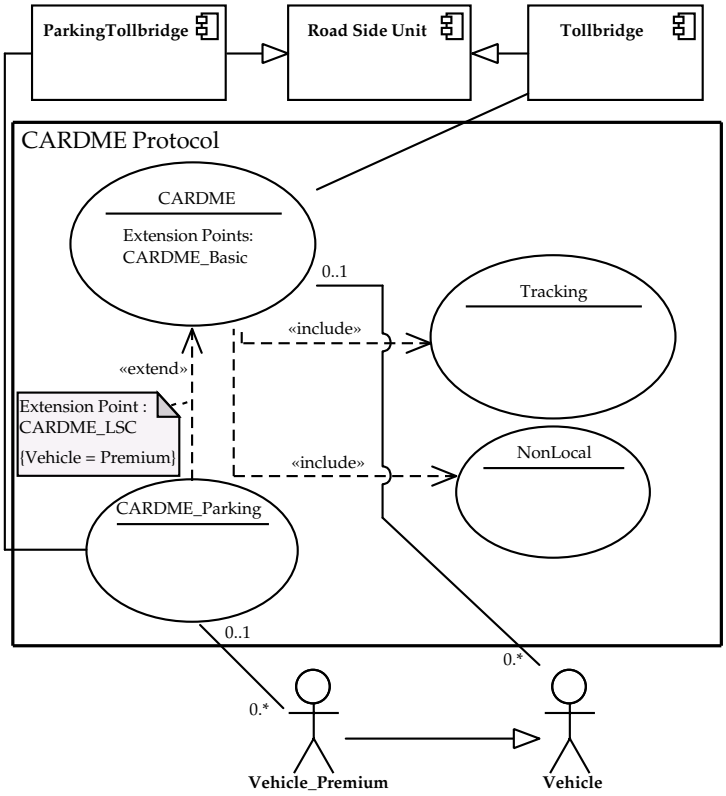


Abbildung 3.2: CARDME-Protokoll als UML2-Use-Case-Diagramm

Systemgrenze: Die Systemgrenze wird als Rechteck dargestellt, das mit dem Namen des Systems innerhalb des Rechtecks annotiert wird. Sie kann in der UML durch einen beliebigen *Classifier* (z. B. Klasse, Schnittstelle, Komponente oder Subsystem) substituiert werden, der dann die Funktionalität des Classifiers spezifiziert.

Beispiel *Systemgrenze*

Die Systemgrenze des Use-Case-Diagramms in Abbildung 3.2 repräsentiert das CARDME-Protokoll. Sie ist als Rechteck dargestellt und mit dem Namen `CARDME Protocol` beschriftet.

Anwendungsfälle (*engl. use cases*): Der *Anwendungsfall* wird durch eine Ellipse dargestellt, deren Name entweder innerhalb oder außerhalb der Ellipse annotiert wird. Dieser Name spiegelt die Aktion bei der Ausführung des Anwendungsfalls aus Sicht des Benutzers (Aktor) wieder. Ein Anwendungsfall besteht aus einer Menge von Aktionen, die in einer bestimmten Reihenfolge abgearbeitet werden müssen, um das entsprechende Verhalten abzubilden. Hierbei können neben dem Normalverhalten auch Spezialfälle und Fehlerfallaktionen in einem Anwendungsfall zusammengefasst werden.

Beispiel *Anwendungsfälle*

Das Use-Case-Diagramm enthält vier Use-Cases, die sich innerhalb der Systemgrenze befinden – `CARDME`, `CARDME_Parking`, `Tracking` und `NonLocal`. Hier-

bei repräsentiert `CARDME` die nach außen sichtbare, grundlegende Kommunikation zwischen Mautbrücke und Fahrzeug, `Tracking` die Kommunikation einer Nachverfolgung des Fahrzeugs durch die Mautbrücke und `NonLocal` eine zusätzliche Kommunikationssequenz zur Identifikation fremder Fahrzeugen. `CARDME_Parking` beschreibt eine spezielle Variante der grundlegenden Kommunikation, die in Parkhäusern eingesetzt werden kann.

Akteure (*engl. actors*): Der *Akteur* wird entweder als Strichmännchen oder, da alle Arten von *Classifiern* der UML2 als Akteur erlaubt sind, in der entsprechenden UML2-Notation für Klassen dargestellt. In Anwendungsfalldiagrammen gehört der Akteur nicht zum System selbst, sondern löst eine Aktion des Systems aus. Somit werden Akteure außerhalb der Systemgrenzen platziert und über *Kommunikationsbeziehungen* mit den für den Akteur zugänglichen Anwendungsfällen verbunden. Diese Anwendungsfälle können dann durch den Akteur ausgelöst werden und er erhält ein Ergebnis nach Ausführung der spezifizierten Interaktionsfolge.

Beispiel *Akteure*

Die Akteure des CARDME-Protokolls sind zum einen die beiden Fahrzeuge (`Vehicle`, `Vehicle_Premium`) und zum anderen zwei Varianten der Mautbrücke (`Tollbridge`, `ParkingTollbridge`). Während die Fahrzeuge als Strichmännchen dargestellt sind, werden die Mautbrücken als UML2-Komponenten der Systembeschreibung repräsentiert. Durch Kommunikationsbeziehungen sind die Akteure mit den Use-Cases verbunden, in denen sie direkt teilnehmen.

Kommunikationsbeziehungen (*engl. communication relationship*): Die *Kommunikationsbeziehungen* zwischen Akteuren und Anwendungsfällen sind die aus den Klassendiagrammen bekannten *Assoziationen*, die weiter verfeinert werden können. Es stehen sowohl binäre als auch gerichtete Assoziationen zur Verfügung, um den Ein- und Ausgabefluss darzustellen. Zusätzlich lassen sich u. a. Multiplizitäten festlegen, die angeben, wie oft ein Akteur einen Anwendungsfall ausführen bzw. initialisieren darf und wie viele Akteure den Anwendungsfall auf einmal auslösen dürfen.

Beispiel *Kommunikationsbeziehungen*

Die beiden Mautbrücken-Akteure im Use-Case-Diagramm des CARDME-Protokolls sind jeweils durch eine ungerichtete Assoziation zu dem Use-Case, an dem sie teilnehmen, dargestellt. Die Assoziationen sind ungerichtet, da eine beidseitige Kommunikation stattfindet. Die Mautbrücke initiiert die Kommunikation und bekommt vom ebenfalls am Use-Case beteiligten Fahrzeug Informationen zurück. Die Assoziationen zwischen den Fahrzeugen und den Use-Cases sind zusätzlich noch mit Multiplizitäten versehen. So darf ein Fahrzeug immer nur `0..1` Kommunikationen mit der entsprechenden Mautbrücke zur gleichen Zeit aufbauen. Im Gegensatz hierzu dürfen `0..*` viele Fahrzeuge auf einmal den Use-Case instanziieren.

Generalisierungsbeziehung (*engl. generalisation relationship*): Auch in den Anwendungsfalldiagrammen steht die Vererbung zwischen Akteuren und zwischen Anwendungsfällen zur Verfügung. Hierzu wird die aus den Klassendiagrammen bekannte Generalisierungsbeziehung, eine durchgezogene Kante mit einer nicht gefüllten geschlossenen Spitze, eingesetzt. So können zwischen Akteuren die Kommunikationsbeziehungen vererbt werden und bei den Anwendungsfällen ein allgemeingültiges Verhalten in einen Basisanwendungsfall, von dem andere Anwendungsfälle erben, ausgelagert werden.

Beispiel *Generalisierung*

Die Generalisierungsbeziehung wird in diesem Use-Case-Diagramm zwischen dem einfachen Fahrzeug (`Vehicle`), das nur die Kommunikation mit Mautbrücken (`Trollbridge`) unterstützt, und der Premium-Variante des Fahrzeugs (`Vehicle_Premium`), die auch die Bezahlung im Parkhaus abwickeln kann, eingesetzt. Hierdurch erbt das Premiumfahrzeug die Kommunikationsbeziehung des einfachen Fahrzeugs.

Beziehungen zwischen Anwendungsfällen können durch *Abhängigkeitsbeziehungen* (*engl. dependency*), dargestellt als gestrichelter Pfeil, modelliert werden. Der Pfeil zeigt immer auf das Modellelement, von dem das Element am anderen Ende abhängt. Die Abhängigkeitsbeziehung wird durch die Annotation eines Stereotypen genauer spezifiziert. In den Anwendungsfalldiagrammen sind dies «include» und «extend».

«include»-Beziehung: Die «include»-Beziehung wird mit dem Schlüsselwort «include» annotiert. Sie visualisiert den Import des Verhaltens des Anwendungsfalls an der Pfeilspitze in den Anwendungsfall am anderen Ende. Hierdurch wird die zentrale Beschreibung von gemeinsamen Verhalten in separaten Anwendungsfall ermöglicht. Durch mehrfache Inklusion kann dieses Verhalten in mehreren Anwendungsfällen wiederverwendet werden. Die «include»-Beziehungen dürfen hierbei jedoch keinen Zyklus bilden.

Beispiel *«include»-Beziehung*

Die «include»-Beziehung wird im Use-Case-Diagramm des CARDME-Protokolls zwischen dem Use-Case `CARDME` und den ausgelagerten Verhaltensbeschreibungen `Tracking` und `NonLocal` eingesetzt. Hierdurch wird die Möglichkeit gegeben, bei der Erweiterung des Use-Case-Diagramms, diese Verhaltensbeschreibungen wiederzuverwenden.

«extend»-Beziehung: Die «extend»-Beziehung wird mit dem Schlüsselwort «extend» annotiert. Sie erweitert die Interaktionsfolge des Anwendungsfalls an der Spitze der Beziehung mit der im Anwendungsfall am Ende der Beziehung angegebenen Interaktionsfolge an der durch den Erweiterungspunkt (*engl. extension point*) angegebenen Stelle. Dieser Erweiterungspunkt wird der Beziehung über ein Notizelement zugeordnet und ist abhängig vom Eintreten der im Notizelement notierten Bedingung. Wie auch die «include»-Beziehung dürfen «extend»-Beziehungen keinen Zyklus bilden.

Beispiel *«extend»-Beziehung*

Die «extend»-Beziehung zwischen den Use-Cases `CARDME` und `CARDME_Parking` beschreibt, dass bei Erreichen des Erweiterungspunktes `CARDME_Basic` das Verhalten in `CARDME_Parking` eine Alternative zu dem in `CARDME` beschriebenen Verhalten darstellt. Das Notizelement ordnet unter der Bedingung, dass das Fahrzeug ein Premiumfahrzeug ist (`{Vehicle = Premium}`), den Erweiterungspunkt `CARDME_Basic` der Beziehung zu.

Szenarien sind zur Spezifikation der Interaktionen von Aktorensystemen geeignet, indem Protokolle der Interaktionen zwischen den einzelnen Aktoren als Anwendungsfälle modelliert werden. Diese Szenarien beschreiben so erlaubte Sequenzen des Nachrichtenflusses und die Kollaboration zwischen den strukturellen Elementen. Hierbei wird zwischen Normal- und Ausnahmeverhalten unterschieden.

Beschreibungen der Anwendungsfallszenarien: Zur Modellierung von einzelnen Interaktionsfolgen der Szenarien bieten die UML neben strukturiertem Text, der sich nur für kurze, klare Abläufe mit wenigen Sonderfällen eignet, UML2-Interaktionsdiagramme an. Tabelle 3.1 zeigt eine Empfehlung, welche Abläufe mit welcher Sprache bzw. Interaktionsdiagrammtypen modelliert werden sollten.

Merkmale des Ablaufs	Empfohlene Notation
kurze klare Abläufe, wenige Sonderfälle	(strukturierter) Text
ablauf- oder schrittorientiert (1.,2.,...)	Aktivitätsdiagramme
einfache daten- oder entitätsorientierte Abläufe (viele Elemente)	Kommunikationsdiagramme
komplexe daten- oder entitätsorientierte Abläufe (viele Elemente)	Sequenzdiagramme
kein typischer Ablauf existiert, gleichwahrscheinliches Auftreten von Abfolgen und Ereignissen	Zustandsautomaten
Use-Case bündelt viele Szenarien	Interaktionsübersichtsdiagramme

Tabelle 3.1: Empfohlene Notationen zur Use-Case-Beschreibung nach [RQJZ07]

3.1.2 *Misuse-Case-Diagramme*

Bei der Modellierung funktionaler Anforderungen als Use-Cases wird angenommen, dass Akteure, die mit dem System interagieren, keine „schlechten“ Absichten besitzen. Da diese Annahme bei der Spezifikation der Szenarien eines Systems zu einer Missachtung möglicher Fehlbedienungsmöglichkeiten und Sicherheitslücken führt, wurden verschiedene aufeinander aufbauende Erweiterungen der Use-Cases vorgeschlagen. Sindre und Opdahl präsentieren erstmals in [SO00] als Erweiterung der UML-Anwendungsfalldiagramme bösartige Akteure (*engl. mal-actors*) und zur Repräsentation von nicht gewolltem bzw. verbotenem Verhalten Misuse-Cases. Diese Misuse-Cases ergänzen die vorhandenen Abhängigkeitsbeziehungen der UML-Anwendungsfalldiagramme durch die Beziehungen «prevents» und «detects». Der Use-Case an der Quellseite der «prevents»-Beziehung verhindert die Ausführung des Misuse-Cases an der Zielseite der Beziehung. Die

«detects»-Beziehung beschreibt, dass der Use-Case an der Quellseite der Beziehung die Aktivierung des Misuse-Cases an der Zielseite erkennt.

Dieses Konzept der Misuse-Cases wird durch Alexander [Ale02, Ale03b] zur Steigerung der Ausdrucksstärke verfeinert und die Beziehungen «prevents» und «detects» werden durch «threatens», «mitigates», «aggravates» und «conflicts with» ersetzt.

Beispiel *Das CARDME-Protokoll als Misuse-Case-Diagramm*

Das Use-Case-Diagramm des CARDME-Protokolls in Abbildung 3.2 beschreibt ausschließlich das erlaubte Verhalten des Systems. Mithilfe der Konzepte der Misuse-Cases wird diese Beschreibung in Abbildung 3.3 um zwei Angriffe auf das System als Misuse-Cases und eine Gegenmaßnahme als Use-Case erweitert. Die Misuse-Cases `Init_DoS_Attack` und `Tracking_DoS_Attack` sind grau hinterlegt und beschreiben jeweils einen Denail-of-Service-Angriff (DoS-Angriff) auf die Mautbrücke in verschiedenen Phasen des CARDME-Protokolls. Diese Angriffe können durch den bösartigen Akteur `Vehicle_Attack` ausgelöst werden, der ebenfalls grau eingefärbt ist. Die Gegenmaßnahme `CloseConnection` erlaubt der Mautbrücke Nachrichten zu ignorieren und somit den Angriff auf ihre Verfügbarkeit zu vereiteln.

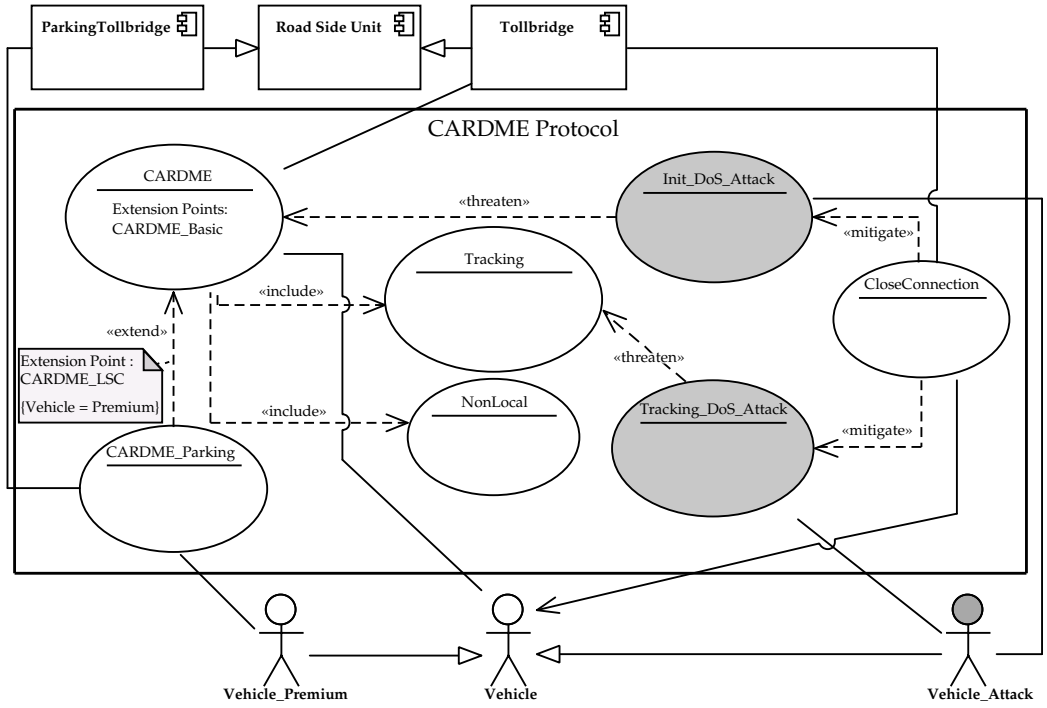


Abbildung 3.3: CARDME-Protokoll als Misuse-Case-Diagramm

«threatens»-Beziehung: Die «threatens»-Beziehung ersetzt die von Sindre und Opdahl generisch eingesetzte «extends»-Beziehung zwischen Misuse-Case und Use-Case, der bedroht wird. Hierdurch wird die Bedrohung für den Use-Case eindeutiger im Anwendungsfalldiagramm modellierbar.

Beispiel *«threatens»-Beziehung*

Im Misuse-Case-Diagramm des CARDME-Protokolls findet die Zuordnung der Misuse-Cases `Init_DoS_Attack` und `Tracking_DoS_Attack` zu den durch sie bedrohten Use-Cases durch die «threaten»-Beziehungen statt.

«mitigates»-Beziehung: Die «mitigates»-Beziehung ist eine abgeschwächte Variante der «prevents»-Beziehung, die auch einen Teilerfolg bei der Verhinderung eines Misuse-Cases erlaubt.

Beispiel *«mitigates»-Beziehung*

Die Zuordnung der Gegenmaßnahme `CloseConnection` zu den Misuse-Cases findet im Misuse-Case-Diagramm des CARDME-Protokolls über die beiden `mitigate`-Beziehung statt. Tritt eine der beiden DoS-Angriffe auf die Mautbrücke ein, vermindert bzw. verhindert die im Use-Case `CloseConnection` hinterlegte Gegenmaßnahme die Auswirkungen des Angriffs.

«aggrevates»-Beziehung: Die «aggrevates»-Beziehung ist das funktionale Gegenteil zu der «mitigates»-Beziehung. Der Anwendungsfall an der Quellseite der Beziehung verschärft die Auswirkungen des Misuse-Cases an der Zielseite.

«conflicts with»-Beziehung: Die «conflicts with»-Beziehung ist eine binäre gerichtete Beziehung und beschreibt den gegenseitigen Ausschluss der Ausführung der an der Beziehung beteiligten Use-Cases.

McDermott und Fox [MF99] führen den Begriff *Abuse-Cases* (*engl. abuse cases*) als Konzept zur Herleitung von Sicherheitsanforderungen ein. Diese Abuse-Cases beschreiben ausschließlich vollständige Interaktionen zwischen Akteuren und Use-Cases, die eine schädliche Auswirkung auf das System oder einen der Akteure haben. Diese Abuse-Cases können als Synonym für Misuse-Cases gesehen werden [Ale02].

Wie Alexander [Ale03a] zeigt, erweitern Misuse-Cases die UML2-Anwendungsfalldiagramme um die Möglichkeit nicht-funktionale Anforderungen wie Zuverlässigkeit, Safety und Security zu spezifizieren. Des Weiteren werden in vielen Arbeiten [PX06, PX05, SO05, WWH08] Misuse-Cases angewendet um Sicherheitsanforderungen an ein System zu spezifizieren. Zur Spezifikation der Interaktionsfolgen werden häufig Sequenzdiagramme verwendet, die im Folgenden genauer betrachtet werden.

3.2 Sequenzdiagramme

Zur Beschreibung komplexer Szenarien, die die Interaktion zwischen vielen Objekten oder Prozessen in Use-Cases beschreiben, dienen häufig Sequenzdiagramme als konkrete Syntax [RQJZ07]. Die verschiedenen Ausprägungen dieser Sequenzdiagramme unterscheiden sich jedoch in ihrer Ausdrucksstärke. Der in der Telekommunikationsindustrie sehr populäre Standard der International Telecommunication Union (ITU), die MSCs [IT92], bilden die Grundlage für viele Erweiterungen. Standard-MSCs basieren auf einer einfachen semantischen Notation der partiellen Ordnung von Ereignissen und sind somit in ihrer Ausdrucksstärke

schwach [HM08]. Auch die meisten Erweiterungen der MSCs, wie die Erweiterung mit regulären Ausdrücken [RGS12], ändern an der Semantik der partiellen Ordnung nichts. Dies trifft auch auf die Sequenzdiagramme des De-Facto-Standards UML2 zu, die viele Konzepte aus den MSCs übernommen und zusätzliche Konstrukte zur Strukturierung des Kontrollflusses eingeführt haben. Eine Erweiterung der MSCs mit dem Namen Live Sequence Charts (LSCs), die 1998 von Damm und Harel eingeführt wurde, erweitert die Semantik der MSCs um die Unterscheidung von möglichem und notwendigem Verhalten auf Ebene ganzer Diagramme und auf Elementebene in den Diagrammen der LSCs. Hierdurch wird die Modellierung von multimodalen Eigenschaften wie Lebendigkeit und Safety, sowie die Beschreibung notwendiger, möglicher und verbotener Szenarien ermöglicht. Im Folgenden werden die soeben angesprochenen Sequenzdiagrammausprägungen genauer betrachtet und ihre Eigenschaften basierend auf dem Mautbrückenbeispiel aus Abschnitt 1.5 vorgestellt. Hierbei wird nur auf eine Untermenge aller Konzepte und Modellierungselemente der Sprachen eingegangen, die für die Modellierung in einer verhaltensbeschreibenden Signatursprache relevant sind.

3.2.1 *Message Sequence Charts*

Message Sequence Charts (MSCs) [GRS01] sind eine grafische Sprache zur Beschreibung von Szenarien, die basierend auf Prozessalgebra formalisiert wurden und weit verbreitet sind. Vor 1992 wurden verschiedene Dialekte von Sequenzdiagrammen in der Telekommunikationsindustrie zur Beschreibung von Interaktionen zwischen Prozessen verwendet. Die erste Version der durch die International Telecommunication Union (ITU) standardisierten MSCs wurde 1993 in der „Recommendation Z.120“ [IT92] als MSC-92 standardisiert. Diese einfachen Sequenzdiagramme unterstützten die Modellierung von Prozessen als Instanzen und Nachrichten, die zwischen diesen Instanzen ausgetauscht werden. Nachrichten können mit Ereignissen annotiert und Bedingungen und Aktionen als Elemente mit informellen Beschriftungen modelliert werden. High-Level-Konstrukte, die den Kontrollfluss beschreiben, waren nicht vorgesehen und durch die auf Lebenslinien fokussierte textuelle Syntax des MSC-92-Standards nicht definierbar [LRS10]. Somit ließen sich mit diesen grundlegenden Modellierungselementen nur einfache MSCs beschreiben.

Beispiel *Grundlegende Elemente der MSCs*

Angewendet auf das Szenario in Abschnitt 1.5, ergibt sich folgende Modellierung des CARDME-Protokolls als Basic Message Sequence Chart (BMSC) für die Kommunikation zwischen den Instanzen Mautbrücke (`Tollbridge`) und Fahrzeug (`Vehicle`). In Abbildung 3.4 wird der grundlegende Nachrichtenaustausch zwischen den Kommunikationspartnern `Tollbridge` und `Vehicle` modelliert. Die ersten beiden Kommunikationsphasen (Initialisierungs- und Präsentationsphase), die mit `(1)` markiert sind, lassen sich schon im MSC-92-Standard beschreiben. Die beiden Kommunikationspartner werden als *Instanzen* repräsentiert, die aus einem beschrifteten Rechteck mit einer vertikalen Li-

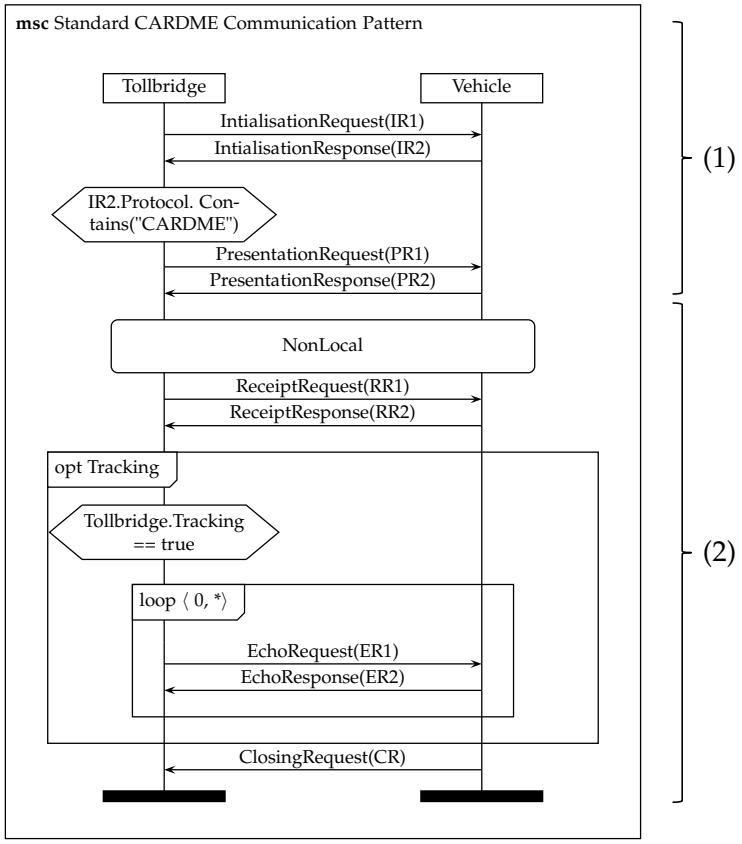


Abbildung 3.4: CARDME-Protokoll als Message Sequence Chart

nie (*Instanzlinie*) über die Zeit, die von oben nach unten voranschreitet, dargestellt werden. Die Mautbrücke sendet eine asynchrone *Nachricht* zur Initialisierung einer Verbindung mit dem On-Board Equipment (OBE) des Fahrzeugs aus (`InitialisationRequest`) und erwartet eine Antwort vom Fahrzeug (`InitalisationResponse`). Hierbei werden Daten ausgetauscht, die hier abstrakt als Parameter `IR1` und `IR2` der Nachricht angegeben sind. Diese Parameter lassen sich im MSC-92-Standard zwar nur informell beschreiben, die späteren Revisionen des MSC-Standards erlauben jedoch die Definition von Variablen in einer durch den Modellierer frei wählbaren Datensprache. Antwortet das Fahrzeug, überprüft die Mautbrücke, ob in der Antwort des Fahrzeugs als mögliches Protokoll, das die OBE unterstützt, das CARDME-Protokoll enthalten ist. Dies wird durch eine *Globale Bedingung* modelliert, die als Sechseck, über eine oder mehrere vertikale Zeitachsen der Instanzen geht. Nachdem diese Initialisierungsphase abgeschlossen ist, findet der Datenaustausch zwischen der Mautbrücke und dem Fahrzeug ebenfalls durch asynchrone Nachrichten statt.

Durch die fehlende Konstrukte zur Beschreibung des Kontrollflusses zwischen und in MSCs, lassen sich mit dem MSC-92-Standard nur lineare Abläufe beschreiben. Aus diesem Grund wurden MSCs nur in frühen Phasen des Systementwurfs zur Formulierung von einzelnen Szenarien (Beispielabläufen von Protokollen) verwendet. Zur Beschreibung eines Systems werden jedoch immer eine Menge von Szenarien, die eine Ordnungsrelation haben, benötigt. Hierzu wurde außer-

halb des Standards die Sprache der Message Sequence Graphs (MSGs) [MP99] eingeführt, um den Kontrollfluss zwischen den MSCs zu modellieren. Sie erlaubt die Beschreibung von Alternativen, sequenzielle Komposition und die Iteration über MSCs, indem sie MSCs in die Knoten eines Graphen einbettet und zusätzlich Entscheidungsknoten einführt.

Erst durch die Erweiterung des MSC-92-Standards 1998 wurde im MSC-96-Standard eine neue Sprache zur Beschreibung des Kontrollflusses zwischen den MSCs, die nun Basic Message Sequence Charts (BMSCs) genannt werden, eingeführt. Diese Erweiterung erforderte eine neue Syntax- und Semantikdefinition, die wie die des MSC-92-Standards auf Prozessalgebra basiert, jedoch nicht mehr auf die Instanzlinien fokussiert ist, sondern auf Ereignisse, die durch Senden und Empfangen von Nachrichten ausgelöst werden [LRS10]. Hierdurch wurde, wie vorher durch die MSGs, die Beschreibung alternativer Abläufe, und somit die Modellierung auf einer höheren Abstraktionsebene ermöglicht. Die High-level Message Sequence Charts (HMSCs) stellen dem Modellierer im Gegensatz zu den MSGs einen verbesserten hierarchischen Strukturierungsmechanismus zur Verfügung. HMSCs repräsentieren einen Graphen, dessen Knoten entweder BMSCs oder wiederum HMSCs sein können. Die MSCs werden hierbei nicht direkt in den Knoten gespeichert, sondern die MSCs durch die Beschriftung der Knoten mit ihrem Namen referenziert. Durch die Beschriftung verschiedener Knoten mit den gleichen MSC-Namen können somit BMSCs wiederverwendet werden. Zusätzlich werden HMSCs zur Modellierung von Alternativen durch mehrere auslaufende Kanten zur Beschreibung von Schleifen und zur Einschränkung der möglichen Ausführungen durch restriktive Bedingungen verwendet.

Beispiel *High-Level-MSCs*

Unter Einsatz der HMSCs kann der erste Nachrichtenaustausch inklusive der Bedingung `IR2.Protokoll.Contains("CARDME")` in Teil (1) in Abbildung 3.4 als Vorbedingung für den Rest der Signatur genutzt werden. Abbildung 3.5 zeigt ein HMSC, das dies realisiert. Das HMSC beginnt mit einem umgedrehten Dreieck als Startknoten. Der mit dem Startknoten über eine gerichtete Kante verbundene Referenzknoten `Initialisation`, referenziert über seine Namensgleichheit das BMSC `Initialisation`, das den initialen Nachrichtenaustausch beschreibt. Der darauf folgende Bedingungskonten überprüft die Bedingung `IR2.Protokoll.Contains("CARDME")` und erreicht bei positiver Auswertung den Referenzknoten `CARDME`, der das entsprechende MSC referenziert. Ist da Protokoll nicht das CARDME-Protokoll, wird das HMSC am Endknoten – ein Dreieck mit der Spitze nach oben – beendet.

Neben der Einführung der HMSCs wurden auch die BMSCs im MSC-96-Standard um strukturelle Sprachkonstrukte wie einem Referenzmechanismus (MSC-Referenz) erweitert. Durch diesen Referenzmechanismus können Abläufe in separate BMSCs ausgelagert werden und aus mehreren BMSCs oder auch HMSCs referenziert werden. Zur Modellierung des Kontrollflusses innerhalb der BMSCs führte der MSC-96-Standard zusätzlich noch weitere strukturelle Sprachkonstrukte ein. Durch Inline-Ausdrücke (*engl. Inline Expressions*) für verschiedene Operatoren können Teilabläufe innerhalb eines BMSCs zu komplexeren Abläufen kombi-

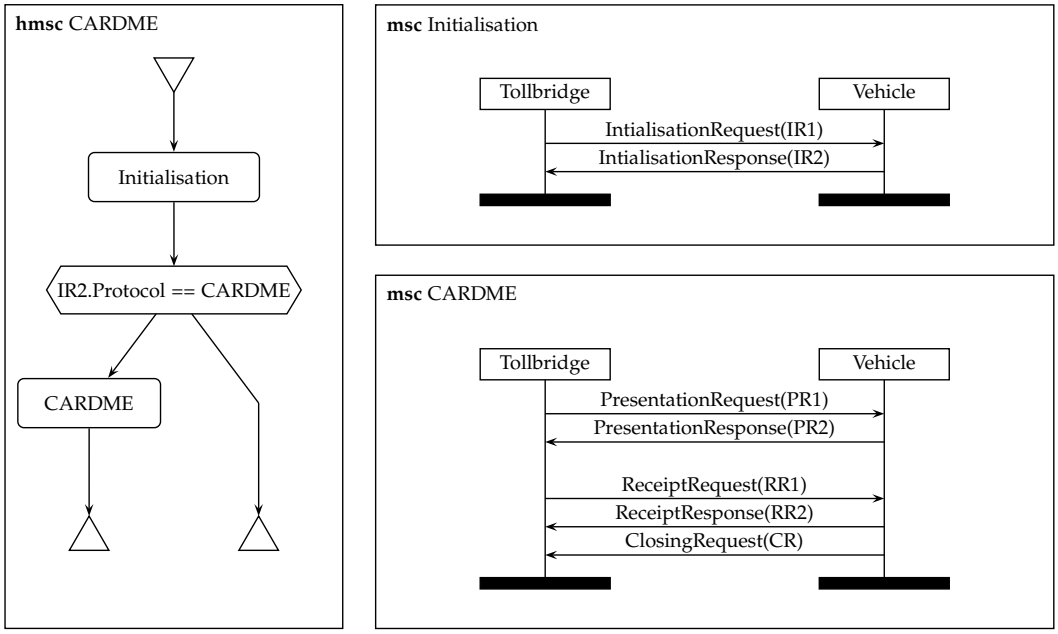


Abbildung 3.5: Vereinfachtes CARDME-Protokoll als HMSC

niert werden. Diese Operatoren ermöglichen die Modellierung alternativer Abläufe (*alt*-Operator), wiederholter Teilabläufe (*loop*-Operator), paralleler Kombination (*par*-Operator), optionaler Abläufe (*opt*-Operator), lose Ordnung (*seq*-Operator) und von Ausnahmen (*exc*-Operator).

Beispiel *Inline-Ausdrücke*

Eines dieser zusätzlichen strukturellen Sprachkonstrukte – der Inline-Ausdruck mit dem Operator *opt* – wird im BMSC in Abbildung 3.4 in Kombination mit einer Bedingung eingesetzt. Zusätzlich wird eine unbegrenzte Schleife durch den Inline-Ausdruck mit dem Operator *loop* spezifiziert.

Abhängig von den Daten, die in der ersten Präsentationsphase bestehend aus `PresentationRequest` und `PresentationResponse` als Parameter `PR2` übermittelt werden, bestimmt die Mautbrücke, ob eine weitere Präsentationsphase (Presentation2) benötigt wird. Dieses Verhalten ist in ein weiteres MSC `NonLocal` ausgelagert, das, wenn es durch das Protokoll gefordert wird, diese zweite Präsentationsphase durchführt.

Auf diese Präsentationsphase folgt die Quittungsphase, ein einfacher Nachrichtenaustausch, der für alle Fahrzeuge verpflichtend ist. Die Ausführung der nächsten Phase – Verfolgungsphase – wird durch einen Inline-Ausdruck mit dem opt-Operator an die Konfiguration der Mautbrücke gekoppelt. Ist die Mautbrücke auf Verfolgen konfiguriert, ist die als Sechseck dargestellte Bedingung (`Tollbridge.Tracking == true`) erfüllt und es findet in dieser Phase ein regelmäßiger Nachrichtenaustausch (`EchoRequest` und `EchoResponse`) statt, bis der Verfolgungsbereich verlassen wird. Diese wird durch eine Schleife (Inline-Ausdruck mit dem *loop*-Operator) modelliert. Diese Schleife wird `0` bis beliebig oft (⟨ `0, ∗`⟩) durchlaufen, wobei bei jeder Iteration ein vollständiger Nachrichtenaustausch (`EchoRequest` und `EchoResponse`) stattfindet. Wenn das Fahrzeug

den Bereich der Mautbrücke verlässt, wird unabhängig von der Verfolgungsphase eine Schließennachricht (`ClosingRequest`) vom Fahrzeug gesendet und die Transaktion abgeschlossen.

Der Standard wurde zuletzt 1999 signifikant zum MSC-2000-Standard [IT00] aktualisiert und erweitert. Zur besseren Strukturierung der auf MSCs basierenden Spezifikationen, wurde eine grafische Notation des MSC-Dokuments eingeführt. Dieses definiert explizit, welche Instanzen in dem Dokument verwendet werden, und trägt somit zu der Objektorientierung der MSCs bei. Zusätzlich findet eine explizite Unterscheidung zwischen definierenden MSCs, die sichtbar sind, und Hilfs-MSCs, die nur in definierenden MSCs genutzt werden, statt. Dynamische Variablen müssen nun zwingend im MSC-Dokument deklariert werden. Statische Variablen werden in der Parameterliste des jeweiligen MSCs deklariert und können in MSCs nicht verändert werden. Auch in dieser Aktualisierung wird keine eigene Datensprache zur Formulierung von Constraints und weiteren Annotationen festgelegt. Der MSC-Standard ermöglicht die Verwendung beliebiger Datensprachen. Des Weiteren können nun Methodenaufrufe zur Synchronisierung der MSCs eingesetzt werden.

Beispiel *MSC-Dokument*

Abbildung 3.6 stellt ein vereinfachtes MSC-Dokument für das CARDME-Protokoll dar. In diesem Dokument wird im Kopf des Diagramms hinter der Diagrammbezeichnung `mscdoc` der Name des Dokuments annotiert. Darunter folgt die Deklaration aller in den MSCs beteiligten Instanzen mit dem Schlüsselwort `inst` gefolgt von dem Namen der Instanz. Auf die Definition von Variablen der Instanzen `variables`, Nachrichten, mit ihren Parametern `msg`, der Datensprache `data` und andern Definitionen wurde in diesem Beispiel verzichtet. Unter diesen Definitionen ist das Dokument in zwei Teile durch eine horizontale gestrichelte Linie geteilt. Oberhalb dieser Linie im definierenden Teil werden als Referenzenknoten die Verweise auf die im Dokument enthaltenden MSCs notiert, die für den Nutzer sichtbar sind. Dies sind das in Abbildung 3.4 vorgestellte MSC `CARDME` und die alternative Variante für den Einsatz im Parkraummanagement `CARDME_Parking`. Unterhalb der Linie befindet sich der Hilfs-MSC-Teil, der die vom CARDME-Protokoll genutzten MSCs enthält, die nur durch die definierenden MSCs genutzt werden dürfen – `NonLocal`.

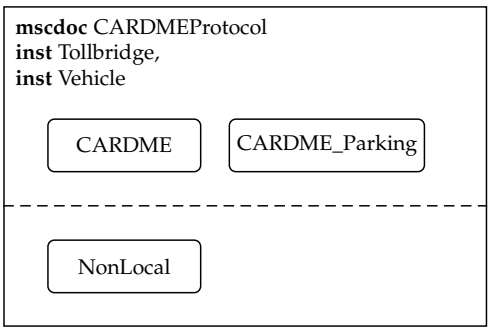


Abbildung 3.6: Vereinfachtes MSC-Dokument des CARDME-Protokolls

3.2.2 *Eignung der MSCs zur Signaturbeschreibung*

Die vorgestellten MSCs als eine Variante der Sequenzdiagramme eignen sich zur Modellierung von funktionalem Verhalten eines Systems. Die im MSC-96-Standard eingeführte übergeordnete Kontrollflusssprache (HMSCs) und die strukturellen Sprachkonstrukte der BMSCs wie die Inline-Ausdrücke können zur Beschreibung komplexer Zusammenhänge zwischen und in den Signaturen eingesetzt werden.

Allerdings besteht die Einschränkung, dass mit MSCs nur positives funktionales Verhalten des Systems beschrieben werden kann. So unterstützt die Semantik des MSC-96-Standards keine Notation für negatives Verhalten. Da Beschreibungen durch Sequenzdiagramme im Normalfall nicht umfassend sind, wird nur zwischen positiven und undefinierten Ereignissequenzen unterschieden, und es findet keine Unterscheidung zwischen negativem und undefiniertem Verhalten statt [LRS10]. Diese explizite Unterscheidung ist jedoch für eine verhaltensbeschreibende Signatursprache essentiell, da auch bekanntes Fehlverhalten des Systems und Angriffe auf das System modelliert werden müssen.

Der MSC-96- und der MSC-2000-Standard hat die Entwicklung der Sequenzdiagramme der UML2 (Unified Modelling Language in Version 2) stark beeinflusst und viele Konzepte wurden in diese übernommen. Zusätzlich findet in UML2-Sequenzdiagrammen eine den MSCs fehlende Unterscheidung zwischen positiven, negativen und undefinierten Sequenzen statt.

3.2.3 *UML2-Sequenzdiagramme*

Die Unified Modeling Language (UML) [OMG11b, OMG11c, RQJZ07] ist ein De-Facto-Standard, der durch die Object Management Group (OMG) standardisiert und gepflegt wird. Dieser stellt neben strukturbeschreibenden Diagrammen, wie in Abschnitt 2.1 vorgestellt, auch verhaltensbeschreibende Diagramme zur Verfügung. Eine Gruppe dieser verhaltensbeschreibenden Diagramme, die sich auf die Beschreibung der Interaktion zwischen Klassen bzw. Komponenten fokussieren, sind die Interaktionsdiagramme. Die Interaktionsdiagramme der UML und somit auch die im Folgenden betrachteten Sequenzdiagramme sind im Gegensatz zu MSCs in die strukturbeschreibenden Diagramme (Abschn. 2.1) der UML2 eingebettet.

Zur Strukturierung der als Interaktionsdiagramme modellierten Szenarien bietet die UML2 die in Abschnitt 3.1 vorgestellten Use-Case-Diagramme an. In diesen ist der Standardablauf der Use-Cases und dessen Alternativen entweder textuell oder grafisch (z. B. durch Sequenzdiagramme) spezifiziert. Zur Beschreibung des Kontrollflusses zwischen einzelnen Interaktionen dienen die Interaktionsübersichtsdiagramme, die in den UML2-Standard eingeführt wurden. Sie entsprechen den High-level Message Sequence Charts (HMSCs) des ITU-Standards und beschreiben den Kontrollfluss zwischen den einzelnen Interaktionen, die durch Timing-Diagramme, Kommunikationsdiagramme, Aktivitätsdiagramme und Sequenzdiagramme ausgerückt werden können. In der UML2 wurden die UML1.4-Sequenzdiagramme um die im MSC-96-Standard eingeführten Kon-

strukte zur Beschreibung von Abläufen erweitert. Somit stellen die UML2-Sequenzdiagramme nahezu alle Modellierungsmöglichkeiten der BMSCs zur Verfügung. Tabelle 3.2 zeigt die Entsprechungen zwischen den Elementen des MSC-2000-Standards und den UML2-Sequenzdiagrammen.

Beispiel *Modellierung des CARDME-Protokolls mit UML2-Sequenzdiagrammen* ———

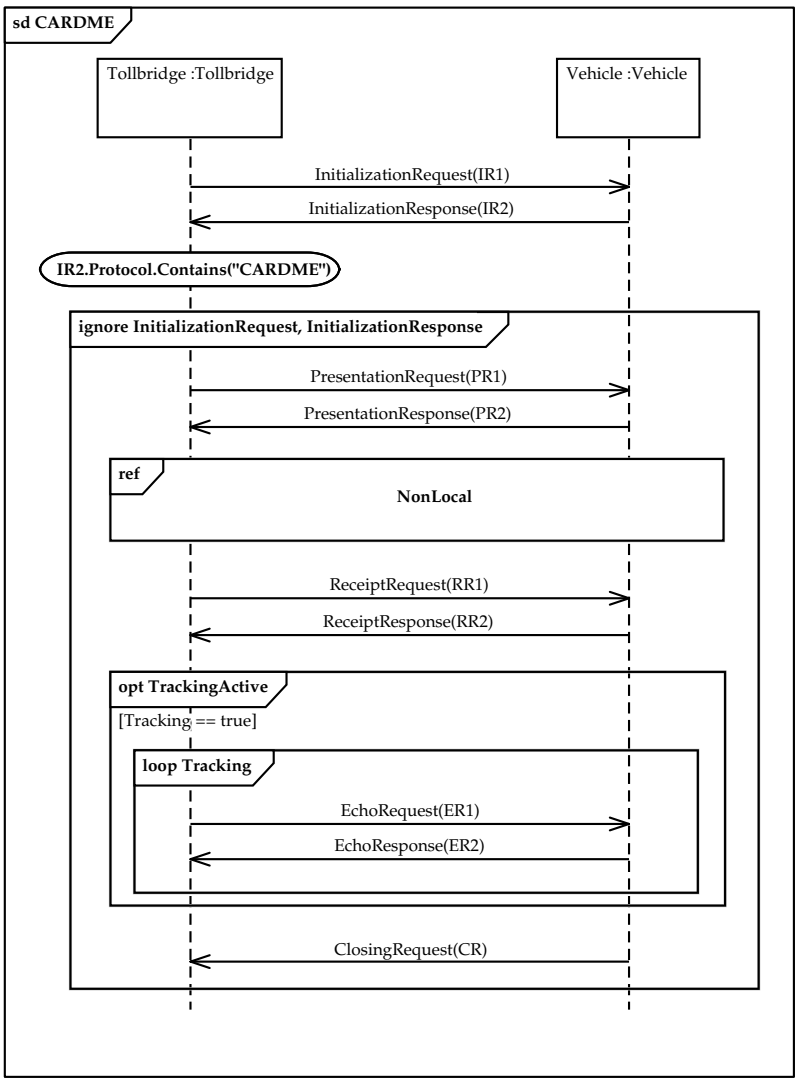


Abbildung 3.7: CARDME-Protokoll als UML2-Sequenzdiagramm

Durch die Übernahme der Modellierungselemente der MSCs in die UML2-Sequenzdiagramme, unterscheidet sich das Sequenzdiagramm des Mautbrückenszenarios in Abbildung 3.7 nahezu nur in der konkreten Syntax von dem in Abbildung 3.4 vorgestellten MSC-Diagramm. Die Instanzen, die an der Kommunikation teilnehmen, werden im UML2-Standard als Lebenslinien bezeichnet. Sie haben in diesem Beispiel eine Zuordnung zu einem strukturbeschreibenden Diagramm (Abschn. 2.1), hier das Komponentendiagramm in Abbildung 2.2, das die möglichen Beziehungen durch Modellierung von Ports, die durch Interfaces beschrieben werden, darstellt. In diesem Sequenzdiagramm

werden die Ports der Komponenten nicht als einzelne untergeordnete Lebenslinien dargestellt, sondern sie definieren über ihre zugeordneten Interfaces nur, welche Nachrichten zur Verfügung stehen. In der Initialisierungsphase findet die Kommunikation wie bei den MSCs über den Austausch asynchroner Nachrichten statt. Abhängig von der Auswertung der Fortsetzungsmarke (*engl. Continuation*), die als ovales Element dargestellt wird, wird das Sequenzdiagramm, beendet oder weiter abgearbeitet.

Zur Beschreibung von Zeitbedingungen, die über die partielle Ordnung und den einfachen Zeitannotationen (Zeitpunkt, Zeitintervall und Zeitdauer) der Sequenzdiagramme hinausgeht, setzt die UML2 auf Timing-Diagramme. Jedoch ist die Modellierung von Zeitgeber-Instanzen (*engl. Timer*), die gestartet, pausiert, wieder gestartet und einen Time-out auslösen können, in UML2-Sequenzdiagrammen und allgemein in den Interaktionsdiagrammen nicht vorgesehen. Diese Lücke überbrückt die UML2 durch die Definition verschiedener Erweiterungen, häufig als UML-Profile, die Zeitgeber [OMG08] und darüber hinaus gehende Konzepte den Sequenzdiagrammen hinzufügen. Eine solche ebenfalls von der OMG standardisierte Erweiterung ist das *Modeling and Analysis of Real-Time and Embedded Systems (MARTE)*-Profil [OMG11d], das u. a. die Annotation von Zeitgebern in Sequenzdiagrammen, wie es die MSCs unterstützen, ermöglicht.

In Tabelle 3.3 sind die Inline- bzw. Interaktionsoperatoren der MSCs bzw. UML2 gegenübergestellt. Inline-Ausdrücke bzw. Interaktionen können in beiden Sprachen geschachtelt werden. Zusätzlich bietet die UML2 die Möglichkeit Operatoren direkt in einem Fragment zu kombinieren und so Schachtelungen, die das Diagramm überfrachten würden, zu vermeiden. Neben den sechs Operatoren (*Alt*, *Loop*, *Par*, *Opt*, *Seq* und *Exc*, der in der UML2 mit *Break* bezeichnet wird), die aus dem MSC-2000-Standard übernommen wurden, führt die UML2 weitere Operatoren ein. Der *Strict*-Operator beschreibt im Gegensatz zum *Seq*-Operator unabhängig von den Lebenslinien und Operanden, dass die Ereignisse durch die Anordnung im Diagramm, von oben nach unten, auftreten müssen. Bereiche, die als kritisch (*Critical*-Operator) gekennzeichnet sind, führen zu einer nicht unterbrechbaren (atomaren) Ausführung der modellierten Ereignisse. Innerhalb dieses Bereichs treten keine anderen Ereignisse, die außerhalb des Bereiches an den Lebenslinien annotiert sind, auf. Zur Filterung von Nachrichten unterstützt die UML2 zwei weitere Operatoren – *Ignore* und *Consider*. Durch die Annotation der zu ignorierenden Nachrichten als kommaseparierte Liste hinter dem Operatornamen *ignore*, können so bewusst Modellierungsaspekte in einem Diagramm ignoriert oder ausgeblendet werden. Der *Consider*-Operand stellt die entgegengesetzte Funktionalität zum *Ignore*-Operanden zur Verfügung, indem alle nicht angegebenen Nachrichten ignoriert werden.

Beispiel *Interaktionsfragmente der UML2-Sequenzdiagramme*

Wie auch im MSC-Diagramm des CARDME-Protokolls (Abb. 3.4) wird in Abbildung 3.7 eine Interaktionsreferenz zu der ausgelagerten Signatur modelliert, die die zweite Präsentationsphase `NonLocal` beschreibt. Für die optionale Verfolgungsphase wird ein Interaktionsfragment mit dem Operator *Opt* eingesetzt und die beliebig häufige Wiederholung durch ein *Loop*-Interaktionsfragment

MSC-2000	**UML2-Sequenzdiagramme**
MSC (Message Sequence Chart)	Interaktionen (Sequenzdiagramme; Kommunikationsdiagramme; Timing-Diagramme; Interaktionsübersichtsdiagramme)
Ereignis (*engl. Event*)	Ereignisvorkommen (*engl. EventOccurence*)
MSC-Dokument (*engl. MSC Document*)	Klasse oder Kollaboration (*engl. Class or Collaboration*)
High-Level-MSC	Interaktionsübersichtsdiagramm (*engl. Interaction Overview Diagram*)
Instanz (*engl. Instance*)	Lebenslinie (*engl. Lifeline*)
Nachricht (*engl. Message*)	Nachricht (*engl. Message*)
Methodenaufruf (*engl. Method call*)	Operationsaufruf (*engl. Operation call*)
Methodengebiet (*engl. Method (area)*)	Aktionssequenz (*engl. ExecutionOccurence*)
Aktion (*engl. Action*)	Aktionssequenz (*engl. ExecutionOccurence*)
Suspensionsgebiet (*engl. Suspension(area)*)	-
Verknüpfungspunkt (*engl. Gate*)	Verknüpfungspunkt (*engl. Gate*)
-	Interaktionsfragment (*engl. Interaction Fragment*)
Inline-Ausdruck (*engl. Inline Expression*)	kombiniertes Interaktionsfragment (*engl. Combined Fragment*)
Coregionen (*engl. Coregion*)	Coregionen (*engl. Coregion*)
MSC-Referenz (*engl. MSC Reference*)	Interaktionsreferenz (*engl. Interaction Occurrance*)
Dekomposition (*engl. Decomposition*)	Zerlegung von Lebenslinien (*engl. PartDecomposition*)
Ordnungsbeziehung (*engl. General Ordering*)	Ordnungsbeziehung (*engl. General Ordering*)
Globale Bedingung (*engl. Global Condition*)	Fortsetzungsmarke (*engl. Continuation*)
Bedingung (*engl. Condition (predicate)*)	Bedingung (*engl. Interaction Constraint*)
Relative Zeit (*engl. Relative time*)	Zeitdauer (*engl. Duration*)
Absolute Zeit (*engl. Absolute time*)	Zeitpunkt, -intervall (*engl. Time*)
Zeitmessung (*engl. Time measurement*)	Zeitpunkt, Zeitpunktintervall (*engl. TimeObservationAction, DurationObservationAction*)
Zeitgeber (*engl. Timer*)	-

Tabelle 3.2: Vergleich der Sprachkonstrukte von MSC-2000 und UML2-Sequenzdiagrammen (basierend auf [Hau05])

	Kürzel im Diagramm		
Operator-Bez. (UML)	**MSC-2000**	**UML2 SD**	**Funktion**
Alternative Fragmente	Alt	Alt	alternative Ablaufmöglichkeiten
Schleife	Loop	Loop	iterative Interaktion
Parallele Fragmente	Par	Par	nebenläufige Abläufe
Optionales Fragment	Opt	Opt	optionale Interaktionsteile
Abbruchfragment	Exc	Break	Interaktion in Ausnahmefällen
Lose Ordnung	Seq	Seq	von Lebenslinie und Operanden *abhängige* chronologische Abläufe
Strenge Ordnung		Strict	von Lebenslinie und Operanden *unabhängige* chronologische Abläufe
Kritischer Bereich		Critical	atomare Interaktion
Irrelevante Nachrichten		Ignore	Filter für unwichtige Nachrichten
Relevante Nachrichten		Consider	Filter für wichtige Nachrichten
Negation		Neg	ungültige Interaktion
Sicherstellung		Assert	unabdingbare Interaktion

Tabelle 3.3: Vergleich der MSC-2000-Inline- und UML2-Interaktionsoperatoren (basierend auf [Hau05] und [RQJZ07])

spezifiziert. Zusätzlich zu den im MSC eingesetzten Operatoren wurde in diesem Diagramm in Abbildung 3.7 ein *Ignore*-Interaktionsfragment als Filter für die Nachrichten `InitialisationRequest` und `InitialisationResponse` spezifiziert. Hierdurch muss nicht explizit an jeder Stelle, an der eine weitere Initialisierungsnachricht von der Mautbrücke gesendet werden kann, diese auch explizit modelliert werden.

Die letzten beiden Operatoren, der *Assert*- und *Neg*-Operator, wurden in der UML2-Spezifikation eingeführt, um zwischen benötigtem und verbotenem Verhalten zu unterscheiden. Diese Operatoren können sowohl in den Sequenzdiagrammen selbst als auch in den Interaktionsübersichtsdiagrammen eingesetzt werden. Der *Assert*-Operator gibt eine verbindliche Reihenfolge für die modellierten Nachrichten an, von der nicht abgewichen werden darf, und definiert so indirekt negatives Verhalten. Der *Neg*-Operator dient zur expliziten Markierung ungültiger Abfolgen, wobei alle andern Reihenfolgen positives Verhalten darstellen. Diese zur Beschreibung von negativem Verhalten essenziellen Operatoren werden jedoch nach [RQJZ07] nicht oft eingesetzt. Dies resultiert nicht aus der Überflüssigkeit der beiden Operatoren, sondern ist der schlechten Einbettung in die UML2-Semantik geschuldet [HM08].

3.2.4 *Eignung der UML2-SD zur Signaturbeschreibung*

Zum Einsatz der UML2-Sequenzdiagramme als verhaltensbeschreibende Signatursprache ist eine klare Spezifikation von verbotenem (negativem) Verhalten unabdingbar. Die auf einem Paar von gültigen und ungültigen eindeutigen Mengen (*engl. sets*) an Ereignissequenzen (*engl. Traces*) basierende Semantik der UML2-Sequenzdiagramme führt jedoch zu Problemen bei der Definition der Operatoren.

Da die Vereinigung der gültigen und ungültigen Mengen an Ereignissequenzen der Sequenzdiagramme nicht umfassend ist, ergibt sich wie in Abbildung 3.8 dargestellt eine dritte Menge – die unspezifizierte Ereignissequenzen. Diese sind weder den gültigen noch den ungültigen Ereignissequenzen zugeordnet. In der UML2-Spezifikation [OMG11c] ist diese Zuordnung der aus den beiden Operatoren abgeleiteten Ereignissequenzen zu diesen drei Mengen nach Harel und Maoz [HM08] inadäquat und vage. So ist der Gültigkeitsfokus der beiden Operatoren nicht klar definiert. Zum einen schreibt die Spezifikation vor [OMG11c, S. 484], dass die Sequenzen im *Assert*-Operator alle gültigen Fortsetzungen beschreibt und alle anderen Fortsetzungen zu ungültigen Ereignissequenzen führen. Zum andern legt die Spezifikation fest [OMG11c, S. 495], dass ungültige Ereignissequenzen nur durch den Einsatz des *Negate*-Operators entstehen. Des Weiteren wird in der Spezifikation nicht definiert, was die genaue Bedeutung von „gültig“ (*engl. valid*) ist – *manchmal* möglich, *zu Beginn* möglich oder *immer* möglich.

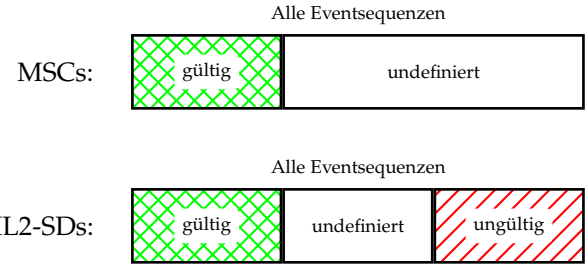


Abbildung 3.8: Kategorien der Ereignissequenzen in der MSC- und UML2-Semantik

Aufgrund dieser Widersprüche in der Semantik der beiden Operatoren, die Harel und Maoz [HM08] attestieren, und aufgrund der verschiedenen Interpretationsmöglichkeiten, die zusätzlich in [Stö04] aufgezeigt werden, können diese Interaktionsfragmente nicht effektiv eingesetzt werden. Harel und Maoz stellen als Wurzel des Problems heraus, dass *Negate* und *Assert* als einfache Operatoren eingeführt wurden, obwohl sie von Natur aus Modalitäten der Diagramme sind. Um diese Unzulänglichkeit zu beheben, hat Harel et al. [HM08] ein Konzept zur Unterscheidung zwischen notwendigem und möglichem Verhalten aus den LSCs als eine leichtgewichtige Erweiterung (UML Profil) der UML-Sequenzdiagramme vorgeschlagen – die Message Sequence Diagrams (MSDs). Dieses Profil erweitert die Sequenzdiagramme um die Möglichkeit, einzelne Elemente der Sequenzdiagramme wie in den Live Sequence Charts (LSCs) (Abschn. 4.1) als möglich (*kalt*) und notwendig (*heiß*) zu markieren. Der *Negate*- und *Assert*-Operator dienen hier nur noch als syntaktische Notation, ob dessen Inhalt *heiß* oder *kalt* markiert ist.

Somit sind sowohl in den MSDs als auch in den LSCs, aus denen das Konzept der Modalitäten entnommen wurde, diese beiden Operatoren eindeutig definiert. Hierdurch lassen sich in beiden Sprachen die für eine Signatursprache essenzielle Unterscheidung zwischen negativen und positiven Ereignissequenzen mit einer klaren Semantik spezifizieren. Da MSDs und LSCs in ihrer Ausdrucksstärke nahezu identisch sind, LSCs jedoch die klarere Aufteilung zwischen Vorbedingung und Hauptteil der Signatur haben, wird im Folgenden die Eignung der LSCs als Basis für die in Kapitel 5 vorgestellte MBSecMon-Spezifikationssprache unter-

sucht. Hierzu werden in Abschnitt 3.3 zunächst bereits existierende Signaturbeschreibungssprachen betrachtet.

3.3 Verwandte Arbeiten (Signaturspezifikationssprachen)

Neben verschiedenen Arten der Sequenzdiagramme werden zur Spezifikation von erlaubtem und verbotenem Verhalten und zur Spezifikation von nicht-funktionalem Verhalten als Signaturen eine Vielzahl von Formalismen eingesetzt. In Abschnitt 3.3.1 wird ein Überblick über diese Formalismen gegeben und in Abschnitt 3.3.2 herausgestellt, warum Live Sequence Charts eine gute Basis für eine verhaltensbeschreibende Signatursprache bilden. Im Folgenden werden in Abschnitt 3.3.3 Sprachen zur Strukturierung von Sequenzdiagrammen untersucht und in Abschnitt 3.3.4 gezeigt, warum Anwendungsfalldiagramme als Strukturierungsmechanismus für die MBSecMonSL eingesetzt werden.

3.3.1 *Verhaltensbeschreibende Spezifikationssprachen*

Der MBSecMon-Entwicklungsprozess für Security-Monitore benötigt für die Modellierung der Signaturen eine Sprache zur Spezifikation von erwartetem und verbotenem Verhalten. Diese Sprache muss sowohl zur Beschreibung *funktionaler* als auch zur Beschreibung *nicht-funktionaler* Anforderungen (NFA), die überwacht werden sollen, geeignet sein. Während funktionale Anforderungen in szenariobasierten Ansätzen einfach spezifiziert werden können, stellt die Spezifikation von NFAs eine größere Herausforderung dar. NFAs werden in der Anforderungsanalyse häufig in natürlicher Sprache verfasst und sind hierdurch ungenau und abstrakt. Im Sicherheitsumfeld werden zur formaleren Beschreibung solcher Anforderungen als Signaturen für Intrusion Detection Systeme (IDS) spezielle Policysprachen oder temporale Logiken eingesetzt.

Expertensysteme: Eines dieser Konzepte zur Beschreibung von Signaturen für IDS sind die komplexen Expertensysteme (ES) wie das *C Language Integrated Production System* (CLIPS) [GR98] und P-BEST [LP99]. Diese Systeme verwenden deklarative Inferenzregeln, die eine Wenn-Dann-Struktur besitzen. Sobald die Bedingung des Wenn-Teils einer Regel erfüllt ist, wird die Aktion im Dann-Teil der Regel ausgeführt, die die Fakten des Expertensystems verändert. Diese Fakten beschreiben den Zustand des überwachten Systems. Ausgelöst durch die Veränderung der Fakten durch das System oder durch eine Regel kann wiederum der Wenn-Teil weiterer Regeln erfüllt sein, die wiederum die Fakten ändern. Auch die Modellierung zeitlicher Bedingungen in Expertensystemen muss auf Basis der Faktenlage erfolgen. Dies stellt große Herausforderungen an den Modellierer bei der Modellierung der Regeln und der Faktenbasis, da diese Herangehensweise nicht intuitiv ist. Zwischen den vielen kleinen Regeln existieren in der Spezifikation nur implizite Zusammenhänge basierend auf den Änderungen der Faktenbasis. Zusätzlich sind ES für Nicht-Experten aufgrund der Aufteilung der Spezifikationen in Inferenzregeln, temporale Regeln, Fakten und ihrer Interaktion nur schwer verständlich.

Temporale Logiken: Andere, besser zur Modellierung temporaler Aspekte für Laufzeitmonitore geeignete Ansätze, verwenden temporale Logiken (TL) wie die Linear-Time Temporal Logic (LTL). Meredith et al. [MJG+12] betrachten verschiedene Ansätze zur Monitorgenerierung, die Varianten der TL als Spezifikationssprache verwenden. Java PathExplorer (JPaX) [HR01], ein Werkzeug zur Überwachung der Ausführung von Java-Programmen, setzt LTL (Past- und Future-Time LTL) zur Beschreibung von High-Level-Anforderungen ein. Das kommerzielle Werkzeug Temporal Rover [Dru00] übersetzt Kommentare, die in Future-time TL (FTTL) formuliert sind, in ausführbaren Java-Quelltext, der die Einhaltung der so spezifizierten Bedingungen an das System überwacht. Das Monitoring Oriented Progamming (MOP) Laufzeitverifikationsrahmenwerk [MJG+12] unterstützt als Spezifikationssprache für die Laufzeitmonitore temporale Logiken wie LTL und Past-Time LTL (PTLTL). Diese Spezifikationen werden in MOP jedoch wie anderen Eingabesprachen (Erweiterte Reguläre Ausdrücke, ...) in endliche Automaten (*engl. finite state machines*) (FSMs) zur Weiterverarbeitung und Ausführung übersetzt. PTLTL ist für die Spezifikation von Laufzeitmonitoren besser als LTL geeignet [MJG+12, HR01], da den aus PTLTL-Spezifikationen generierten Monitoren die Vergangenheit bekannt ist. Ein Monitor, der auf LTL basiert, stellt Annahmen über die Zukunft auf und kann sich somit in einem unbestimmten Auswertungszustand befinden, solange die LTL-Formel weder verletzt wurde, noch alle möglichen zukünftigen Ereignissequenzen die Formel erfüllen. PLTL erfordert jedoch die Speicherung der Vergangenheit in der Implementierung, wie in MOP durch eine Tabelle realisiert, und führt zu mehr Speicherverbrauch.

Diesen auf temporalen Logiken basierenden Spezifikationssprachen fehlt für den Einsatz im MBSecMon-Entwicklungsprozess eine allgemein verständliche grafische Notation. Die in ihnen verfassten Spezifikationen sind aufgrund ihrer abstrakten Notation und oftmals großen Umfangs schwer zu formulieren und zu verstehen [MN08]. Zur schlechten Lesbarkeit trägt zusätzlich bei, dass zur besseren Übersetzbarkeit der TL-Formeln in Monitore häufig Sammlungen von Eigenschaften eingesetzt werden. Diese Sammlungen beschreiben z. B. die Reihenfolge aufeinanderfolgender Ereignisse, da eine globale Formulierung der Gesamteigenschaft bzw. der Sequenz von Ereignissen zu einem exponentiellen Wachstum des Monitors führt [KHP+05]. Eine grafische Repräsentation der Spezifikationssprache vereinfacht die Modellierung und das Verständnis der Signaturen [Wu93].

Erweiterungen der UML: Zur Überwindung der schweren Erlernbarkeit der bisher vorgestellten Sprachen und zur Steigerung der Verständlichkeit für Nicht-Experten werden verschiedene Erweiterungen der UML vorgeschlagen. Eine leichtgewichtige Erweiterung der UML stellt die *UMLsec* [Jür05, Jür02] dar, die in UML-Diagrammen verfasste Systemspezifikationen um formal spezifizierte Sicherheitsanforderungen wie Vertraulichkeit oder Verschlüsselung der Kommunikation erweitert. Durch ein UML-Profil werden Stereotypen und Tagged-Values definiert, mit denen das Systemmodell annotiert wird. Diese Stereotypen beschreiben Sicherheitsannahmen auf physikalischer Ebene, Sicherheitsanforderungen der logischen Struktur eines Systems und Sicherheitsrichtlinien (*engl. security policies*), die von Teilen des Systems bzw. des Modells eingehalten werden müssen. Auf solchen angereicherten statischen und dynamischen UML-Modellen (z. B. Klassen- und

Sequenzdiagramme) werden Konsistenzüberprüfungen der Modelle selbst und zwischen den Modellen durchgeführt, um z. B. die Einhaltung erlaubnisbasierter Zugriffskontrollrichtlinien abzusichern [Jür08]. Für diese statische Verifizierung werden Security-Checker der UMLsec-Toolsuite[1] eingesetzt. Zur dynamischen Absicherung des Systems während der Laufzeit generiert die UMLsec-Toolsuite Java Runtime-Checks aus den UMLsec-Spezifikationen, die die Sicherheitsrichtlinien festlegen. In aktuellen Veröffentlichungen von Bauer et al. [BJ08, BJ10, BJY11] dienen UMLsec-Spezifikationen als Ausgangspunkt, um über Temporale Formeln FSMs zu generieren, die als Laufzeitmonitore eingesetzt werden. Hierbei wird nicht explizit der Schritt der Abbildung der UMLsec-Annotationen in Temporale Logiken vorgestellt, sondern der Übersetzungsprozess der Temporalen Formeln in FSMs beschrieben. Auf diese Übersetzung wird im Rahmen der Betrachtung verwandter Arbeiten in Abschnitt 7.2 genauer eingegangen.

Im Gegensatz hierzu wurde mit *SecureUML* [LBD02, BDL06] eine schwergewichtige Erweiterung am UML-Metamodell vorgenommen. Dieser Dialekt der UML-Sprache dient der Modellierung von Einschränkungen in Systemen mit rollenbasierter Zugriffskontrolle. Der Fokus liegt bei diesem Ansatz auf der Beschreibung einer semantische Basis zur Annotation von Klassen- oder Zustandsdiagrammen, mit dem Ziel Implementierungen zu generieren und die modellierten Policies zu analysieren. Beide Ansätze erweitern die UML, um nicht-funktionale Sicherheitsbedingungen in einem modellbasierten Entwicklungsprozess beschreiben zu können. Hierbei fokussieren sie sich jedoch hauptsächlich auf die Beschreibung statischer Systemeigenschaften und sehen keine Modellierung verhaltensbeschreibender Signaturen für IDS bzw. Monitore vor. Zur Spezifikation von Sicherheitseigenschaften des Verhaltens schlägt Hussein et al. [HZ06] das UML-Profil *UMLintr* vor, dass Stereotypen zur Annotation verhaltensbeschreibender UML-Diagramme definiert. Hierfür werden Use-Cases zur Szenariobeschreibung, Klassendiagramme zur Modellierung der Struktur und Zustandsdiagramme zur Modellierung potenzieller Verhaltensabläufe des Systems eingesetzt.

Neben UML-Zustandsdiagrammen werden auch andere Zustandstransitionssysteme zur Modellierung verhaltensbeschreibender Signaturen eingesetzt. Eine einfach strukturierte Variante zur Modellierung von Angriffsszenarien verwendet die *State Transition Analysis Technique* (STAT) [VEK00]. Die Modellierungssprache STATL des STAT-Ansatzes besitzt eine textuelle Syntax, die jedoch auch grafisch als Zustandsdiagramm repräsentiert werden kann. Auf Basis von Übergabeparametern an das Szenario und Zuständen, die durch Transitionen verbunden sind und die Schritte des Angriffs beschreiben, werden komplette Szenarien spezifiziert. Zustände besitzen einen eindeutigen Namen zur Identifikation und Beschreiben über eine Bedingung, ob der Zustand betreten werden darf und der in ihnen annotierte Codeblock ausgeführt wird. Transitionen sind ebenfalls eindeutig über ihren Namen identifizierbar. Ihr Typ legt fest, ob die Transition konsumierend, nicht-konsumierend oder rückabwickelned (*engl. unwinding*) ist. Schaltet eine nicht-konsumierende Transition, bleibt der Quell- und Zielzustand aktiv,

[1] UMLsec-Toolsuite: http://www.umlsec.de/ (Seit 2011 existiert eine Neuimplementierung dieser Toolsuite unter dem Namen CARiSMA (http://carisma.umlsec.de). CARiSMA basiert auf dem Eclipse Modeling Framework (EMF) und unterstützt somit alle Sprachen, die auf dem EMF basieren – somit auch UML2.)

eine Unwinding-Transition beschreibt, unter welchen Bedingungen in den vorherigen Zustand zurückgekehrt werden muss. Das Schalten wird durch eines an der Transition annotiertes Ereignis (in STATL Aktion genannt) bestehend aus einem Typ und einem Namen ausgelöst. Zusätzlich lassen sich Transitionen wie Zustände bedingen und für den Fall des Schaltens ein auszuführender Codeblock definieren. STATL unterstützt weder Fork- noch Join-Operatoren, wodurch eine Transition immer nur einen Vorgänger und einen Nachfolger haben kann. Dies vereinfacht zwar die Interpretation der Signaturen, resultiert jedoch bei der Modellierung von nebenläufigen Verhalten in einer Zustandsexplosion, da jede Permutation der sonst als nebenläufig modellierten Zustände als separater Zustand modelliert werden muss [SEJK08].

Petrinetze: Im Gegensatz zu STAT, verwendet Kumar die ausdrucksstarken gefärbten Petrinetze (*engl. colored Petri nets (CPN)*) zur Signaturbeschreibung in seinem Werkzeug *Intrusion Detection In Our Time* (IDIOT) [Kum95]. Die Petrinetze werden zusätzlich mit ML – einer funktionalen Programmiersprache – annotiert. Die komplexe Syntax der Spezifikationssprache, bestehend aus Petrinetzen und einer funktionalen Programmiersprache, erlaubt eine kompakte Modellierung der Signaturen, macht sie jedoch schwer verständlich.

Eine grundlegende Variante der Petrinetze wird von Frankowiak et al. [FGP09] zur Spezifikation von Prozessmonitoren genutzt, die auf preiswerten Mikrocontrollern eingesetzt werden sollen. Hierfür werden einfache Petrinetze um einen Markengenerator und spezielle Endplätze erweitert, die jedoch keine Unterscheidung zwischen negativen und positiven Enden der Ausführung des Netzes zulassen. Die einzelnen modellierten Teilnetze werden über ein Kontrollnetz zusammengeschaltet. Aufgrund der expliziten Modellierung aller möglichen Abläufe wird die Spezifikation von Monitoren für komplexere Systeme sehr groß.

Eine umfassende Signatursprache für die Modellierung von Misuse-Szenarien in Intrusion Detection Systemen, die ebenfalls auf Petrinetzen basiert, ist die Event Description Language (EDL) [Sch04, MS05, FM12]. Diese Variante der Petrinetze weicht sowohl in der Syntax als auch der Semantik von Standardpetrinetzen ab. Die Syntax definiert vier verschiedene Platzarten – *Initial-*, *Interior-*, *Escape-* und *Final*-Plätze – und erlauben hierdurch die Unterscheidung zwischen positivem und negativem Beenden der Überwachung der Signatur. Zusätzlich enthalten Plätze Merkmale (*engl. features*), die durch die an den Transitionen annotierten Bindungsausdrücke verändert werden können. An Transitionen können neben Ereignissen auch Bedingungen, Variablen- oder Merkmalbindungen und Aktionen annotiert werden. Diese Annotationen greifen auf die Merkmale der Ereignisse und Plätze zu und können die den Markierungen zugeordneten Merkmalswerte der Plätze ändern. Kanten zwischen Platz und Transition können als konsumierend oder nicht-konsumierend markiert werden. Zusätzlich unterstützt die EDL nicht nur ein Hierarchisierungskonzept für Ereignistypen, sondern erlaubt es ganze Signaturnetze als Schaltbedingung in Transitionen zu hinterlegen. Die Einbettung der Netze in Transitionen fördert zwar die Übersichtlichkeit der Signaturspezifikationen, erschwert jedoch durch die starke Färbung (Merkmale der Plätze und Transitionen) der Petrinetze das Verständnis durch Nicht-Experten.

Sequenzdiagramme: Eine Steigerung der Verständlichkeit der Spezifikationen auch für Nicht-Experten und eine erleichterte Modellierung der Signaturen kann durch die Entwicklung und Repräsentation der Signaturen auf einer höheren Abstraktionsebene erreicht werden. Hierzu setzen Massacci und Naliuka [MN08] zur Modellierung von verhaltensbeschreibenden Signaturen UML-Sequenzdiagramme (SDs) ein, die mit temporalen Logiken erweitert sind. Da die Semantik von SDs nicht zur Modellierung deontischer Bedingungen wie Verpflichtungen, Erlaubnissen und Verboten (Def. 4) geeignet sind, wurde ihre Semantik für den Einsatz als Signatursprache angepasst.

Definition 4 (*Deontische und Modale Logik*)**.** Die *deontische Logik*, ein Spezialfall der Modallogik, ist eine Erweiterung der klassischen Aussagenlogik. Neben den Operatoren der Aussagenlogik wie die Negation, Konjunktion und Disjunktion, unterstützt die deontische Logik zusätzliche Operatoren (Verpflichtung O und Möglichkeit P) zur Formulierung normativer Aussagen. Zusammen mit dem Negationsoperator ($\neg$) der Aussagenlogik lassen sich *Verpflichtungen*, *Erlaubnisse* und *Verbote* formulieren (Tab. 3.4). Basierend auf [CF08] können Operationen der deontischen Logik (Verpflichtung, Erlaubnis, Verbot) auf Operatoren der *modalen Logik* (Notwendigkeit, Möglichkeit, Unmöglichkeit) abgebildet werden.

	Deontische Deutung		Modale Deutung	
Verpflichtung/Notwendigkeit	Es ist geboten, dass φ	$(O\varphi)$	Es ist notwendig, dass φ	$(\Box\varphi)$
Erlaubnis/Möglichkeit	Es ist erlaubt, dass φ	$(P\varphi)$	Es ist möglich, dass φ	$(\Diamond\varphi)$
Verbot/Unmöglichkeit	Es ist verboten, dass φ	$(\neg P\varphi)$	Es ist unmöglich, dass φ	$(\neg\Diamond\varphi)$

Tabelle 3.4: Abbildung von deontischer Logik auf modale Logik

Hierdurch besteht die Gefahr, dass Entwickler basierend auf ihrem bestehenden UML-Wissen, fehlerhafte Signaturen modellieren und sie von Nicht-Entwicklern schwer oder falsch verstanden werden. Ein sehr ähnlicher Ansatz von Solhaug et al. [SES07] teilt dieselben Nachteile. In dieser Fallstudie werden Sequenzdiagramme zur Definition von Policyspezifikationen verwendet, deren Semantik auf STAIRS [HHRS05] basiert. Der STAIRS-Ansatz definiert eine formale Grundlage für die Verwendung von UML2-Interaktionsdiagrammen und beschreibt einen Prozess für einen schrittweisen, inkrementellen Entwicklungsprozess eines Systems.

Die Semantik der UML2-Interaktionsdiagramme wird in STAIRS durch Traces beschrieben. Jedes positive bzw. negative Verhalten, beschrieben durch Interaktionsdiagramme, wird in eine Menge positiver Traces bzw. negativer Traces eingeordnet. Undefinierte Traces sind zunächst nicht Teil der Spezifikation, können jedoch durch die Verfeinerungsmechanismen einem der beiden Tracesets zugeordnet werden. Zur Definition von möglichen und notwendigen sich ausschließenden Alternativen bietet STAIRS zusätzlich zum *alt*-Operator der UML2 einen *xalt*-Operator an. Um diese Unterscheidung zwischen verpflichtendem und möglichem Verhalten in der Semantik zu realisieren, führt der STAIRS-Ansatz eine

Menge von Interaktionsverpflichtungen (*engl. interaction obligations*) ein. Jede Interaktionsverpflichtung enthält eine Menge positiver und negativer Traces, die die möglichen Alternativen dieser Verpflichtung angeben, und für eine gültige Implementierung des Systems alle erfüllt sein müssen.

Live Sequence Charts: Als alternative Modellierungssprache werden Live Sequence Charts (LSCs) vorgeschlagen, da diese durch ihre Temperaturen nativ in ihrer Semantik Modalitäten unterstützen [SES07]. Zur Spezifikation von Hardwareprotokollen auf Registerebene setzt Bunker et al. [BG01, BGS05] *Protocol LSCs (PLSCs)* ein, um HDL-Implementierungen zu verifizieren. Hierbei handelt es sich um eine reduzierte Variante der LSCs, die jedoch auf die LSCs von Harel [DH01] abgebildet werden kann. Die Einschränkungen dienen zur einfacheren Übersetzbarkeit in temporale Logiken und beziehen sich u. a. auf das Variablenmodell, in dem die Variablen nur inkrementiert, dekrementiert und zurückgesetzt werden dürfen. Anstelle des Precharts der Universal LSCs verwenden PLSCs kalte Bedingungen zur Modellierung der Vorbedingung eines Charts. Nachrichten werden durch heiße und kalte Subcharts gruppiert und Coregionen zur Modellierung von beliebigen Reihenfolgen der Nachrichten verwendet. Sowohl Locations auf Sende- und Empfangsseite der Nachrichten als auch Subcharts können mit Multiplizitäten annotiert werden, die die Anzahl der Wiederholungen angeben. Nachbedingung werden durch heiße Bedingungen am Ende des Subcharts modelliert. Zur Definition des Timings auf Registertransferebene definieren PLSCs ein eigenes Timing-Modell mit der Notation der Anzahl an Ticks des Zeitgebers (*engl. clock*) als horizontale annotierte Linie, die alle Lebenslinien des Diagramms überspannt. Diese eingeschränkte Variante der LSCs eignet sich zur Modellierung auf der niedrigen Abstraktionsebene der Registerebene, ist jedoch nicht ausdrucksstark genug Protokolle auf einer höheren Abstraktionsebene kompakt zu spezifizieren.

Kumar und Mercer [KM09] zeigen, dass LSCs zur Spezifikation von Protokollen zur Beschreibung von Inter-Prozess-Kommunikation geeignet sind. Diese Protokolle werden in Automaten übersetzt, die zur Verifikation des Systems durch einen Modelchecker eingesetzt werden. Hierbei wird herausgestellt, dass Subcharts, unbeschränkte Schleifen und hierarchische Charts der LSCs essenziell für die Protokollspezifikation sind.

3.3.2 *Eignung der Formalismen zur Signaturbeschreibung*

Die in Abschnitt 3.3.1 vorgestellten Formalismen zur Beschreibung von funktionalen und nicht-funktionalen Anforderungen unterscheiden sich in ihrer Ausdrucksstärke, ihrem Formalisierungsgrad und ihrer Verständlichkeit für Nicht-Experten stark voneinander.

Das größte Potential dieser Formalismen als verhaltensbeschreibende Spezifikationssprache, die auch für Nicht-Experten verständlich ist, zeigen die LSCs. Im Gegensatz zu den vorgestellten Expertensystemen und Temporalen Logiken, besitzen die LSCs eine grafische konkrete Syntax. Diese Syntax entspricht zum Großteil der aus dem szenariobasierten Design bekannten Beschreibung von Interaktionssequenzen als Sequenzdiagramme, die zur Kommunikation mit Nicht-

Experten in der Praxis eingesetzt werden. Zusätzlich unterstützen LSCs im Gegensatz zu anderen Sequenzdiagrammen nativ die Spezifikation von Modalitäten als Temperaturen der Diagramme und ihrer Elemente. Hierdurch lassen sich die zur Signaturbeschreibung zwingend benötigten deontischen Bedingungen (Def. 4) durch vorhandene Sprachmittel spezifizieren. Durch den Einsatz dieser Modalitäten können Signaturen kompakter, sich auf weniger Diagramme erstreckend, modelliert werden [BG01].

Eine genauere Untersuchung in Abschnitt 4.3 wird jedoch zeigen, dass selbst die ausdrucksstarken LSCs zwar viele der Anforderungen an eine verhaltensbeschreibende Signatursprache erfüllen, jedoch einige Erweiterungen (Abschn. 5.1) notwendig sind, um Signaturen komfortabel modellieren zu können.

3.3.3 *Strukturierung der Spezifikation*

Die Erstellung einer Sicherheitsmonitorspezifikation für große Systeme ist selbst mit dem Einsatz von Sequenzdiagrammen eine komplexe Aufgabe, da jedes Sequenzdiagramm seine Gültigkeit durch eine Sequenzabfolge selbst beschreiben muss. Um die Spezifikationen übersichtlicher zu gestalten und die Generierung von effizienten Monitoren zu vereinfachen, wird ein Strukturierungsmechanismus auf höherer Abstraktionsebene benötigt. In der Literatur werden hauptsächlich zwei Varianten zur Strukturierung von Sequenzdiagrammen eingesetzt. Zum einen existieren Übersichtsdiagramme, die den Kontrollfluss explizit modellieren, und zum anderen Anwendungsfalldiagramme, die auf einer abstrakteren Ebene Zusammenhänge zwischen Szenarien beschreiben.

Übersichtsdiagramme: Die meisten Ansätze zur Strukturierung von Sequenzdiagrammdialekten haben zum Ziel mögliche Ausführungsreihenfolgen zwischen den einzelnen Diagrammen auf einem höheren Abstraktionsniveau als die Sprache selbst zu modellieren. Eine Kompositionstechnik für MSCs sind die in Abschnitt 3.2.1 vorgestellten HMSCs, die die sequenzielle und parallele Komposition von BMSCs ermöglichen [UKM04]. Ein sehr ähnlicher Ansatz zu den HMSCs stellen die UML2-Interaktionsübersichtsdiagramme (IODs) [OMG11b] dar, die eine Kombination aus UML2-Zustandsdiagrammen und Aktivitätsdiagrammen sind. Als Knoten des Graphen sind diese Diagramme nicht wie die HMSCs auf einen Diagrammtyp eingeschränkt, sondern alle Interaktionsdiagrammtypen der UML2 können verwendet werden. Alle diese Spezifikationssprachen zur Strukturierung der Spezifikation sind vom Abstraktionsniveau sehr nahe an der Implementierung angesiedelt, da sie explizit die möglichen Ausführungsreihenfolgen der einzelnen Diagramme beschreiben.

Anwendungsfalldiagramme: Greenyer [Gre11] verwendet in seinem Ansatz Anwendungsfalldiagramme als Sammelbehälter zur Strukturierung einer Spezifikation aus Modal Sequence Charts [HM08], eine vereinfachte Version der LSCs, zur Definition des Verhaltens mechatronischer Systeme. Diese Anwendungsfalldiagramme visualisieren das Zusammenspiel zwischen den statischen und dynamischen Modellen der Spezifikation. Hierfür werden die an der Verhaltensbeschreibung beteiligten Instanzen der statischen Modelle in den Anwendungsfällen annotiert, jedoch stehen die Anwendungsfälle nicht untereinander in Beziehung.

Jayaraman et al. [JW07] präsentiert einen Ansatz für ausführbare Anwendungsfälle – UCSim. Das Verhalten der ausführbaren Anwendungsfälle wird durch Zustandsautomaten beschrieben, die in den Anwendungsfällen hinterlegt sind. Die Spezifikation der erlaubten Ausführungsreihenfolgen erfolgt über einen Kontrollfluss, der explizit zwischen den Anwendungsfällen modelliert wird. Dieser Ansatz betrachtet jedoch nur die Spezifikation von erwartetem Verhalten.

Anwendungsfälle können in frühen Phasen des Entwicklungsprozesses zur Beschreibung funktionaler Anforderungen eingesetzt werden, wobei jedoch nichtfunktionale Anforderungen wie Sicherheitsanforderungen nur indirekt beschrieben werden können. Zur Überwindung dieses Nachteils wurden von Alexander et al. die Misuse-Cases [Ale03a] eingeführt, die einen Mechanismus zur direkten Beschreibung nicht-funktionaler Anforderungen bereitstellt (Abschn. 3.1).

Diese Use- und Misuse-Cases werden häufig informell durch eine natürlichsprachliche textuelle Repräsentation beschrieben [SO05]. Whittle et al. [WWH08] beschreiben als Erstes einen Ansatz zur Modellierung von Sicherheitsanforderungen als ausführbare Misuse-Cases und vermeidet hierbei informelle Notationen. Hierfür erweitern sie Extended Interaction Overview Diagrams (EIODs) [Whi07] um das Konzept der Misuse-Cases, die in die Basisspezifikation eingewoben werden. EIODs erweitern IODs der UML2 um zwei weitere übergeordnete Ebenen, die die Ausführungsreihenfolge der Interaktionsdiagramme festlegen. Die erste Ebene bildet das Use-Case-Diagramm (*engl. use case chart*), ein erweitertes Aktivitätsdiagramm, dessen Knoten Use-Cases sind. Diese Use-Cases beinhalten jeweils ein Szenariodiagramm (*engl. scenario chart*), dessen Notation ebenfalls auf Aktivitätsdiagrammen basiert und das als Knoten Szenarien enthält. Jeder Szenarioknoten wird durch ein Interaktionsdiagramm beschrieben. Hierbei wird die Syntax der Aktivitätsdiagramme wiederverwendet, jedoch die informelle auf Petrinetzen basierende Semantik durch eine formale Funktionensemantik, die auf Traces basiert, ersetzt. Für die Modellierung von Misuse-Cases stehen in den sogenannten Misuse-EOID die in Abschnitt 3.1 vorgestellten Misuse-Cases mit den Beziehungen «threatens» und «mitigates» zur Verfügung. Auch dieser Ansatz ist für die Modellierung der Ausführungsreihenfolge von Szenarien und deren Ausführung vorgesehen, zeigt zusätzlich, dass Sequenzdiagramme und die Erweiterung von Use-Case-Diagrammen um Misuse-Cases die Modellierung nicht-funktionaler Anforderungen ermöglicht.

3.3.4 *Eignung der Ansätze zur Signaturstrukturierung*

Die beiden vorgestellten Ansätze zur Strukturierung von Sequenzdiagrammen unterscheiden sich grundlegend. Während die Übersichtsdiagramme die Ausführungsreihenfolge der Sequenzdiagramme explizit auf einer höheren Abstraktionsebene beschreiben, strukturieren Anwendungsfallbeschreibungen die Sequenzdiagramme nach ihrer Funktionalität und beschreiben die Beziehungen zwischen den Sequenzdiagrammen nur implizit. Somit sind Übersichtsdiagramme näher als die Anwendungsfallmodellierung an der Implementierung des Systems bzw. der Monitore angesiedelt, da die Reihenfolge der Ausführung bzw. Überwachung direkt modelliert werden muss.

Die auf einer höheren Abstraktionsebene angesiedelte Anwendungsfallmodellierung kann somit früher im Entwicklungsprozess eingesetzt werden und es kann diese Spezifikation Schritt für Schritt verfeinert werden. Zudem ist die Beschreibung von nach außen sichtbarer Funktionalität der zu entwickelnden Systeme häufig Bestandteil der Anforderungsanalyse. Die aus diesem Ansatz des szenariobasierten Designs resultierenden Entwicklungsartefakte können so direkt in die Signaturbeschreibung einfließen.

Das Konzept der Use- und Misuse-Cases bildet im folgenden Kapitel die Grundlage für die MUC-Sprache der MBSecMonSL. Hierin dienen die „Cases“ als Behälter der Signaturen, die zur Beschreibung von erwartetem und verbotenem Verhalten des zu überwachenden Systems genutzt werden. Durch die Kapselung der einzelnen eLSC-Signaturen wird dem Entwickler die Modellierung der Beziehungen zwischen den einzelnen Signaturen vereinfacht, jedoch auf eine explizite Spezifikation der Ausführungsreihenfolge wie in den vorher erwähnten Ansätzen verzichtet.

Das nächste Kapitel untersucht LSCs auf ihre Eignung zur Modellierung verhaltensbeschreibender Signaturen. Darauf aufbauend wird die in dieser Arbeit entwickelte MBSecMon-Spezifikationssprache (MBSecMonSL) eingeführt, die aus einer angepassten Version der LSCs (erweiterte LSCs [PPS11]) und einer Variante der Anwendungsfallmodellierung basiert. Hierzu werden LSCs in ihrer Syntax zur kompakteren Modellierung der Signaturen erweitert und die Semantik für die Überwachung von Systemen und der Kommunikation zwischen ihnen zu erweiterten LSCs (eLSCs) angepasst.

4

LIVE SEQUENCE CHARTS UND IHRE EINSETZBARKEIT ALS VERHALTENSBESCHREIBENDE SIGNATURSPRACHE

Nachdem die Betrachtung existierender Signaturbeschreibungssprachen in Abschnitt 3.3 gezeigt hat, dass die LSC-Sprache potenziell zur Modellierung von Signaturen geeignet ist, wird in diesem Kapitel die Einsetzbarkeit der LSCs im MBSecMon-Entwicklungsprozess untersucht. Hierzu wird zunächst die Sprache der LSCs in Abschnitt 4.1 eingeführt. Um die Eignung der grafischen Notation der LSCs in einem modellbasierten Entwicklungsprozess zu evaluieren, werden in Abschnitt 4.2 Anforderungen an den MBSecMon-Entwicklungsprozess aufgestellt und gezeigt, wie diese Anforderungen erfüllt werden. Anschließend wird in Abschnitt 4.3 die Ausdrucksstärke der LSC-Sprache als verhaltensbeschreibende Signaturbeschreibungssprache untersucht und von ihr nicht unterstützte Konzepte identifiziert.

4.1 Live Sequence Charts

Live Sequence Charts (LSCs) [DH01] wurden 1998 von Damm und Harel eingeführt, um die schwerwiegenden Einschränkungen in der Ausdrucksstärke der MSCs [IT96] und somit auch der UML2-Sequenzdiagramme aufzuheben. Diese Einschränkungen [HK02] resultieren aus der ausdrucksschwachen Semantik der MSCs, die auf dem Konzept der schwachen partiellen Ordnung der Ereignisse basiert. So können MSCs nur die richtige Reihenfolge der Sende- und Empfangsereignisse beschreiben, sagen jedoch nichts über das Gesamtsystem, über das Verhalten beim Auftreten von nicht zutreffenden Bedingungen und über verbotene Szenarien aus. Dies verhindert den effektiven Einsatz der Sequenzdiagramme zur Beschreibung von Systemverhalten und somit zur Konkretisierung von Use-Cases, wie in Abschnitt 3.1 vorgestellt.

Im Gegensatz zu der UML2 beheben LSCs diese Einschränkungen nicht durch Einführung weiterer Operatoren (*Assert* und *Negate*), die nicht in die Semantik der Sprache passen, sondern durch die Erweiterung der MSCs mit dem Konzept der Modalitäten. Die in diesem Abschnitt verwendete Syntax und Semantik der LSCs basiert auf [HM03].

Definition 5 (*Modalität in LSCs*)**.** Der Begriff Modalität stammt aus der Modallogik und erweitert die Aussagenlogik um die Modalbegriffe *möglich* und *notwendig*. In LSCs werden Diagramme und ihre Modellierungselemente mit diesen Modali-

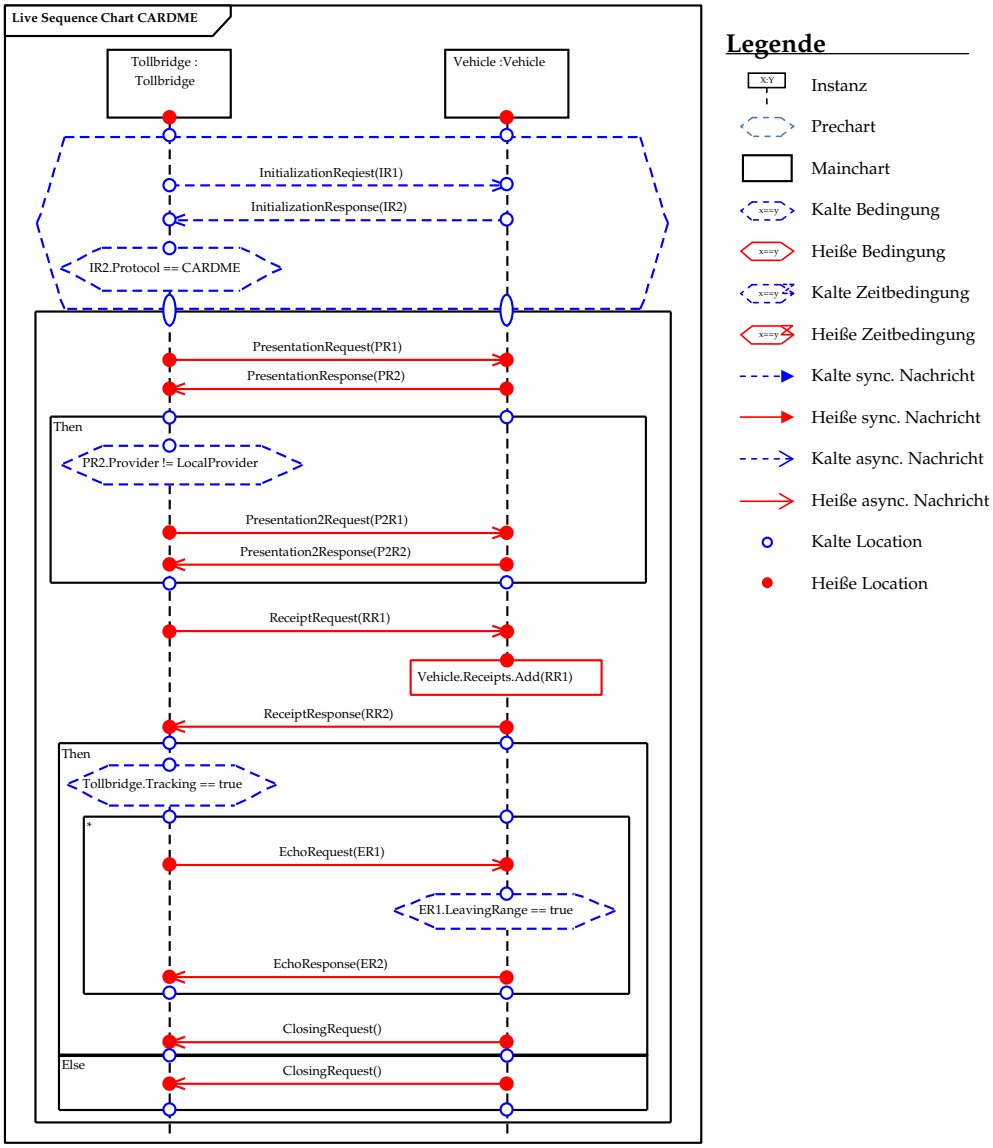


Abbildung 4.1: CARDME-Protokoll als LSC-Diagramm

täten annotiert, indem ihnen eine *Temperatur* zugeordnet wird. Hierbei steht *heiß* für notwendig und *kalt* für möglich.

Modalitäten (Temperaturen): Diese Modalitäten werden zur Definition von notwendigem und möglichem Verhalten auf globaler und auf lokaler Ebene eingesetzt. So können ganze LSCs global als notwendiges (Universal LSC) oder als mögliches (Existential LSC) Verhalten modelliert werden. Innerhalb dieser Diagramme können zudem lokal Ereignisse, Nachrichten und Bedingungen als notwendiges heißes (*engl. hot*) oder mögliches kaltes (*engl. cold*) Element modelliert werden. Durch die saubere semantische Einbindung der Temperaturen in die Semantik der LSCs können die Operatoren *Assert* und *Negate* der UML2 implizit

modelliert werden, ohne die Probleme der UML2-Operatoren (Abschn. 3.2.4) zu verursachen. In den MSDs wurden diese Operatoren aus der UML2 zwar übernommen, dienen jedoch nur als eine syntaktische Notation für die Modalitäten der enthaltenen Elemente.

Universal/Existential Charts: Basierend auf dem Konzept der Modalitäten beschreibt die Menge der Universal LSCs das szenariobasierende Verhalten, das für alle Systemabläufe zutreffen muss, während Existential Charts, wie die MSCs, einen Beispielablauf beschreiben, der mindestens einmalig in der Menge aller möglichen Systemausführungen auftreten muss. Damit Universal Charts auch Verhalten beschreiben können, das nur in bestimmten Zuständen des Systems auftritt, bieten sie die Möglichkeit, eine Vorbedingung durch ein Prechart zu modellieren. Ein Prechart wird als kaltes Element modelliert und durch ein gestricheltes, blaues Sechseck repräsentiert. Es legt fest, dass eine Abweichung des Inhalts zu keinem Fehlverhalten, sondern nur zur Nichtausführung des zugehörigen Maincharts führt. Wird das Prechart erfüllt, beschreibt das Mainchart, das als durchgezogenes Rechteck modelliert wird, das Verhalten des Systems, das zwingend eintreten muss. Eine Sammlung von Szenarien, die durch Universal Charts beschrieben werden, bildet eine Verhaltensbeschreibung des Systems, in der die Universal Charts durch die Precharts implizit verknüpft sind. Wenn das Mainchart eines Szenarios das Prechart eines anderen Universal Charts erfüllt, aktiviert es somit das weitere Verhalten.

Beispiel *Universal LSC*

Im Mautbrückenszenario, in Abbildung 4.1, werden zur Modellierung des Protokolls Universal Charts verwendet, die mit ihrem Prechart die Vorbedingung für die Ausführung des Maincharts beschreiben. Die im Mainchart beschriebene Kommunikation ist nur unter der Vorbedingung gültig, dass in der Initialisierungsphase das CARDME-Protokoll als mögliches Protokoll übermittelt wurde. Dies wird in LSCs als Prechart abgebildet und somit die Ausführung des Maincharts eingeschränkt.

Instanzen und Locations: Die Instanzen werden wie in den MSCs durch ein Symbol, bzw. durch ein Rechteck mit ausgehender vertikaler Linie (Lebenslinie) dargestellt. Auf ihr schreitet wie bei den MSCs die Zeit von oben nach unten fort. Auf den Lebenslinien werden durch kleine Kreise *Locations* definiert, die Ereignisse darstellen. Jede Lebenslinie hat in einem Existential Chart mindestens drei Locations, für den Start der Lebenslinie und je eine Location für den Beginn und das Ende des Maincharts. In Universal Charts haben alle Lebenslinien, die an dem Prechart teilnehmen, eine zusätzliche Location für den Beginn des Precharts. Das Ende des Precharts und der Anfang des Maincharts teilen sich jeweils eine Location je teilnehmender Lebenslinie und stellt einen Synchronisationspunkt der Ausführung dar. Erst wenn alle Ereignisse auf den Lebenslinien des Precharts beendet sind, findet die Ausführung des Maincharts statt. Diese Locations können ebenfalls mit den Modalitäten heiß bzw. kalt versehen werden. Heiß bedeutet, dass die Ausführung des Charts an dieser Stelle fortschreiten muss, kalt, dass an dieser Stelle angehalten und gewartet werden darf, ohne das Chart zu verletzen.

Fall	Syntax			Temperaturkonflikt	Semantische Interpretation		
	l_s	M	l_e		gesendet	empfangen	Kommentar
1	heiß	heiß	heiß	-	+	+	
2	kalt	heiß	heiß	l_s zu M und l_e	+	+	entspricht 1
3	kalt	kalt	heiß	l_s und M zu l_e	+	+	entspricht 1
4	heiß	kalt	heiß	M zu l_e	+	+	entspricht 1
5	kalt	kalt	kalt	-	?	?	
6	kalt	heiß	kalt	M zu l_s und l_e	?	?	entspricht 5
7	heiß	kalt	kalt	-	+	?	
8	heiß	heiß	kalt	M zu l_e	+	?	entspricht 7

+ (muss); ? (kann)

Tabelle 4.1: Konflikte in der Interpretation von asynchronen Nachrichten und ihren Locations (basierend auf [HM03])

Beispiel *Locations auf den Lebenslinien der Instanzen*

Das Universal LSC in Abbildung 4.1 besitzt zwei Instanzen, die `Tollbridge` und das `Vehicle`. Die Locations sind hier als Kreise auf den Schnittpunkten von Modellierungselementen mit den Lebenslinien der Instanzen eingezeichnet. Rote, gefüllte Punkte stehen für heiße Locations und blaue, leere Punkte für kalte Locations.

Nachrichten: Nachrichten werden ebenfalls mit den Modalitäten heiß (durchgezogene rote Linie) und kalt (gestrichelte blaue Linie) annotiert. Das Sende- und das Empfangsereignis werden jeweils durch eine Location auf der Instanzline gekennzeichnet. Eine heiße Nachricht muss empfangen werden, nachdem sie gesendet wurde (notwendig), kalte Nachrichten können gesendet, aber müssen nicht empfangen werden (möglich). Wie bei den MSCs existieren zwei Arten von Nachrichten: synchrone mit geschlossenem Pfeilkopf und asynchrone mit offenem Pfeilkopf. Synchrone Nachrichten werden verwendet, wenn, wie bei Methodenaufrufen, kein Kommunikationskanal vorliegt, der nach dem Senden das Empfangen verzögern oder verhindern kann. Im anderen Fall werden asynchrone Nachrichten zur Modellierung verwendet. Eigennachrichten, bei denen das Sende- und Empfangsereignis auf derselben Lebenslinie liegen, werden meist ebenfalls synchron modelliert. Da sich die Modalitäten der Locations und der Nachrichten, die mit ihnen verbunden sind, widersprechen können, ergeben sich acht verschiedene Kombinationen der Temperatur der Sende-Location (l_s), der Nachricht (M) und der Empfangs-Location (l_e).

Wie in [HM03] diskutiert, sind hierbei jedoch nicht alle Kombinationen sinnvoll bzw. teilweise von ihrer semantischen Bedeutung redundant. In Tabelle 4.1 werden die Ergebnisse der Diskussion basierend auf der Semantik der Locations und Nachrichten dargestellt. In den Fällen 2-4 verursacht die heiße Location l_e ein Fehlschlagen des LSCs, wenn die Nachricht nicht ankommt. In den Fällen 6 und 8 müsste ein aktives Ablehnen der empfangenden Instanz angenommen werden. Diese Konflikte werden in der Semantik gelöst, indem die Temperatur der Location der empfangenden Instanz l_e die der Nachricht überschreibt. Somit

dienen die Temperaturen der Nachrichten nur zur grafischen Hervorhebung der Kommunikationsfehler.

Beispiel *Nachrichten zwischen Instanzen*

In Abbildung 4.1 werden nur zwei Fälle (1 und 5) der in Tabelle 4.1 genannten Kombinationen zur Modellierung des Mautbrückenprotokolls verwendet. Start- und End-Location haben in diesen Fällen dieselbe Temperatur wie die Nachricht. Die Nachrichten selbst sind asynchron modelliert, da im Mautbrückenszenario ein Kommunikationskanal (Funkverbindung) verwendet wird. Im Prechart werden zwischen Mautbrücke und Fahrzeug zunächst Initialisierungsinformationen als kalte Nachrichten ausgetauscht. Bei diesem Informationsaustausch übermittelt das Fahrzeug über den Parameter `IR2` unter anderem die unterstützten Protokolle seiner OBE. In diesen LSCs werden im Prechart alle Elemente (Locations, Nachrichten, ...) kalt modelliert, da es sich um eine zu überwachende Vorbedingung handelt. Im Mainchart sind alle Nachrichten heiß und asynchron modelliert, da diese im Protokoll verpflichtend auftreten müssen.

Zuweisungen: Nicht alle Datenmanipulationen können als Nachrichtenaustausch zwischen Instanzen dargestellt werden. Hierfür bieten die LSCs Zuweisungen an – einfache Rechtecke, die immer als notwendig heiß markiert sind. Diese Zuweisungen können zur Speicherung von Eigenschaften der Instanzen, Werten, die mit Nachrichten als Parameter übermittelt werden oder Ergebnissen aus Funktionsaufrufen dienen. Die nach Auswertung der rechten Seite der Zuweisung ermittelten Werte können Eigenschaften der Instanz oder einer anderen vordefinierten Variable zugeordnet werden.

Beispiel *Zuweisungen*

Im Hauptszenario des CARDME-Protokolls in Abbildung 4.1 wird eine Zuweisung genutzt, um die von der Mautbrücke ausgestellte Quittung zu speichern. Die Mautbrücke sendet die Quittung in der `ReceiptRequest`-Nachricht als Parameter `RR1` an das Fahrzeug. Dieses speichert durch die Zuweisung die Quittung ab und sendet anschließend als `ReceiptResponse`-Nachricht eine Empfangsbestätigung.

Globale und Prädikatenbedingungen: Diese durch Zuweisungen auf Eigenschaften der Instanz oder vordefinierte Variablen gespeicherten Werte können verwendet werden, um zusätzlich zu den Precharts, auch innerhalb der Szenarien einen Kontrollfluss zu spezifizieren. Diese Bedingungen können, wie in Tabelle 4.2 dargestellt, ebenfalls mit den Modalitäten heiß oder kalt versehen werden und ändern somit die Ausführung des LSCs. Kalte Bedingungen werden als blaues gestricheltes Sechseck dargestellt und führen bei Nichteinhaltung der Bedingung zu einer erlaubten Beendigung der Ausführung des LSCs. Im Gegensatz hierzu sind heiße Bedingungen mit einem durchgängigen roten Rand dargestellt. Eine Verletzung dieser Bedingung führt zu einem fehlerhaften Abbruch des LSCs. Um die Auswertung der Bedingungen zu forcieren, kann die Bedingung über beliebig viele Lebenslinien platziert werden und so eine strikte Ordnung zwischen

	Prädikat (Subchart/If-Then-Else ≡ an erster Location) kalt	**Global** heiß	kalt
Bedingung	entsp. Glob. Bedingung	+ LSC bzw. Chart wird weiter überwacht – LSC schlägt fehl	+ LSC bzw. Chart wird weiter ausgeführt – LSC bzw. Chart wird abgebrochen
Zeitbedingung (minimal)	entsp. Glob. Zeitb.	+ LSC bzw. Chart wird weiter ausgeführt – wartet bis Bedingung erfüllt	+ LSC bzw. Chart weiter ausführen – LSC bzw. Chart wird abgebrochen
Zeitbedingung (maximal)	entsp. Glob. Zeitb.	+ LSC bzw. Chart wird weiter ausgeführt – LSC schlägt fehl	+ LSC bzw. Chart wird weiter ausgeführt – LSC bzw. Chart wird abgebrochen

Tabelle 4.2: Definition von Bedingungen in LSCs (+ erfüllt, – nicht erfüllt)

den Locations verschiedener Lebenslinien modelliert werden. Hierdurch findet eine Synchronisierung der an der Bedingung beteiligten Instanzen statt, da für ein Fortschreiten des LSCs erst alle Ereignisse (Locations) auf den Lebenslinien vor der Bedingung abgearbeitet werden müssen.

Beispiel *Globale und Prädikatenbedingungen*

Eine globale Bedingung wird im Prechart des CARDME-Protokolls eingesetzt, um zu überprüfen, ob im Parameter `IR2` der Nachricht, als mögliches Protokoll das CARDME-Protokoll übermittelt wurde (`IR2.Protocol == CARDME`). Da die Bedingung im Prechart modelliert ist, wurde sie als kalte Bedingung modelliert, die an einer kalten Location an die Instanz der `Tollbridge` eingehängt ist.

Anti-Szenarien: Basierend auf dem Konzept der Modalitäten, unterstützt die LSC-Sprache, im Gegensatz zu MSCs, die Spezifikation von Anti-Szenarien, die verbotenes Verhalten abbilden. Anti-Szenarien werden durch ein Universal Chart spezifiziert. Das Prechart beschreibt das verbotene Verhalten und das Mainchart besteht nur aus einer heißen Bedingung, die mit FALSE annotiert ist. Diese führt nach Tabelle 4.2 zum fehlerhaften Abbruch des LSCs, wenn das Anti-Szenario eingetreten ist.

Beispiel *Anti-Szenario*

Am Beispiel eines DoS-Angriffs auf die Mautbrücke beschreibt das LSC in Abbildung 4.2, welches Verhalten nicht erwünscht ist. Die Vorbedingung wird komplett im Prechart modelliert, da diese nur überwacht und nicht ausgeführt werden soll. Das Fehlschlagen der Anti-Szenarien wird durch eine einzelne heiße Bedingung mit der Annotation `FALSE` im Mainchart modelliert.

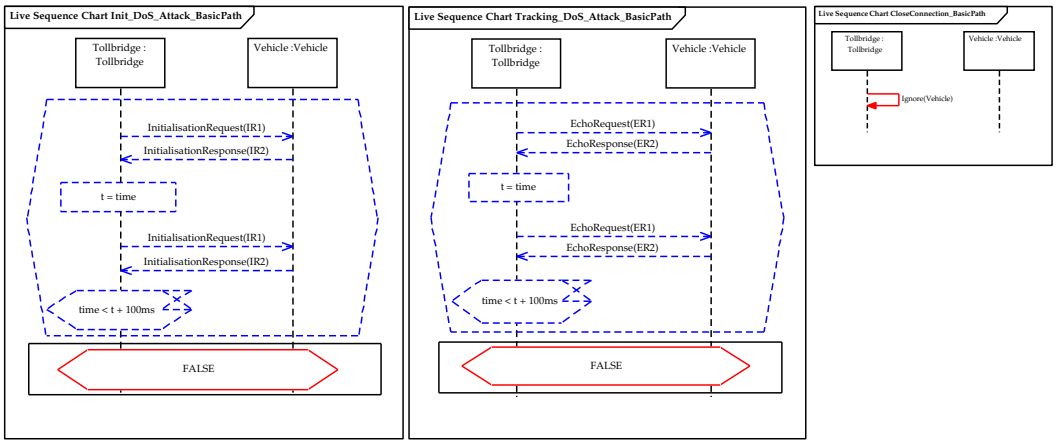


Abbildung 4.2: Anti-Szenarien des CARDME-Protokolls als LSC-Diagramm

Zeitbedingungen: Harel et al. [HM02] führt für die LSCs ein Zeitkonzept basierend auf einem diskreten System mit Zeitvariablen ein. Dieses besteht aus einem *Clock*-Objekt mit einer Variablen $\mathtt{Time}$ und der Möglichkeit, das Clock-Objekt als Instanz in ein LSC einzubinden. Hier kann dann die Methode *tick()* des Objekts verwendet werden, um einen diskreten Zeitschritt im LSC zu modellieren. Die Modellierung von Zeitbedingungen erfolgt durch die Zuweisungen ($\mathtt{Time\text{-}Variable} = \mathtt{Time}$) und die Bedingungen, die durch Ausnutzung der Modalitäten einen großen Bereich der Zeitbedingungen abdecken. Zeitbedingungen werden wie Bedingungen als Sechseck mit einem annotierten Sanduhrsymbol oben rechts dargestellt. Die Interpretation der Zeitbedingung für untere Zeitgrenzen ($\mathtt{Time} > \mathtt{Time\text{-}Variable} - \mathtt{Min\text{-}Delay}$) und für obere Zeitgrenzen ($\mathtt{Time} < \mathtt{Time\text{-}Variable} + \mathtt{Max\text{-}Delay}$) ist Tabelle 4.2 zu entnehmen. Auf diese Weise lassen sich vertikale Verzögerungen, Nachrichtenverzögerungen und Timer modellieren. Timer sind hierbei nicht wie in den MSCs auf eine Instanz festgelegt, sondern können auch zwischen verschiedenen Instanzen geteilt werden. Watchdogtimer können durch die später in diesem Abschnitt eingeführten *Forbidden Elements* modelliert werden, indem z. B. einem Subchart eine obere Zeitgrenze zugeordnet wird.

Beispiel *Zeitbedingungen*

Eine Zuweisung der Zeitvariablen `time` auf die Variable `t` wird im Anti-Szenario in Abbildung 4.2 im Prechart eingesetzt. Hierbei wird weder die Instanz *Clock* noch die Methode *tick()* explizit modelliert, sondern nur deren *Time*-Variable verwendet. Das modellierte Szenario stellt eine zu schnell wiederkehrende Initialisierungsphase dar. Nach der Initialisierungsphase wird der Zeitpunkt des Empfangs auf Mautbrückenseite zwischengespeichert, um eine spätere Auswertung zu ermöglichen. Nach Auftreten einer weiteren Initialisierungsphase wird eine obere Zeitgrenze abgeprüft. Wird diese Bedingung erfüllt, wird das Mainchart ausgeführt und das Anti-Szenario ist eingetreten.

Subcharts: Bisher wurden Bedingungen zur Steuerung des Kontrollflusses innerhalb von LSCs vorgestellt. Durch Subcharts, wohlgeformte Fragmente eines LSCs, die durch ein dickes schwarzes Rechteck dargestellt werden, findet eine Synchronisierung beim Eintritt und beim Austritt statt. Wie auch das Pre- und das Mainchart besitzt das Subchart auf jeder teilnehmenden Instanz zwei Locations.

If-Then-Else: Das If-Then-Else-Konstrukt nutzt zwei direkt untereinander angeordneten Subcharts, die den Then- und den Else-Fall darstellen. Hier teilen sich die beiden Subcharts beim Übergang zwischen Then- und Else-Subchart eine Location auf jeder beteiligten Instanz. Eine kalte Bedingung im ersten Subchart entscheidet, ob das Then- oder das Else-Subchart ausgeführt wird.

Beispiel *Alternativen durch If-Then-Else*

Der Then-Teil des If-Then-Else-Konstrukts wird im CARDME-Protokoll in Abbildung 4.1 eingesetzt, um die zweite Präsentationsphase nur auszuführen, wenn die kalte Prädikatenbedingung (`PR2.Provider != LocalProvider`) erfüllt ist. Der Else-Teil kann in diesem Fall als leeres Subchart betrachtet werden. Im Gegensatz hierzu wird für die Beschreibung der Verfolgungsphase das vollständige If-Then-Else-Konstrukt genutzt. Wenn `Tollbridge.Tracking == true` zu wahr evaluiert, wird die Verfolgung ausgeführt, andernfalls wird nur ein `CloseRequest` vom Fahrzeug versendet.

Die Subcharts und If-Then-Else-Konstrukte erweitern die Ausdrucksstärke der LSCs auf die der HMSCs. Abweichend zu aufeinanderfolgenden BMSCs, die in einem HMSC durch eine Bedingung verknüpft sind, müssen beim Eintreten in und Austreten aus einem Subchart alle teilnehmenden Instanzen synchronisiert werden.

Verbotene Elemente (*engl. Forbidden Elements*): Verbotene Elemente werden auf derselben Ebene wie Pre- und Mainchart als Subchart unter der Annotation `Forbidden Elements` modelliert. Jedes Subchart unter dieser Annotation enthält nur eine verbotene Nachricht (*engl. forbidden message*) oder eine verbotene Bedingung (*engl. forbidden condition*). Die Verwendung des *Forbidden*-Fragments führt zu einer direkteren und flexibleren Modellierung des nicht erwünschten Verhaltens. Anti-Szenarien beschreiben in ihrem Prechart die Bedingung bzw. Interaktionssequenz, die nicht eintreten darf, immer als ganzes global gültiges Universal LSC. Verbotene Elemente spezifizieren hingegen verbotenen Nachrichten bzw. Bedingungen, die während der Ausführung des Charts nicht auftreten bzw. eintreten dürfen. Durch die Zuordnung der Forbidden-Fragmente zu anderen Subcharts desselben LSCs kann ein Intervall während der Ausführung des LSCs definiert werden, in dem das modellierte Verhalten nicht auftreten darf. Das Forbidden-Fragment wird mit einer Temperatur versehen, die die Modalität des enthaltenen verbotenen Elements beschreibt. So brechen kalte Forbidden-Fragmente nur den Bereich des LSCs ab, dem sie zugeordnet sind, während heiße Forbidden-Fragmente bei ihrem Eintreten zu einem Fehlschlagen des gesamten LSCs führen. Als Modellierungsvarianten für *verbotene Nachrichten* schlagen Harel et al. [HM03]

- einzelne spezifische Nachrichten zwischen zwei Instanzen des LSCs (annotiert mit `Name der Nachricht`),
- alle Nachrichten zwischen zwei Instanzen des LSCs (annotiert mit $*$) und
- das Verbot aller nicht modellierten Nachrichten (`globale Nachricht`, die nicht an Lebenslinien gebunden ist, mit $*$)

vor. *Verbotene Bedingungen* dürfen in ihrem definierten Intervall zu keinem Zeitpunkt eintreten. So lassen sich auch Invarianten wie obere Zeitgrenzen bzw. Watchdogs durch kalte verbotene Elemente mit einer Zeitbedingung ausdrücken.

Schleifen: LSCs unterstützen verschiedene Varianten von Schleifen. Alle werden durch ein Subchart mit einer entsprechenden Iterationsbedingung oben links modelliert. Die beschränkte Schleife (*engl. fixed loop*) gibt durch eine Konstante oder durch eine vorher im LSC zugewiesenen Variable die Anzahl der Iterationen fest vor. Die unbeschränkte Schleife (*engl. unbounded loop*) ist mit einem $*$ als Iterationsanzahl markiert und kann beliebig oft ausgeführt werden. Als Abbruchbedingung kann wie bei allen Subcharts und wie im Speziellen beim If-Then-Else-Konstrukt eine kalte Bedingung genutzt werden, um den Abbruch der Schleife zu definieren. Durch die Positionierung dieser Bedingung oben oder unten in der Schleife lassen sich kopf- oder fußgesteuerte Schleifen realisieren.

Beispiel *Direkte Modellierung von Schleifen*

Im Hauptszenario des CARDME-Protokolls in Abbildung 4.1, wird eine unbeschränkte Schleife zur Modellierung der Verfolgungsphase des Fahrzeuges verwendet. Hier werden beliebig viele EchoRequest- und EchoResponse-Nachrichten ausgetauscht. Der Abbruch erfolgt durch die kalte Bedingung auf der Seite des Fahrzeugs, wenn die Mautbrücke feststellt, dass das Auto die Kommunikationsreichweite verlässt, und dies über den Parameter `ER1` dem Fahrzeug mitteilt.

Generische Abläufe: Zur Spezifikation generischer Abläufe bieten LSCs *symbolische Instanzen* zur kompakteren Modellierung an. In objektorientierten Sprachen beschreiben Klassen das gemeinsame Verhalten einer Menge von Objekten. Diese Klassen werden während der Designphase der Softwareentwicklung erstellt. In LSCs interessiert jedoch nicht die interne Beschreibung des Verhaltens, die Implementierung der Klassen, sondern die Kommunikation zwischen ihnen. Die Klassen werden in LSCs als Platzhalter für eine Menge von Objekten verwendet, die mit der entsprechenden Klasse verknüpft sind. Die Bindung der Objekte an die Symbolischen Instanzen findet in LSCs zum einen über einen Bindungsausdruck (*engl. binding expression*) statt und zum anderen über die Modalität der Symbolischen Instanz. Bindungsausdrücke definieren Bedingungen auf den Eigenschaften der Instanz, die entscheiden, ob ein Objekt der Symbolischen Instanz zugeordnet wird. Des Weiteren bindet eine kalte Symbolische Instanz zufällig ein Objekt, dass der Bindungsausdruck erfüllt, während eine heiße Instanz bewirkt, dass für jedes infrage kommende Objekt eine neue Instanz der Signatur initialisiert wird.

Ausführung mehrerer LSCs: Das Grundkonzept bei der Ausführung mehrerer LSCs ist die gegenseitige Aktivierung anderer oder desselben LSCs. So können Ereignisse, die durch die Ausführung der Maincharts entstehen, wiederum

Precharts vorantreiben und ein weiteres Mainchart, das dann ausgeführt werden kann, aktivieren.

Beispiel *Überlappende Charts/Rekursive Charts*

Ein Beispiel für die Erfüllung eines Precharts durch das eigene Mainchart des LSCs ist in dem Anti-Szenario in Abbildung 4.2 (`Tracking_DoS_Attack`) zu sehen. Das in Abbildung 4.1 dargestellte Protokoll führt, wenn das Tracking der Mautbrücke aktiviert ist, in einer Schleife eine Verfolgung des Fahrzeugs durch. Dies erfolgt, indem die Mautbrücke in regelmäßigen Abständen eine `EchoRequest`-Nachricht sendet und das Fahrzeug mit `EchoResponse` darauf antwortet. Das Prechart des Anti-Szenarios wird durch die Ausführung dieser Kommunikation vorangetrieben und der Zeitpunkt des Empfangs der `Response`-Nachricht wird durch eine Zuweisung auf die Variable `t` gespeichert. Ein weiteres Auftreten einer EchoRequest-Nachricht löst jetzt nicht nur die aktuelle Instanz des LSCs aus, sondern gleichzeitig eine weitere neu initialisierte Instanz des gleichen LSCs. Dies führt somit zu einer überlappenden Ausführung der Szenarien.

Für die Auswertung und Ausführung mehrerer Szenarien existieren in der Literatur zwei Varianten [HM03, Plo08]. Der Ausführungsalgorithmus geht hierbei nach der minimalen oder der maximalen Richtlinie vor, nach der die Ereignisse ausgewählt werden. Die minimale Richtlinie für Precharts sieht vor, so früh wie möglich eine Verletzung des Precharts zu erkennen. Dies reduziert die Anzahl der Bedingungen, die während der Ausführung einer Menge von LSCs auszuwerten sind. Im Play-Out-Ansatz von Harel wird jedoch die maximale Richtlinie als Standard für Precharts sowie Maincharts angewendet. Hierbei verhindert der Ausführungsalgorithmus so lange wie möglich eine Verletzung der Precharts, indem Ereignisse gewählt werden, die das Prechart wenn möglich vorantreiben. Erst wenn kein aktives Ereignis mehr existiert, das kein Prechart verletzt, wird ein verletzendes gewählt und des Prechart beendet. Da Precharts ausschließlich kalte Locations enthalten, setzt dies zwingend die Anwendung der maximalen Richtlinie für den Ausführungsalgorithmus voraus. Die Anwendung der minimalen Richtlinie könnte dazu führen, dass das Mainchart nicht erreichbar ist.

Neben diesen Modellierungselementen der LSCs existieren noch weitere Konstrukte, wie die Modellierung von Coregionen, erweiterte Bindungskonzepte für symbolische Instanzen, die für die Modellierung der in dieser Arbeit entwickelten verhaltensbeschreibenden Signatursprache nicht relevant sind und aus diesem Grund hier nicht genauer vorgestellt werden. Interessierte Leser finden ausführlichere Informationen zu LSCs in [HM03].

FAZIT Obwohl LSCs durch die Einführung der Modalitäten in ihrer Ausdrucksstärke stark gegenüber MSCs erweitert wurden, sind sie hauptsächlich zur Spezifikation von Interobjektkommunikation in frühen Entwicklungsphasen und zur Ausführung im PlayIn/PlayOut-Kontext entwickelt worden [HM03]. Die in Abschnitt 4.1 vorgestellte Semantik der LSCs basiert auf ihrer Ausführung durch einen Interpreter, um mögliche Kommunikationssequenzen aus ihnen abzuleiten. Für den Einsatz als verhaltensbeschreibende Signatursprache im MBSecMon-Ent-

wicklungsprozess muss die Syntax und Semantik zusätzliche Anforderungen an ihre Anwendbarkeit und Ausdrucksstärke erfüllen, die im Folgenden untersucht wird.

4.2 Anforderungen an den MBSecMon-Entwicklungsprozess

Die Beschreibung verhaltensbeschreibender Signaturen stellt sowohl Anforderungen an die Signatursprache selbst als auch an den gesamten modellbasierten Entwicklungsprozess, in dem sie als Spezifikationssprache eingesetzt wird. Wie in Abschnitt 3.3 vorgestellt, gibt es viele Varianten erwartetes Verhalten, verbotenes Verhalten und Angriffe auf ein System zu modellieren. Da die Signaturspezifikationssprache in einem modellbasierten Umfeld eingesetzt wird und auch für Nicht-Experten verständlich sein soll, bietet sich eine grafische konkrete Syntax an.

LSCs, als eine um Modalitäten erweiterte Variante der weitverbreiteten Sequenzdiagramme, erfüllen diese beiden abstrakten Kriterien und erlauben zusätzlich auch mögliches und verbotenes Verhalten zu spezifizieren. Im Folgenden wird gezeigt, dass LSCs im MBSecMon-Entwicklungsprozess viele der in [SBP08] definierten Anforderungen an einen Softwareentwicklungsprozess erfüllen und welche Erweiterungen der LSCs notwendig sind, um allen Anforderungen an eine verhaltensbeschreibende Signatursprache zu genügen. Angepasst an den MBSecMon-Prozess ergeben sich folgende Anforderungen:

(1) Einfache Erlernbarkeit Der Zugang zur Modellierungssprache muss einfach sein. Eine flache Lernkurve ist wünschenswert.

(2) Verständlichkeit Ein grundlegendes Verständnis des modellierten Systems muss auch für Nicht-Experten möglich sein.

(3) Vorhersehbarkeit Die modellierten Spezifikationen sollten eine klar definierte Semantik haben, die analysiert und simuliert werden kann, um nicht offensichtliche Eigenschaften aufzudecken.

(4) Effektiver Übergang zur Implementierung Der Übergang von den Modellen bis zum generierten Quelltext muss korrekt definiert und implementiert sein.

(5) Kosteneffektivität Die Erstellung eines Modells und Generierung des Systems muss günstiger sein, als das System in anderen geeigneten Arten (z. B. direkte Implementierung basierend auf der Spezifikation) zu implementieren.

(6) Ausdrucksstärke Die verwendete Sprache muss so ausdrucksstark sein, dass sie alle Schlüsselkonzepte der Domäne abbilden kann, ohne wichtige Details zu verlieren.

(1) Die meisten Diagrammtypen, die im MBSecMon-Prozess zur Spezifikation verwendet werden, sind Softwareentwicklern bekannt, da sie an die UML2 bzw. die MSCs angelehnt sind. Das Konzept zur Strukturierung der einzelnen Sequenzen basiert auf den UML2-Use-Cases, die um Misuse-Cases zur Modellierung von

verbotenem Verhalten und Angriffen erweitert wurde. UML2-Klassendiagramme bzw. Komponentendiagramme beschreiben die Struktur des Systems und die Schnittstellen, über die die Nachrichten ausgetauscht werden. Eine Hierarchie von Nachrichten wird ebenfalls durch Vererbung zwischen Klassen modelliert. LSCs – eine Erweiterung der MSCs – werden in einer erweiterten Form zur Beschreibung der Signaturen eingesetzt. Hierdurch bleiben die grafische Attraktivität und die intuitive Verwendung der MSCs erhalten [WT06]. Zur Beschreibung von Interaktionen in Systemen haben sich szenariobasierte Ansätze in der Praxis als geeignet erwiesen [WPJH98]. Sequenzdiagramme als Beschreibungssprache für diese Szenarien sind aufgrund ihrer konkreten Semantik selbst für Nicht-Experten vergleichsweise einfach erlernbar [BS05] und insbesondere LSCs eignen sich aufgrund ihrer Ausdrucksstärke zu deren Beschreibung [WT06]. Hierdurch kann der Entwickler mit der Modellierung sofort anzufangen und schrittweise sein Wissen über die speziellen Eigenschaften der Sprache erweitern.

(2) Im Gegensatz zu vielen existierenden speziellen Protokollbeschreibungssprachen auf Basis von Expertensystemen oder zustands-transitions-basierende Ansätze, bei denen die Lesbarkeit keine Grundanforderung ist [SES07, SL02], sind LSCs durch ihre konkrete Semantik und grafische konkrete Syntax für Nicht-Experten leichter verständlich [BS05].

(3) LSCs besitzen eine klar definierte Syntax und eine strenge formale Semantik [DH01]. Da im Rahmen dieser Arbeit die Syntax und Semantik der LSCs zur Modellierung von Signaturen angepasst werden muss, findet in Abschnitt 8.1 eine Definition der neu eingeführten Modellierungselemente und der Semantik statt. Die Definition der Semantik der eLSCs erfolgt durch die Übersetzung der eLSCs in die formal klar definierten MPNs [Pat14]. Im MBSecMon-Prozess wird die MPN-Repräsentation der Signaturen für die generische Quelltextgenerierung für verschiedene Zielplattformen genutzt. Zusätzlich kann die MPN-Repräsentation durch Simulation und Analyse zum Nachweis der Korrektheit der Spezifikation genutzt werden.

(4) Der MBSecMon-Entwicklungsprozess (Abb. 3.1) basiert auf dem Konzept der modellgetriebenen Softwareentwicklung (MDD), die von der OMG[1] als *Model Driven Architecture (MDA)* für die UML vorgeschlagen wurde. Die Transformation im Schritt *Transformation in Zwischensprache* zwischen den Spezifikationen in der MBSecMonSL und den MPNs findet automatisiert ohne Einwirkung des Modellierers statt. Zusatzinformationen, die die Schnittstellen des Monitors beeinflussen, werden aus vorhanden Entwicklungsartefakten extrahiert oder als Bibliotheken für die Zielplattform bzw. -sprache bereitgestellt.

(5) Die Kosten für die Entwicklung der Monitore werden durch die Wiederverwendung der szenariobasierten Spezifikationen aus frühen Entwicklungsphasen reduziert. Die als LSCs verfassten Anforderungen, die für die Entwickler bekannt und intuitiv sind, können im Laufe des Entwicklungsprozesses immer weiter verfeinert werden. So dienen sie gleichzeitig als Spezifikation für die Monitore und als Anforderungen für die Implementierung. Somit sind neben der hierdurch vorhandenen Beschreibung des Normalverhaltens, nur noch die zusätzliche Spezifi-

[1] Object Management Group: http://www.omg.org

kation des Fehlverhaltens und der Angriffe auf das System als Misuse-Cases zu modellieren.

(6) Die Modellierungssprache und somit die LSCs müssen alle notwendigen Konstrukte für die verhaltensbeschreibende Signaturmodellierung unterstützen. Anhand einer Evaluation verschiedener Protokollbeschreibungssprachen haben Meier und Schmerl [Mei04, Sch04] Anforderungen aufgestellt, die von einer allgemeinen Signaturbeschreibungssprache für Intrusion Detection Systeme (IDS) basierend auf Petrinetzen erfüllt werden muss. Hierbei hat er Ereignismuster basierend auf unterschiedlichen Aspekten des semantischen Modells kategorisiert, die in Tabelle 4.3 aufgelistet sind. Im folgenden Abschnitt wird gezeigt, dass LSC diese Anforderungen zum Großteil erfüllen, jedoch Erweiterungen zur Steigerung der Ausdrucksstärke und zur kompakteren Modellierung notwendig sind.

Neben diesen Anforderungen muss die Sprache die Modellierung von Verpflichtungen (*engl. obligations*), Erlaubnissen (*engl. permissions*) und Verboten (*engl. prohibitions*) (Def. 4) unterstützen. Solhaug et al. [SES07] hat gezeigt, dass deontische Modalitäten durch LSCs ausgedrückt werden können. Obligationen können als Use-Cases beschrieben durch Universal Charts (ohne Prechart), Erlaubnisse als Use-Cases beschrieben durch Universal Charts (mit Prechart) und Verbote durch Misuse-Cases beschrieben durch Universal Charts (mit Prechart) modelliert werden.

4.3 LSCs als verhaltensbeschreibende Signatursprache

In der Anforderungsphase des MBSecMon-Entwicklungsprozesses werden LSCs in Kombination mit strukturellen Spezifikationen genutzt. Hierfür werden UML2-Klassendiagramme bzw. Komponentendiagramme zur Beschreibung der Struktur des Systems, der Teilnehmer und ihrer Beziehungen eingesetzt. Beziehungen zwischen einzelnen Signaturen als LSCs werden durch UML-Use-Cases, die um das Konzept der Misuse-Cases erweitert sind, modelliert. Sie deklarieren, ob die Signatur ein erwartetes Verhalten (Use-Case) oder ein fehlerhaftes Verhalten bzw. einen Angriff auf das System (Misuse-Case) beschreiben.

Als Spezifikationssprache, die zur detaillierten Modellierung von Szenarien in Use- und Misuse-Cases eingesetzt wird, sollen im MBSecMon-Prozess die ausdrucksstarken LSCs genutzt werden. Basierend auf den ermittelten Anforderungen an eine Signaturbeschreibungssprache, die in Abschnitt 4.2 aufgestellt wurden, wird im Folgenden gezeigt, dass LSCs diese zum Großteil erfüllen. Die in diesem Abschnitt identifizierten fehlenden Eigenschaften und fehlende Ausdrucksstärke können durch wenige Erweiterungen, die in die eLSC-Sprache einfließen, behoben werden. Diese erweiterten LSCs (eLSCs) bilden die Grundlage der MBSecMonSL.

Im Folgenden wird in diesem Abschnitt und im Kapitel 5 zur Vorstellung der Anforderungen nicht die Kommunikation zwischen Mautbrücke und Fahrzeug, sondern exemplarisch die interne Kommunikation im Fahrzeug über den CAN-Bus verwendet. An dieser Kommunikation sind nicht nur zwei, sondern drei verschiedene Komponenten beteiligt, die die Modellierung von komplexeren Signa-

Typ und Reihenfolge	
Sequenzen	Mehrere Ereignisse mit einer strikten sequenziellen Ordnung.
Konjunktionen	Mehrere Ereignisse, die in beliebiger Reihenfolge auftreten können.
Negationen	Ein Ereignis, das nicht auftreten darf.
Disjunktionen	Genau ein Ereignis von mehreren möglichen darf auftreten.
Simultan	Zwei Ereignisse treten zur gleichen Zeit auf.
Wiederholungen	
Genau	Ein Muster muss genau n-Mal auftreten.
Mindestens	Ein Muster muss mindestens n-Mal auftreten.
Höchstens	Ein Muster darf höchstens n-Mal auftreten.
Kontinuität	
Kontinuierlich	Jede auftretende Ereignisinstanz muss explizit in dem Ereignismuster modelliert sein.
Nicht-kontinuierlich	Das Ereignismuster ist erfüllt, wenn alle modellierten Ereignisse in der richtigen Reihenfolge aufgetreten sind. Alle zusätzlichen, nicht modellierten Ereignisse werden ignoriert.
Nebenläufigkeit	
Nicht-überlappend	Zwei oder mehr Ereignismuster müssen sequenziell nacheinander auftreten.
Überlappend	Die Ereignisse von zwei oder mehr Ereignismustern dürfen verschränkt zueinander auftreten. Allerdings muss im Gegensatz zur Konjunktion vorherige Ereignismuster vor dem nachfolgenden Ereignismuster abgeschlossen sein.
Kontextbedingungen	
Intra-Step-Bedingungen	Ein einfacher boolescher Ausdruck, der basierend auf Merkmalen der auslösenden Ereignisinstanz ausgewertet wird.
Inter-Step-Bedingungen	Eine komplexe Bedingung, die auf Merkmalen mehrerer Ereignisinstanzen ausgewertet wird.
Schritt-Instanz-Selektion	
Erste	Die erste auftretende Ereignisinstanz eines Ereignisses wird selektiert.
Letzte	Die letzte auftretende Ereignisinstanz eines Ereignisses wird selektiert.
Alle	Alle auftretenden Ereignisinstanzen werden an ein Ereignis gebunden.
Schritt-Instanz-Konsum	
Konsumierend	Die auftretenden Ereignisinstanzen werden durch ihre erste Übereinstimmung in einem Ereignismuster konsumiert und stehen für weitere Zuordnungen nicht zur Verfügung.
Nicht-konsumierend	Die auftretenden Ereignisinstanzen werden mehrfach für die Zuordnung in einem einzelnen Ereignismuster verwendet.

Tabelle 4.3: Anforderungen an eine verhaltensbeschreibende Signaturbeschreibungssprache (basierend auf [Sch04, Mei04])

turen erlaubt. Hierdurch lassen sich die Anforderungen an die Signaturbeschreibungssprache besser und klarer visualisieren.

Beispiel *Car2X-CAN-Bus*

Das Car2X-CAN-Bus-Szenario wird in Abbildung 4.3 dargestellt. Koscher et al. [KCR+10] hat gezeigt, dass Sicherheitsgefahren in modernen Fahrzeugen existieren und diese somit angreifbar sind. Diese Angreifbarkeit resultiert aus den Schwächen des CAN-Busprotokolls. Die Pakete enthalten weder ein Authentifizierungsfeld noch einen Identifikator für die Quelle des Pakets. Der CAN-ID-Header enthält nur Informationen über den Pakettyp. Diese Pakete werden über einen Broadcast zu jedem Knoten im Netzwerk des Fahrzeugs weitergeleitet und das Steuergerät entscheidet selbst, ob das Paket für es relevant ist. Ein kompromittiertes Steuergerät ist ausreichend, um Nachrichten auf dem CAN-Bus einzuschleusen und so die Kontrolle über andere Steuergeräte zu erhalten. Dies erlaubt u. a. fehlerhafte Informationen auf den an dem Bus angeschlossenen Anzeigeinstrumenten (*engl. Driver Information Center (DIC)*) anzuzeigen. In diesem Beispiel resultieren diese fehlerhaften Nachrichten aus der Kommunikation zwischen zwei Steuergeräten, die drahtlos mit *Road Site Units* (RSUs) außerhalb des Fahrzeugs kommunizieren. Das *Tollbridge Module* (TBM) dient zur Gebührenerfassung und das *Enterprise Module* (EM) kommuniziert mit Enterprise RSUs, die weitere Services anbieten.

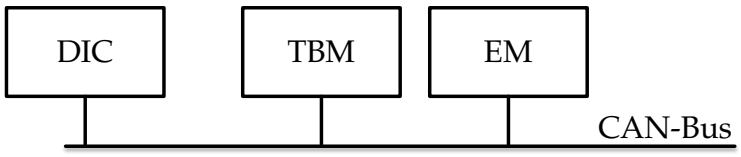
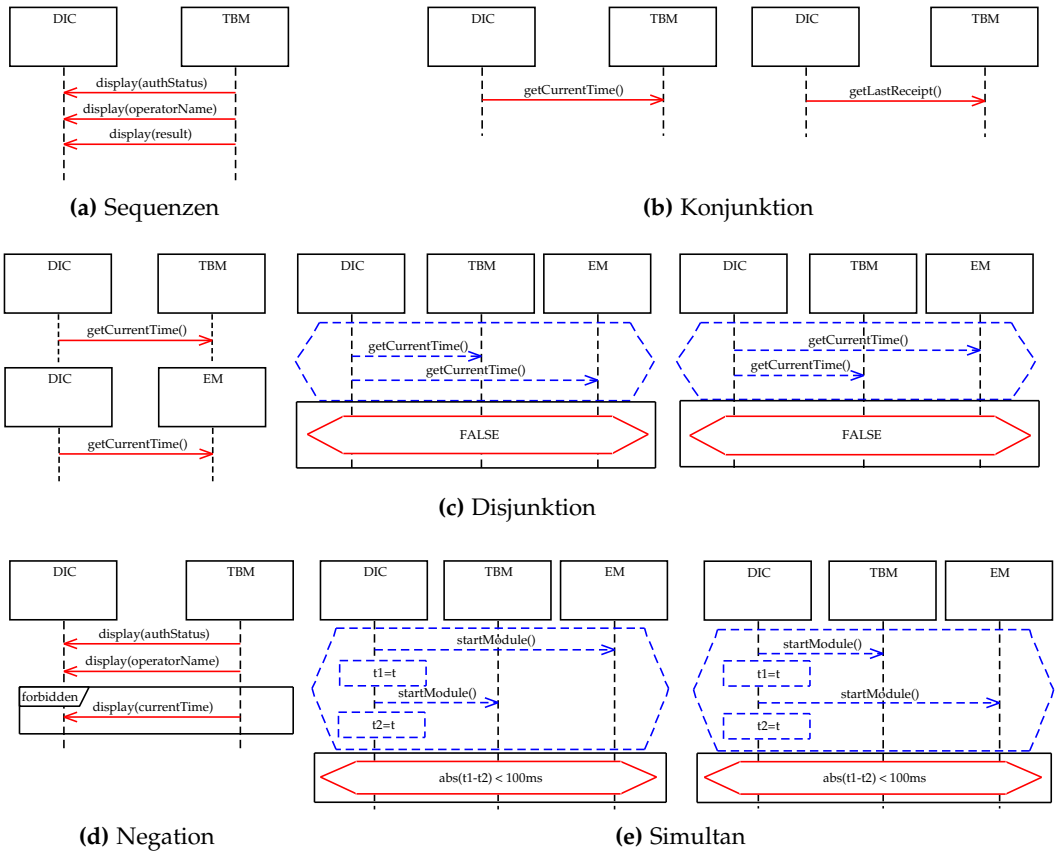


Abbildung 4.3: Car2X-Beispielszenario – Systemarchitektur

Um die Unterstützung der Anforderungen an eine Signatursprache aus Tabelle 4.3 durch die LSCs zu evaluieren, werden zur einfacheren Darstellung im Folgenden Universal Charts ohne Prechart verwendet. Ausschließlich für die Spezifikation von Anti-Szenarien werden Precharts eingesetzt, da Anti-Szenarien in der LSC-Sprache, wie in Abschnitt 4.1 vorgestellt, verbotene Abläufe durch ihre Precharts beschreiben. Die Beispiele in den Abbildungen 4.4 und 4.5 verwenden zur einfacheren Verständlichkeit größtenteils einzelne asynchrone Nachrichten, anstelle derer jedoch auch komplexe Muster stehen können.

Die ersten fünf Anforderungen aus Tabelle 4.3 an eine verhaltensbeschreibende Signatursprache bilden Muster, die den Typ und die Reihenfolge von Ereignissen beschreiben. Diese Ereignisse sind in LSCs z. B. das Senden und Empfangen von Nachrichten. Die Abbildungen 4.4a bis e stellen beispielhaft Spezifikationen zur Beschreibung von Sequenzen, Konjunktionen, Negationen, Disjunktionen und simultanem Auftreten von Ereignissen durch LSCs dar. Zusätzlich wird im Folgenden auf Probleme, die bei der Spezifikation dieser Muster in der LSC-Sprache für den Modellierer auftreten, hingewiesen.



(a) Sequenzen **(b)** Konjunktion

(c) Disjunktion

(d) Negation **(e)** Simultan

Abbildung 4.4: Muster zur Signaturbeschreibung als LSCs (Typ und Reihenfolge)

Sequenzen: Die Semantik der LSCs unterstützt die geforderte Spezifikationsmöglichkeit von *Ereignissequenzen*. Die `display`-Nachrichten an das `DIC` in Abbildung 4.4a besitzen eine partielle Ordnung auf jeder Lebenslinie und zwischen den Ereignissen des Sendens und Empfangens einer Nachricht. Das `TBM` sendet zuerst den Status der Authentifizierung (`authStatus`) an das `DIC`, gefolgt vom Operatornamen der Mautbrücke (`operatorName`) und dem Ergebnis der Aushandlung zwischen `TBM` und Mautbrücke (`result`). Die Mautbrücke ist hier nicht explizit modelliert.

Konjunktion: Eine *Konjunktion* von Ereignissen kann in der LSC-Sprache wie in Abbildung 4.4b als zwei parallel zueinander überwachte LSCs modelliert werden. Diese zwei Nachrichten können in beliebiger Reihenfolge vom `DIC` an das `TBM` gesendet werden. Hierfür müssen die beiden parallel zu überwachenden LSCs jedoch Nachrichten des jeweiligen anderen LSCs ignorieren, wenn diese Nachrichten nicht explizit im LSC spezifiziert sind. Sind komplexere Ereignisstrukturen von mehr als einer Nachricht je Diagramm modelliert, muss die partielle Ordnung innerhalb der einzelnen LSCs eingehalten werden.

Problem: Die Modellierung paralleler Ereignissequenzen als einzelne LSCs führt zu unübersichtlichen und redundanten Spezifikationen. Während bei vollständig unabhängigen Signaturen, die parallel überwacht werden sollen, die vorgestellte

Variante eingesetzt werden kann, führt sie bei Signaturen, die sich nur in einem Teil der Ereignissequenz unterscheiden zu einer Duplizierung des Großteils der Signatur.

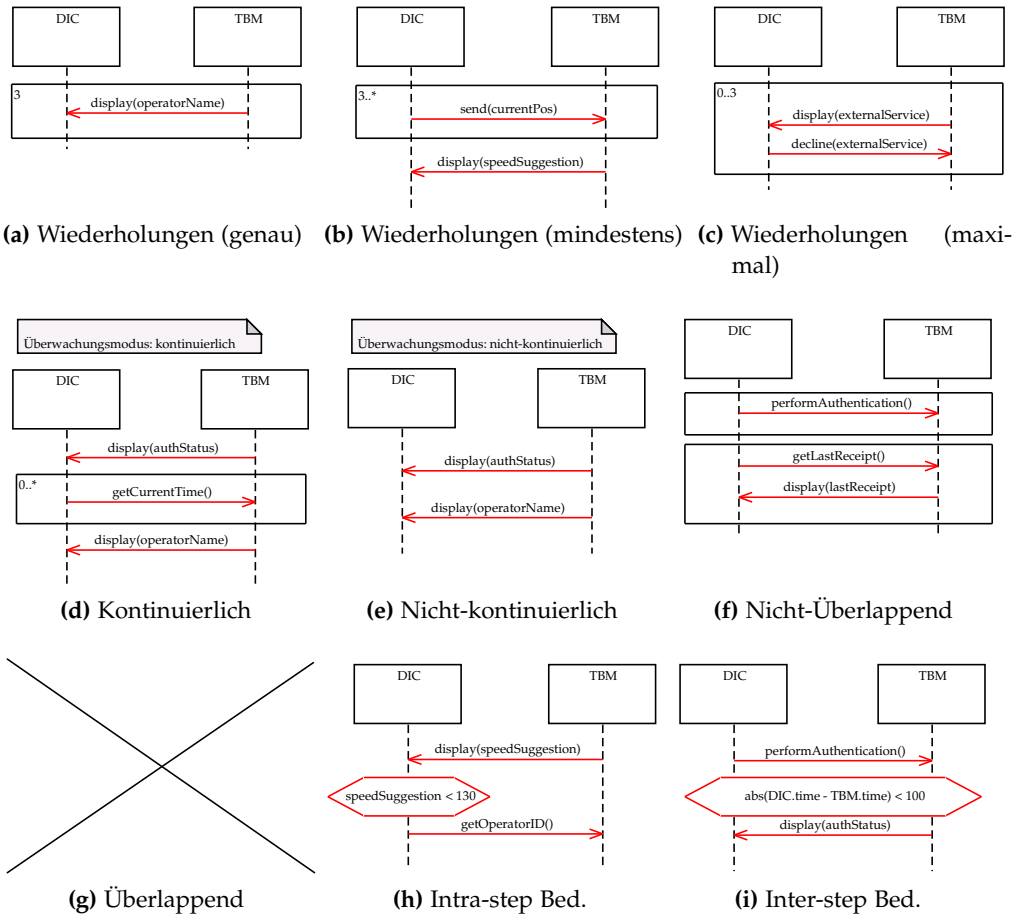
Negation: Die Modellierung einer *Negation* erfolgt durch das Forbidden-Fragment der LSCs. Wie in Abbildung 4.4d dargestellt, kann das Forbidden-Fragment eingesetzt werden, um in einer Sequenz von Ereignissen, bestimmte Ereignisse zu verbieten. Das LSC-Diagramm beschreibt den Fall, dass zwischen den `display`-Nachrichten für `authStatus` und `operatorName` keine zusätzliche Zeitaktualisierung (`currentTime`) auf dem `DIC` stattfinden darf. Zusätzlich zu dem globalen Einsatz in diesem Beispiel erlauben LSCs die Gültigkeit des Forbidden-Fragments auf Subcharts sowie auf Pre- und Maincharts einzuschränken, indem sie ihnen zugeordnet werden.

Disjunktion: Eine *Disjunktion*, die nicht durch eine Bedingung eines If-Then-Else-Fragments formulierbar ist, kann nicht explizit in einem LSC dargestellt werden. Ein Beispiel hierfür ist, dass das `DIC` nur von einer `OBE`, dem `TBM` oder dem `EM`, die aktuelle Zeit anfragen darf. Die Modellierung kann in der LSC-Sprache in diesem Beispiel nur umständlich, wie in Abbildung 4.4c dargestellt, über zwei zueinander parallel überwachte LSCs und zwei Anti-Szenarien realisiert werden. Jede `getCurrentTime`-Nachricht zu den verschiedenen OBEs wird in einem einzelnen LSC beschrieben und die zwei Anti-Szenarien sichern für beide möglichen Reihenfolgen ab, dass nicht beide Nachrichten nacheinander auftreten. Alternativ können durch den Einsatz einer Symbolischen Instanz für die `OBE` die zwei parallel überwachten LSCs zu einer Signatur vereinigt werden.

Problem: Dies ändert jedoch nichts daran, dass diese Art der Modellierung für den Modellierer der Signatur eine große Herausforderung darstellt – insbesondere bei komplexeren Zusammenhängen und umfangreichen Spezifikationen.

Simultan: Zur Beschreibung des *simultanen* Auftretens mehrerer Sendeereignisse bieten LSCs im Gegensatz zu MSCs ebenfalls keinen expliziten Modellierungsmechanismus, da in LSCs im PlayIn/PlayOut-Konzept immer nur ein Ereignis ausgelöst werden kann. Für die Spezifikation eines Monitors ist es jedoch häufig nicht sinnvoll allgemein von simultanem Auftreten zu sprechen, da dies eine genauere Definition von Gleichzeitigkeit benötigt. Da z. B. der CAN-Bus als Kommunikationsmedium immer nur eine Nachricht auf einmal übertragen kann, ist die Definition eines Zeitintervalls sinnvoll. Dieses Zeitintervall zwischen zwei Nachrichten kann wie in Abbildung 4.4e durch die Nutzung von Zuweisungen der aktuellen Zeit auf Variablen und Zeitbedingung überwacht werden. Nach jedem Senden einer der beiden Nachrichten durch das `DIC`, wird die Zeit des Ereignisses durch eine Zuweisung der aktuellen Zeit `t` auf eine Zeitvariable (`t1` bzw. `t2`) gespeichert und nach Auftreten beider Nachrichten der Betrag der Differenz auf kleiner als `100 ms` geprüft. Da jedoch die Reihenfolge der beiden Nachrichten nicht feststeht, müssen auch hier alle Alternativen explizit modelliert werden.

Problem: Hierdurch treten ähnliche Probleme wie bei der Modellierung einer Konjunktion auf. In diesem Fall müssen beide Alternativen des Auftretens der beiden Nachrichten als Prechart der jeweiligen LSCs modelliert werden. Bei komplexe-



(a) Wiederholungen (genau) **(b)** Wiederholungen (mindestens) **(c)** Wiederholungen (maximal)

(d) Kontinuierlich **(e)** Nicht-kontinuierlich **(f)** Nicht-Überlappend

(g) Überlappend **(h)** Intra-step Bed. **(i)** Inter-step Bed.

Abbildung 4.5: Muster zur Signaturbeschreibung als LSCs (Kontrollfluss)

ren Signaturen mit mehr als zwei Nachrichten in der Vorbedingung führt dies zu unübersichtlichen Signaturen und die Modellierung wird hierdurch fehleranfällig.

Neben diesen den Typ und die Reihenfolge beschreibenden Mustern muss eine verhaltensbeschreibende Signatursprache verschiedene Arten der Iterationen beschreiben können.

Wiederholungen: Die drei Iterationsarten aus Tabelle 4.3 mit den Eigenschaften *genau*, *mindestens* und *maximal* sind als LSCs in Abbildung 4.5a bis c dargestellt. Zu ihrer Modellierung wird der vorhandene Loop-Operator der LSCs genutzt und mit den Annotationen n (genau), $n..^*$ (mindestens), und $n..m$ (maximal), die untere, obere Grenzen der Anzahl der Iterationen festgelegt werden.

Für eine Signatursprache ist es zudem wichtig festlegen zu können, wie die Signaturen interpretiert werden sollen. Ein Aspekt der Interpretation ist die Art der Mustererkennung, die auf die modellierten Signaturen angewendet wird. Dies kann, wie in Tabelle 4.3 gefordert, *kontinuierlich* oder *nicht-kontinuierlich* stattfinden.

Kontinuierlich: Im *kontinuierlichen Fall* müssen alle Ereignisse (Nachrichten), die während der Überwachung im System auftreten, explizit modelliert werden.

Da LSCs zur Ausführung bzw. Ermittlung möglicher Kommunikationsabläufe eingesetzt werden, findet sowohl die Überwachung des Precharts als auch die Ausführung des Maincharts kontinuierlich statt. Zur Überwachung einer Signatur müssen in diesem kontinuierlichen Modus alle Nachrichten, die auftreten können wie in Abbildung 4.5d explizit modelliert werden. Zwischen den zwei `display`-Nachrichten an das `DIC` kann jederzeit in der Kommunikation eine oder mehrere `getCurrentTime`-Anfragen an das `TBM` geschickt werden. Dies kann in LSCs durch eine zusätzliche unbeschränkte Schleife mit der entsprechenden Nachricht zwischen den `display`-Nachrichten modelliert werden.

Problem: Bei diesem Beispiel handelt es sich um einen einfachen Fall, in dem nur eine Nachricht mehrfach auftreten darf. Werden Signaturen für komplexere Systeme modelliert, führt eine solche explizite Darstellung zu unübersichtlichen großen Signaturen, die nicht wartbar sind.

Nicht-Kontinuierlich: Erst durch eine zusätzliche explizite Einführung der beiden Überwachungsmodi (kontinuierlich und nicht-kontinuierlich) in der Semantik der LSCs kann der *nicht-kontinuierliche Fall* auch durch LSCs, wie in Abbildung 4.5e dargestellt, modelliert werden. Die LSCs, die nicht-kontinuierlich überwacht werden sollen, werden mit einer Annotation – hier als Notizelement realisiert – versehen.

Problem: Diese zusätzliche Unterscheidung zwischen den zwei verschiedenen Semantiken erschweren jedoch wiederum die Modellierung der Signaturen durch Softwareentwickler und das Verständnis durch Nicht-Experten, da für ein und dieselbe Notation zwei Semantiken der LSC-Diagramme betrachtet werden müssen.

Ein weiterer Aspekt, den [Mei04] als essenziell herausstellt, ist das verschränkte nebenläufige (*überlappende*) oder sequenzielle (*nicht-überlappende*) Auftreten mehrerer Ereignissequenzen.

Nicht-Überlappend: *Nicht-Überlappende* Ereignissequenzen können in LSC-Diagrammen als zwei oder mehrere aufeinanderfolgende Subcharts dargestellt werden. Das LSC in Abbildung 4.5f beschreibt, dass zunächst über das `TBM` eine Authentifizierung stattfinden muss, bevor die letzte Quittung (`getLastReceipt`) abgefragt werden darf.

Überlappend: Im überlappenden Modus werden Ereignissequenzen wie bei einer *Konjunktion* nebenläufig verschachtelt interpretiert, allerdings wird zusätzlich gefordert, dass das abschließende Ereignis der letzten Ereignissequenz als Letztes auftritt. Die geforderte Variante der überlappenden Überwachung von Ereignissequenzen lässt sich durch LSCs nicht modellieren, da durch die Modellierung parallel überwachter LSCs wie bei der Konjunktion der eindeutige Abschluss nicht spezifiziert werden kann. Dieser eindeutige Abschluss der Signaturen ist für die Überwachung von Systemen durch Monitore aus LSCs nicht notwendig, da die Erkennung einer Signatur durch die Abarbeitung aller Ereignisse der Signatur gegeben wird.

Problem: Sollte trotz allem der Abschluss der Ereignissequenz durch ein einzelnes Ereignis notwendig sein, kann diese Einschränkung der LSCs durch eine kontroll-

flussbeschreibende Sprache wie die Interaktionsübersichtsdiagramme der UML2 mit ihren Parallelisierungsknoten und Synchronisierungsknoten gelöst werden. Hierfür wird die abschließende Nachricht in ein separates LSC ausgelagert und nach erfolgreicher paralleler Überwachung beider Teilsignaturen ohne diese abschließende Nachricht, diese abschließende Nachricht überwacht. Hierdurch wird in der Spezifikationssprache jedoch eine komplexe zusätzliche Kontrollflusssprache eingeführt, die eine Monitorgenerierung erschwert.

Zusätzlich wird in Tabelle 4.3 als ein essenzielles Konzept einer Signatursprache gefordert, dass zwei verschiedene Varianten der Bedingungen unterstützt werden – die *Intra-Step-Bedingung* und die *Inter-Step-Bedingung*.

Intra-Step-Bedingung: Die *Intra-Step-Bedingungen* beschreiben Bedingungen, die nur auf dem zuletzt aufgetretenen Ereignis basieren, und lassen sich durch eine Bedingung der LSCs, die nur einer Instanz zugeordnet ist, spezifizieren. Die in Abbildung 4.5h dargestellte Intra-Step-Bedingung, basiert nur auf dem durch das letzte Empfangsereignis übermittelten Wert der aktuellen Geschwindigkeitsempfehlung (`speedSugestion`) und einer Konstanten (`130`), die mit dem Kleiner-Operator verglichen werden.

Inter-Step-Bedingung: Eine *Inter-Step-Bedingungen* greift auf mehr Informationen als das zuletzt aufgetretene Ereignis zu. Die in Abbildung 4.5i modellierte Inter-Step-Bedingung greift auf die Zeiten beider Instanzen (`DIC.time` und `TBM.time`) zu und vergleicht deren Abweichung mit einer Obergrenze von `100ms`.

Die letzten zwei Anforderungen, die verhaltensbeschreibende Signatursprachen erfüllen müssen, sind zum einen die Definition komplexer Ereignisse (Schritt-Instanz-Selektion) und zum anderen, ob ein Ereignis in einer Signatur mehrfach als Übereinstimmung gewertet werden darf (Schritt-Instanz-Konsum).

Schritt-Instanz-Selektion: Die *Schritt-Instanz-Selektion* regelt, wie auftretende Ereignisinstanzen an die Ereignisse einer Signatur (Ereignissequenzen) gebunden werden. Hier schlägt Meier für die Misuse-Case-Erkennung vor drei Fälle zu unterscheiden – *Erste*, *Letzte* und *Alle*. Diese verschiedenen Selektionsarten lassen sich explizit in den LSCs durch Modellierung mit einzelnen Nachrichten und Loop-Fragmenten realisieren. Zur Beschreibung einer Signatur, die *alle* Ereignisinstanzen eines Ereignistyps bindet, kann ein Loop-Fragment, wie in Abbildung 4.5a und c abgebildet, eingesetzt werden. Ein einzelnes Ereignis (senden einer Nachricht) kann wie in Abbildung 4.4a genutzt werden, um die *erste* auftretende Ereignisinstanz zu binden. Um das *letzte* Ereignis (letzte Nachricht) auszuwerten, kann ein Loop-Fragment mit einer nach dem Loop-Fragment folgenden Nachricht eingesetzt werden.

Schritt-Instanz-Konsum: Basierend auf der LSC-Syntax, die durch Harel et al. [HM02] definiert ist, wird bei der Signaturmodellierung mit LSCs ein *konsumierender* Schritt-Instanz-Konsum angenommen. Innerhalb einer Signaturinstanz werden somit Ereignisinstanzen konsumiert und stehen somit nicht mehr für weitere Bindungen in dieser Signaturinstanz zur Verfügung.

FAZIT Die Evaluation der LSCs als verhaltensbeschreibende Signatursprache hat gezeigt, dass, obwohl die LSCs sehr ausdrucksstark sind, viele Anforderungen nur umständlich, mit hohem Modellierungsaufwand spezifizierbar sind. Für die überlappende Überwachung, die mit einem eindeutigen Ereignis (Nachricht) abgeschlossen werden muss, ist die Modellierung mit LSCs ohne Erweiterung der Sprache nicht möglich. Somit ergeben sich folgende Konzepte, um die die LSC-Sprache erweitert werden muss:

K1 Die Modellierung paralleler verschränkter Ereignissequenzen muss auch für komplexe Ereignissequenzen möglich sein, ohne große Teile der Signatur duplizieren zu müssen. (basierend auf Anforderungen: Konjunktion, Simultan, Überlappend).

K2 Disjunktionen müssen in der Signatursprache direkt formulierbar und nicht nur über Anti-Szenarien beschreibbar sein. (basierend auf Anforderung: Disjunktion)

K3 Die Unterscheidung zwischen einer kontinuierlichen und nicht-kontinuierlichen Überwachung sollte nicht durch eine zusätzliche abgeänderte Semantik in die LSC-Sprache eingeführt werden, sondern die Modellierung sollte durch die Sprache selbst unterstützt werden. (basierend auf Anforderungen: Kontinuierlich, Nicht-Kontinuierlich)

K4 Zur Reduzierung der Duplikation von Signaturteilen und somit zur besseren Wiederverwendbarkeit sollten Teilsignaturen als LSCs auslagerbar sein und in mehreren Signaturen genutzt werden können.

Da die modellierten Signaturen nicht als ausführbare Spezifikation dienen sollen, sondern zur Überwachung eingesetzt werden, muss neben diesen zusätzlichen Konzepten ebenfalls eine Anpassung der Semantik der LSCs erfolgen. Hiervon betroffen sind das Prechart, die Interpretation der Bedingungen und eine Zuordnung der eLSC-Lebenslinien auf Strukturbeschreibungen des zu überwachenden Systems. Kapitel 5 beschreibt, wie LSCs zu den eLSCs erweitert werden, um die ermittelten Unzulänglichkeiten der LSCs zu beheben. Des Weiteren wird eine auf Use-Cases und Misuse-Cases basierende Sprache vorgeschlagen, die eine übersichtlichere Strukturierung von Signaturen in komplexen Spezifikationen ermöglicht.

5

DIE MBSECMON-SPEZIFIKATIONSSPRACHE

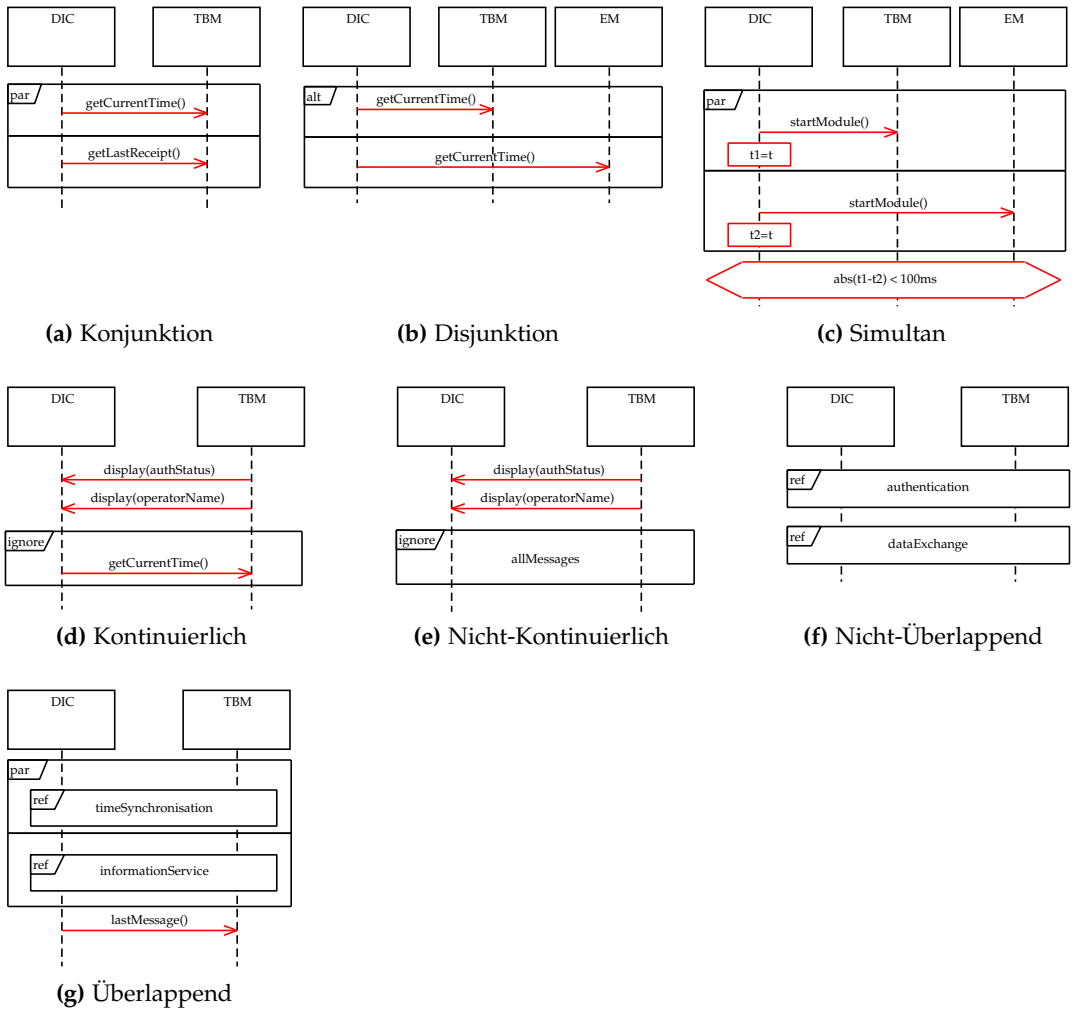
Basierend auf der in Abschnitt 4.1 vorgestellten LSC-Sprache und den Ergebnissen aus der Evaluation der LSCs als verhaltensbeschreibende Signatursprache in Abschnitt 4.3 wird in diesem Kapitel die neue *MBSecMon-Spezifikationssprache* (MBSecMonSL) vorgestellt. Die MBSecMonSL besteht aus einer erweiterten Variante der LSCs – die eLSC-Sprache – und aus einer Strukturierungssprache für die eLSCs, die auf Use- und Misuse-Cases basiert – die MUC-Sprache. Diese werden zunächst informell in ihrer konkreten Syntax in diesem Kapitel eingeführt und anschließend in Kapitel 8 durch eine Übersetzung in die formal definierten Monitor-Petrinetze (MPNs) formalisiert.

Analog zu [PPS11] wird in Abschnitt 5.1 gezeigt, wie LSCs erweitert und in ihre Semantik angepasst werden müssen, um den Anforderungen einer verhaltensbeschreibenden Signatursprache zu genügen. Hierzu müssen die in Abschnitt 4.3 ermittelten fehlenden Konzepte K1 bis K4 durch die eLSCs erfüllt werden. Anschließend beschreibt Abschnitt 5.2 eine Erweiterung der UML2-Use-Case-Diagramme um das Konzept der Misuse-Cases und die Einbettung der eLSCs in dieses Konzept. Hierdurch wird die Spezifikation aus eLSCs durch die zusätzliche Abstraktionsschicht übersichtlicher strukturiert und die wiederholte Definition der Vorbedingungen in jedem Universal Chart minimiert, indem die Überwachung von Signaturen von anderen Signaturen abhängig modellierbar ist. Des Weiteren wird ein Wiederverwendungskonzept für Teilsignaturen zur Steigerung der Skalierbarkeit der Signaturmodellierung in der MBSecMonSL eingeführt.

5.1 Erweiterte Live Sequence Charts

In Abschnitt 4.3 wurde gezeigt, dass LSCs nicht alle Anforderungen an eine Signaturbeschreibungssprache, die in Abschnitt 4.2 aufgestellt wurden, erfüllen. So müssen für die Modellierung einer überlappenden Überwachung, die durch eine eindeutige Nachricht beendet werden muss, zusätzliche Kontrollstrukturen eingeführt werden. Zusätzlich hierzu lassen sich viele der erforderlichen Muster einer verhaltensbeschreibenden Signatursprache nur durch sehr aufwendige und redundante Modellierung realisieren. Hierzu zählen u. a. die Spezifikation von Konjunktionen, Disjunktionen und dem simultanen Auftreten von Ereignissen.

Für eine kompaktere und intuitivere Modellierung von Signaturen mit LSCs wird im Folgenden gezeigt, welche neuen Fragmente in der eLSC-Sprache eingeführt werden, um die von LSCs nicht zufriedenstellend modellierbaren Konzepte K1 bis K4 zu unterstützen. Des Weiteren wird die Semantik der LSC-Diagramme in der eLSC-Sprache an den Verwendungszweck der Überwachung angepasst.



(a) Konjunktion **(b)** Disjunktion **(c)** Simultan

(d) Kontinuierlich **(e)** Nicht-Kontinuierlich **(f)** Nicht-Überlappend

(g) Überlappend

Abbildung 5.1: Muster zur Signaturbeschreibung als eLSCs

5.1.1 *Anpassungen und Erweiterungen der Syntax und Semantik*

Mehreren zueinander parallel auftretenden Ereignissequenzen sind in der LSC-Sprache nur durch die Aufteilung der Sequenzen auf mehrere parallel überwachte LSCs spezifizierbar. Dies führt, wie in Abschnitt 4.3 gezeigt, zu hoch redundanten und hierdurch unübersichtlichen Spezifikationen. Zur Lösung dieses Problems fordert K1 die explizite Modellierung von parallelen, verschränkt auftretenden Ereignissequenzen. Zur Darstellung nebenläufiger Ereignissequenzen bietet die UML2 und die MSCs im Gegensatz zu den LSCs ein kombiniertes Interaktionsfragment bzw. einen Inline-Ausdruck an – das in Abschnitt 3.2.3 vorgestellte *Par*-Fragment. Durch dieses Fragment werden in Sequenzdiagrammen das nebenläufige Auftreten von Ereignissequenzen beschrieben.

Das Par-Fragment der eLSCs: Das *Par*-Fragment erweitert die Syntax und Semantik der LSCs um eine Fragment zur direkten Modellierung nebenläufiger Abläufen in einem eLSC-Diagramm. Es wird wie ein Subchart der LSCs als Rechteck, in dessen oberen linken Eck der Operator `Par` in einem Fünfeck annotiert

ist, dargestellt. Das *Par*-Fragment besteht aus beliebig vielen *Subcharts*, die alle verschränkt zueinander eintreten müssen. Die Semantik entspricht innerhalb des *Par*-Fragments Subcharts, die unabhängig voneinander parallel überwacht werden. Hierbei werden Ereignisse, die mit modellierten eLSC-Elementen in einem oder mehreren der Subcharts übereinstimmen, in allen weiteren Subcharts ignoriert, in denen das Ereignis nicht in die modellierte Sequenz passt.

Durch die Einführung dieses Fragments in der eLSC-Sprache wird die Modellierung des bisher nicht spezifizierbaren Musters *Überlappend* ermöglicht und zusätzlich eine übersichtlichere, den Entwicklern bekannte Notation für die Modellierung nebenläufiger Sequenzen zur Verfügung gestellt. Somit können die Muster *Konjunktion* und *Simultan* kompakter dargestellt werden.

Beispiel *Einsatz des Par-Fragments*

Eine *Konjunktion* von Ereignissequenzen (Nachrichten) kann nun, wie in Abbildung 5.1a dargestellt, kompakt spezifiziert werden. Das neu eingeführte Par-Fragment beschreibt lokal Teildiagramme, die vorher als einzelne LSCs (Abb. 4.4b) modelliert werden mussten.

Auch die Modellierung des *simultanen* Auftretens mehrerer Ereignisse kann, wie in Abbildung 5.1c gezeigt, mithilfe des Par-Fragments spezifiziert werden. Hierdurch müssen nicht alle möglichen Abläufe einzeln als LSCs modelliert werden, da das Par-Fragment alle Reihenfolgen der verschränkten Ereignissequenzen abbildet. Erlaubt das Kommunikationsmedium im Gegensatz zum CAN-Bus eine echte gleichzeitige Übertragung, kann dieses durch ein Intervall von `0ms` ausgedrückt werden.

Das Par-Fragment kann ebenfalls zur Spezifikation einer Signatur, die der Anforderung nach der Beschreibbarkeit überlappender Ereignissequenzen entspricht, eingesetzt werden. Diese Signatur kann als eLSC wie in Abbildung 5.1g explizit modelliert werden. Durch den Einsatz eines Par-Fragments mit einer ausgelagerten letzten Nachricht wird erreicht, dass alle Ereignisse innerhalb des Fragments zuerst auftreten müssen, bevor die ausgelagerte letzte Nachricht auftreten darf. Hierbei repräsentieren die später in diesen Abschnitt eingeführten Ref-Fragmente ausgelagerte Signaturteile.

Die Spezifikation mehrerer Alternativen (*Disjunktion*), die nicht durch eine Bedingung am Anfang der Subcharts (If-Then-Else-Fragment) entschieden werden kann, ist in den LSCs nur, wie in Abbildung 4.4c gezeigt, über eine Kombination von alternativen LSCs und dem Ausschluss des gemeinsamen Auftretens durch Anti-Szenarien formulierbar. Da dies zu umfangreichen, schwer lesbaren und auch schwer zu spezifizierenden Signaturen führt, ist die Umsetzung des geforderten Konzepts K2 für die eLSCs sinnvoll. Auch zur Spezifikation von Alternativen bieten die Sequenzdiagramme der UML und des MSC2000-Standards ein zusätzliches kontrollflussbeschreibendes Fragment an – das Alt-Fragment.

Das Alt-Fragment der eLSCs: Alternativen können in den eLSCs durch zwei oder mehr aneinandergehängte Subcharts modelliert werden. Zur Abhebung von separaten aufeinanderfolgenden Subcharts wird wie beim Par-Fragment, der Operator `Alt` oben links im Rechteck annotiert. Das Alt-Fragment wird ähnlich, wie

das *If-Then-Else*-Fragment interpretiert, wobei jedoch die Auswahl des Subcharts nicht durch eine eLSC-Bedingung erfolgt. Stattdessen werden die Teilsignaturen in allen Subcharts parallel überwacht. Schlägt die Überwachung einer Teilsignatur fehl, bricht diese ab, führt jedoch zunächst nicht zum Fehlschlagen des gesamten eLSCs. Erst wenn die Überwachung aller alternativen Teilsignaturen fehlgeschlagen ist, ist auch die Überwachung des eLSCs fehlgeschlagen. Die Einführung dieses Fragments in die eLSC-Sprache führt durch die explizite Modellierung von Alternativen zu kompakteren Spezifikationen von *Disjunktionen*.

Beispiel *Einsatz des Alt-Fragments*

Die *Disjunktion* ist nun, wie in Abbildung 5.1b dargestellt, in einem eLSC-Diagramm modellierbar. Hierfür werden zwei Subcharts zu einer Oder-Struktur verknüpft und durch den Operator `Alt` markiert.

Die Evaluation der Ausdrucksstärke der LSCs in Abschnitt 4.3 hat gezeigt, dass die explizite Unterscheidung zwischen kontinuierlicher und nicht-kontinuierlicher Überwachung der Signaturen als LSCs nur durch die Einführung von zwei verschiedenen Semantiken lösbar ist. Durch die Umsetzung des Konzeptes K3 in der eLSC-Sprache wird auf die Einführung einer alternativen Ausführungssemantik verzichtet und die native, kontinuierliche Semantik der LSCs beibehalten. Hierzu wird das *Ignore*-Fragment in der eLSC-Sprache eingeführt.

Das Ignore-Fragment der eLSCs: Der zusätzliche Einsatz des *Ignore*-Fragments führt zu einer kompakteren Modellierung von Signaturen in einem kontinuierlichen Überwachungsszenario. Im Gegensatz zum *Ignore*-Fragment der UML2-Sequenzdiagramme, das als übergeordnetes Chart um den betreffenden Bereich der Sequenz platziert wird und der textuellen Angabe der zu ignorierenden Nachrichten, orientiert sich das *Ignore*-Fragment der eLSCs am bereits in den LSCs vorhandenen *Forbidden*-Fragment. Wie die Forbidden-Fragmente werden Ignore-Fragmente unter dem Mainchart im eLSC platziert. Wenn sie nicht für das gesamte LSC gelten, werden sie einem Prechart, Mainchart oder einem Subchart zugeordnet. In diesem definierten Bereich können Ereignisse, die nicht explizit modelliert sind, global ignoriert werden und führen bei der Überwachung nicht zu einem Fehlschlagen der Signatur. Alternativ zur Spezifikation von bestimmten Ereignissen, die ignoriert werden sollen, können auch durch die Annotation von `allMessages` in diesem Fragment, alle nicht modellierten Ereignisse bei der Überwachung des eLSCs ignoriert werden. Hierdurch lassen sich trotz der kontinuierlichen Überwachung der als eLSCs modellierten Signaturen, auch nicht-kontinuierliche Zusammenhänge modellieren. Zusätzlich können die Signaturen deutlich kompakter spezifiziert werden, da nicht mehr alle möglichen Ereignisse im Pre- oder Mainchart des eLSCs explizit modelliert werden müssen.

Beispiel *Einsatz des Ignore-Fragments*

Das *Ignore*-Fragment wird zu Spezifikation von *kontinuierlichen* Signaturen eingesetzt, um nicht wie in Abbildung 4.5d alle Ereignisse, die auftreten können, explizit zu modellieren. Der Einsatz des *Ignore*-Fragments bewirkt, dass die unbeschränkte Schleife der eLSC-Signatur in Abbildung 5.1d durch ein *Ignore*-Fragment ersetzt werden kann.

Um im *nicht-kontinuierlichen* Fall in einer eLSC-Signatur nicht alle Nachrichten explizit modellieren zu müssen, damit sie ignoriert werden, wird das *Ignore*-Fragment wie in Abbildung 5.1e mit dem Schlüsselwort `allMessages` eingesetzt.

Die Evaluation der LSCs als Signatursprache in Abschnitt 4.3 hat ebenfalls gezeigt, dass Signaturen häufig gleiche Anteile enthalten, die mehrfach modelliert werden müssen und so zu einer schlechteren Wartbarkeit führen. Dieser Nachteil der LSCs, der durch das vorgeschlagene Konzept K4 adressiert wird, wird in den eLSCs durch die Einführung des aus der UML2 bekannten *Ref*-Fragments behoben.

Ref-Fragment: Das *Ref*-Fragment bindet ausgelagerte eLSC-Teilsignaturen, die nicht nur in einer Signatur benötigt werden, in ein eLSC ein, wodurch diese Teilsignaturen in mehreren eLSCs wiederverwendet werden können. Die *Ref*-Fragmente werden wie in UML2-Sequenzdiagrammen dargestellt und referenzieren über ein Namensmapping die ausgelagerten eLSC-Teilsignaturen. Wird ein *Ref*-Fragment bei der Überwachung der Hauptsignatur erreicht, wird es durch die referenzierte Teilsignatur ersetzt und mit der Überwachung der so ergänzten Signatur fortgefahren.

Beispiel *Einsatz des Ref-Fragments*

Das Ref-Fragment kann wie in Abbildung 5.1f eingesetzt werden, um Subcharts aus der Signatur auszulagern. So kann die Teilsignatur der `authentication`-Phase auch in anderen Signaturen wiederverwendet werden. Auch in der Signatur in Abbildung 5.1g in der der überlappende Überwachungsmodus mithilfe eines Par-Fragments modelliert ist, sind Teile der Signatur ausgelagert.

Neben diesen vier neu eingeführten Elementen zur Kontrollflusssteuerung innerhalb der LSCs sind für den Einsatz der eLSCs als Signaturbeschreibungssprache Änderungen an der Semantik vorhandener Modellierungselemente notwendig. Hierfür muss die Semantik des *Precharts* zur Spezifikation komplexerer Vorbedingungen einer Signatur erweitert werden, die Interpretation von *Bedingungen* angepasst und eine Zuordnung der eLSC-Lebenslinien zu Strukturbeschreibungen des zu überwachenden Systems realisiert werden.

Prechart: In LSCs werden Precharts komplett kalt modelliert und die maximale Richtlinie für den Ausführungsalgorithmus (vgl. Abschnitt 4.1) angewendet. Diese Semantik ist für die Ausführung der LSCs im Play-In/Play-Out-Ansatz [HM03] geeignet, schränkt jedoch die Spezifikation komplexerer Vorbedingungen einer Signatur ein. Da bei einer Signaturbeschreibungssprache keine Auswahl der nächsten auszuführenden Nachricht stattfindet, sondern die Ereignisse durch das zu überwachende System ausgelöst werden, ist es sinnvoll ebenfalls im Prechart Modalitäten zu verwenden, um optionale Teile der Vorbedingung modellieren zu können. Durch diese Kombination von heißen und kalten Elementen können komplexere Vorbedingungen kompakt modelliert werden und somit ähnliche Signaturen, die sich nur in ihrem Prechart unterscheiden, in einer Signatur kombiniert werden.

Somit wird in der eLSC-Sprache das Prechart selbst wie bei den LSCs kalt modelliert, d. h. als blaues und gestricheltes Hexagon dargestellt. Die Vorbedingung innerhalb des Prechart-Elements wird wie ein Mainchart durch kalte und heiße Elemente modelliert. Die kalte Temperatur des Precharts wirkt sich jedoch auf die Auswertung der Beendigung des Precharts aus. Wird die Überwachung des Precharts nicht erfolgreich abgeschlossen, bedeutet dies nur, dass das Mainchart nicht überwacht werden muss. Wird das Prechart erfolgreich erkannt, muss das Mainchart erfüllt werden.

Globale Bedingungen und Prädikatenbedingungen: *Prädikatenbedingungen*, die als Vorbedingung eines Subcharts bzw. eines *If-Then-Else*-Fragments eingesetzt werden, sind immer kalt modelliert. Trifft diese Bedingung zu, wird das Chart weiter überwacht, während die Nichterfüllung der Bedingung zu einem Abbruch des aktuellen Charts führt und im Fall des *If-Then-Else*-Fragments das Subchart des Else-Teils ausgeführt wird. Dies entspricht der Semantik der LSCs. Die Semantik der *Globalen Bedingungen* für LSCs (Tab. 4.2) unterscheidet sich jedoch von denen der eLSCs (Tabelle 5.1). *Globale Bedingungen*, die *kalt* modelliert sind, verhalten sich abweichend von den Prädikatenbedingungen. Sie können an jeder Stelle im eLSC eingesetzt werden. Wenn die Bedingung nicht erfüllt ist, bleibt der Überwachungszustand der Signatur auf den von der Bedingung überlappten Lebenslinien stehen. Hierdurch muss erst die Bedingung erfüllt werden, damit die weiteren Ereignisse auf den überlappten Lebenslinien der Signatur (z. B. Senden und Empfangen von Nachrichten) auftreten dürfen. Wartet das eLSC vor einer bisher nicht erfüllten Bedingung und ein Ereignis tritt auf, das nicht zum aktuellen Zustand der Signatur passt, findet ein Abbruch der Überwachung des restlichen Subcharts bzw. des gesamten eLSCs statt. *Heiße globale Bedingungen* sind bei Erreichen zwingend einzuhalten und führen bei Nichterfüllung zu einem sofortigen Fehlschlagen der eLSC-Signatur und somit zur Fehlererkennung.

Zeitbedingungen: Die Semantik der *Zeitbedingungen* teilt sich wie bei den LSCs wieder auf *minimale* und *maximale* Zeitbedingungen auf (Abschn. 4.1). Als *Prädikatenbedingung* bedeutet eine *kalte minimale* oder *maximale Zeitbedingung*, dass bei positiver Auswertung das entsprechende Subchart weiter überwacht wird, bei negativer Auswertung wird das Subchart nicht weiter überwacht. Gibt es ein alternatives Subchart wie bei dem *If-Then-Else*-Fragment, wird dieses stattdessen überwacht. Wie auch bei den einfachen Prädikatenbedingungen werden diese auch als Zeitbedingungen immer kalt modelliert. *Globale Zeitbedingungen* können *heiß* oder *kalt* sein. Ist die Auswertung der *minimalen* oder der *maximalen Zeitbedingung* positiv, wird die Überwachung in beiden Fällen fortgesetzt. Bei negativer Auswertung der annotierten Bedingung ergeben sich Unterschiede in der Semantik, basierend auf der Erfüllbarkeit der Zeitbedingung. Eine *globale minimale Zeitbedingung* schlägt nicht sofort fehl, sondern wartet vor dieser Bedingung, bis die Zeit überschritten wird. Tritt währenddessen ein Ereignis auf, das nicht zu der modellierten Signatur in diesem Zustand passt, schlägt das gesamte eLSC fehl. Die *maximale* Variante schlägt sofort fehl, da der Zeitpunkt schon überschritten ist und die Bedingung nie wieder zu wahr evaluieren kann. *Globale, kalte Zeitbedingungen* brechen die Überwachung des Charts in dem sie sich befinden oder das gesamte eLSC, wenn sie in keinem Subchart platziert sind, ab.

	Prädikat (Subchart/If-Then-Else ≡ an erster Location)	**Global**	
	kalt	heiß	kalt
Bedingung	+ Chart überwachen – Chart nicht überwachen	+ eLSC bzw. Chart weiter überwachen – eLSC schlägt fehl	+ eLSC bzw. Chart weiter überwachen – eLSC- bzw. Chart-Überwachung für überlappte Lebenslinien pausieren, wenn Ereignis kommt, das nicht passt, abbrechen
Zeitbedingung (minimal)	+ Chart überwachen – Chart nicht überwachen, mögliche Alternative überwachen	+ eLSC bzw. Chart weiter überwachen – warten bis Bedingung erfüllt (andere nicht modellierte Ereignisse führen zum Fehlschlagen des eLSCs)	+ eLSC bzw. Chart weiter überwachen – eLSC bzw. Chart-Überwachung abbrechen
Zeitbedingung (maximal)	+ Chart überwachen – Chart nicht überwachen, mögliche Alternative überwachen	+ eLSC bzw. Chart weiter überwachen – eLSC- bzw. Chart-Überwachung fehlgeschlagen	+ eLSC bzw. Chart weiter überwachen – eLSC- bzw. Chart-Überwachung abbrechen

Tabelle 5.1: Definition von Bedingungen in eLSCs zur Modellierung von Monitorsignaturen (+ erfüllt, – nicht erfüllt)

Rekursive Ausführung: Die rekursive bzw. überlappende Ausführung mehrerer eLSCs, die in Abschnitt 4.1 für LSCs vorgestellt wurde, wird für die Überwachung von Signaturen geringfügig angepasst. Die nebenläufige Überwachung ein und desselben eLSCs erfolgt nur, wenn das Universal eLSC als nebenläufig (*engl. concurrent*) gekennzeichnet ist. Ist es nicht als nebenläufig markiert, werden zwar verschiedene eLSCs gleichzeitig überwacht, jedoch muss die Überwachung einer Instanz eines eLSCs erst abgeschlossen sein, bevor einer neuen Instanz desselben eLSCs Ereignisse zugeordnet werden.

Locations der Nachrichten: Die Semantik der Locations der LSCs und ihrer Temperatur wird bei den eLSCs, durch die Einschränkung der möglichen Kombinationen bei Nachrichten, auf vier Fälle, der in Tabelle 4.1 angegebenen 8 Kombinationen, reduziert. Diese Kombinationen sind:

- Fall 1, in dem alle Locations und die Nachricht heiß sind,
- Fall 2, in dem eine gesendete Nachricht auch empfangen werden muss,
- Fall 5, in dem alle Locations und Nachrichten kalt sind und
- Fall 7, in dem eine gesendete Nachricht nicht empfangen werden muss.

Während in der Ausführungssemantik der LSCs die Fälle 1 und 2 als gleich angesehen werden, da in beiden Fällen die Nachricht empfangen und somit auch gesendet werden muss, beschreiben die beiden Fälle bei den eLSCs aufgrund des Einsatzes zur Überwachung unterschiedliche Sachverhalte. In Fall 1 muss die Nachricht sowohl gesendet als auch empfangen werden und in Fall 2 muss die

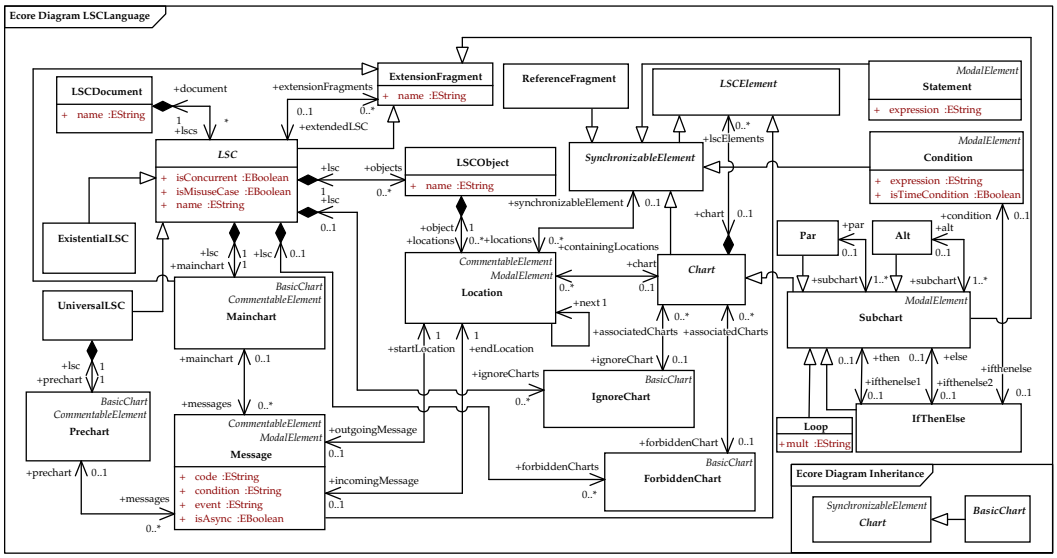


Abbildung 5.2: Metamodell der eLSC-Sprache

Nachricht nur empfangen werden, wenn sie auch vorher gesendet wurde. Fall 5 und Fall 7 entsprechend den gleich benannten Fällen der LSCs aus Tabelle 4.2. Ein weiterer für die Signaturbeschreibung wichtiger Fall, in dem eine Nachricht nur empfangen werden muss und das Sendeereignis unerheblich ist, kann mit einer *Environment*-Instanz modelliert werden. Diese Instanz dient als eine unspezifizierte Quelle, aus der die Nachricht stammt oder an die die Nachricht gesendet wird. Mithilfe dieser fünf verschiedenen Varianten der Modellierung von Nachrichten lassen sich alle wichtigen Konzepte des Nachrichtenaustauschs modellieren, die zur verhaltensbeschreibenden Signaturspezifikation benötigt werden.

Anbindung von eLSCs an Strukturdiagramme der UML: LSCs bieten zur Modellierung *generischer Abläufe*, wie in Abschnitt 4.1 vorgestellt, Klassen und *Symbolische Instanzen* an. Durch diese Konzepte lassen sich die eLSCs an strukturbeschreibende UML2-Diagramme anbinden. Hierfür kommen neben den in den LSCs angedachten UML2-Klassendiagrammen auch die abstrakteren UML2-Komponentendiagramme in Betracht.

5.1.2 *Die abstrakte Syntax der eLSCs*

In diesem Abschnitt wird die abstrakte Syntax der eLSCs anhand ihres Metamodells in Abbildung 5.2 vorgestellt. Zur Visualisierung des Metamodells wird wie im MOF-Standard (Abschn. 2.2) die Notation der UML2-Klassendiagramme verwendet. Die beiden primitiven Datentypen `EString` und `EBoolean` entsprechen dem Ecore-Standard. Als übergeordneter Behälter für eine Sammlung von eLSCs (`LSC`) dient die Klasse `LSCDocument`. Die Ausprägungen der Klasse `LSC` sind zum einen Existential LSCs (`ExistentialLSC`) und zum anderen Universal LSCs (`UniversalLSC`). Beide besitzen genau ein `Mainchart`, während das `UniversalLSC` zusätzlich noch ein `Prechart` besitzen kann. `Prechart` und `Mainchart` enthal-

ten Nachrichten (`Message`), die zwischen den im `LSC` enthaltenen Lebenslinien (`LSCObject`) verschickt werden. Als Ankerpunkte der Nachrichten dienen Instanzen der Klasse `Location`, die den `LSCObject`-Instanzen zugeordnet sind. Eine `Location` dient als Startpunkt (Senden) und eine Zweite als Zielpunkt (Empfangen) der Nachricht. Zusätzlich besitzen die Instanzen von `Location` eine übergeordnete Ordnung im `LSC`, die über ihre `next`-Referenz beschrieben wird. Alle Elemente des eLSC-Metamodells, mit Ausnahme der Ausprägungen der Klasse `LSC` und die Klasse `LSCObject`, sind direkte oder indirekte Ausprägungen der abstrakten Klasse `LSCElement`. Alle Modellierungselemente der eLSCs, die in der konkreten Syntax Lebenslinien überspannen und somit eine Synchronisierung der Lebenslinien bewirken, sind Ausprägungen der Klasse `SynchronizableElement`. Alle `SynchronizableElements` sind mit denen ihnen zugeordneten Lebenslinien über `Locations` verbunden. So können Zuweisungen (`Statement`) und Bedingungen (`Condition`) sowohl mit einer oder mit beliebig vielen Lebenslinien verbunden werden. Zeitbedingungen werden im eLSC-Metamodell durch die Boolean-Variable `isTimeCondition` markiert. Das in den eLSCs zusätzlich eingeführte *Ref*-Fragment wird durch die Klasse `ReferenceFragment` repräsentiert. Werden eLSCs ohne die im nächsten Kapitel eingeführte Misuse-Case-Sprache eingesetzt, erfolgt die Zuordnung der referenzierten LSCs über ein Namensmapping im Attribut `name:EString`. Die abstrakte Klasse `Chart` stellt die Elternklasse für alle eLSC-Elemente dar, die weitere `LSCElement`-Objekte enthalten können. Das `Chart` kennt alle `Location`-Instanzen und somit auch alle LSC-Elemente, die in ihm enthalten sind. Wie in Abbildung 5.2 unten rechts zu sehen ist, erbt von `Chart` sowohl die Klasse `Subchart` als auch die abstrakte Klasse `BasicChart`. Unter `BasicCharts` fallen alle Charts, die auf oberster Ebene im LSC angesiedelt sind. Dies sind die Klassen `Prechart`, `Mainchart`, `IgnoreChart` und `ForbiddenChart`. `IgnoreChart` und `ForbiddenChart` können wiederum einem `Chart` zugeordnet werden und können somit ihren Gültigkeitsbereich eingeschränkten. Zu den spezialisierten Subcharts gehört das Schleifen- (`Loop`), das If-Then-Else-Fragment (`IfThenElse`), das Alt-Fragment (`Alt`) und das Par-Fragment (`Par`). Das Attribut `mult:EString` der Klasse `Loop` definiert die Iterationsanzahl der Schleife. Die Klasse `IfThenElse` dient als Behälter für zwei Subcharts, die den Then- und Else-Teil repräsentieren. Als Bedingung des Fragments wird eine Bedingung (`Condition`) verwendet. Das Klasse `Par` dient als Behälter für beliebig viele Subcharts. Zur Abbildung der Modalitäten (Temperaturen der eLSCs) dient als Elternklasse `ModalElement`, die ihr Attribut `isHot` an alle modalen Elemente der eLSC-Sprache vererbt. Die Klasse `ExtensionFragment` von der `LSC`, `Mainchart` und alle Ausprägungen von `Subchart` erbt, dient zur Einbettung der eLSCs in die im nächsten Kapitel vorgestellte MUC-Sprache.

5.2 Use-/Misuse-Case-Sprache (MUC-Sprache)

Nachdem in Abschnitt 5.1 gezeigt wurde, welche Erweiterungen in den eLSCs zur Beschreibung von verhaltensbeschreibenden Signaturen eingeführt werden mussten, um die identifizierten Schwachpunkte der LSCs zu beheben, wird in diesem Abschnitt auf die Einsetzbarkeit der Sprache zur Beschreibung komplexer Syste-

me eingegangen. Es hat sich herausgestellt, dass Spezifikationen aus eLSCs trotz der in Abschnitt 5.1.1 neu eingeführten vier Fragmente durch die implizite Modellierung ihrer Zusammenhänge schnell unübersichtlich werden. Jede einzelne eLSC-Signatur beschreibt ihre Vorbedingung, die auch sehr komplex sein kann, explizit als Prechart. Häufig kommt es vor, dass Vorbedingungen von verschiedenen alternativen Abläufen nahezu oder komplett gleich sind und somit von mehreren Signaturen geteilt werden könnten. Für den Modellierer der Signaturen bedeutet dies, dass er die Vorbedingung, die festlegt, wann ein Mainchart gültig ist, mehrfach modellieren muss und somit der Spezifikationsprozess sehr komplex, fehleranfällig und zeitaufwendig ist.

Eine schon in Abschnitt 4.3 angesprochene Möglichkeit Zusammenhänge zwischen Teilsignaturen und somit Kontrollfluss auf einer höheren Abstraktionsebene zu spezifizieren sind die Interaktionsübersichtsdiagramme der UML2. Durch sie können u. a. über die Parallelisierungs- und Synchronisierungskonten zwar gemeinsame Vorbedingungen als einzeln referenziertes Sequenzdiagramm modelliert werden, allerdings wird hierdurch das Konzept der Precharts der LSCs und eLSCs ausgehebelt. Gegen den Einsatz einer solchen kontrollflussbeschreibenden Sprache spricht ebenfalls, dass hierdurch viele Konzepte, die schon in den LSCs vorhanden sind und durch die neuen Fragmente abgedeckt werden, dupliziert werden würden.

Beispiel *Implizite Zusammenhänge zwischen eLSCs*

Die in Abschnitt 4.1 vorgestellten LSC-Signaturen des CARDME-Protokolls, beschreiben unabhängig voneinander durch ihr Prechart wann sie überwacht werden sollen und was überwacht werden soll. Im CARDME-Beispiel gibt es jedoch zwei Varianten des Protokolls zur Gebührenabrechnung. Die eine ist zur Kommunikation mit einer Mautbrücke vorgesehen, die andere (Abb. 5.3) zur Kommunikation mit einer RSU, die für das Parken in einem Parkhaus verantwortlich ist. Beide Signaturen haben eine identische Vorbedingung, die sich beide Signaturen teilen könnten.

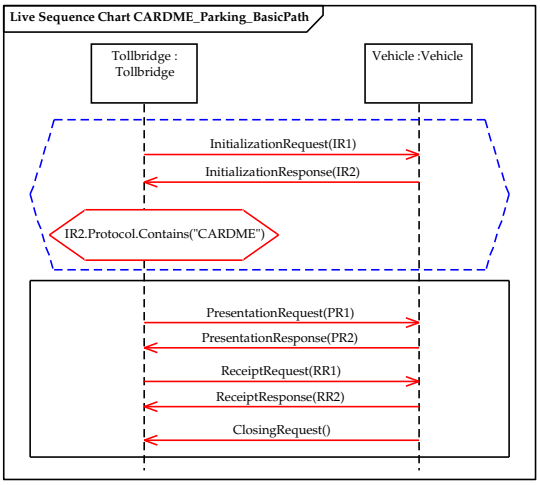


Abbildung 5.3: eLSC-Signatur zum Einsatz des CARDME-Protokolls zum Parkraummanagement

Die redundante Spezifikation hat jedoch nicht nur Auswirkungen auf ihre Erstellung, Verständlichkeit und Wartbarkeit, sondern erschwert ebenfalls, wie im Beispiel gezeigt, die Generierung eines effizienten Monitors, der möglichst wenig Ressourcen (Rechenzeit, Speicher) beansprucht. Entweder muss jede der mehrfach modellierten Vorbedingungen einzeln ausgewertet werden, wodurch die Rechenzeit des Monitors je Ereignis ansteigt, oder während der Codegenerierung eine aufwendige Analyse stattfinden, die mehrfache Vorbedingungen zusammenfasst. Diese Probleme werden durch die Einführung einer den eLSC-Signaturen übergeordneten Sprache, die zur Strukturierung der Spezifikation aus eLSCs dient, gelöst. Zusammen mit den eLSCs bildet diese im Folgenden vorgestellte Misuse-Case-Sprache (MUC-Sprache), die MBSecMon-Spezifikationssprache (*engl. MBSecMon Specification Language (MBSecMonSL)*).

5.2.1 *Die konkrete Syntax und Semantik der MUC-Sprache*

Die MUC-Sprache ist eine Erweiterung der UML2-Anwendungsfalldiagramme (*engl. use case diagrams*), die um Konzepte zur Modellierung von Misuse-Cases erweitert wurde. Die MUC-Sprache dient zur Kapselung der Signaturen, die als eLSCs beschrieben sind, in positive und negative Anwendungsfälle (Use- und Misuse-Cases) und zur expliziten Modellierung der Beziehungen zwischen ihnen. Die neu eingeführten Misuse-Cases und Beziehungen zwischen ihnen und den Use-Cases basieren auf den in Abschnitt 3.1 vorgestellten Konzepten, die für die Strukturierung von Signaturen angepasst und erweitert wurden.

Konkrete Syntax der Use- und Misuse-Cases: Anwendungsfälle werden durch Ellipsen repräsentiert, die mit einem Namen beschriftet sind, der die Funktionalität der enthaltenen eLSCs beschreibt. Use-Cases besitzen den Stereotyp «usecase» und werden durch eine weiße Füllung der Ellipse repräsentiert. Die in Abschnitt 3.1 vorgestellten Misuse-Cases werden zur Unterscheidung zu den Use-Cases mit dem Stereotyp «misusecase» annotiert und grau hinterlegt dargestellt.

Beispiel *MUC-Diagramm des CARDME-Protokolls*

Abbildung 5.4 zeigt eine mögliche Strukturierung der eLSC-Spezifikation des CARDME-Protokolls durch ein MUC-Diagramm in konkreter Syntax. Dieses MUC-Diagramm wird Schritt für Schritt mit der Einführung der Elemente und Konzepte der MUC-Sprache in den Beispielabschnitten vollständig vorgestellt. Zur Verdeutlichung, welche eLSCs in welchem Behälter (Use-Cases und Misuse-Cases) enthalten sind, wird hier ein Namensmapping von Anwendungsfällen mit ihren enthaltenden eLSCs angenommen. Ein Use-Case mit dem Namen `CARDME` enthält ein eLSC mit dem Namen `CARDME_BasicPath`. Jeder weitere alternative Ablauf wird nach dem Namensschema `CARDME_Alternate<X>` benannt, wobei `<X>` $\in \mathbb{N}$ für Nummer des entsprechenden alternativen Szenarios (eLSC) steht. Zur Vereinfachung wird in diesem Beispiel angenommen, dass in jedem Use- bzw. Misuse-Case nur ein eLSC enthalten ist. Abbildung 5.4 zeigt fünf Use-Cases, benannt mit `CARDME`, `CARDME_Parking`, `Tracking`, `NonLocal` und `CloseConnection`, und zwei Misuse-Cases mit den Namen `Init_DoS_Attack` und `Tracking_DoS_Attack`.

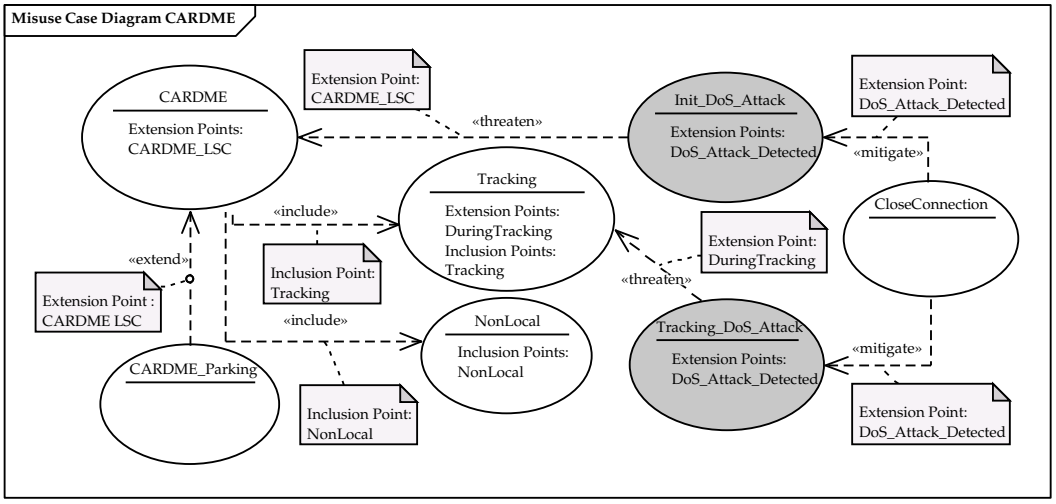


Abbildung 5.4: CARDME-Protokoll strukturiert mit MUC-Sprache

Zusätzlich zur Kapselung können eLSCs über *Erweiterungspunkte* (*engl. extension points*) und *Inklusionspunkte* (*engl. inclusion points*) miteinander in Beziehung gesetzt werden. Diese werden unter dem Namen der Anwendungsfälle, abgetrennt durch eine horizontale Linie, aufgelistet und durch eine Beschriftung *Extension Points:* bzw. *Inclusion Points:* gruppiert. Die unter diesen beiden Beschriftung annotierten Punkte müssen im Gültigkeitsbereich des Anwendungsfalls eindeutig benannt sein.

Ein *Erweiterungspunkt* eines Use-Cases stellt einen Ankerpunkt für die Beschreibung alternativer Abläufe zu einer vorhanden (Teil-)Signatur, modelliert als eLSC, dar. Er wird hierzu entweder einem vollständigen eLSC oder Erweiterungsfragmenten (Def. 6) der eLSC-Sprache zugeordnet, zu dem oder zu denen Alternativen modelliert werden sollen. Ein *Inklusionspunkt* stellt im Gegensatz zum Erweiterungspunkt keine Alternative dar, sondern visualisiert im Use-Case, welche ausgelagerten Teilsignaturen in ihm enthalten sind. Die Inklusionspunkte spiegeln somit die in dem Use-Case enthaltenen Signaturen wider, die in anderen Anwendungsfällen in eLSCs über ein *Ref*-Fragment eingefügt werden können.

Definition 6 (*Erweiterungsfragmente in eLSCs*)**.** Ein Erweiterungsfragment (*engl. extension fragment*) ist ein komplettes eLSC, ein Mainchart oder ein Subchart-Fragment der eLSC-Sprache. Zu den erweiterbaren Subcharts gehören neben dem Subchart selbst das *If-Then-Else*-Fragment, das *Loop*-Fragment, das *Par*-Fragment und das *Alt*-Fragment.

Beispiel *Erweiterungs- und Inklusionspunkte*

Der Use-Case `CARDME` in Abbildung 5.4 enthält einen *Erweiterungspunkt* mit dem Namen `CARDME_LSC`. Diesem ist das eLSC `CARDME_BasicPath` in Abbildung 5.5 zugeordnet, das im selben Use-Case enthaltenen ist. Über die im Folgenden

vorgestellten Beziehungen zwischen den Anwendungsfällen können durch Referenzierung des Erweiterungspunktes alternative (Teil-)Signaturen definiert werden. Die Use-Cases `Tracking` und `NonLocal`, stellen ausgelagerte Teilsignaturen dar, die an mehreren Stellen verwendet werden können. Der Inklusionspunkt `Tracking` visualisiert im MUC-Diagramm das im Use-Case enthaltene eLSC `Tracking_BasicPath` und entsprechend repräsentiert der Inklusionspunkt `NonLocal` ebenfalls die enthaltene Teilsignatur.

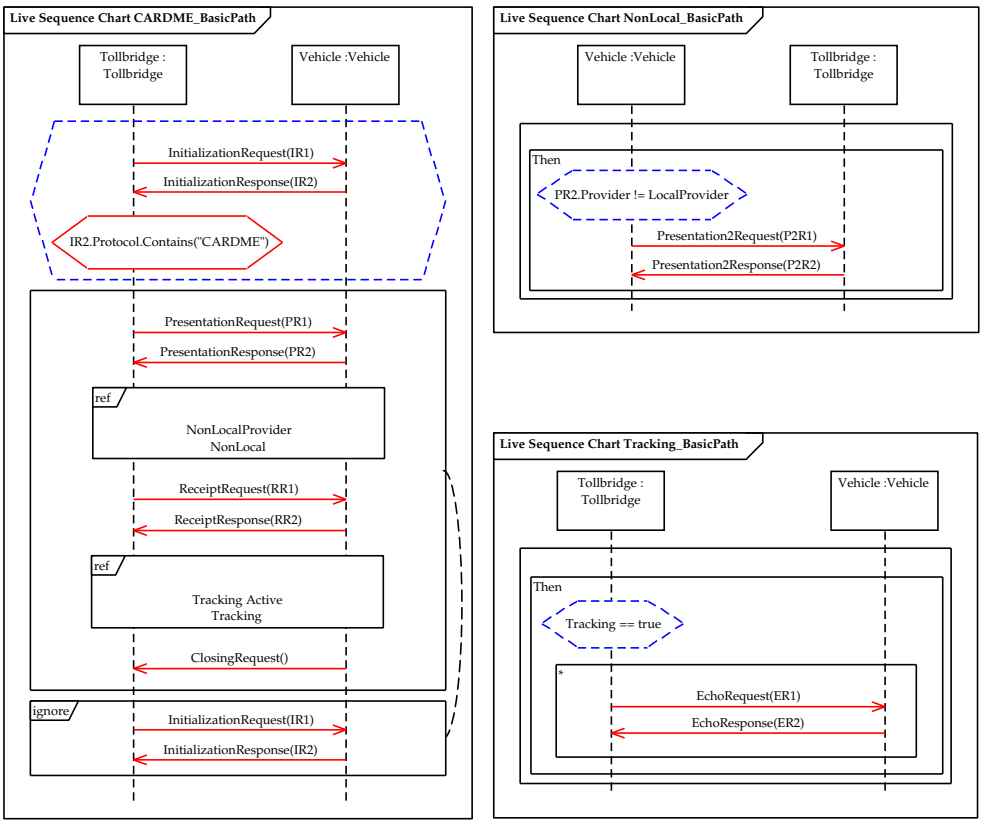


Abbildung 5.5: eLSC-Signatur des CARDME-Protokolls zur Gebührenerfassung durch die Mautbrücke

Die Ankerpunkte – Erweiterungs- und Inklusionspunkte – erlauben es die Anwendungsfälle und somit die enthaltenen eLSC-Signaturen miteinander explizit in Beziehung zu setzen und Teilsignaturen wiederzuverwenden. Dies geschieht durch vier verschiedene Beziehungen.

Beziehungen zwischen Anwendungsfällen: Die Modellierung der Abhängigkeitsbeziehungen zwischen den Anwendungsfällen wird durch vier verschiedene Beziehungstypen realisiert: «include», «extend», «mitigate» und «threaten». Diese vier Beziehungen werden durch unterbrochene gerichtete Kanten dargestellt, die mit dem entsprechenden Stereotypen annotiert sind. Tabelle 5.2 zeigt, wie die Beziehungen zwischen Use- und Misuse-Cases eingesetzt werden dürfen, während Abbildung 5.6 alle dort beschriebenen Kombinationen mit der entsprechenden Anwendungsfallbenennung in konkreter Syntax darstellt. Aus Übersichtsgründen wird hier immer nur ein Erweiterungs- bzw. Inklusionspunkt der

	Quellseite	
Zielseite	**Use-Case**	**Misuse-Case**
Use-Case	«extend», «include», «mitigate»	«threaten», «include»
Misuse-Case	«mitigate»	«extend», «include»

Tabelle 5.2: Erlaubte Beziehungen zwischen Use- und Misuse-Cases in der MUC-Sprache

	eLSCs im Anwendungsfall an der		Art der
Beziehung	**Quellseite**	**Zielseite**	**Überwachung**
«include»	Ein *Ref*-Fragment referenziert einen Inklusionspunkt der Zielseite. Die zur Verfügung stehenden Inklusionspunkte sind an der Beziehung annotiert.	Inklusionspunkte definieren Mengen verschiedener eLSCs	eingefügt
«extend»	Menge aller eLSCs im Use-/ Misuse-Case	Erweiterungspunkte definieren Mengen von Erweiterungsfragmenten.	parallel
«threaten»	Menge aller eLSCs im Misuse-Case	Erweiterungspunkte definieren Mengen von Erweiterungsfragmenten	parallel
«mitigate»	Ein Existential eLSC in einem Use-Case	eLSCs, die die Vorbedingung des Existential eLSC sind.	sequenziell

Tabelle 5.3: Erweiterungs- und Inklusionspunkte in der MBSecMonSL

Beziehung zugeordnet. Im Allgemeinen können einer Beziehung jedoch beliebig viele dieser Punkte zugeordnet werden. Tabelle 5.3 liefert eine zusammenfassende abstrakte Übersicht über die Verwendung dieser Beziehungen zwischen den Anwendungsfällen, die im Folgenden genauer beschrieben wird.

«include»-Beziehungen verbinden in den MUC-Diagrammen Use-Cases mit Use-Cases bzw. Misuse-Cases mit Use-Cases oder Misuse-Cases wie in Abbildung 5.6 dargestellt. Der *inkludierte Anwendungsfall* auf der Zielseite der Beziehung, der als Use-Case modelliert ist, wird vom *Basis-Anwendungsfall* auf der Quellseite inkludiert. Diese Beziehung ermöglicht die Auslagerung von Signaturbestandteilen in einen separaten Anwendungsfall, wodurch die Bestandteile in beliebig vielen Anwendungsfällen wiederverwendet werden können. Abhängig von der Intention der ausgelagerten Signaturteile werden sie in einen Use- oder Misuse-Case ausgelagert. Ist der inkludierte Anwendungsfall ein Misuse-Case, bedeutet dies, dass die dort hinterlegten Teilsignaturen nur in anderen Misuse-Cases sinnvoll eingesetzt werden dürfen. Ausgelagerte Signaturteile in einem Use-Case sind sowohl in Use-Cases als auch in Misuse-Cases referenzierbar.

Die Einführung der Inklusionspunkte, die in der UML2 nicht vorgesehen sind, erlaubt es die «include»-Beziehung in der konkreten Syntax diese Inklusionspunkte wie bei den Erweiterungspunkten der UML2 mittels eines Notizelements zu referenzieren. Diese Notiz definiert einen Filter, der beschreibt, welche eLSCs aus dem inkludierten Anwendungsfall dem Basis-Anwendungsfall zur Verfügung gestellt bzw. dort eingesetzt werden. In den eLSCs des Basis-Anwendungsfalls werden diese sichtbaren Inklusionspunkte genutzt, um einen der ausgelagerten

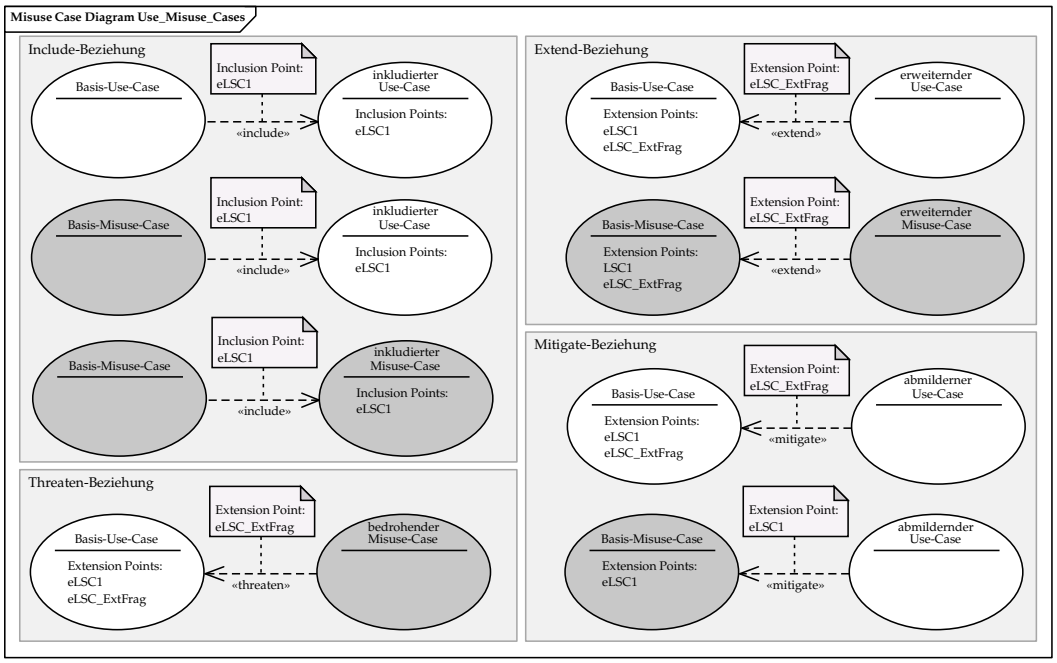


Abbildung 5.6: Erlaubte Beziehungen zwischen Use- und Misuse-Cases in konkreter Syntax

Signaturteile mit einem *Ref*-Fragment der eLSCs in den Ablauf einzubinden. Hierdurch werden die Zusammenhänge der Signaturen auf der abstrakten Ebene der MUC-Sprache explizit und somit übersichtlich representiert. Während der Laufzeit des Monitors wird die Überwachung des eLSCs im Basis-Anwendungsfall bei Erreichen des *Ref*-Fragments für die entsprechenden Lebenslinien angehalten und das inkludierte eLSC aus dem inkludierten Anwendungsfall überwacht. Nach erfolgreicher Überwachung des inkludierten eLSCs wird die angehaltene Überwachung der Lebenslinien des *Ref*-Fragments im Basis-Anwendungsfall an derselben Stelle fortgesetzt. Das Bilden von Zyklen von «include»-Beziehungen ist nicht erlaubt.

Beispiel *«include»-Beziehungen, Inklusionspunkte und Ref-Fragmente*

Im MUC-Diagramm in Abbildung 5.4 werden aus dem Basis-Use-Case `CARDME` die Teilsignaturen aus den beiden inkludierten Anwendungsfällen `Tracking` und `NonLocal` referenziert, indem jeweils eine «include»-Beziehung zwischen den Anwendungsfällen gezogen wird. Hierzu werden die in den beiden inkludierten Anwendungsfällen enthaltenen eLSC-Signaturen, `Tracking_BasicPath` und `NonLocal_BasicPath` in Abbildung 5.5, als Inklusionspunkte – `Tracking` und `NonLocal` – im jeweiligen Use-Case eingetragen. Die Inklusionspunkte, die im Basis-Use-Case `CARDME` referenziert werden, sind als Notizelement an der Beziehung notiert. Die so sichtbar gemachten Inklusionspunkte werden durch die im eLSC `CARDME_BasicPath` modellierten *Ref*-Fragmente in die Signatur eingebunden.

«extend»-Beziehungen verbinden Use-Cases mit Use-Cases und Misuse-Cases mit Misuse-Cases wie in Abbildung 5.6 dargestellt. Der Anwendungsfall am Ziel der Beziehung nennt sich *Basis-Anwendungsfall* (*engl. basic case*) und der andere *erweiternder Anwendungsfall* (*engl. extending case*). Diese Beziehung wird genutzt, um zueinander alternative Überwachungssignaturen zu modellieren. Die Modellierung alternativer Abläufe ist in der eLSC-Sprache durch das Alt-Fragment, das in Abschnitt 5.1.1 vorgestellt wurde, stark vereinfacht worden. Allerdings führt diese Modellierung bei großen oder vielen alternativen Teilsignaturen trotz allem zu unübersichtlichen Spezifikationen. Zudem ist die Wiederverwendung kompletter parallel überwachter Teilsignaturen bisher nur über die zusätzliche Auslagerung in einen weiteren Anwendungsfall und die Einbindung über das Ref-Fragment möglich. Dieser Nachteil wird durch die Einführung der «extend»-Beziehung in der MUC-Sprache behoben. Mit ihr können Alternativen zu ganzen eLSC-Signaturen bzw. Teilsignaturen der eLSCs auf einer höheren Abstraktionsebene spezifiziert werden.

Die «extend»-Beziehung kann an Erweiterungspunkte des Basis-Anwendungsfalls gebunden werden, indem, wie in der UML2 eine Notiz, mit den relevanten Erweiterungspunkten, im MUC-Diagramm an der Beziehung angebracht wird. Bedingungen für die Gültigkeit des Erweiterungspunktes, die in der UML2 direkt in den Notizelementen annotiert werden, werden in der MBSecMonSL direkt in den alternativen Signaturen als Prechart des Universal eLSCs modelliert.

Den *Erweiterungsfragmenten* der eLSCs können Erweiterungspunkte des beinhaltenden Anwendungsfalls zugeordnet werden. Während der Laufzeit des Monitors werden die alternativen eLSCs im erweiternden Anwendungsfall parallel zum eLSC des Basis-Anwendungsfalls überwacht. Wenn die Überwachung eines der alternativen eLSCs oder des Erweiterungsfragments im Basis-Anwendungsfall erfolgreich abgeschlossen ist, wird die Überwachung der alternativen eLSCs abgebrochen und die Überwachung des erweiterten eLSCs im Basis-Anwendungsfall hinter dem Erweiterungsfragment fortgesetzt. Das Bilden von Zyklen durch «extend»-Beziehungen ist wie bei der «include»-Beziehung verboten. Auch Kombinationen der beiden Beziehungen dürfen keine Zyklen bilden.

Beispiel *«extend»-Beziehungen, Erweiterungspunkte und -fragmente*

Im MUC-Diagramm in Abbildung 5.4 wird die «extend»-Beziehung zur Modellierung einer Alternative zum Mautbrückenszenario eingesetzt. Der Erweiterungspunkt `CARDME_LSC` dient im Basis-Use-Case `CARDME` als Referenz auf das enthaltene eLSC `CARDME_BasicPath` in Abbildung 5.5. Die alternative Signatur im erweiternden Use-Case `CARDME_Parking` beschreibt den Standardablauf des CARDME-Protokolls für den Einsatz zur Gebührenerfassung in einem Parkhaus. Die «extend»-Beziehung ordnet dem Erweiterungspunkt über die Notiz die alternative Signatur zu. Wird der Erweiterungspunkt nicht an das gesamte eLSC `CARDME_BasicPath`, sondern an das Mainchart des eLSCs gebunden, kann die Signatur in `CARDME_Parking` vereinfacht werden, da das Prechart gemeinsam für beide Fälle überwacht werden kann. Bei Erreichen des Maincharts des Basis-Use-Cases `CARDME` wird die Überwachung des eLSCs im erweiternden Use-Case gestartet.

«mitigate»-Beziehungen verbinden Use-Cases mit anderen Use-Cases oder Misuse-Cases wie in Abbildung 5.6 dargestellt. Der Anwendungsfall auf der Zielseite der Beziehung wird *Basis-Anwendungsfall* und der an der Quellseite *abmildernder Use-Case* (*engl. mitigating usecase*) genannt. Die Beziehung ordnet Use-Cases bzw. Misuse-Cases eine Gegenmaßnahme zu, die bei fehlgeschlagener bzw. erfolgreicher Überwachung des Basis-Anwendungsfalls ausgeführt wird. Die «mitigate»-Beziehung wird wie die «extend»-Beziehung eingesetzt und kann auch über Erweiterungspunkte an Erweiterungsfragmente angehängt werden. Hierbei wird zwischen Use-Cases und Misuse-Cases als Basis-Anwendungsfall unterschieden. Für einen Basis-Use-Case können alle Erweiterungsfragmente der eLSCs über Erweiterungspunkte referenziert werden. Der Basis-Use-Case, d. h., das entsprechende eLSC oder Erweiterungsfragment, agiert somit als Vorbedingung für die Ausführung des abmildernden Use-Cases. Im Gegensatz hierzu können in Basis-Misuse-Cases nur gesamte eLSCs über Erweiterungspunkte referenziert werden, da der Misuse-Case erfolgreich überwacht werden muss, um die Gegenmaßnahme auszulösen. Tritt diese Vorbedingung ein, d. h., die Überwachung des Erweiterungsfragments des Use-Cases schlägt fehl oder die Überwachung des eLSCs des Misuse-Cases beendet sich erfolgreich, wird das Existential eLSC des abmildernden Anwendungsfalls ausgeführt. In diesem Existential eLSC können Nachrichten an andere Teilnehmer verschickt und über Zuweisungen die Ausführung von Aktionen spezifiziert werden.

Beispiel *«mitigate»-Beziehungen, Erweiterungspunkte und -fragmente*

Die als eLSCs verfassten Misuse-Case-Signaturen in Abbildung 5.7, die in den Misuse-Cases `Init_DoS_Attack` und `Tracking_DoS_Attack` hinterlegt sind, führen bei ihrer Erkennung zur Ausführung der Gegenmaßnahme, die in dem Use-Case `CloseConnection` hinterlegt ist. Das eLSC `CloseConnection_BasicPath` sendet bei Erkennen des DoS-Angriffs eine Fehlernachricht an die OBE des Fahrzeugs und trennt die Verbindung zum Angreifer. Diese Gegenmaßnahme ist als Existential eLSC modelliert, in dem die Zuweisung einen Methodenaufruf `Ignore(Vehicle)` enthält, der auf der Mautbrücke ausgeführt wird. Um die im Misuse-Case als eLSCs spezifizierte Vorbedingung mit dem Use-Case `CloseConnection` zu verbinden, wird eine «mitigate»-Beziehung eingesetzt, die dem Erweiterungspunkt `DoS_Attack_Detected` mittels eines Notizelements zugeordnet wird. Der Erweiterungspunkt ist dem gesamten eLSC im entsprechenden Misuse-Case zugeordnet.

«threaten»-Beziehungen verbinden im Gegensatz zu «extend»-Beziehungen nur Misuse-Cases mit Use-Cases wie in Abbildung 5.6 dargestellt. Der Use-Case auf der Zielseite der Beziehung nennt sich *Basis-Use-Case* und der Misuse-Case auf der Quellseite *bedrohender Misuse-Case*. Diese Beziehung ähnelt einer «extend»-Beziehung und definiert, wann eLSCs aus dem bedrohenden Misuse-Case parallel zum annotierten Erweiterungsfragment des Basis-Anwendungsfalls überwacht werden sollen. Bedingungen werden auch wie bei «extend»-Beziehungen bei der «threaten»-Beziehung nicht in der Notiz annotiert, sondern als Prechart der enthaltenen bedrohenden Universal eLSCs modelliert.

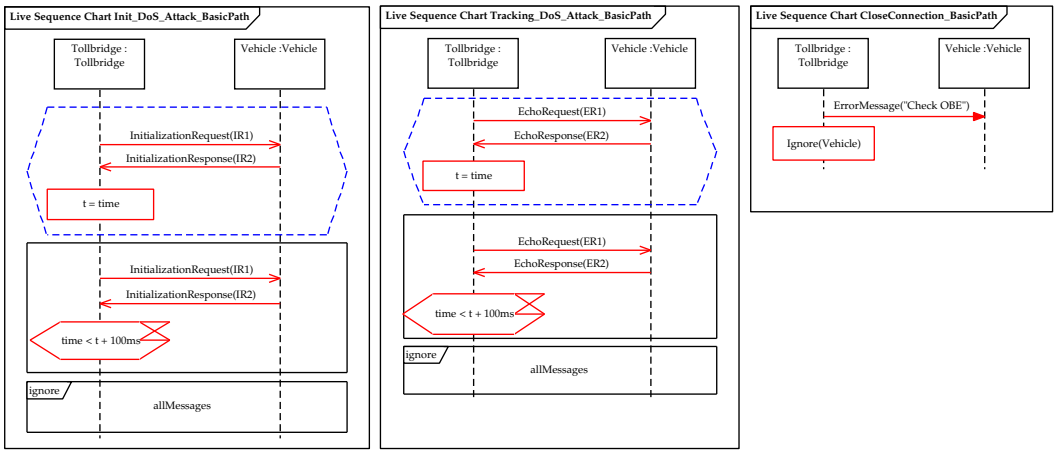


Abbildung 5.7: Angriffssignaturen des CARDME-Protokolls zur Gebührenerfassung durch die Mautbrücke als eLSC

Die «threaten»-Beziehung wird zur expliziten Modellierung des Überwachungszeitpunktes eingesetzt, an dem der Misuse-Case auftreten kann. Dies reduziert zum einen den Umfang der Precharts der eLSCs der Misuse-Cases, da nicht der gesamte Kontext, in dem die bedrohende Signatur gültig ist, im eLSC als Prechart spezifiziert werden muss. Zum anderen wird hierdurch die Generierung effizienter Monitore vereinfacht, da Zusammenhänge explizit spezifiziert sind und nicht für jedes eLSC einzeln komplexe Precharts überwacht werden müssen.

Wird die Signatur eines der bedrohenden Misuse-Cases erkannt, während die Überwachung des Basis-Use-Cases im entsprechenden Erweiterungsfragment ist, ist der Misuse-Case eingetreten. Die Gegenmaßnahmen, die über eine «mitigate»-Beziehungen mit dem erkannten bedrohenden Misuse-Case verknüpft sind, werden wie vorgestellt bei Eintreten der modellierten Misuse-Case-Signatur ausgeführt.

Beispiel *«threaten»-Beziehungen, Erweiterungspunkte und -fragmente*

Die «threaten»-Beziehung wird im MUC-Diagramm des CARDME-Beispiels zum einen zwischen dem Basis-Use-Case `CARDME` und dem bedrohenden Misuse-Case `Init_DoS_Attack` und zum anderen zwischen dem Basis-Use-Case `Tracking` und dem bedrohenden Misuse-Case `Tracking_DoS_Attack` eingesetzt. Im ersten Fall wird der bedrohende Anwendungsfall über den Erweiterungspunkt `CARDME_LSC` an das eLSC `CARDME_BasicPath` gehängt und legt somit fest, dass der Misuse-Case nur parallel zu dieser Signatur auftreten kann. Im zweiten Fall findet die Zuordnung über den Erweiterungspunkt `DuringTracking` statt, während dessen Überwachung der bedrohende Misuse-Case eintreten kann.

Überwachung der Signaturen: Bei der Überwachung einer mit der MUC-Sprache strukturierten Spezifikation, die in der MBSecMonSL verfasst ist, wird die Überwachung der Signaturen durch Auftreten der Ereignisse des überwachten Systems vorangetrieben. Nachdem alle eLSCs der Signatur das letzte Ereignis verarbeitet haben, findet die Auswertung der Signaturen statt, in der ermittelt

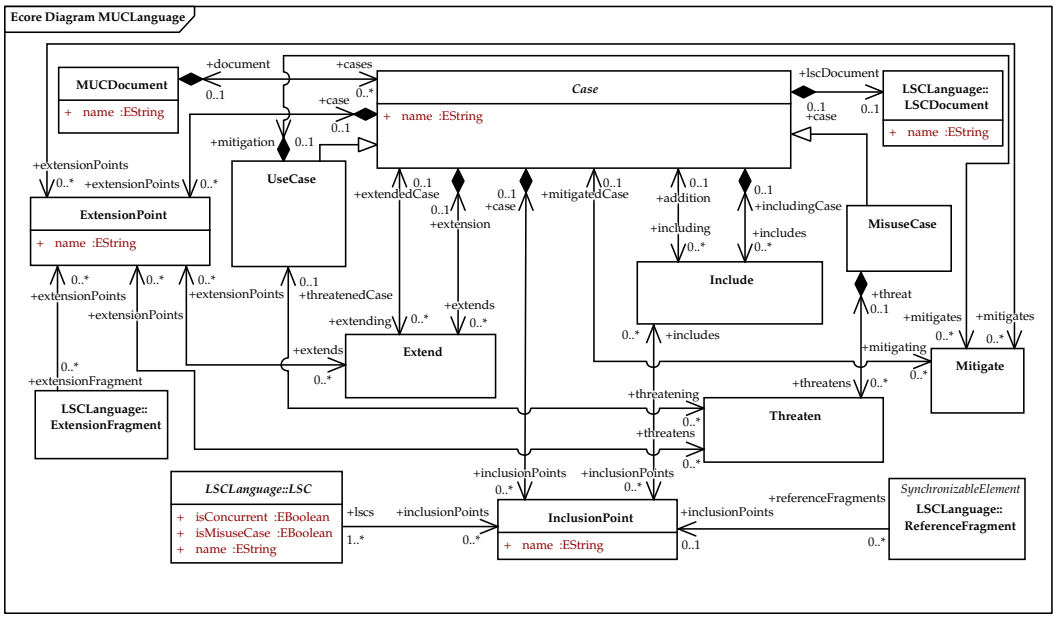


Abbildung 5.8: Metamodell der MUC-Sprache

wird, ob die Signatur noch überwacht wird, sich beendet hat oder fehlgeschlagen ist. Hierbei wird zwischen Use-Cases und Misuse-Cases unterschieden. Ein Misuse-Case löst eine Gegenmaßnahme aus, die ihm über eine «mitigate»-Beziehung zugeordnet ist, wenn die Signatur erkannt wurde. Im Gegensatz hierzu muss ein Use-Case fehlgeschlagen sein, damit seine ihm zugeordneten Gegenmaßnahme ausgelöst wird.

Solange nur ein Use-Case oder Misuse-Case eine Gegenmaßnahme auslöst, kann diese einfach ausgeführt werden. Sobald jedoch einem Use-Case und einem Misuse-Case, der den Use-Case bedroht, jeweils eine Gegenmaßnahme zugeordnet ist, und beide gleichzeitig fehlschlagen, muss entschieden werden, welche Gegenmaßnahme zuerst angewendet werden soll. Hierbei sieht die MBSecMonSL vor, dass zunächst Gegenmaßnahmen der Misuse-Cases ausgeführt werden, da Misuse-Cases explizit modellierte verbotene Ereignissequenzen beschreiben. Erst danach werden die den Use-Cases zugeordneten Gegenmaßnahmen ausgeführt. Somit findet die Ausführung der ausgelösten Gegenmaßnahmen von den Blättern des MUC-Diagramms (Anwendungsfälle mit nur auslaufenden Beziehungen) zu den Wurzeln statt. «include»-Beziehungen werden hierbei rückwärts navigiert.

5.2.2 *Die abstrakte Syntax der MUC-Sprache*

Nach der Präsentation der konkreten Syntax und Semantik der MUC-Sprache in Abschnitt 5.2.1 wird in diesem Abschnitt die abstrakte Syntax der MUC-Sprache anhand seines Metamodells in Abbildung 5.8 vorgestellt. Als übergeordneter Behälter eines MUC-Diagramms dient die Klasse `MUCDocument`, die alle Use- und Misuse-Cases enthält. Diese sind als Ausprägungen der abstrakten Klasse `Case` modelliert und werden durch die Klassen `UseCase` und `MisuseCase` repräsentiert.

Die Klasse `Case` dient als Behälter für eine Instanz der Klasse `LSCDocument` aus dem Metamodell der eLSC-Sprache, die in Abbildung 5.2 vorgestellt wurde. Die Verwendung der Klasse `LSCDocument` ermöglicht es beliebig viele eLSCs in einem `Case` einzuhängen.

Des Weiteren kann jede Instanz der Klasse `Case` beliebig viele Instanzen der Klassen `ExtensionPoint` und `InclusionPoint` beinhalten, die Erweiterungs- und Inklusionspunkte repräsentieren. `ExtensionPoints` werden durch Erweiterungsfragmente des eLSC-Metamodells (`ExtensionFragment`) referenziert. Diese Erweiterungsfragmente müssen vom selben Anwendungsfall gehören wie die Erweiterungspunkte selbst. `InclusionPoints` werden von `LSCs` der eLSC-Sprache referenziert. eLSCs können unter einem Inklusionspunkt gruppiert oder Inklusionspunkte können genutzt werden, um eine einzelne Teilsignatur im MUC-Diagramm zu visualisieren. Ein *Ref*-Fragment (`ReferenceFragement`) der eLSC-Sprache referenziert einen Inklusionspunkt (`InclusionPoint`) des inkludierten Anwendungsfalls, um das eLSC anzugeben, das das Ref-Fragment im eLSC des Basis-Anwendungsfalls ersetzen soll.

Use-Cases bzw. Misuse-Cases können jeweils untereinander mit den aus der UML2 bekannten «extend»-Beziehungen verbunden werden und selbst Erweiterungspunkte (`ExtensionPoint`) enthalten. Die Klasse `Extend` repräsentiert diese «extend»-Beziehung, die durch den `Case` auf der Quellseite der Beziehung besessen wird und beliebig viele Instanzen der Klasse `ExtensionPoint` referenziert. Die «include»-Beziehung wird durch die Klasse `Include` repräsentiert.

Misuse-Cases (`MisuseCase`) sind analog zu den Use-Cases modelliert und erben die Referenzen zu den Klassen `Extend` und `Include` von der abstrakten Elternklasse `Case`. Zusätzlich repräsentiert die Klasse `Threaten` die «threaten»-Beziehung, die wie die Klasse `Extend` Erweiterungspunkte referenzieren kann. Die «mitigate»-Beziehung wird durch die Klasse `Mitigate` repräsentiert und kann ebenfalls an Erweiterungspunkte gebunden werden.

5.2.3 *Zusammenfassung: Einbettung von eLSCs in die MUC-Sprache*

Universal eLSCs werden zur Beschreibung von zu überwachenden Nachrichtensequenzen und Bedingungen verwendet. Hierdurch eignen sich die eLSCs zur Beschreibung von Use-Cases und Misuse-Cases. Eine Ausnahme bilden hier die abmildernden Use-Cases, die als Gegenmaßnahmen zu erkanntem Fehlverhalten ausgeführt werden. Die Feststellung dieses Fehlverhaltens basiert auf dem Ergebnis der Überwachung des abgemilderten Anwendungsfalls. Wenn der überwachte Misuse-Case erkannt wurde bzw. der abzumildernde Use-Case fehlgeschlagen ist, wird die als Existential eLSC modellierte Gegenmaßnahme ausgeführt. Der Erfolg der Ausführung der Gegenmaßnahme kann wiederum durch einen Misuse-Case, der dem abmildernden Use-Case über eine «threaten»-Beziehung zugeordnet ist, überwacht werden. Auf dessen Fehlschlagen kann mit einer auf diesen Fall spezialisierten Gegenmaßnahme reagiert werden. Die Precondition des Universal eLSCs wird nicht benötigt, da die Vorbedingung für die Gegenmaßnahme durch die Überwachung des abzumildernden Anwendungsfalls gegeben ist.

In Universal eLSCs können Erweiterungspunkte des zugehörigen Anwendungsfalls durch Erweiterungsfragmente referenziert werden. Diese Fragmente geben den Zustand der Überwachung bzw. den Bereich im eLSC an, in dem der erweiternde Anwendungsfall auftreten kann und überwacht werden muss. Durch die Verknüpfung der «extend»- und «threaten»-Beziehung und der Erweiterungsfragmente der eLSCs zu den Erweiterungspunkten, ist eine Verbindung zwischen eLSCs in den verschiedenen Anwendungsfällen realisiert. Inklusionspunkte eines Anwendungsfalls referenzieren ein oder mehrere eLSCs desselben inkludierten Anwendungsfalls und werden an der «include»-Beziehung annotiert. Ref-Fragmente der eLSCs im Basis-Anwendungsfall referenzieren diese inkludierten Anwendungsfälle, die an der Beziehung annotiert sind, und binden so diese Teilsignaturen ein.

Die vorgestellte MUC-Sprache erweitert die eLSC-Sprache um die Strukturierung der Spezifikation und die explizite Beschreibung der Beziehung zwischen eLSCs. Die Erweiterungspunkte stellen dem Entwickler ein Konzept zur Verfügung, um eLSCs an Erweiterungsfragmente anderer eLSCs zu binden. Hierdurch wird die Anzahl und die Komplexität der Vorbedingungen der eLSCs reduziert, die als Precondition modelliert und während der Laufzeit im generierten Monitor evaluiert werden muss. Außerdem zieht der Entwickler den Vorteil aus dem Konzept, dass er eine strukturierte Spezifikation mit einer klaren Präsentation der Beziehungen zwischen den eLSCs erstellen kann.

Die Erweiterung der eLSCs zur MBSecMonSL führt zu einer verbesserten Modellierbarkeit der verhaltensbeschreibenden Signaturen durch die Modellierung der Abhängigkeiten zwischen den einzelnen Signaturen auf einer höheren Abstraktionsebene. So können Teilsignaturen transparent in Use-Cases bzw. Misuse-Cases ausgelagert werden, um in mehreren Signaturen über die «include»-Beziehung wiederverwendet zu werden und durch die «extend»-Beziehung die Anzahl der redundanten Vorbedingungen zu reduzieren. Dies führt zu einer besseren Skalierbarkeit und Wartbarkeit der Monitorsignaturspezifikation. Zusätzliche können Anforderungen, die in der Anforderungsanalyse des Entwicklungsprozesses des zu überwachenden Systems nach dem Konzept des szenariobasierten Designs erstellt wurden, als Use-Cases in den Monitorspezifikationsprozess einfließen.

6

DER EDITOR (ENTERPRISE ARCHITECT ADD-IN)

Der Editor zur Modellierung der Spezifikationen in der MBSecMon-Spezifikationssprache (MBSecMonSL) wurde als Erweiterung des UML2-Modellierungswerkzeuges SparxSystems Enterprise Architect (EA) entwickelt. EA unterstützt zwar bereits nativ sowohl die Modellierung von UML2-Use-Case-Diagrammen als auch die Modellierung der in ihnen eingebettete Sequenzdiagramme, jedoch fehlen Modellierungselemente der Misuse-Cases und der eLSCs. Diese notwendigen Erweiterungen werden in Abschnitt 6.1 vorgestellt und in Abschnitt 6.2 auf den Export in ein weiterverarbeitbares Format eingegangen.

6.1 MBSecMonSL-Editor

Zur Definition zusätzlicher Sprachkonstrukte und zur Erweiterung seiner Funktionalität bietet EA zwei Mechanismen an: UML-Profile (Abschn. 2.2.2) dienen zur Definition neuer bzw. angepasster Diagrammtypen und Modellierungselemente, die in EA eingebunden werden können. Zusätzlich finden über einen Add-in-Mechanismus die Einbindung des definierten Profils und die Erweiterung der Funktionalität von EA statt.

6.1.1 *MBSecMon-Profile*

Zur Anpassung des UML2-Modellierungswerkzeuges Enterprise Architect werden zwei UML-Profile – das MUC-Profil und das eLSC-Profil – in EA modelliert. Diese werden anschließend in einer auf dem XML-Format basierenden *Model Driven Generation*-Technologiedatei zusammengefasst und diese in EA eingebunden. Das eLSC-Profil basiert auf den in EA bereitgestellten UML2-Sequenzdiagrammen und definiert auf Basis der Metaklassen der Sequenzdiagramme die Elemente der eLSCs. Im Gegensatz hierzu erweitert das MUC-Profil die Metaklassen der UML2-Use-Case-Diagramme.

Das EA-Projekt, in dem das Profil der eLSC- und das Profil der MUC-Diagramme beschrieben wird, enthält, wie in Abbildung 6.1 gezeigt, für jedes Profil ein übergeordnetes Paket. Diese enthalten jeweils ein mit dem Stereotyp «profile» versehenes Pakete beinhalten, das als Behälter für die Profile dient. Diese Profile werden als UML2-Klassendiagramm modelliert. Es existieren drei Varianten der Profile:

- Profile, die neue Diagrammtypen definieren,
- Profile, die neue Elemente definieren und

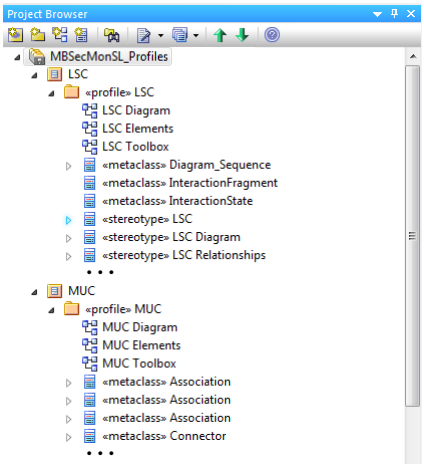


Abbildung 6.1: Struktur der UML-Profile der MBSecMonSL

- Profile, die den Aufbau der Toolbox beschreiben, in der die definierten Elemente auf der Benutzeroberfläche zur Verfügung gestellt werden.

Alle drei Varianten werden mit der in Abschnitt 2.2.2 vorgestellten Notation für UML-Profile in EA modelliert.

eLSC-UML-Profil: Den Hauptteil des eLSC-UML-Profils bilden die neu definierten Stereotypen im Klassendiagramm `LSC Elements` in Abbildung 6.2. Hierzu werden die Metaklassen, die EA für die UML2-Sequenzdiagramme bereitstellt, mit Stereotypen erweitert. Diese für die eLSC-Elemente genutzten Metaklassen sind:

- `Object`, die UML2-Objekte und somit die Lebenslinien in Sequenzdiagrammen repräsentieren,
- `Message`, die die Nachrichten repräsentieren,
- `InteractionFragment`, die Interaktionsfragmente repräsentieren, und
- `InteractionState`, die in EA u. a. für Fortsetzungsmarken eingesetzt werden.

In den Stereotypklassen werden verschiedene Eigenschaften (*engl. tagged values*) zur Konfiguration des EA-Editors und der Repräsentation der Elemente in EA gesetzt. In `_image` kann mit einem sogenannten *ShapeScript* die grafische Darstellung eines Elements in einem Diagramm angepasst werden. Die Eigenschaften `_sizeX` und `_sizeY` geben die Standardgröße der Elemente im Diagramm an und `_metatype` legt fest, unter welchem Namen das Element auf der Benutzeroberfläche von EA (z. B. in Eigenschaftsdialogen) angezeigt wird.

Neben diesen neuen Elementen der eLSCs muss zusätzlich noch ein neuer Diagrammtyp in EA zur Verfügung gestellt werden. Das Klassendiagramm `LSC Diagramm` in Abbildung 6.3 definiert einen neuen Stereotypen «LSC Diagram», der die Metaklasse `Diagram_Sequence`, die in EA die Basisklasse der Sequenzdiagramme darstellt, erweitert. Wie auch die neu definierten Elemente wird dem eLSC-Diagramm über das Attribut `_metatype` der Name `LSC Diagramm` zur Repräsentation in EA zugeordnet. Durch die Verwendung der Metaklasse für Se-

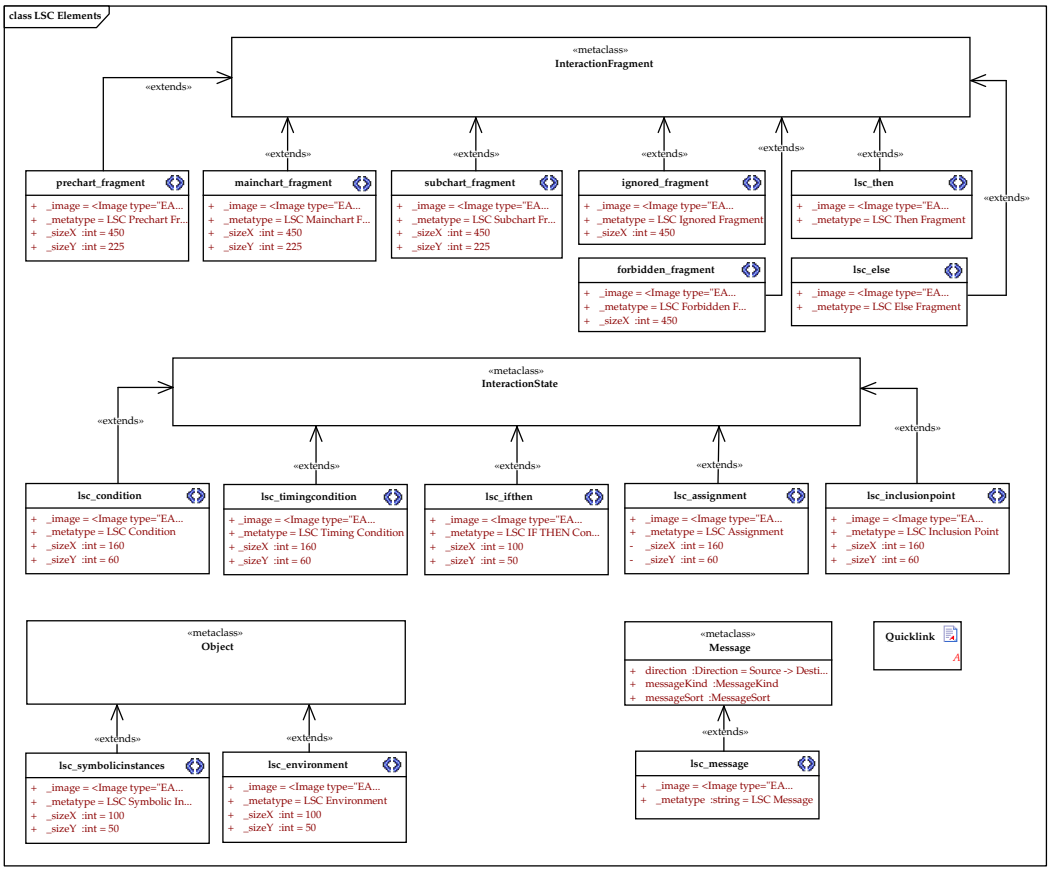


Abbildung 6.2: Das Profil der speziellen eLSC-Elemente

quenzdiagramme (`Diagram_Sequences`) werden deren Eigenschaften wie z. B. das spezielle automatische Layout der Diagramme in das neu definierte eLSC-Diagramm übernommen. Zur Anpassung des Namens und des Identifizierers werden die Attribute `alias`, `diagrammID` der Metaklasse mit den Werten, die die neue Diagrammart beschreiben, initialisiert. Das Attribut `toolbox` ordnet über die Namensgleichheit den neuen definierten eLSC-Diagrammen ein weiteres Profil zu, das die zugehörige Toolbox in Abbildung 6.4 definiert.

Das Profil der Toolbox beschreibt, welche Elemente und Beziehungen der eLSC-Sprache dem Nutzer beim Öffnen eines eLSC-Diagramms zur Verfügung steht. Als Basisklasse, die durch zwei Stereotype erweitert wird, dient die von EA bereitgestellte Metaklasse `ToolboxPage`. Die Namen `LSC` und `LSC Relationships` dieser beiden Stereotypen entsprechen bei diesem Profil den Namen, mit dem die Unterseite (*engl. subpage*) der Toolbox beschriftet wird. Der Inhalt dieser Seiten (Modellierungselemente und Beziehungen) wird durch Attribute bestimmt, deren Namen dem vollständigen Namen des Elements in EA (siehe [Spa13]) entspricht und der Initialwert gibt den angezeigten Namen in der Toolbox an. So wird die aus der UML2 wiederverwendete Lebenslinie durch den Attributnamen `UML::Lifeline` referenziert und ihr der neue Anzeigenamen `LSC Lifeline` zugeordnet. Alle weiteren Elemente der eLSC-Sprache, die in der Toolbox angezeigt werden, stammen aus dem Profil `LSC Elements`. Ihr Attributname setzt sich aus

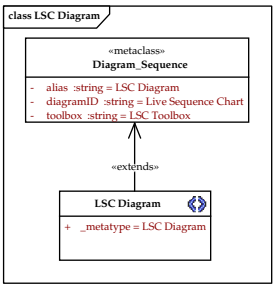


Abbildung 6.3: Das Profil des eLSC-Diagramms

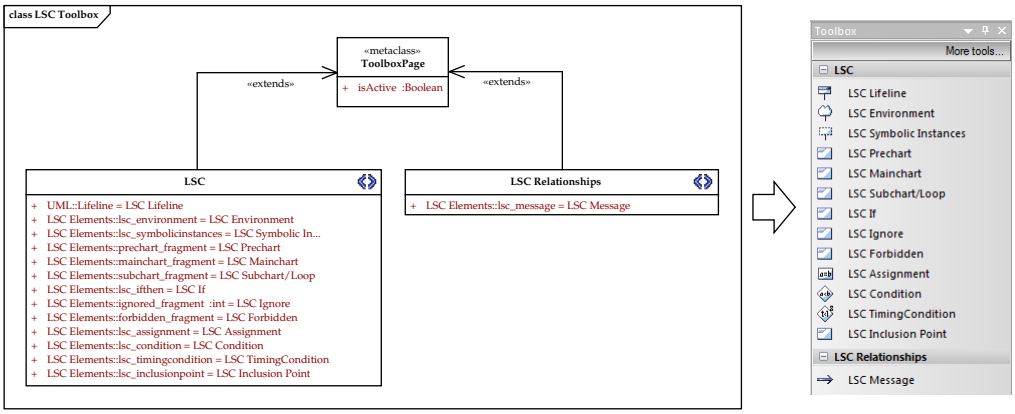


Abbildung 6.4: Das Profil der eLSC-Toolbox

dem Präfix `LSC Elements::` (Diagrammname des Profils gefolgt von „::“) und dem Namen des Stereotyps zusammen. Auch hier legt der Initialwert den Anzeigenamen des Elements in der Toolbox fest. So wird die neue Variante der Lebenslinie, die die Umgebung repräsentiert, durch das Attribut mit dem Namen `LSC Elements::lsc_environment` eingebunden.

MUC-UML-Profil: Das Profil für die MUC-Sprache ist in drei Klassendiagrammen `MUC Elements`, `MUC Diagram` und `MUC Toolbox` (Abb. 6.5 bis 6.7) modelliert. Die Use- und Misuse-Cases der MUC-Sprache basieren auf der von EA bereitgestellten Metaklasse `UseCase` und erweitern diese Klasse mit den Stereotypklassen `muc_usecase` und `muc_misusecase`. Die beiden zusätzlichen Beziehungen zwischen Use- und Misuse-Cases – «threaten» und «mitigate» – werden durch die Stereotypklassen `mitigate` und `threaten` realisiert. Zusätzlich wird ein Stereotyp `muc_interaction` definiert, der Elemente als Behälter für eLSC-Diagramme markiert. Wie in der UML2-Modellierung von Use-Cases können ein oder mehrerer dieser Behälter als untergeordnete Elemente in die Use- und Misuse-Cases eingehängt werden. Sie enthalten jeweils ein eLSC-Diagramm.

Wie auch im eLSC-UML-Profil wird für die MUC-Diagramme eine eigene Diagrammart durch das Klassendiagramm `MUC Diagram` in Abbildung 6.6 definiert. Hierfür erweitert der Stereotyp `MUC Diagram` die von EA bereitgestellte Metaklasse `Diagram_UseCase`, auf der auch die UML2-Anwendungsfälle in EA basieren. Durch Setzten der Attribute `alias`, `diagramID` und `toolbox` der Metaklasse und

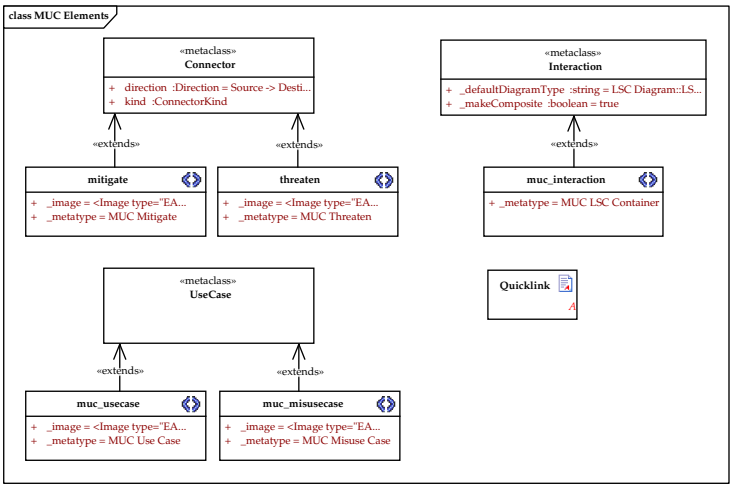


Abbildung 6.5: Das Profil der speziellen MUC-Elemente

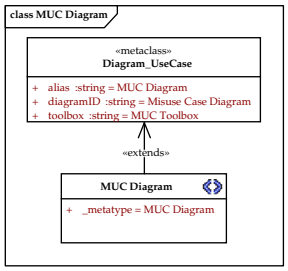


Abbildung 6.6: Das Profil des MUC-Diagramms

dem Attribut `_metatype` der Stereotypklasse, wird das neue Diagramm als MUC-Diagramm definiert.

Die Elemente und Beziehungen, die im MUC-UML-Profil definiert sind, werden in einer Toolbox bereitgestellt. Hierfür wird die Metaklasse `ToolboxPage` mit zwei Stereotypklassen erweitert, die die Unterseiten `MUC` und `MUC Relationships` der Toolboxseite beschreiben. Auch hier werden in EA vorhandene Elemente der UML2 unter einem neuen Namen wiederverwendet. Diese sind der Aktor (`UML::Actor`), die Begrenzung des modellierten Systems (`UML::Boundary`), die «include»- und «extend»-Beziehung (`UML::UCInclude`, `UML::UCExtend`) und die Assoziation (`UML::Association`). Für die von den Use-Case-Diagrammen der UML abweichenden Elemente und Beziehungen werden die neu definierten Stereotypklassen aus dem Profil `MUC Elements` in die Toolbox eingebunden.

MDG-Technologie: Zur Einbindung dieser beiden Profile in EA stellt EA die *Model Driven Generation*-Technologie (MDG) zur Verfügung. Zunächst werden alle sechs Klassendiagramme – `LSC Elements`, `LSC Diagram`, `LSC Toolbox`, `MUC Elements`, `MUC Diagram` und `MUC Toolbox` – als UML-Profile in ein XML-Format exportiert. Diese einzelnen XML-Repräsentationen der Diagramme werden durch den *MDG Technology Creation Wizzard* von EA in eine Technologie-Datei gewandelt. Diese Datei lässt sich dann direkt oder durch ein Add-in in EA einbinden. Sie

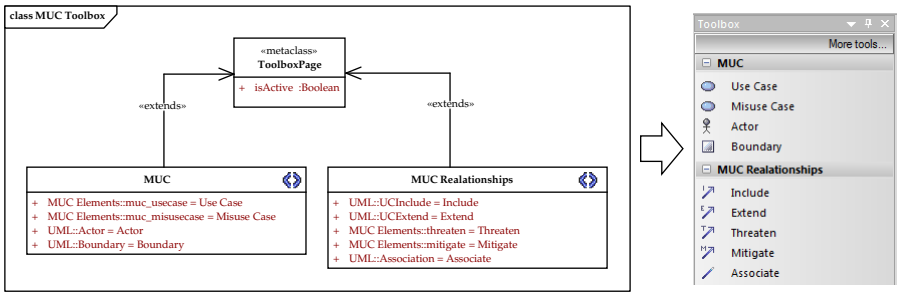


Abbildung 6.7: Das Profil der MUC-Toolbox

fügt in EA die Elemente und Beziehungen der MUC- und eLSC-Diagramme ein, stellt die MUC- und eLSC-Diagramme in der Diagrammauswahl zur Verfügung und erweitert die Toolbox von EA um Seiten für die Modellierung von MUC und eLSC-Diagrammen.

6.1.2 *Das MBSecMon-EA-Add-in*

Die im vorherigen Abschnitt vorgestellten UML-Profile verändern das Aussehen und die Bezeichnung der UML-Elemente und Diagramme. Das Verhalten von EA, spezialisierte Eigenschaftsfenster für die neuen Elemente und Beziehungen und weitergehende Anpassung der Benutzeroberfläche werden von der MDG-Technologie nicht unterstützt. Hierfür bietet EA eine Schnittstelle – das *Automation Interface* – an, die mittels eines Add-ins angesprochen wird. Über das in der Programmiersprache C# verfasste Add-in kann sowohl die Benutzeroberfläche angepasst als auch auf Eingaben des Modellierers auf der Benutzeroberfläche von EA reagiert werden. Das MBSecMon-EA-Add-in basiert auf dieser Technik.

Struktur und Funktionsweise des Add-ins: Der Einstiegspunkt des MBSecMon-EA-Add-ins wird durch eine *Main*-Klasse definiert, in der Methoden implementiert werden, die EA über ein *Listener*-Konzept ausführt. Hierzu löst EA bei jeder Benutzeraktion ein Ereignis aus, das zur Ausführung der entsprechenden Methode in der Main-Klasse führt. Diese Methoden beginnen mit dem Präfix `EA_`.

Laden der MBSecMonSL-Technologie: Wird EA gestartet, wird als erstes die `EA_Connect`-Methode, die sich in der Main-Klasse des Add-ins befindet, ausgeführt. Hierbei wird an EA ein Bezeichner des Add-ins als String zurückgegeben. Beim Beenden führt EA die `EA_Disconnect`-Methode jedes Add-ins aus, in der der *Garbage-Collector* aufgerufen werden muss, um alle nicht freigegebenen Referenzen des Add-ins zu lösen. Nachdem sich EA zum Add-in verbunden hat, wird die Methode `EA_OnInitializeTechnologies` der Main-Methode ausgeführt. Diese lädt die in Abschnitt 6.1.1 vorgestellte MDG-Technologie-Datei in einen *Stream* und übergibt sie EA als *String*.

Interaktion mit dem Benutzer: Weitere *Listener*-Methoden können im Add-in implementiert werden, um auf Benutzeraktionen zu reagieren. Zur Anpassung der Modellierung in EA kommen hauptsächlich Methoden zum Einsatz, die:

- vor der Erstellung von Objekten oder Eigenschaften der Objekte in EA aufgerufen werden (`EA_OnPreNew...`).
- nach der Erstellung von Objekten oder Eigenschaften der Objekte in EA aufgerufen werden (`EA_OnPostNew...`).
- vor dem Löschen von Objekten oder Eigenschaften der Objekte in EA aufgerufen werden (`EA_OnPreDelete...`).
- nach Veränderungen der Auswahl der Elemente in EA oder der Eigenschaften eines Objektes ausgeführt werden (`EA_OnContextItemDoubleClicked`, `EA_OnNotifyContextItemModified`).

Durch die Methoden `EA_OnPreNew...` wird entschieden, ob die Benutzeraktion – das Erstellen eines Objektes – im aktuellen Kontext abgebrochen oder durchgeführt wird. Die `EA_OnPostNew...`-Methode entscheidet anhand des erstellten Objektes, welche Eigenschaften des Objektes initialisiert oder verändert werden müssen. Zusätzlich wird diese Methode genutzt, um den nativen Eigenschaftsdialog von EA zu unterdrücken und ihn durch eine angepasste Variante zu ersetzen.

Beispiel *Nachbearbeitung einer Bedingung der eLSCs*

Wird ein Bedingungselement der eLSC-Sprache in EA durch den Modellierer einem Diagramm hinzugefügt, würde sich ohne die Implementierung der Methode `EA_OnPostNewElement` der Standard-Eigenschaftsdialog öffnen. Der in Auflistung 6.1 dargestellte Ausschnitt der Implementierung der Methode bewirkt, dass falls der Metatyp des Elements einer Bedingung entspricht, der Eigenschaftsdialog von EA unterdrückt wird (Zeile 8-9). In Zeile 10-14 wird ein TaggedValue `temp` angelegt, der die Temperatur des Elements repräsentiert, und mit `"cold"` initialisiert. Abschließend öffnet die Methode einen angepassten Eigenschaftsdialog (Zeile 15-18).

Auflistung 6.1: Ausschnitt der EA_OnPostNewElement-Methode für eLSCs

```
1  public void EA_OnPostNewElement(EA.Repository Repository, EA.EventProperties Info)
2  {
3      // Typ des ausgewählten Objekts ermitteln
4      EA.Element object = Repository.GetElementByID(int.Parse((String)Info.Get(0).Value));
5      String type = object.MetaType;
6      if (type == HelperMethods.LSC_CONDITION_METATYPE || ...)
7      {
8          // Standard-Eigenschaftsdialog von EA unterdrücken
9          Repository.SuppressEADialogs = true;
10         // Eigenschaft setzen: Temperatur des Objekts
11         EA.TaggedValue tag = ((EA.TaggedValue)object.TaggedValues.AddNew("temp", "cold"));
12         tag.Update();
13         object.TaggedValues.Refresh();
14         object.Update();
15         // Angepassten Eigenschaftsdialog öffnen
16         Form prop =
               new PropertyDialog(Repository, object.ElementGUID, EA.ObjectType.otElement);
17         prop.ShowDialog();
18     }
19     ...
```

Da das Add-in nicht nur auf die Neuerstellung von Elementen, sondern auch auf andere Benutzereingaben reagieren soll, die das Diagramm verändern, wer-

den die Methoden `EA_OnContextItemDoubleClicked` und `EA_OnNotifyContext-ItemModified` implementiert. `EA_OnContextItemDoubleClicked` repräsentiert hierbei einen Doppelklick auf ein Element, wodurch EA der Eigenschaftsdialog des Objektes öffnet. Um auch den angepassten Dialog wie bei der Neuerstellung zu öffnen, wird bei der Implementierung der Methode wie in Auflistung 6.1 vorgegangen. Auf Änderungen, die z. B. im Eigenschaftsdialog durch den Modellierer durchgeführt werden, kann mit der Methode `EA_OnNotifyContextItemModified` reagiert werden.

Zusätzliche Funktionen werden dem Modellierer über Menüeinträge zur Verfügung gestellt. Hierzu werden über die Implementierung der `EA_GetMenuItems`-Methode Haupt- bzw. Kontextmenüeinträge der Benutzeroberfläche hinzugefügt, die die Modellierung in der MBSecMonSL erleichtern. Die Implementierung der Methode `EA_GetMenuState` legt beim Öffnen des Menüs fest, welche Einträge im aktuellen Kontext der Benutzeroberfläche angezeigt werden sollen. Sobald der Benutzer einen Menüeintrag auswählt, wird die in der Methode `EA_MenuClick` hinterlegte Aktion ausgeführt. Eine solche Funktion stellt auch der Export der in der MBSecMonSL modellierten Signaturen in ein Ecore-kompatibles XML-Format, das dem Metamodell aus Abschnitt 5.1.2 entspricht, dar.

6.2 Export in ein auf EMF basierendes XMI-Format

Zur Weiterverarbeitung der in der MBSecMonSL verfassten Signaturen in einem auf EMF (Abschn. 2.2.3) basierenden Werkzeug müssen diese Signaturen als eine XMI-Instanz der MBSecMonSL-Metamodelle aus Abschnitt 5.2.2 und 5.1.2 exportiert werden. Dieser Export ist in drei Schritte unterteilt (Abb. 6.8) – die Analyse der eLSC-Diagramme, die Überführung der MUC- und eLSC-Diagramme in ein Zwischenmodell und die Serialisierung in das XMI-Format.

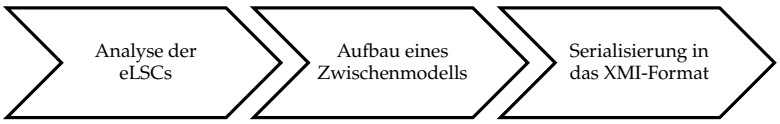


Abbildung 6.8: Export der MBSecMonSL-Signaturen aus EA

Analyse der eLSCs: Sequenzdiagramme in EA im Allgemeinen und somit auch die eLSC-Diagramme besitzen eine explizite Reihenfolge der Nachrichten auf den Lebenslinien. Werden allerdings neben Nachrichten auch Subcharts eingesetzt, die wiederum Nachrichten und weitere Subcharts enthalten, wird diese Schachtelung ausschließlich als Positionsdaten der Diagrammobjekte in EA gespeichert. Für den Export ist diese Schachtelung jedoch essenziell, um die Reihenfolge der zu exportierenden Elemente festzulegen. Hierfür findet im ersten Schritt des Exports eine Analyse über die Positionierung der Elemente im Diagramm statt.

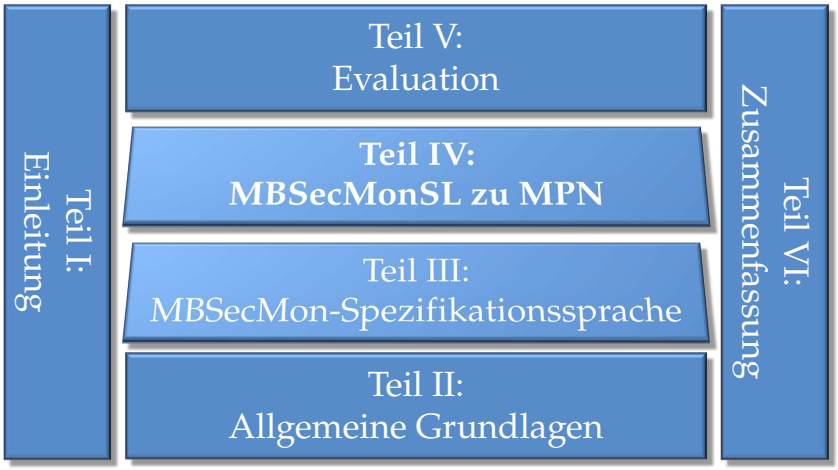
Erzeugung des Zwischenmodells: Anhand dieser Positionierungsinformationen wird im zweiten Schritt ein einfaches Zwischenmodell angelegt, das alle Informationen, die zur Serialisierung benötigt werden enthält. In diesem Schritt

wird das Zwischenmodell mit zusätzlichen Informationen, die in EA nicht explizit modelliert sind, angereichert. So wird z. B. für jede Nachricht, jede Bedingung, jede Zuweisung und jedes Pre-, Main- und Subchart zusätzliche Locations angelegt, um diese im nächsten Schritt serialisieren zu können. Das Ecore-XMI-Format verwendet zur Referenzierung anderer Elemente (Konten in der erstellten XMI-Datei) Pfadausdrücke, ausgehend vom Quellknoten des Dokuments über die Kindknoten bis zum Knoten des referenzierten Elements. Diese Knotennamen bestehen in den Pfadausdrücken aus dem Namen des Knotens und der Position unter ihrem Elternknoten. Somit wird beim Aufbau des Zwischenmodells auch diese Information anhand des Erstellungszeitpunktes automatisch abgespeichert.

Serialisierung in XMI-Format: Für die Serialisierung besitzt jede Klasse des Zwischenmodells eine Methode `SerializeToXMI`, die eine XMI-Repräsentation der Instanz der Klasse in den bestehenden XMI-Baum einhängt. Diese vollständige Repräsentation der Spezifikation wird dann in eine Datei mit der Endung `.xmi` serialisiert.

Teil IV

MBSECMONSL ZU MONITOR-PETRINETZE



7

TRANSFORMATIONEN VON SEQUENZDIAGRAMMEN ZU AUTOMATEN

Die in der in Kapitel 5 informell eingeführten MBSecMonSL verfassten Monitorsignaturen dienen als Spezifikation für die Generierung von Laufzeitmonitoren durch den MBSecMon-Prozess. Zur Vorbereitung einer einfacheren, direkteren Codegenerierung findet, wie in Abbildung 7.1 dargestellt, eine Transformation (Def. 7) der Signaturen in die Zwischensprache MPN statt. Diese formal definierte Sprache vereinfacht zum einen die Codegenerierung durch ihre explizite Repräsentation der Signaturen und dient zum anderen der Formalisierung der Semantik der MBSecMonSL.

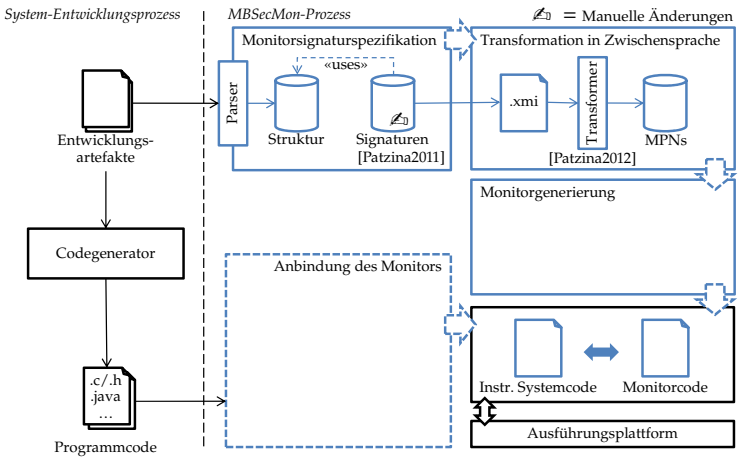


Abbildung 7.1: MBSecMon-Prozess: Übersetzung in die Zwischensprache

Definition 7 (*Modelltransformation*)**.** Eine Modelltransformation stellt eine syntaktische Umformung eines Modells oder eine automatische Abbildung eines Quellmodells auf ein Zielmodell dar. Hierzu müssen sowohl Quell- als auch Zielmodell formal definiert sein und in einem maschinenlesbaren Format vorliegen.

In diesem Kapitel werden in Abschnitt 7.1 die Transformationssprachen *Atlas Transformation Language* (ATL) und *Story Driven Modelling* (SDM) eingeführt. Diese beiden Sprachen bilden die Grundlage für die Evaluation ihrer Einsetzbarkeit zur konkreten Implementierung der Transformation zwischen der MBSecMonSL und den MPNs in Kapitel 9. In Abschnitt 7.2 werden verwandte Arbeiten vorgestellt, in denen Transformationen zur Formalisierung bzw. Verifikation eingesetzt werden,

und nach ihrer Einsetzbarkeit im MBSecMon-Prozess bewertet. Hierbei wird neben der Übersetzung in Automaten insbesondere auf die in diesem Umfeld häufig eingesetzten Temporalen Logiken und Übersetzungen von Sequenzdiagrammen in diese Temporalen Logiken eingegangen.

7.1 Transformationssprachen

Transformationssprachen sind Computersprachen, die der Formulierung einer Transformation eines Eingabedokuments in ein modifiziertes Zieldokument dienen. Hierbei ist das Eingabedokument in einer formalen Sprache verfasst und die Ausgabe dient einem bestimmten Zweck. In der Modellgetriebenen Softwareentwicklung (MDE) werden zum automatisierten Übergang zwischen den verschiedenen Abstraktionsebenen der Modelle (plattformunabhängige zu plattformabhängige Modelle bis hin zur Codegenerierung) Transformationssprachen genutzt, die als Eingabe und Ausgabe Modelle erzeugen, die zu Metamodellen konform sind. Im metamodellorientierten Ansatz werden die Beziehungen zwischen Quell- und Zielmodell durch ein gemeinsames Meta-Metamodell (Abschn. 2.2) repräsentiert.

Eine Menge an Transformationsregeln definiert hierbei die Transformation und beschreibt somit, wie Elemente des Quellmodells in Elemente des Zielmodells übersetzt werden. Hierbei bestehen die Transformationsregeln immer aus einem linken Teil (*engl. Left-Hand-Side (LHS)*), der sich auf Elemente des Quellmodells bezieht, und einem rechten Teil (*engl. Right-Hand-Side (RHS)*), der sich auf Elemente des Zielmodells bezieht.

Eine spezielle Variante aktueller Transformationssprachen sind regelbasierte Modelltransformationssprachen wie die ATLAS Transformation Language (ATL) und das Story Driven Modeling (SDM). Diese beiden Sprachen unterscheiden sich grundlegend in der Formalisierung und in der Modellierung der Regeln voneinander. In ATL, als eine in der Eclipse-Community weit verbreitete Sprache, wird die Anwendungsreihenfolge der hauptsächlich deklarativen Regeln implizit anhand der Quellmuster (LHS), die durch OCL-Ausdrücke auf Attributen bedingt werden, geplant. Das SDM sieht hingegen eine explizite Modellierung des Kontrollflusses vor. Auf diese und weitere Unterschiede der beiden Sprachen, die Einordnung zu anderen Transformationssprachen und die Einsetzbarkeit im MBSecMon-Entwicklungsprozess geht Abschnitt 9.2 ein. Im Folgenden werden zunächst die beiden Sprachen ATL und SDM genauer vorgestellt.

7.1.1 *Atlas Transformation Language*

Die ATLAS Transformation Language [JK06b, JABK08] ist ein hybrider, unidirektionaler Modell-zu-Modell-Transformationsansatz, der sowohl deklarative als auch imperative Konstrukte unterstützt. Für die Anwendung der Sprache wird jedoch die ausschließliche deklarative Spezifikation der Transformation empfohlen [JK06b], da so Eigenschaften wie die Terminierung der Transformation garantiert werden und die Spezifikationen besser nachvollziehbar sind. Aus diesem Grund und da sowohl die in Abschnitt 9.2 vorgestellten Regeln als auch die ge-

samte Transformation rein deklarativ modelliert sind, wird nur der für die deklarative Modellierung benötigte Teil der ATL-Sprache vorgestellt. Da es sich bei der ATL um eine Sprache basierend auf der MDE handelt, werden sowohl Quell- als auch Zielmodell durch Metamodelle beschrieben. Die Transformationsregeln zwischen ihnen werden textuell spezifiziert, und auf ein Modell, das konform zum ATL-Metamodell ist, abgebildet. Diese drei Metamodelle basieren in der Literatur auf dem MOF-Standard [OMG06], der jedoch in der Eclipse-Integration von ATL durch den EMF-Standard substituiert wird. Für die Formulierung der Transformationsregeln ist darauf zu achten, dass das Quellmodell nur lesbar ist und das Zielmodell nur geschrieben werden kann. Somit kann die Anwendungsbedingung der Regeln nur im Quellmodell navigieren.

Eine ATL-Transformationsdefinition (Auflistung 7.1) besteht aus einer Import-Sektion, die das Quell- und Zielmetamodell der Transformation festlegen und dem Kopf des Moduls.

Beispiel *Kopf eines ATL-Moduls*

Dieser Kopf definiert zum einen den Namen `LSCToMPN` und zum anderen auf Basis des Imports den Namen und Typ des Zielmodells hinter dem Schlüsselwort *create* und des Quellmodells hinter *from*.

Auflistung 7.1: Import und Kopf einer ATL-Transformation

```
-- @path MPNMM=/MPNLanguage/model/MPNLanguage.ecore
-- @path LSCMM=/LSCLanguage/model/LSCLanguage.ecore

module LSCToMPN;
create OUT : MPNMM from IN : LSCMM;
```

Ein Modul enthält eine Menge von Helper-Methoden und Transformationsregeln. Helper-Methoden werden zur Spezifikation von Hilfsmethoden der Transformationsregeln eingesetzt und dürfen nur auf Elementen mit dem Typ des Quellmodells oder OCL-Typen arbeiten, da die Navigation nur im Quellmodell erlaubt ist. Hierbei wird zwischen Operationshilfsmethoden, die auf ihrem Kontext oder Modul arbeiten, Eingabeparameter und Rekursion unterstützen, und Attributhilfsmethoden, die nur auf ihrem Kontext arbeiten, unterschieden.

Beispiel *Helper in ATL*

Auflistung 7.2 zeigt die Operationshilfsmethode `isAsyncSendingLocation`, die überprüft, ob das Kontextelement vom Typ `LSCMM!Location` eine Location an der Sendeseite einer asynchronen Nachricht ist und gibt den entsprechend einen Wahrheitswert zurück. Hierzu werden die Attributbelegungen des Kontextelements *self* auf ihre Belegung überprüft.

Auflistung 7.2: OCL-Helper isAsyncSendingLocation

```
helper context LSCMM!Location def: isAsyncSendingLocation : Boolean =
  if self.incomingMessage.oclIsUndefined()
  then
    if not self.outgoingMessage.oclIsUndefined() then
      self.outgoingMessage.isMessageAsync
    else
      false
```

```
    endif
  else
    false
  endif
;
```

Transformationsregeln werden in ATL als *Matched Rules* bezeichnet und basieren auf dem Konzept der Linken (LHS) und Rechten Seite (RHS) der Transformationsregel. Die LHS (*from*-Teil) beschreibt ein Muster bestehend aus Elementen von Typen des Quellmetamodells und Bedingungen, die zu einem Wahrheitswert auswertbar sind. Zur Strukturierung der Bedingungen können auch Helper eingesetzt werden. Die RHS (*to*-Teil) spezifiziert eine Menge an Elementen des Zielmetamodells und eine Menge an Zuweisungen (*engl. bindings*). Diese Zuweisung initialisiert eine Eigenschaft des Elements mit einem Ausdruck, der aus Helper-Methoden, aus einer String-Konkatenation oder aus einer direkten Navigation im Quellmodell besteht.

Beispiel *Standard Regel in ATL*

Auflistung 7.3 zeigt die Standard-Regel *AsyncMPNPattern*, die auf jede asynchrone Nachricht angewendet werden soll. Der *from*-Teil der Regel beschreibt alle Elemente des Quellmodells, die vom Typ `LSCMM!Message` sind. Die in Klammern folgende Bedingung schränkt die Übereinstimmungen auf die Nachrichten ein, deren Eigenschaft *isAsync* mit dem Wahrheitswert `true` belegt ist. Ausgelagert ist diese Bedingung in die Helper-Methode `isMessageAsync`. Für jede Übereinstimmung im Quellmodell, werden im Zielmodell die im *to*-Teil definierten Elemente (u. a. `sPA` vom Typ `MPNMM!StandardPlace`) erstellt und deren Eigenschaften mit Werten initialisiert.

Auflistung 7.3: Standard-Regel für asynchrone Nachrichten

```
rule AsyncMPNPattern {
  from
    lscAM : LSCMM!Message(lscAM.isMessageAsync)
  to
    sPA : MPNMM!StandardPlace(
      name <- lscAM.startLocation.comment,
      mpn <- thisModule.resolveTemp(lscAM.endLocation.object.lsc, 'mpn')
    ),
    ...
}

helper context LSCMM!Message def : isMessageAsync : Boolean =
  self.isAsync
;
```

In ATL existiert ein Vererbungsmechanismus, der zur Verfeinerung der Elternregeln eingesetzt wird. Die Kindregel beschreibt in ihrem *from*-Teil eine eingeschränkte Menge an Elementen der Elternregel. Hierzu werden die Typen der Elemente der Elternregel durch spezielle Untertypen ersetzt oder die Elemente durch Bedingungen eingeschränkt, sodass sie nur noch auf eine Untermenge matchen. Die Elemente im *to*-Teil der Elternregel können durch Namensgleichheit durch Untertypen überschrieben und durch neue Zielelemente erweitert werden. Zusätzlich können Zuordnungen zu Eigenschaften hinzugefügt oder ersetzt werden.

Beispiel *Standard Regel in ATL*

Die Regel `LastAsyncMessageMainchartUsecase` in Auflistung 7.4 ist eine Verfeinerung der Regel `AsyncMPNPattern` und beschreibt die Sonderbehandlung der letzten Nachricht in einem Mainchart für einen Use-Case. Hierfür muss im MPN des Zielmodells eine zusätzliche Synchronisierung auf einen Endplatz erfolgen, indem eine weitere Transition hinzugefügt wird. Die Einschränkung des *from*-Teils erfolgt durch die zwei zusätzlichen Bedingungen an die Nachricht. Der Helper `isLastMessageInMainchart` überprüft, ob es die letzte Nachricht im Mainchart ist und die letzte Bedingung, ob es sich um einen Use-Case handelt. Im *to*-Teil sind zusätzliche Elemente definiert, die in diesem Sonderfall erstellt werden sollen.

Auflistung 7.4: Vererbung

```
rule LastAsyncMessageMainchartUsecase extends AsyncMPNPattern{
  from
    lscAM : LSCMM!Message(lscAM.isMessageAsync and lscAM.isLastMessageInMainchart
        and not lscAM.startLocation.object.lsc.isMisuseCase)
  to
    tS: MPNMM!Transition(
      comment <- 'null_'.concat(lscAM.startLocation.comment).concat('_').concat(
          lscAM.endLocation.comment),
      event <- 'stop',
      mpn <- thisModule.resolveTemp(lscAM.endLocation.object.lsc, 'mpn'),
    ...
}
```

Die so definierten Regeln werden in einer Virtuellen Maschine ausgeführt. Der Ausführungsalgorithmus findet zunächst alle Übereinstimmungen der Quellmuster (*from*-Teil der Regeln) im Quellmodell und erstellt bei jeder Übereinstimmung im Zielmodell die im *to*-Teil definierten Elemente. Hierbei werden Korrespondenzlinks erstellt. Im zweiten Schritt werden die Zuweisungen des *to*-Teils ausgewertet, die Korrespondenzlinks z. B. für die Auflösung der `resolveTemp`-Methode genutzt und die Eigenschaften der Elemente gesetzt.

7.1.2 *Story Driven Modeling*

Story Driven Modeling (SDM) [BDHR11] ist ein hybrider, unidirektionaler Modell-zu-Modell-Transformationsansatz, der ursprünglich für *In-Place*-Transformationen, d. h. für Veränderungen der Objektstruktur in einem Modell, vorgesehen war. Wie ATL ist SDM ebenfalls in das MDE-Konzept eingebettet und Quell- und Zielmodell basieren auf Metamodellen. Die Transformationen des Quellmodells in ein Zielmodell, die als Storydiagramme modelliert werden, werden als Methodenimplementierung einer Klasse hinterlegt, da der SDM-Formalismus u. a. zur Spezifikation von Verhalten von Softwaresystemen eingesetzt wird. Beim Einsatz von SDMs als *In-Place*-Transformationssprache, befinden sich diese Methoden im gemeinsamen Quell-/Zielmetamodell, während beim Einsatz als Modell-zu-Modell-Transformation sie in einem separaten Transformationsmetamodell spezifiziert sind, das auch zusätzliche Hilfsmethoden oder Korrespondenzlinktypen zwischen Quell- und Zielmetamodell spezifizieren kann.

Beispiel *Transformationsmetamodell*

Abbildung 7.2 zeigt ein Ausschnitt eines solchen Transformationsmetamodells für die Transformation zwischen eLSCs und MPNs. Die Klasse `Transformator` besitzt eine Methode `transform`, die ein Storydiagramm enthält. Die Klassen mittig unter der Klasse `Transformator` bilden eine mögliche Implementierung eines Korrespondenzmodells zwischen den Elementen des Quellmetamodells auf der linken Seite und dem Zielmetamodell auf der rechten Seite.

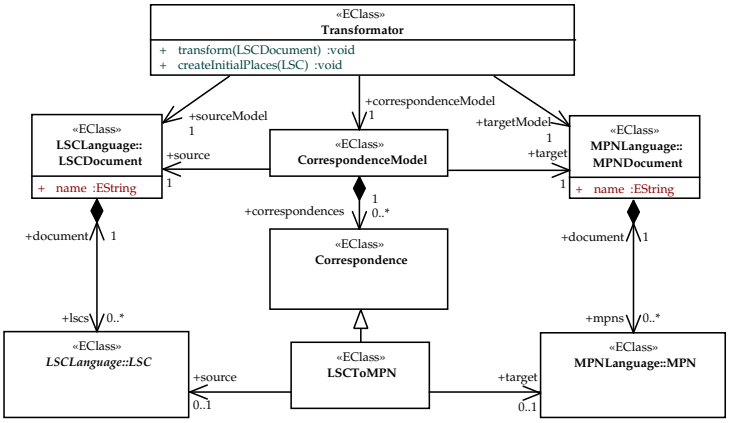


Abbildung 7.2: Transformationsmetamodell der SDM-Transformation

Die Storydiagramme verbinden deklarative Graphtransformationen, die aus Objektmustern (*engl. story pattern*) bestehen, mit einer imperativen Beschreibung des Kontrollflusses der Transformation, die als eine Variante der UML-Aktivitätsdiagramme modelliert wird. Hierzu werden die Objektmuster in die Aktionen eingebettet. Die Objektmuster in den Aktionen werden bei SDM-Regeln in einer Kurzschreibvariante notiert, bei der die LHS und RHS zusammen als ein Objektmuster beschrieben werden. Sie bestehen aus Objekten (Rechtecke) und Links (Verbindungen zwischen den Objekten). Das Erstellen bzw. Löschen von Objekten und Links wird durch die Annotation der Stereotypen «create» bzw. «destroy» annotiert. Repräsentiert wird dies zusätzlich im Objektmuster durch die Einfärbung des entsprechenden Objekts oder Links in Grün bzw. Rot. Im Modell wird während der Transformation die LHS der Objektmuster gesucht, bestehend aus Objekten und Links, die entweder schwarz oder mit «destroy» (rot) markiert sind. Die RHS stellt das Ergebnis der Transformation dar und besteht aus den schwarzen und mit «create» (grün) markierten Objekten und Links.

Jedes Storydiagramm besitzt einen Startknoten und einen oder mehrere Endknoten. Der Kontrollfluss wird durch Transitionen zwischen diesen Aktionen modelliert, die mit dem Erfolg der strukturellen Suche nach dem eingebetteten Objektmuster (*engl. matching*) bedingt werden können (`[success]`- bzw. `[failure]`-Transitionen). Neben diesen einfachen Verzweigungen zwischen den Aktionen, bieten zwei weitere Varianten der Aktionen die Möglichkeit Schleifen mit einer *For-Each*-Aktion und Aufrufe anderer Methoden und damit auch anderer Storydiagramme durch die *Statement*-Aktion zu modellieren. Während in einer Aktion nur nach einem einzigen Vorkommen der LHS eines Objektmusters in der Ob-

jektstruktur aus Quell- und Zielmodell gesucht wird, iteriert eine *For-Each*-Aktion, dargestellt durch einen doppelten Rand, über alle Vorkommen. Hierbei wird die spezifizierte Transformation auf den gefundenen Strukturen ausgeführt und dann nach dem nächsten Vorkommen gesucht. Die *For-Each*-Aktion kann durch eine `[each time]`-Transition erweitert werden, indem hierdurch weitere Aktionen angehängt werden, die nach jeder Transformation ausgeführt werden. Sind alle Vorkommen des Objektmusters in den Modellen gefunden, wird die *For-Each*-Aktion über die obligatorische `[end]`-Transition verlassen.

Beispiel *Kontrollfluss*

Abbildung 7.3 zeigt ein Storydiagramm, das einen Ausschnitt aus der eLSC-zu-MPN-Transformation darstellt. Das Storydiagramm beginnt wie ein Aktivitätsdiagramm mit einem Startknoten, der mit der Signatur der beinhaltenden Methode annotiert ist. Über eine Transition führt der Kontrollfluss zu einer einfachen Aktion, deren Objektmuster einmalig im Modell gesucht wird und die spezifizierte Transformation ausgeführt wird. Ist die Suche im Modell nicht erfolgreich, wird die Aktion durch die `[Failure]`-Transition verlassen. Bei Erfolg wird über die `[Success]`-Transition die *For-Each*-Aktion betreten und die Transformation im Objektmuster auf alle Vorkommnisse im Modell angewendet. Nach jeder Anwendung, wird die *For-Each*-Aktion über die `[each time]`-Transition verlassen und u. a. eine weitere Methode `createInitialPlaces` in einer *Statement*-Aktion aufgerufen. Nach der Rückkehr in die *For-Each*-Aktion wird nach einem weiteren Vorkommen gesucht, und erst wenn keines mehr gefunden wird, wird die Aktion über die `[End]`-Transition verlassen und das Storydiagramm beendet.

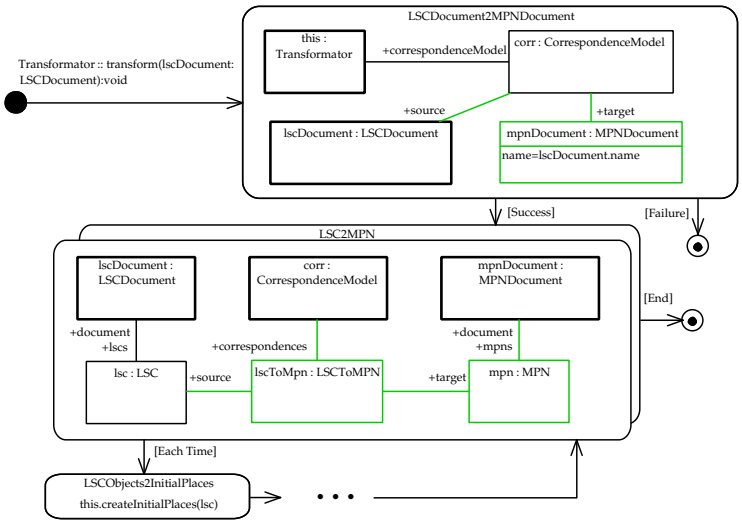


Abbildung 7.3: Teilausschnitt aus der SDM-Transformation eLSC zu MPN

Die Speicherung der gefundenen Muster wird als Bindung bezeichnet. Die initiale Bindung der Modellelemente in einem Storydiagramm findet auf Basis der durch den Methodenaufruf übergebenen Parameter statt, die durch die Auswertung der weiteren Objektmuster des Storydiagramms erweitert wird. Die durch

die Objektmuster gebundenen Objekte der Modelle (Quell-, Ziel- und Transformationsmodell) werden durch die Transitionen zwischen den Aktionen weitergegeben, wenn die LHS des Objektmusters in der vorherigen Aktion vollständig im Modell gefunden wurde. Ist dies nicht der Fall, wird die RHS der Regel nicht angewendet und die gebundenen Objektstrukturen z. B. über eine `[Failure]`-Transition ohne Änderungen weitergegeben. In folgenden Aktionen können bereits gebundene Objekte durch Namensgleichheit als Ausgangspunkt für weitere Suchen verwendet werden. In jeder Aktion findet diese Suche isomorph statt, d. h. beim Suchen der LHS des Objektmusters dürfen zwei verschiedene Teile des Objektmusters nicht demselben Objekt des Modells zugeordnet werden. Bei der Weitergabe der bisher gebundenen Objektstrukturen gibt es zwei Ausnahmen. Objekte, die in einer Schleife oder durch eine Aktion, die durch eine *For-Each*-Transition angebunden ist, einem Objektmuster zugeordnet werden, stehen nach Verlassen der Schleifen nicht mehr als gebundene Objekte zur Verfügung. Die durch *Statement*-Aktionen aufgerufenen Storydiagramme sind unabhängig vom aufrufenden Storydiagramm und somit stehen ihnen nur die initiale Bindung der übergebenen Parameter zur Verfügung. Diese Parameter können jedoch beim Aufruf durch bereits gebundene Objekte belegt werden.

Beispiel *Objektmuster (Story-Pattern)*

Die Objektmuster in Abbildung 7.3 beschreiben eine Transformation, die zu einem als Parameter übergebenen Objekt `lscDocument` ein `mpnDocument` erstellt. Hierbei stellen die Objekte mit einem dicken schwarzen Rahmen, die bereits gebundenen Objekte dar. Das `this`-Objekt repräsentiert eine Instanz der Transformator-Klasse des Transformationsmetamodells, die in jedem Storydiagramm gebunden ist. Ausgehend von diesen bekannten Objekten im Modell wird ein zum `this`-Objekt gehörendes Objekt vom Typ `CorrespondenceModel` gesucht. Wird dies in den Modellen gefunden, erstellt die Transformation (RHS) ein `mpnDocument`-Objekt und verbindet es über `corr` mit dem `lscDocument`-Objekt. Diese gefunden und erstellten Objekte sind nun für die nächste Aktion bekannt und können als gebundene Objekte eingesetzt werden.

7.2 Generierung von Monitoren aus Spezifikationssprachen

Die MBSecMonSL, die im MBSecMon-Prozess zur Spezifikation von Ereignissequenzen und Bedingungen für diese eingesetzt wird, wird zur Generierung der Monitore im MBSecMon-Entwicklungsprozess auf den Formalismus der Monitor-Petrinetze (MPNs) abgebildet. Alternativ könnte eine Formalisierung der MBSecMonSL mit ihren Eigenschaften und die Monitorgenerierung auch über andere Formalismen, wie Temporale Logiken, erfolgen.

In der Literatur werden Temporale Logiken [Var01] häufig zur Spezifikation von unendlichen Ereignissequenzen verwendet, die als Eingabe für Modelchecker genutzt werden, um Spezifikationen von Systemen zu validiert. Temporale Logiken unterscheiden sich in ihrem zugrundeliegenden Zeitmodell. Die linearen Temporalen Logiken (*engl. linear temporal logic*) betrachten jeden Zeitpunkt, als ob es nur eine mögliche Zukunft gibt, und werden zur Beschreibung linearer

Sequenzen verwendet. Im Gegensatz hierzu sehen die verzweigende Temporale Logiken (*engl. branching temporal logic*) jederzeit eine Aufteilung in verschiedene mögliche Zukünfte vor.

Die Formeln der Linear Temporal Logic (LTL) setzten sich aus Aussagen (*engl. proposition*) zusammen, die über boolesche Operatoren und temporalen Operatoren verknüpft werden. Eigenschaften werden durch die Kombination aus Aussagen über temporale Operatoren gebildet. LTL bieten die vier grundlegenden temporalen Operatoren G („immer"), F („letztendlich"), X („als nächstes") und U („bis") an. CTL* [EH86] erweitert LTL um die Unterscheidung zwischen den Pfadquantifikatoren E, dem Existenzquantor, und A, dem Allquantor. CTL ist eine Teilmenge der CTL*, in der jeder einzelne temporale Operator einen Pfadquantifikator vorangestellt hat.

Auch Live Sequence Charts, auf denen die eLSCs der MBSecMonSL aufbauen, werden zur Formalisierung der Semantik und Validierung der Spezifikationen in Temporale Logiken übersetzt. Kugler et al. [KHP+05] übersetzt den Kern der LSC-Sprache u. a. ohne Variablenkonzept und der Einschränkung, dass jedes Chart nur eine Instanz einer Nachricht enthält, in CTL*-Formeln, da die Ausdrucksstärke von LTL und CTL separat zur Repräsentation der Existential und Universal LSCs nicht ausreicht. Diese beiden Ansätze [KHP+05, DTW07] übersetzen nicht jedes LSC-Konstrukt in die Temporale Logik, sondern bilden jeden Eintrag der partiellen Ordnung der LSCs in einzelne Ordnungsbeziehungen ab, die als Eigenschaften repräsentiert werden. Diese Übersetzung produziert somit für jede Kombination von zwei beliebigen Nachrichten (x_i, x_j) in einem Chart eine Eigenschaft der Form $\neg x_j \,\mathrm{U}\, x_i$, die aussagt, dass x_j nicht vor x_i eintreten darf. Zur Absicherung, dass eine Nachrichteninstanz, die nicht in der partiellen Ordnung enthalten ist, mehrfach auftritt, werden zusätzliche Eigenschaften der Form $(\neg x_i \wedge \neg x_j) \,\mathrm{U}\, (x_i \wedge \mathrm{X}((\neg x_i \wedge \neg x_j) \,\mathrm{U}\, x_i))$ definiert. Diese LTL-Formel beschreibt, dass die Nachricht x_i zweimal vor x_j auftritt. Durch ihre Negation kann somit sichergestellt werden, dass x_i nicht mehrfach auftreten kann. Neben dieser Einschränkung, dass nur eine Instanz einer Nachricht in einem Chart vorkommen darf, können manche komplexere LSC-Elemente nicht direkt in die Temporale Logik übersetzt werden. So werden beschränkte Schleifen durch Entfalten (*engl. unfolding*) abgebildet und unbeschränkte Schleifen, die x Instanzen derselben Nachricht enthalten, sind aufgrund der fehlenden Möglichkeit *Modulo n* zu zählen, nicht übersetzbar. Trotz der Reduzierung der benötigten Eigenschaften zur Beschreibung der partiellen Ordnung und zur Absicherung des einmaligen Vorkommens der Nachrichten durch Kumar et al. [KMB09], wächst die Größe der Formeln (Anzahl der geschachtelten Temporalen Operatoren) quadratisch proportional zu der maximalen Anzahl der Nachrichten im Mainchart des LSCs. Die extreme Größe dieser Temporalen Formeln und die fehlende Unterstützung der gesamten LSC-Grammatik schränken die Anwendbarkeit dieser Übersetzungen stark ein.

Als Alternative zur Übersetzung der LSCs in Temporale Logiken stellen Kumar et al. [KM08, KM09] eine Übersetzung der LSCs in Automaten vor, die zur Verifikation des Systemmodells gegen die LSC-Spezifikation genutzt werden. Dieser Ansatz bildet nahezu die gesamte LSC-Grammatik in Automaten ab. So kön-

nen auch unbeschränkte Schleifen und hierarchische Charts als Automaten repräsentiert werden, wobei nebenläufige Konstrukte durch parallele Komposition der Einzelautomaten repräsentiert werden. Durch die Negation der aus den LSCs erstellten Automaten erkennt ein Modelchecker in Zusammenspiel mit dem Systemmodellautomaten auch verbotenes Verhalten. Die Komposition der Automaten, die aus der Abbildung nebenläufiger LSC-Konstrukte entstehen, führen auch bei diesem Ansatz zu einem starken Wachstum der Anzahl der Zustände.

Sequenzdiagramme im Allgemeinen werden nicht nur zur Validierung der Spezifikation in Automaten übersetzt, sondern auch zur Ausführung der Spezifikation. Whittle et al. [WJ06, WWH08] verwendet hierfür das Werkzeug UCSIM [JW07], das für jeden Teilnehmer eines jeden Sequenzdiagramms eine einzelne kommunizierende FSM generiert. Die Synchronisierung der FSMs erfolgt über durch UCSIM zusätzlich eingefügte Synchronisierungsnachrichten. Im Gegensatz zum vorherigen Ansatz führt dies zwar zu weniger Zuständen der Automaten, jedoch zu einem großen Kommunikationsoverhead.

Neben diesen direkten Übersetzungen der Sequenzdiagramme, insbesondere der LSCs, in Automaten, existieren weitere Entwicklungsprozesse, die zunächst abstrakte Spezifikationen in Temporale Formeln übersetzen, die dann in Automaten überführt werden. Im Bereich der Modellbasierten Sicherheitsengineering wird UMLsec [Jür05] in der Designphase zur Annotation der UML-Modelle zur Entwicklung sicherheitskritischer Software von wiederkehrenden Sicherheitsanforderungen (z. B. Integrität, Authentifizierung und Geheimhaltung) und zur Formulierung von Annahmen über die Umgebung eingesetzt [Jür05]. Diese annotierten Designmodelle werden mit dem UMLsec-Tool formal gegen die in den Modellen annotierten High-Level-Sicherheitsanforderungen verifiziert. In [BJ08, BJ10, BJY11] wird gezeigt, dass auch die Implementierung durch Laufzeitverifikation gegen das Designmodell verifiziert werden kann. Hierzu wird angenommen, dass die Sicherheitseigenschaften der UMLsec-Annotationen im Designmodell als LTL-Formeln vorliegen oder in LTL-Formeln übersetzt werden können. Basierend auf diesen LTL-Formeln werden durch das Werkzeug LTL_3[1] [BLS06, BLS07] in mehreren Übersetzungsschritten endliche Zustandsautomaten (FSM) erzeugt, die dann in einen Monitor in der Programmiersprache Java übersetzt werden. Die Übersetzungsschritte erzeugen aus den LTL-Formeln, die unendliche Ereignissequenzen beschreiben, zunächst die negierte LTL-Formel. Danach werden beide Formeln einzeln in nicht-deterministische Büchi-Automaten (NBA) überführt, die nach einer erfolgreichen Validierung aller Zustände auf *Emptiness*, in Nichtdeterministische Endliche Automaten (NFA) übersetzt werden. Um den Aufbau einer FSM durch Potenzmengenkonstruktion der beiden Automaten durchzuführen, werden die NFAs in Deterministische Endliche Automaten (DFA) übersetzt. In diesen Übersetzungsschritten muss sowohl bei der Konstruktion der NBAs aus den LTL-Formeln als auch bei der Überführung in die DFAs mit dem exponentiellen Wachstum der Automaten umgegangen werden.

Im MBSecMon-Prozess wird dieses exponentielle Wachstum der Automaten durch den Einsatz der MPNs als Zwischensprache zur Codegenerierung, mit ihrer parallelen und impliziten Semantik, umgangen. Diese kompakte Repräsentation

[1] LTL_3-Tools: http://ltl3tools.sourceforge.net/ (besucht am 29.09.2013)

der Spezifikation als MPN ermöglicht eine einfache Generierung speichereffizienter Laufzeitmonitore. Im folgenden Kapitel wird diese Übersetzung der MBSecMonSL in die in [Pat14] formalisierten MPNs, der wichtige Schritt zu einer Monitorgenerierung und gleichzeitig eine Formalisierung der MBSecMonSL durchgeführt.

8

MBSECMONSL ZU MONITOR-PETRINETZE

In der modellgetriebenen Softwareentwicklung (MDE) werden Modell-zu-Modell-Transformationen zwischen Modellen verschiedener Abstraktionsebenen eingesetzt. So werden plattformunabhängige Modelle durch Transformationen verfeinert und über plattformspezifische Modelle in eine Systemimplementierung überführt. Dieses Vorgehen wird auch vom MBSecMon-Prozess verfolgt, indem die abstrakte Spezifikation in der MBSecMonSL zunächst in die Zwischensprache MPN transformiert wird und dann über weitere Transformationen plattformabhängige Laufzeitmonitore generiert werden.

In diesem Kapitel wird exemplarisch, Schritt für Schritt die Modelltransformation zwischen der MBSecMonSL und den MPNs in konkreter Syntax vorgestellt. Hierzu wird in Abschnitt 8.1 zunächst auf die Übersetzung der eLSC-Signaturen in ihre MPN-Repräsentation eingegangen. In Abschnitt 8.2 wird gezeigt, wie die zur Strukturierung der Signaturen genutzte MUC-Sprache in das Referenzsystem der MPNs überführt wird und welche Auswirkungen dies auf die Übersetzung der eLSCs hat. Abschließend werden in Abschnitt 8.4 sechs Modellierungsregeln für Spezifikationen, die in der MBSecMonSL verfasst sind, aufgestellt. Diese Regeln stellen die korrekte Transformation in die MPN-Sprache und somit die Generierung korrekter Laufzeitmonitore sicher.

8.1 Abbildung der eLSC-Sprache in konkreter Syntax

Die Übersetzungsregeln der Abbildung der eLSC-Sprache in die MPN-Sprache können in vier Ebenen unterteilt werden. Die erste Ebene besteht aus der Übersetzung der grundlegenden Elemente und die zweite Ebene aus der Betrachtung der Auswirkungen der Temperaturen auf die resultierenden MPNs, die dritte aus der Übersetzung der kontrollflusssteuernden Fragmente und die vierte Ebene aus der Übersetzung der *Forbidden*- und *Ignore*-Fragmente.

8.1.1 *Grundlegende Elemente*

Die Basis eines eLSCs bilden die Lebenslinien, die, wie in Abbildung 8.1 gezeigt, in einzelne Initialplätze übersetzt werden. Diese Initialplätze, die mit dem Namen der Lebenslinie beschriftet sind, bilden den Ausgangspunkt der MPN-Signatur und beschreiben die Anfangsbelegung der MPNs. Jeder Platz, der in der Übersetzung erstellt wird, wird ebenfalls wie die Initialplätze den Lebenslinien des eLSCs zugeordnet. Diese Zuordnung ist für die Identifikation der Plätze im MPN wäh-

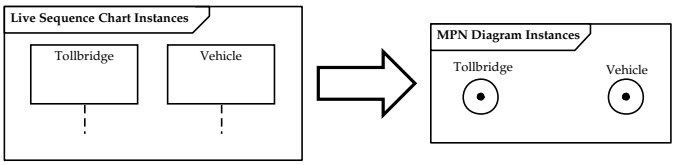


Abbildung 8.1: MPN-Repräsentation der eLSC-Lebenslinien

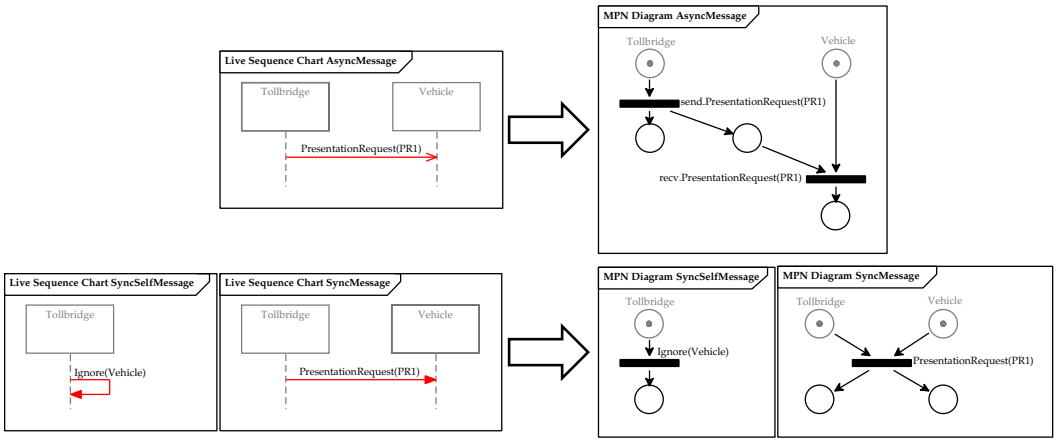


Abbildung 8.2: MPN-Repräsentation einer eLSC-Nachricht

rend der Übersetzung notwendig, an die das nächste übersetzte eLSC-Element angehängt werden soll.

Zur Modellierung der Kommunikation zwischen den Lebenslinien werden in eLSCs Nachrichten eingesetzt, die asynchron oder synchron definiert sein können. Abbildung 8.2 zeigt die Übersetzung für eine asynchrone heiße Nachricht (oben) und synchrone Nachrichten (unten) in eine semantisch äquivalente MPN-Repräsentation. In beiden Fällen dienen die grau dargestellten Initialplätze nur als Platzhalter für beliebige Plätze des MPNs, die von der Übersetzung des zeitlich auf deren entsprechenden Lebenslinien vorher liegenden Elementen stammen.

Eine *asynchrone Nachricht* wird in zwei Transitionen, eine für das Sendeereignis und eine für das Empfangsereignis und drei Plätze übersetzt. Hierbei dienen die Präfixe `send.` und `recv.` zur Unterscheidung des Ereignisses, die an den Transitionen annotiert werden. Der linke Nachplatz codiert den Zustand, dass das Sendeereignis eingetreten ist, und der rechte Nachplatz, dass das Empfangsereignis eingetreten ist. Diese beiden Plätze sind jeweils der Lebenslinie auf der Sende- bzw. Empfangsseite zugeordnet. Der mittlere Platz zwischen der Sendetransition und der Empfangstransition stellt die Reihenfolge zwischen Senden und Empfangen der Nachricht sicher, da die Empfangstransition nur feuern kann, wenn alle Vorplätze belegt sind. Dieser mittlere Platz synchronisiert das Sende- und Empfangsereignis und ist somit beiden Lebenslinien zugeordnet.

Eine *synchrone (Eigen-)Nachricht* kann als ein einzelnes Ereignis modelliert werden, da Senden und Empfangen zusammenfallen. Somit werden die zuletzt übersetzten Plätze, die zu den an der Nachricht beteiligten Lebenslinien gehören, über

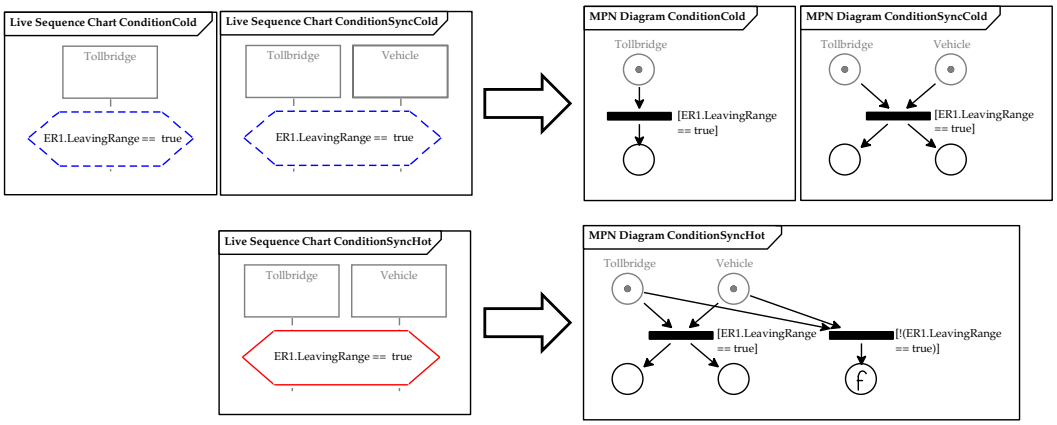


Abbildung 8.3: MPN-Repräsentation einer eLSC-Bedingung

eine gemeinsame Transition synchronisiert und die Transition mit dem Nachrichtennamen annotiert, der das Ereignis repräsentiert.

Bedingungen der eLSC-Sprache werden, wie in Abbildung 8.3 dargestellt, ähnlich zu synchronen Nachrichten übersetzt, können jedoch beliebig vielen Lebenslinien gleichzeitig zugeordnet sein. Statt eines Ereignisses wird an der Transition die Bedingung in eckigen Klammern annotiert. In Abbildung 8.3 sind exemplarisch zwei Varianten einer *kalten Bedingung* dargestellt, `ConditionCold` und `ConditionSyncCold`. Die Übersetzung des Diagramms `ConditionCold` bildet eine Bedingung, die einer Lebenslinie zugeordnet ist, auf die entsprechende MPN-Repräsentation ab. `ConditionSyncCold` repräsentiert die Übersetzung einer Bedingung, die zwei Lebenslinien zugeordnet ist. Dieses Konzept kann auf beliebig viele Lebenslinien übertragen werden, wobei nicht jede überspannte Lebenslinie auch der Bedingung zugeordnet sein muss. Im Gegensatz zu einer kalten Bedingung führt die Nichteinhaltung einer *heißen Bedingung* (Übersetzung `ConditionSyncHot` in Abbildung 8.3) zu einer positiven oder negativen Auswertung der Überwachung des eLSCs. Hierfür wird bei der Übersetzung eine weitere Transition mit negierter Bedingung an die Vorplätze der positiven Transition angehängt. Diese Transition führt zu einem Endplatz oder Fehlerplatz, abhängig davon, ob das eLSC einen Use- oder Misuse-Case beschreibt und ob sich die Bedingung in einem Pre- oder Mainchart befindet.

Zeitbedingungen werden wie Bedingungen auf Transitionen abgebildet. Aufgrund der abweichenden Semantik der Zeitbedingungen von den einfachen Bedingungen (Tab. 5.1) unterscheiden sich ihre Übersetzungen. Während *heiße maximale Zeitbedingungen* wie heiße Bedingungen übersetzt werden, werden *heiße minimale Zeitbedingungen* äquivalent zu kalten Bedingungen übersetzt. Sowohl *minimale* als auch *maximale kalte Zeitbedingungen* werden grundlegend wie heiße Bedingungen in der MPN-Sprache repräsentiert. Da kalte Zeitbedingungen jedoch zu keiner Fehlererkennung führen, endet die zweite Transition, die mit der negierten Bedingung annotiert ist, auf einem Endplatz statt auf einem Fehlerplatz.

Eine *Zuweisung* der eLSC-Sprache kann wie eine Bedingung beliebig vielen Lebenslinien zugeordnet sein, wird jedoch immer *heiß* modelliert, da es sich um eine auszuführende Aktion handelt. Abbildung 8.4 zeigt beispielhaft die Zuordnung

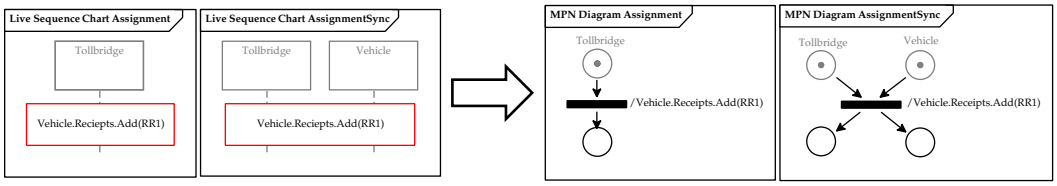


Abbildung 8.4: MPN-Repräsentation einer eLSC-Zuweisung

der Zuweisung zu einer (`Assignment`) und zu zwei Lebenslinien (`AssignmentSync`). Da dieses Element ebenfalls zu einer Synchronisierung der entsprechenden Plätze der Lebenslinien führt, erfolgt die Übersetzung analog zur kalten Bedingung, mit der Ausnahme, dass an der Transition die Aktion der eLSC-Zuweisung hinter einem „/“ annotiert wird.

Diese Übersetzungen der grundlegenden Elemente der eLSC-Sprache in die MPN-Sprache definiert die Semantik der eLSC-Elemente als verhaltensbeschreibende Signatursprache. Da nicht nur einzelne Elemente der eLSCs in MPNs repräsentiert werden sollen, müssen gesamte eLSC-Diagramme übersetzt werden. Dies erfolgt durch die einzelne Übersetzung der grundlegenden eLSC-Elemente, die an die *offenen Plätze* (Def. 8) der zuvor übersetzten Elemente in der MPN-Repräsentation gehängt werden.

Definition 8 (*Offene Plätze im Übersetzungsprozess für grundlegende Elemente*)**.** *Offene Plätze* in der MPN-Repräsentation sind Plätze, die eindeutig einer einzelnen Lebenslinie des eLSCs zugeordnet sind und eine der beiden Eigenschaften erfüllen: (1) Sie besitzen keine ausgehende Verbindung zu einer Transition. (2) Sie sind ausschließlich zu nachfolgenden Transitionen verbunden, die durch Übersetzung kalter Nachrichten (Nachrichten mit kalter Location auf der Sende- oder Empfangsseite) entstanden sind.

Wird ein weiteres eLSC-Element übersetzt, das über eine heiße Location mit einer Lebenslinie verbunden ist, werden die bisherigen offenen Plätze der Lebenslinien durch die neu entstandenen der Lebenslinie zugeordneten Plätze ersetzt.

Beispiel *Offene Plätze im Transformationsprozess*

Abbildung 8.5 zeigt den Übersetzungsprozess für ein eLSC, dass aus zwei Instanzen (die Mautbrücke `Tollbridge` und das Fahrzeug `Vehicle`), einer heißen asynchronen Nachricht `EchoRequest(ER1)` und einer Bedingung, die den durch die Nachricht übertragenen Parameter `ER1` überprüft, besteht. Im ersten Schritt werden die Instanzen in jeweils einen Initialplatz übersetzt. Diese beiden Initialplätze sind nun für den zweiten Schritt die offenen Plätze, an die das als nächstes übersetzte Element angehängt wird. Der zweite Schritt übersetzt nun die asynchrone Nachricht in die in Abbildung 8.2 dargestellte Repräsentation. Hierbei wird die Transition, die dem Sendeereignis entspricht, an den offenen Platz, der der Mautbrücke zugeordnet ist, angehängt. Die Transition des Empfangsereignisses wird an den offenen Platz, der dem Fahrzeug zugeordnet

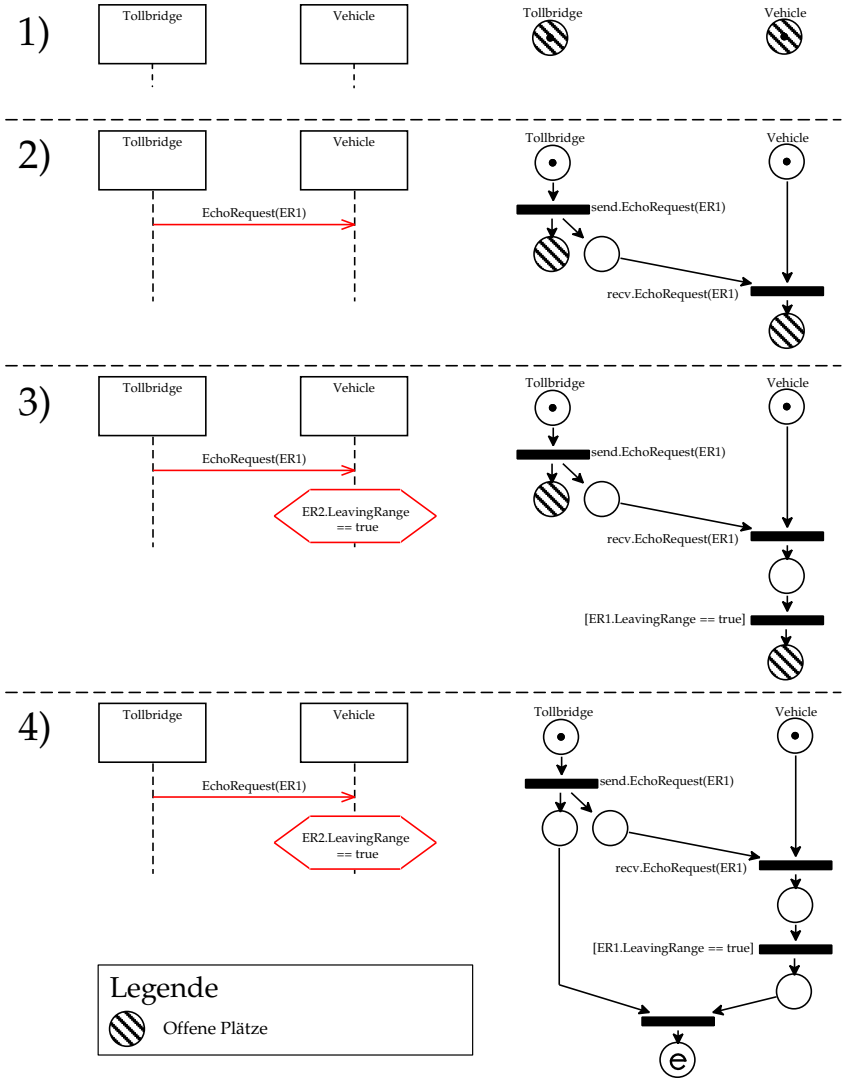


Abbildung 8.5: Offene Plätze in der Transformation in die MPN-Sprache

ist, angehängt. Da es sich um eine heiße Nachricht handelt, werden in diesem Schritt die Initialplätze aus der Menge der offenen Plätze entfernt und die Nachplätze der beiden Transitionen, ohne den Synchronisierungsplatz in die Menge aufgenommen. Im dritten Schritt wird die heiße Bedingung übersetzt, die nur dem Fahrzeug zugeordnet ist. Ihre MPN-Repräsentation wird an den offenen Platz der Empfangstransition der vorherigen Nachricht angehängt und ersetzt diesen in der Menge der offenen Plätze mit dem neu hinzugefügten Platz. Im vierten Schritt findet eine Synchronisierung der beiden noch offen Plätze auf einen Endplatz statt.

Existential eLSC-Diagramme werden basierend auf einer Aneinanderreihung der Übersetzung der einzelnen grundlegenden eLSC-Elemente realisiert und mit einem gemeinsamen Endplatz abgeschlossen. Für das in Abbildung 8.6 dargestellte eLSC-Diagramm bedeutet dies, dass für jede Lebenslinie ein Initialplatz mit ihrem Namen erzeugt wird. Die asynchrone heiße Nachricht `EchoRequest(ER1)`

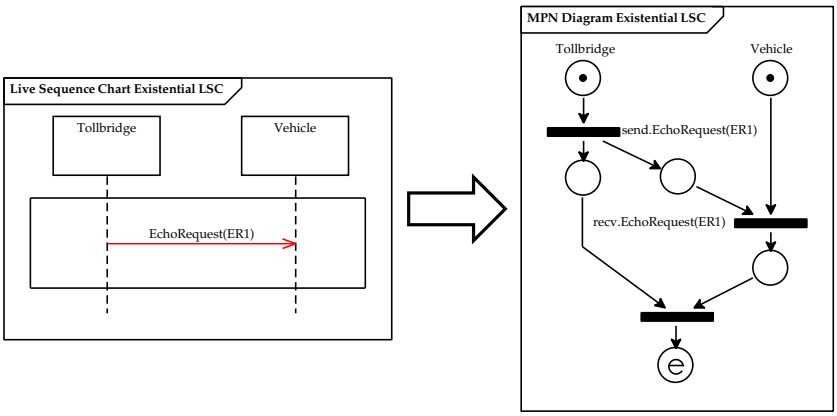


Abbildung 8.6: MPN-Repräsentation eines Existential eLSCs

wird wie vorher beschrieben übersetzt und am Ende des Maincharts werden alle offenen Plätze zu einem Terminalplatz synchronisiert, der den Abschluss der Überwachung symbolisiert. Dieser Terminalplatz ist bei einem eLSC, das einen Use-Case beschreibt, ein Endplatz und bei einem eLSC, das einen Misuse-Case beschreibt, ein Fehlerplatz.

Universal eLSC-Diagramme werden grundlegend wie die Existential eLSC-Diagramme übersetzt. Das übersetzte MPN in Abbildung 8.7 beginnt mit zwei Initialplätzen. Die MPN-Repräsentation der ersten heißen Nachricht im Prechart `InitialisationRequest(IR1)` wird im MPN an die Initialplätze der an ihr beteiligten Lebenslinien angehängt. Die Initialplätze dienen somit als Vorplätze der hinzugefügten Transitionen. Anschließend werden die offenen Plätze beim Übergang zwischen Pre- und Mainchart durch eine Transition ohne Ereignis auf zwei neue Nachplätze synchronisiert. Dies stellt sicher, dass das Prechart vollständig erkannt wurde, bevor mit der Überwachung des Maincharts fortgefahren wird. An diese neuen Nachplätze wird die MPN-Repräsentation der Nachricht `PresentationRequest(PR1)` angehängt und durch eine weitere Transition ohne Ereignis für das Verlassen des Maincharts wie bei der Übersetzung des Existential eLSCs auf einen Terminalplatz synchronisiert.

8.1.2 *Kombinationen von heißen und kalten Nachrichten*

Die bisher präsentierten Übersetzungsregeln der grundlegenden eLSC-Elemente hängen die neu übersetzten MPN-Repräsentationen an die noch nicht weiter verbundenen Plätze der beteiligten Lebenslinien. Durch die Verwendung heißer und kalter Nachrichten muss jedoch durch zusätzliche Transitionen in der MPN-Repräsentation das Überspringen aller oder eines Teils der kalten Nachrichten abgebildet werden. Hierfür muss der zweite Fall der offenen Plätze in Definition 8 beachtet werden. Um dies zu bewerkstelligen, wird über die *offenen Plätze* Buch geführt, an die die MPN-Repräsentation des nächsten eLSC-Elements angehängt wird.

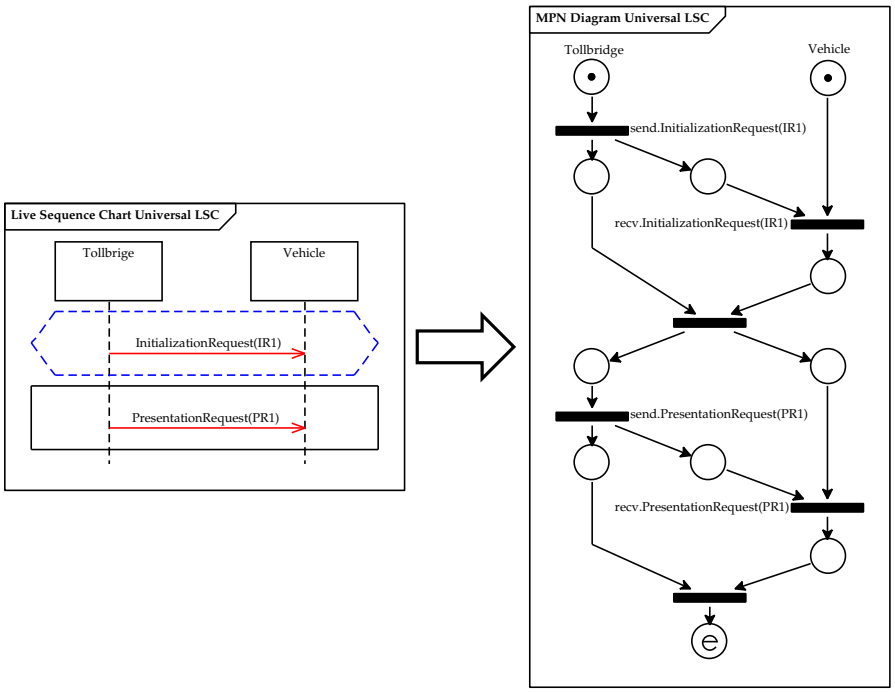


Abbildung 8.7: MPN-Repräsentation eines Universal eLSCs

Asynchrone Nachrichten können in eLSCs in Kombination mit heißen und kalten Locations in den vier Ausprägungen, wie in Abschnitt 5.1.1 festgelegt, in den Signaturen auftreten. Diese Ausprägungen entsprechen Fall 1, Fall 2, Fall 5 und Fall 7 aus Tabelle 4.1. Allgemein wird für asynchrone Nachrichten die Sende- und Empfangsseite im MPN einzeln betrachtet und für jeden weiteren offenen Platz auf einer der beiden Seiten eine Übersprungstransition, auf die Nachplätze der aktuell erzeugten Transition übersetzt.

Abbildung 8.8 zeigt ein eLSC mit einem Mainchart, das zwei heiße asynchrone Nachrichten und eine kalte asynchrone Nachricht, die entsprechend Fall 5 in Tabelle 4.1 modelliert ist, enthält. Nachdem die Mautbrücke die heiße Nachricht `PresentationRequest(PR1)` gesendet hat, kann es durch die partielle Ordnung der eLSCs vorkommen, dass vor der Antwort des Fahrzeugs eine zweite Nachricht desselben Typs gesendet wird. Diese Nachricht und ihre Locations auf den Lebenslinien sind *kalt* modelliert und somit sind sowohl das Sende- als auch das Empfangsereignis optional. Abgeschlossen wird die Kommunikationssequenz durch die Antwort des Fahrzeugs, die als heiße Nachricht modelliert ist. Die Übersetzung des eLSCs in ein MPN findet, wie in Abschnitt 8.1.1 beschrieben, statt. Die ersten zwei Nachrichten werden mit dem Sendeereignis auf der Seite der Mautbrücke im MPN angehängt, die letzte Nachricht umgekehrt mit dem Sendeereignis auf der Fahrzeugseite. Durch die Übersetzung der kalten Nachricht vergrößert sich die Menge der offenen Plätze nach Definition 8 um einen weiteren offenen Platz je beteiligter Lebenslinie. Die letzte asynchrone Nachricht wird zunächst so übersetzt, als ob die vorherige Nachricht heiß wäre, d. h. an die unteren offenen Plätze angehängt. In diesem Zustand entspricht die MPN-Repräsentation

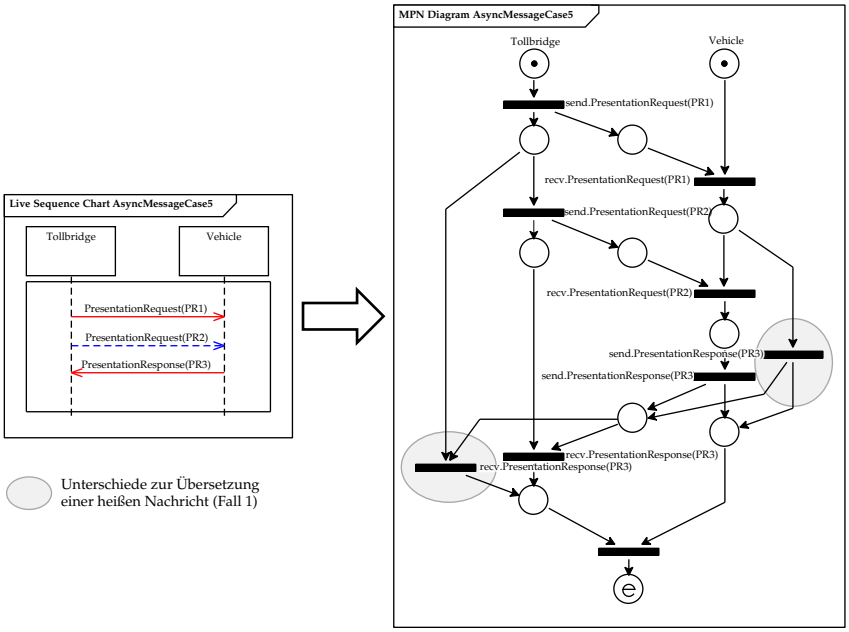


Abbildung 8.8: MPN-Repräsentation eines eLSCs mit einer kalten Nachricht (Fall 5)

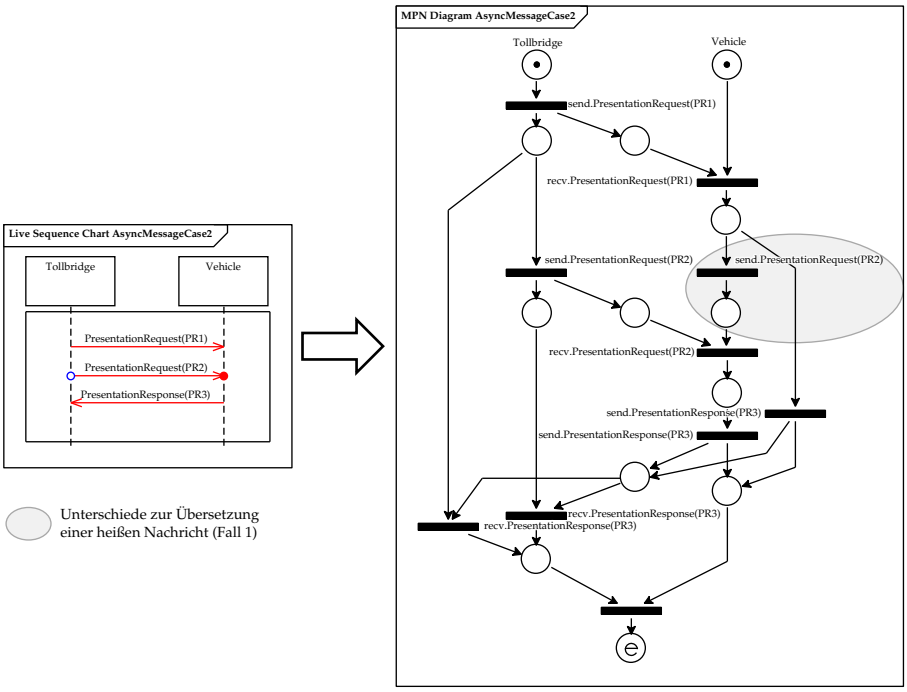


Abbildung 8.9: MPN-Repräsentation eines eLSCs mit einer Nachricht, die empfangen werden muss, wenn sie gesendet wurde (Fall 2)

der Übersetzung von drei heißen Nachrichten (Fall 1). Die kalte Temperatur und somit der optionale Charakter der kalten Nachricht wird durch zwei zusätzliche Übersprungstransitionen an den zwei weiteren offenen Plätzen repräsentiert (grau hinterlegt) – eine für das Sende- und eine für das Empfangsereignis der nächsten Nachricht (`PresentationResponse`). Diese Transitionen werden auf die Nachplätze der MPN-Repräsentation der Nachricht `PresentationResponse` verbunden.

Neben den beiden Standardfällen einer komplett heißen (Fall 1) und einer komplett kalten Nachricht (Fall 5), erlauben die eLSC ebenfalls die Modellierung einer Nachricht, die empfangen werden muss, wenn sie gesendet wurde (Fall 2). Hierzu wird wie in Abbildung 8.9 eine heiße Nachricht modelliert, die eine kalte Location auf der Sende- und eine heiße Location auf der Empfangsseite besitzt. Zur Repräsentation dieser Signatur als MPN wird die kalte Location der Sendeseite der Nachricht `PresentationRequest(PR2)` durch eine Übersprungstransition repräsentiert und mit dem Empfangsereignis der nächsten Location auf der Lebenslinie annotiert. Dies entspricht soweit der Übersetzung der kalten Nachricht aus Fall 5. Für die Empfangsseite der Nachricht muss jedoch eine Ausnahme in der Übersetzung eingeführt werden. Trotz der heißen Location und der heiß modellierten Nachricht wird eine zusätzliche Transition (grau hinterlegt) der Empfangstransition vorgeschaltet und mit dem Sendeereignis der Nachricht annotiert. Der Platz vor der zusätzlichen Transition bleibt jedoch als offener Platz erhalten, sodass die als nächste übersetzte Nachricht eine Übersprungstransition in der MPN-Repräsentation erzeugt. So wird ein Überspringen des Empfangsereignisses, wenn vorher die Nachricht gesendet wurde, verhindert.

Ein eLSC mit einer asynchronen Nachricht, die Fall 7 entspricht, ist in Abbildung 8.10 dargestellt. Eine Nachricht muss gesendet werden, kann jedoch bei der Übermittlung verloren gehen und muss somit nicht empfangen werden. Modelliert wird dies durch eine kalte Nachricht, die auf der Sendeseite eine heiße und auf der Empfangsseite eine kalte Location besitzt. Diese Nachricht kann im Gegensatz zum vorherigen Fall basierend auf der Temperatur der Locations übersetzt werden. Auf der Empfangsseite bewirkt die kalte Location, dass der Platz der vorherigen Nachricht noch als offener Platz weitergeführt und so eine Übersprungstransition mit der Übersetzung der Nachricht `PresentationResponse(PR3)` erzeugt wird.

Im Gegensatz zu asynchronen Nachrichten existieren für *synchrone Nachrichten* zwei Fälle: (1) Die synchronen Nachrichten, für die offene Plätze existieren, sind wie in Abbildung 8.11 zwischen den gleichen zwei Lebenslinien wie die zu übersetzende Nachricht im eLSC aufgespannt. (2) Die zuletzt übersetzten synchronen Nachrichten verbinden wie in Abbildung 8.12 andere Lebenslinien als die aktuell übersetzte.

In Fall 1 für synchrone Nachrichten zwischen zwei Lebenslinien zeigt Abbildung 8.11 das Ergebnis der Übersetzung. Die einzelnen synchronen Nachrichten werden jeweils in eine synchronisierende Transition, die an die zwei offenen Plätze angehängt wird, und zwei neue Nachplätze übersetzt. Für jedes weitere Paar offener Plätze der beteiligten Lebenslinien, die aus der Übersetzung einer gemeinsamen synchronen Nachrichten resultieren, wird auch hier eine zusätzliche

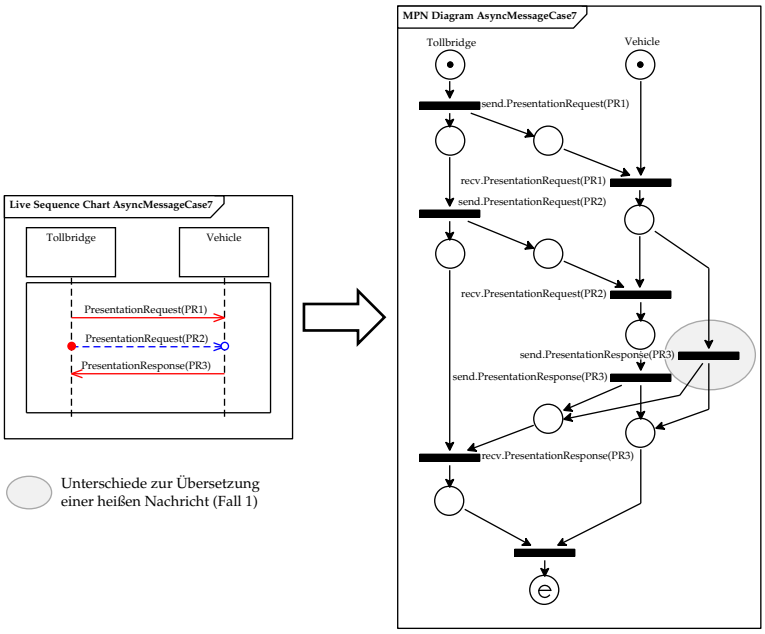


Abbildung 8.10: MPN-Repräsentation eines eLSCs mit einer Nachricht, die gesendet, aber nicht empfangen werden muss (Fall 7)

Übersprungstransition für die Nachricht hinter der kalten Nachricht generiert, um ihren optionalen Charakter zu repräsentieren.

In Fall 2 werden die offenen Plätze der Sende- und Empfangsseite einzeln betrachtet und für jede Kombination eine zusätzliche Übersprungstransition erstellt. Abbildung 8.12 zeigt einen solchen komplexen Fall, in dem drei Instanzen miteinander kommunizieren. Die Besonderheit der Transformation ergibt sich bei der Übersetzung der letzten beiden heißen Nachrichten im eLSC. Da die Sendeseite der Nachricht `PresenstationRequest(PR3) Tollbridge2` ist, sind nur die offenen Plätze der Lebenslinien `Vehicle` (2 offene Plätze) und `Toolbridge2` (1 offener Platz) an der Übersetzung beteiligt. Hier werden für alle Kombinationen aus jeweils einem offenen Platz je Lebenslinie gebildet und jeweils eine Transition, die mit denselben Nachplätzen verbunden sind erstellt. Diese Nachplätze sind nun die neuen offenen Plätze der beiden Lebenslinien. Für die Übersetzung der Nachricht `PresentationResponse(PR4)` existieren somit auf der Lebenslinie `Tollbridge1` noch zwei offene Plätze und auf `Vehicle` noch ein offener Platz. Die Übersetzung resultiert wie bei der vorherigen Nachricht in zwei Transitionen, die auf dieselben zwei Nachplätze führen. Am Ende des Maincharts findet eine Synchronisierung auf den Endplatz statt.

Auch Fall 1 für synchrone Nachrichten könnte wie Fall 2 behandelt werden, wodurch jedoch *tote* Transitionen entstehen würden, deren Vorplätze nie gleichzeitig belegt sein können und so nie schalten. Durch eine statische Analyse basierend auf einem Überdeckungsgraphen, wie sie in [Pat14] vorgeschlagen wird, können diese Transitionen erkannt und aus dem MPN entfernt werden.

Besitzt das während einer Übersetzung bisher erstellte MPN mehr als einen offenen Platz je zugeordneter Lebenslinie, wird für jede Kombination dieser Plätze

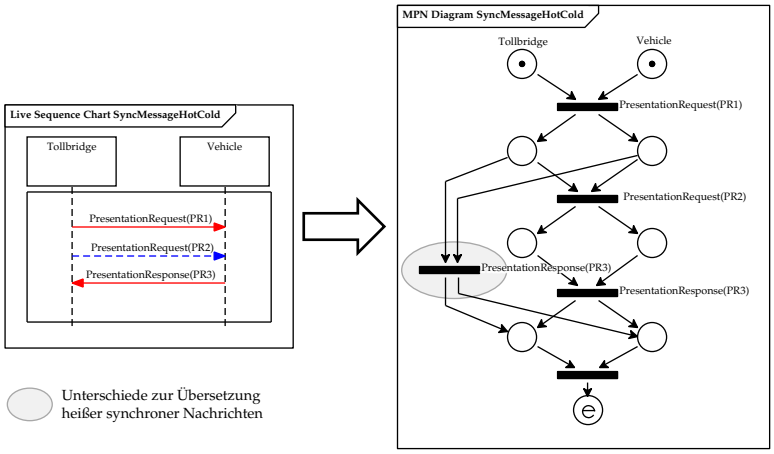


Abbildung 8.11: MPN-Repräsentation eines eLSCs mit heißen und kalten synchronen Nachrichten (Fall 1)

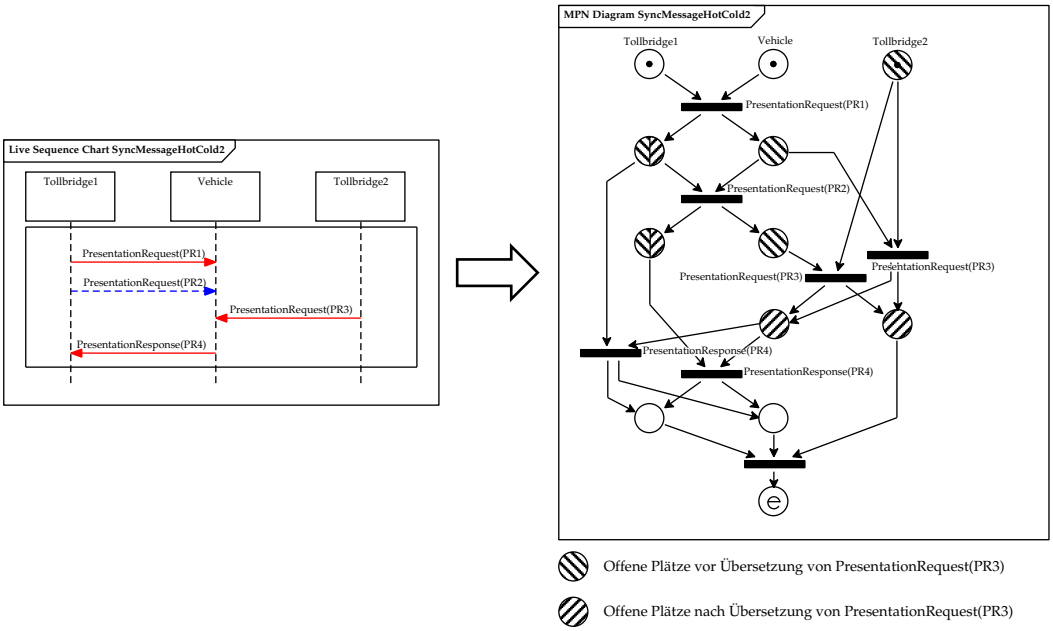


Abbildung 8.12: MPN-Repräsentation eines eLSCs mit heißen und kalten synchronen Nachrichten (Fall 2)

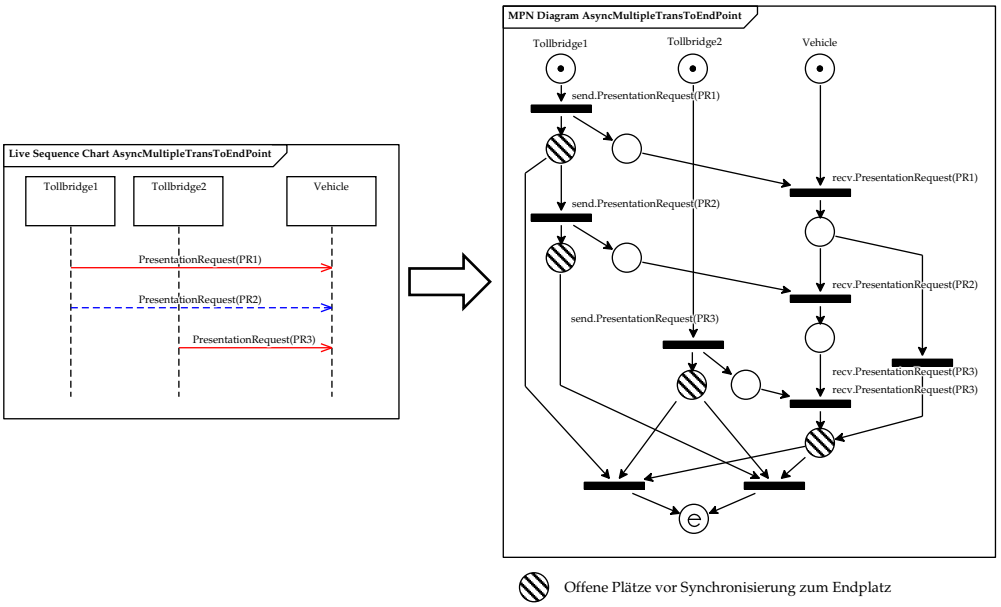


Abbildung 8.13: MPN-Repräsentation eines eLSCs mit kalter Location als letzte Location einer Lebenslinie

eine Transition erstellt und auf dieselben Nachplätze verbunden. Dies ist auch auf die Synchronisation am Übergang zwischen Pre- und Mainchart oder Ende des Maincharts anwendbar. Der Abschluss eines MPNs auf einen gemeinsamen Terminalplatz mit mehreren offenen Plätzen, die der Lebenslinie `Tollbridge1` zugeordnet sind, ist in Abbildung 8.13 dargestellt. Durch die kalte Location auf der Sendeseite der asynchronen Nachricht `PresentationRequest(PR2)` ohne folgende heiße Location auf der Lebenslinie `Tollbridge1`, gibt es zwei Möglichkeiten die Überwachung des MPNs abzuschließen. Hierfür wird wiederum für jede Kombination eine Transition erstellt, die auf denselben Endplatz führen.

8.1.3 *Komplexe Elemente zur Kontrollflusssteuerung*

Neben den grundlegenden Elementen der eLSC-Sprache müssen auch Elemente zur Kontrollflusssteuerung in die MPN-Syntax übersetzt werden, da mit diesen Elementen Signaturen kompakter in der eLSC-Sprache modelliert werden können. Hierzu gehören sowohl die aus den LSCs bekannten *Loop*- und *If-Then-Else*-Fragmente als auch die in eLSCs neu eingeführten *Par*-, *Alt*- und *Ref*-Fragmente.

Die Übersetzung eines *If-Then-Else*-Fragments mit einer eingebetteten unbeschränkten Schleife (*Loop*-Fragment) im *Then*-Subchart ist in Abbildung 8.14 dargestellt. Das *If-Then-Else*-Fragment synchronisiert zunächst alle offenen Plätze der beiden beteiligten Lebenslinien auf zwei neue Plätze des MPNs. Basierend auf der Bedingung des *Then*-Subcharts werden zwei Transitionen erzeugt, die beide die offenen Plätze nach der Synchronisierung als Vorplätze haben. An die Transition des *Then*-Teils des MPNs wird die Bedingung und an die Transition des *Else*-Teils die negierte Bedingung annotiert. Beide Transitionen haben jeweils

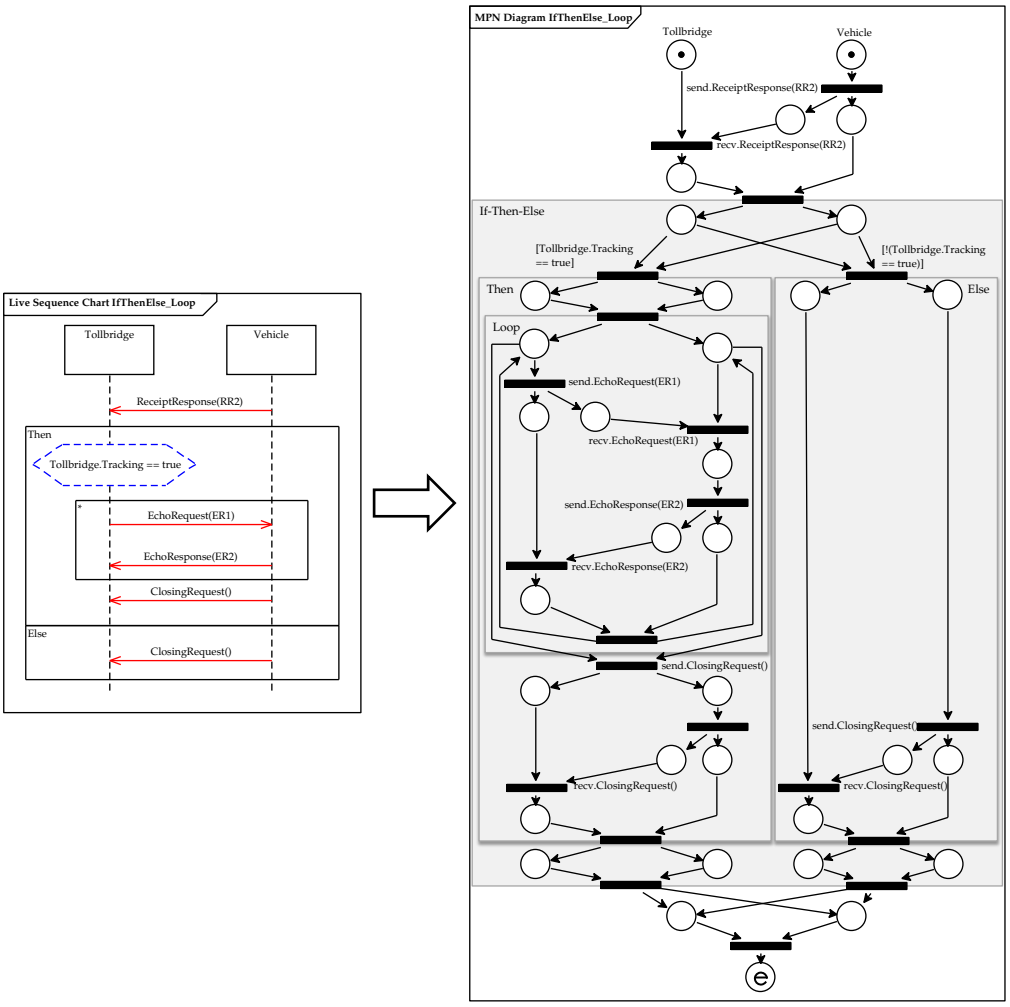
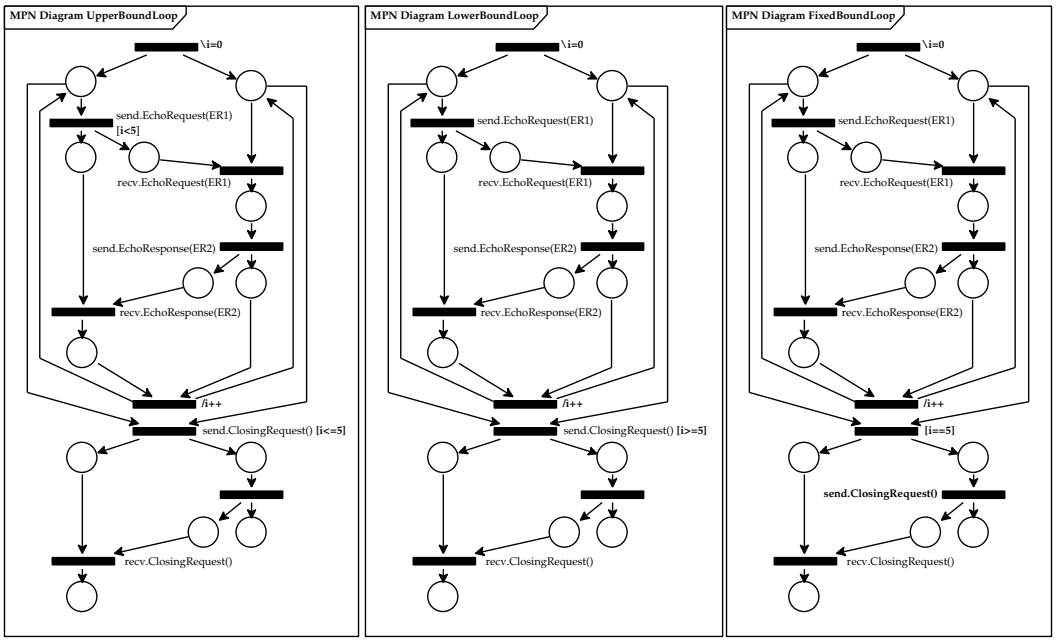


Abbildung 8.14: MPN-Repräsentation eines *If-Then-Else*-Fragments mit einer enthaltenen unbeschränkten Schleife

zwei Nachplätze – für jede Lebenslinie einen. An diese Plätze wird jeweils die MPN-Repräsentation des Inhalts des *Then*- und *Else*-Subcharts gehängt. Während das *Else*-Subchart nur aus einer asynchronen Nachricht `ClosingRequest()` besteht, können im *Then*-Subchart beliebig viele Echo-Nachrichten ausgetauscht werden, bevor die `ClosingRequest()`-Nachricht auftritt. Da es sich bei einer Schleife um einen Übergang zu einem Subchart handelt, werden die offenen Plätze der betroffenen Lebenslinien im *Then*-Teil auf zwei neue Plätze synchronisiert. Bei der im eLSC modellierten Schleife handelt es sich um eine kopfbasierte Schleife, deren Abbruchkriterium das Auftreten der nachfolgenden Nachricht `CloseRequest()` ist. Somit werden die MPN-Repräsentationen der beiden `Echo`-Nachrichten an die offenen Plätze im MPN angehängt und am Ende über eine gemeinsame Transition mit den Vorplätzen der Nachricht `EchoRequest(ER1)` zurück gekoppelt. Um das Eintreten von Null bis beliebig vielen vollständigen Iterationen der Schleife zu repräsentieren, wird die Nachricht, die den Abbruch der Schleife verursacht, ebenfalls an diese Vorplätze gehängt. Hierzu wird eine zusätzliche vorgelager-



a) 0..5 Iterationen b) 5..* Iterationen c) 5 Iterationen

Abbildung 8.15: MPN-Repräsentationen der *Loop*-Fragmente

te Transition mit dem Sendeereignis `send.ClosingRequest()` zur Synchronisation vor die Repräsentation der `ClosingRequest`-Nachricht gesetzt, um Fehlerkennungen zu verhindern. Die ursprüngliche Sendetransition wird zur Epsilon-Transition und schaltet somit im selben Schritt wie die vorgelagerte Transition. Am Ende des *If-Then-Else*-Fragments wird der *Then*- und der *Else*-Teil unabhängig voneinander durch eine eigene Transition synchronisiert und über jeweils eine weitere Transition auf zwei gemeinsame Plätze verbunden. Beendet wird die MPN-Repräsentation durch eine Synchronisierung auf einen Terminalplatz.

Neben der vorgestellten unbeschränkten Schleife können *Loop*-Fragmente mit verschiedenen Multiplizitäten versehen werden, die die obere und untere Schranke der Schleife festlegen. Abbildung 8.15 zeigt die MPN-Repräsentation einer Schleife mit einer oberen Schranke (`0..5`), einer Schleife mit einer unteren Schranke (`5..*`) und einer Schleife mit einer festen Anzahl an Durchläufen (`5`). Im Gegensatz zur MPN-Repräsentation der unbeschränkten Schleife muss für die weiteren Varianten eine Zählvariable `i` eingefügt werden, die die Durchläufe der Schleife bei der Überwachung protokolliert. Diese wird durch eine Annotation der Aktion `/i=0` an die Epsilon-Transition, die in das *Loop*-Fragment führt, initialisiert und durch die Aktion `/i++` an der rücksynchronisierenden Epsilon-Transition erhöht.

Eine Schleife mit einer *oberen Schranke* kann nun durch die Annotation von zwei Bedingungen realisiert werden. Eine Bedingung bewirkt, dass die Schleife nur durch ein Ereignis `send.ClosingRequest()` verlassen werden kann, wenn die Bedingung `[i<=5]`, d. h. die obere Schranke wurde eingehalten, erfüllt ist. Die andere Bedingung `[i<5]` verhindert das Betreten des Schleifenkörpers, wenn die obere Schranke überschritten ist. Ist die obere Schranke erreicht und das MPN wird wiederum mit dem ersten Ereignis der Schleife stimuliert, schaltet das MPN

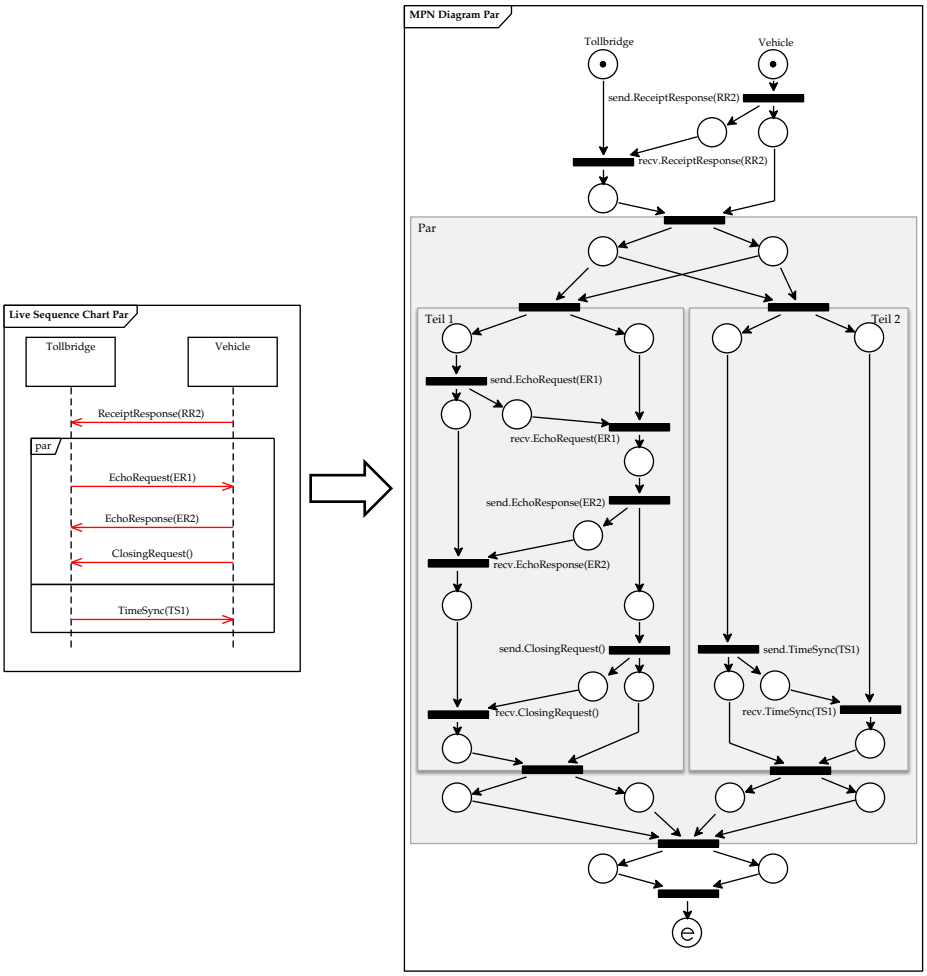


Abbildung 8.16: MPN-Repräsentation eines *Par*-Fragments

nicht und die Signatur wird nicht erkannt. Zur Beschreibung einer Schleife mit einer *unteren Schranke* wird nur das Verlassen der Schleife durch `[i>=5]` bedingt. Ein Ereignis `send.ClosingRequest()`, wenn die Zählvariable kleiner als das Minimum 5 ist, führt dies wie bei der oberen Schranke zu einem Nichtschalten des MPNs. Eine Schleife mit einer *festen Anzahl an Iterationen* wird durch dieselbe Struktur repräsentiert, die an der abschließenden Epsilon-Transition mit der Bedingung `[i==5]` annotiert wird. Das Sendeereignis der MPN-Repräsentation der `ClosingRequest`-Nachricht wird in diesem Fall nicht an die synchronisierende Transition vorgezogen.

Die MPN-Repräsentation des *Par*-Fragments entspricht syntaktisch grundlegend der des *If-Then-Else*-Fragments, wobei jedoch nicht nur zwei, sondern beliebig viele Stränge, parallel überwacht werden können. Hierbei werden, wie in Abschnitt 5.1 eingeführt, alle Subcharts des *Par*-Fragments unabhängig voneinander überwacht, wobei die partielle Ordnung in den einzelnen Subcharts eingehalten werden muss. In Abbildung 8.16 ist diese Übersetzung dargestellt. Der Übergang in das *Par*-Fragment wird in der der MPN-Repräsentation wie bei jedem Subchart durch eine Synchronisation der offenen Plätze der beteiligten Le-

benslinien über eine Transition ohne Ereignis modelliert. Zur Repräsentation der parallel überwachten Abläufe wird im MPN für jedes Subchart des *Par*-Fragments eine Transition erstellt, die die offenen Plätze als Vorplätze verwendet und auf eigene neue Nachplätze verbunden. Die der Anzahl der Nachplätze entspricht der am Fragment beteiligten Lebenslinien. Diese Transition besitzen im Gegensatz zum *If-Then-Else*-Fragment keine Bedingung, da sie keine Entscheidung darstellen, sondern parallel überwacht werden. Die in den Subcharts modellierten eLSC-Elemente werden nun, wie vorher vorgestellt, in die MPN-Repräsentation an die neu entstandenen Plätze angehängt. Als Abschluss der einzelnen Subcharts dient jeweils eine Transition, die die offenen Plätze in jedem Subchart des Fragments synchronisiert. Das Ende des *Par*-Fragments bildet eine Transition, die als Vorplätze alle offenen Plätze des Subcharts besitzt, um sicherzustellen das alle überwachten Subcharts des *Par*-Fragments erkannt wurden.

Die Übersetzung des *Alt*-Fragments in die MPN-Repräsentation stellt die größte Herausforderung an den Übersetzungsprozess dar, da die Entscheidung nicht anhand der kalten Bedingung wie bei einem *If-Then-Else*-Fragment getroffen werden kann. Werden die Alternativen als einzelne, parallel überwachte LSCs spezifiziert, können sie in einzelne unabhängige MPNs übersetzt werden, die unabhängig überwacht werden, worauf die MPN-Semantik ausgelegt ist. Die Schalt- bzw. Auswertungssemantik eines MPNs erlaubt es nicht, das *Alt*-Fragment wie das *Par*-Fragment zu übersetzten, da immer ausgewertet wird, ob das gesamte MPN geschaltet hat. Auch ist die Entscheidung häufig nicht anhand des ersten Ereignisses in den Subcharts des *Alt*-Fragments zu treffen, wie in Abbildung 8.17 zu sehen ist.

Für das *Alt*-Fragment bedeutet dies, dass alle Ereignisse auftreten können, die in einem der Subcharts des Fragments modelliert sind. Um eine fehlerhafte Abbildung des eLSCs zu verhindern, ist bei der Übersetzung die Analyse des Inhaltes der Subcharts auf gleiche Ereignissequenzen notwendig, die in der MPN-Repräsentation zusammengefasst und vor die Entscheidung vorgelagert werden. In Abbildung 8.17 teilen sich die ersten beiden Subcharts (`Teil 1+2`) die Ereignisse der Nachricht `EchoRequest(ER1)` und unterscheiden sich erst durch das Sendeereignis der folgenden Nachricht. Wie bei dem *If-Then-Else*- und *Par*-Fragment, findet bei Eintritt in das Fragment eine Synchronisierung und eine Aufteilung auf die zu überwachenden Stränge statt. Die Transitionen auf die einzelnen Stränge werden jedoch nicht mit der Bedingung oder als Epsilon-Transition modelliert, sondern es wird an ihnen das erste Ereignis im jeweiligen Subchart (`send.EchoRequest(ER1)` bzw. `send.ClosingRequest`) annotiert. Die diesem Ereignis ursprünglich entsprechende Transition aus der Übersetzung der Nachricht bleibt als Epsilon-Transition im MPN erhalten. Da in den Strängen `Teil 1` und `Teil 2` die Entscheidung erst mit der zweiten Nachricht des Charts getroffen werden kann, teilen sich beide Subcharts eine MPN-Repräsentation der ersten Nachricht und die Stränge werden erst für die darauf folgende Nachricht aufgeteilt. Die Synchronisation am Ende eines *Alt*-Fragments erfolgt über getrennte Transitionen zu gemeinsamen Nachplätzen. Die Anzahl der Nachplätze entspricht der Anzahl, der an dem Fragment beteiligten Lebenslinien. Durch diese Übersetzung des *Alt*-Fragments ist sicherge-

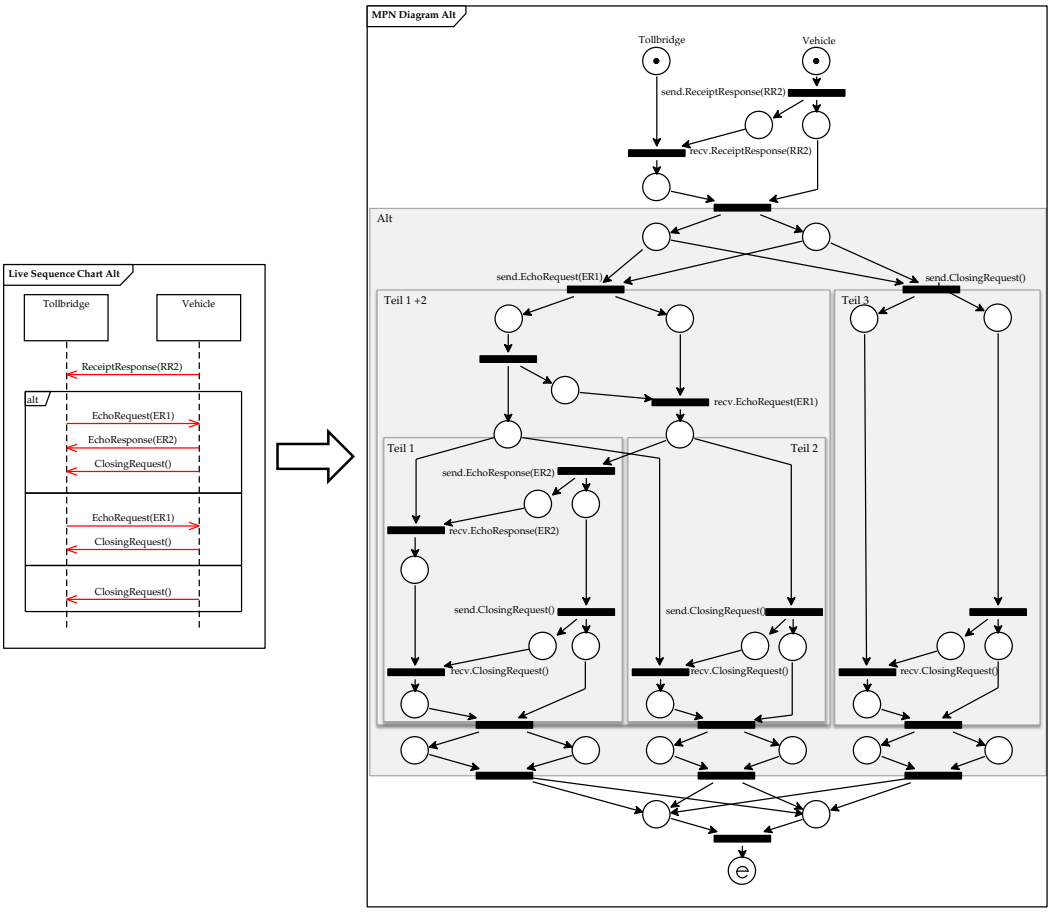


Abbildung 8.17: MPN-Repräsentation eines *Alt*-Fragments

stellt, dass die Auswertungssemantik der MPNs den Fehler sicher erkennt, wenn ein Ereignis auftritt, jedoch das MPN nicht schaltet.

Das *Ref*-Fragment synchronisiert wie alle anderen Fragmente in den eLSCs die offenen Plätze der beteiligten Lebenslinien. An diese wird dann das einzeln übersetzte, ausgelagerte eLSC an die offenen Plätze angehängt. Hierzu werden die Initialplätze durch vorhandene offene Plätze, die zur selben Lebenslinie gehören, ersetzt und am Ende die offen Plätze der MPN-Repräsentation des ausgelagerten eLSC durch eine Transition zu gemeinsamen Plätzen synchronisiert. Hierdurch wird die ausgelagerte Signatur an allen referenzierenden Stellen eingefügt.

8.1.4 *Chartplätze für Verbotene und Ignorierte Elemente*

Die eLSC-Konstrukte *Forbidden-Fragment* und *Ignore-Fragment* können, wie in Abschnitt 4.1 und 5.1 vorgestellt, sowohl für das gesamte eLSC-Diagramm gelten als auch nur dem Prechart, Mainchart oder einem Subchart zugeordnet werden. Zur Übersetzung dieser Elemente in die MPN-Repräsentation bieten sich mehrere Alternativen an. In allen Fällen führt das Auftreten verbotener Elemente zu einem

Abbruch der Überwachung der Signatur und das Auftreten ignorierter Elemente zu keinem Abbruch.

Eine verbotene Nachricht oder Bedingung wird in die MPN-Repräsentation durch Hinzufügen einer zusätzliche Transition an jedem Platz, der zu dem Gültigkeitsbereich des Forbidden-Fragments gehört, übersetzt. Diese Transitionen führen auf einen entsprechenden Terminalplatz. Zu ignorierende Nachrichten können als Eigentransition an jedem zugehörigen Platz annotiert werden. Hierdurch schaltet das MPN auch, wenn ein zu ignorierendes Ereignis an das MPN übergeben wird, und die implizite Abbruchbedingung der MPNs bei Nicht-Schalten, die in Abschnitt 2.3.2 vorgestellt wurde, tritt nicht ein. Dies resultiert jedoch in einer großen Anzahl zusätzlicher Transitionen mit gleichen Ereignissen bzw. Bedingungen, die es zu vermeiden gilt, um die Laufzeit des späteren Monitors zu reduzieren [Pat14].

Zur Reduzierung dieser redundanten Transitionen dient die Übersetzung jedes Pre-, Main- und Subcharts zu einem zusätzlichen Chartplatz (Def. 9) in der MPN-Repräsentation. An diese Plätze werden die Transitionen aus der Übersetzung der Ereignisse des *Ignore*-Fragments einmalig als Eigentransitionen und des *Forbidden*-Fragments als Transition auf einen Terminalplatz angehängt. Dies erleichtert zum einen eine Auswertung der MPN-Konfigurationen bei Fehlschlagen oder Abbruch der Überwachung der Signatur und führt zum anderen zu deutlich weniger Transitionen an den Plätzen des MPNs.

Definition 9 (*Chartplätze*)**.** Für jedes Pre-, Main- und Subchart des eLSC-Diagramms wird in der MPN-Repräsentation ein zusätzlicher Chartplatz angelegt, der die aktuell überwachte Teilsignatur repräsentiert. In Universal eLSCs ist dies für das Prechart ein vorausgehender Platz (*engl. antecedent place*), der als Initialplatz modelliert ist und dessen Belegung die Überwachung der Vorbedingung der Signatur repräsentiert. Das Maincharts wird in der MPN-Repräsentation als ein nachfolgenden Platz (*engl. consequent place*) abgebildet. Für ein Universal eLSC-Diagramm ist dies ein Standardplatz und für ein Existential eLSC ein Initialplatz, dessen Belegung die Überwachung der Hauptsignatur repräsentiert. Über die Synchronisationstransition zwischen Pre- und Mainchart findet im Universal eLSC die Kopplung des vorausgehenden und nachfolgenden Platzes statt. Der Platz des Maincharts wird als Vorplatz der Transition für die Synchronisierung zum Endplatz angebunden. Zusätzlich wird für jedes Subchart in Pre- oder Mainchart ein zusätzlicher Chartplatz angelegt, der bei Eintritt in diesen Bereich belegt und mit dem Verlassen des Bereichs abgeräumt wird.

In Abbildung 8.18 sind diese zusätzlichen Chartplätze auf der rechten Seite des MPNs platziert. Der vorausgehende Initialplatz für das Prechart ist mit $*$ markiert und wird an der Transition für den Übergang zwischen Pre- zu Mainchart auf den nachfolgenden Standardplatz für das Mainchart synchronisiert. Über die letzte Transition im MPN wird dieser nachfolgende Platz zum Endplatz verbunden. Zusätzlich wird ein weiterer Standardplatz als Nachplatz an die Eintrittstransition des Subchart-Bereichs angehängt, der als Vorplatz der Austrittstransition dieses Bereiches dient.

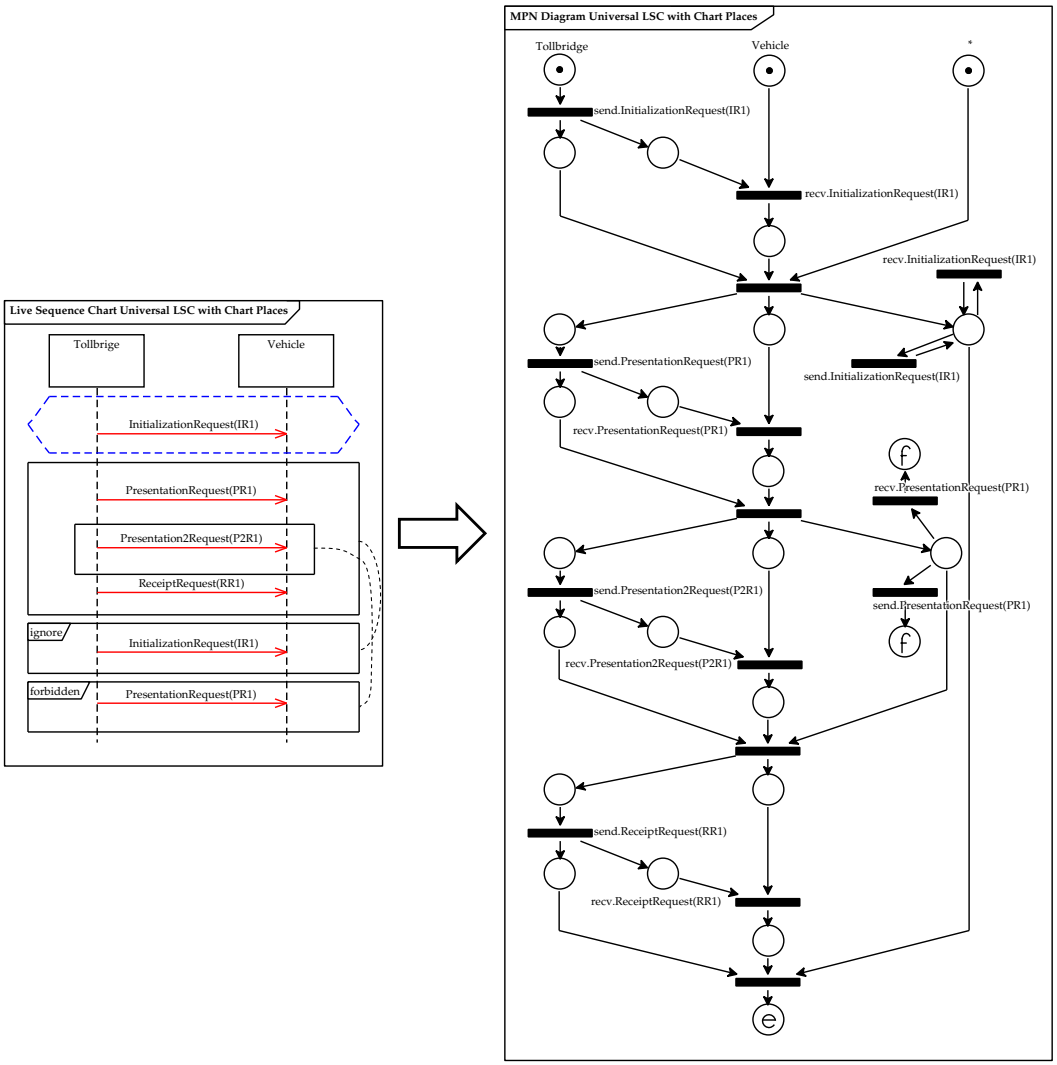


Abbildung 8.18: MPN-Repräsentation eines eLSCs mit verbotenen und zu ignorierenden Nachrichten

Basierend auf den Chartplätzen können nun *Forbidden*- und *Ignore*-Fragmente für eine effiziente Überwachung übersetzt werden, indem die Transitionen nur einmalig an diese Chartplätze angehängt werden[1]. In Abbildung 8.18 ist diese Übersetzung exemplarisch anhand eines Ignore-Fragments, das dem Mainchart und eines Forbidden-Fragments, das dem Subchart im Mainchart zugeordnet ist, dargestellt. Die Sende- und Empfangsereignisse der zu ignorierenden Nachricht werden als Eigentransitionen am Chartplatz des Maincharts übersetzt. Dieser Platz ist mit einer Markierung belegt, solange das Mainchart überwacht wird. Bei Auftreten eines der beiden Ereignisse schaltet das MPN, wodurch die Nachricht im überwachten System zu keinem Fehlschlagen des MPNs führt und somit ignoriert wird. Die Nachricht im Forbidden-Fragment wird ebenfalls zu zwei

[1] Hierbei wird von einer eindeutigen Benennung der Sende- und Empfangsereignisse ausgegangen. So dürfen keine zwei Nachrichten mit derselben Signatur existieren, die sowohl von der Mautbrücke als auch von dem Fahrzeug versendet werden kann, da die Chartplätze keinen Lebenslinien des LSCs zugeordnet sind.

Transitionen an dem Chartplatz des Subcharts übersetzt. Hierbei handelt es sich jedoch um Transitionen, die auf einen Fehlerplatz führen, da diese Nachricht in diesem Bereich des MPNs nicht auftreten darf.

8.2 Abbildung der MUC-Sprache in konkreter Syntax

Durch den Einsatz der MUC-Sprache (Abschn. 5.2) zur Strukturierung der eLSC-Spezifikation können diese nicht nur übersichtlicher spezifiziert werden, sondern es kann ebenfalls der Umfang der durch die Abbildung entstanden MPNs reduziert werden. Die vier Beziehungen der MUC-Sprache zwischen den eLSCs können durch das in Abschnitt 2.3.2 vorgestellte allgemeine Referenzsystem der MPN-Sprache [Pat14] ausgedrückt werden. Hierzu wird in den Abschnitten 8.2.1 bis 8.2.4 diese Übersetzung der «include»-, «extend»-, «threaten»- und «mitigate»-Beziehung in das Referenzsystem der MPN-Sprache vorgestellt. Die Signaturen des CARDME-Protokolls, die als eLSCs in Kapitel 5 vorgestellt wurden, werden zur besseren Darstellbarkeit der Transformation der Beziehungen zwischen den Anwendungsfällen in diesem Abschnitt vereinfacht. Hierfür werden Ignore-Fragmente ausgeblendet und Teilsignaturen durch Ref-Fragmente abstrahiert.

8.2.1 *«include»-Beziehung*

Die Übersetzung der «include»-Beziehung in das Referenzsystem der MPN-Sprache reduziert das unnötige Anwachsen der MPN-Repräsentation der Signatur, das durch das mehrfache Einfügen referenzierter Teilnetze entstehen würde. Wie in Abschnitt 5.2 in Abbildung 5.6 gezeigt, dürfen sowohl Use-Cases (*Fall 1*) als auch Misuse-Cases (*Fall 2*) über eine «include»-Beziehung eLSCs aus Use-Cases inkludieren. Zur grafischen Darstellung von Teilabläufen (eLSCs), die nur aus Misuse-Cases referenziert werden dürfen, existiert ein dritter Fall (*Fall 3*), in dem der inkludierte Anwendungsfall ein Misuse-Case ist. Die Übersetzung dieser drei Fälle unterscheidet sich hierbei nur in der Auswahl der Terminalplätze in der MPN-Repräsentation. In Fall 1 werden die eLSCs des inkludierten Use-Cases, wie in Abschnitt 8.1 beschrieben, übersetzt. Nicht erfüllte heiße Bedingungen führen auf Fehlerplätze und am Ende des eLSCs findet in der MPN-Repräsentation eine Synchronisierung auf Endplätze statt. In Fall 2 muss die MPN-Repräsentation einer nicht erfüllten heißen Bedingung auf Endplätze statt Fehlerplätze führen, da ein Misuse-Case erst eingetreten ist, wenn der gesamte Basis-Misuse-Case inklusive des inkludierten Use-Cases erkannt wurde. Im dritten Fall wird das eLSC als Misuse-Case-Signatur übersetzt, jedoch werden am Ende des eLSCs die offenen Plätze nicht auf Fehlerplätze, sondern auf Endplätze synchronisiert, da es sich hierbei wie im vorherigen Fall nur um eine Teilsignatur handelt.

Abgesehen von der Generierung der passenden Terminalplätze für die inkludierten eLSCs, wird die «include»-Beziehung zwischen Anwendungsfällen, wie in Abbildung 8.19 exemplarisch dargestellt, übersetzt. Im Folgenden werden in diesem Abschnitt alle Chartplätze der referenzierten MPNs (`refMPNs`) ausgeblendet, die für die Transformation der MUC-Sprachelemente nicht relevant sind.

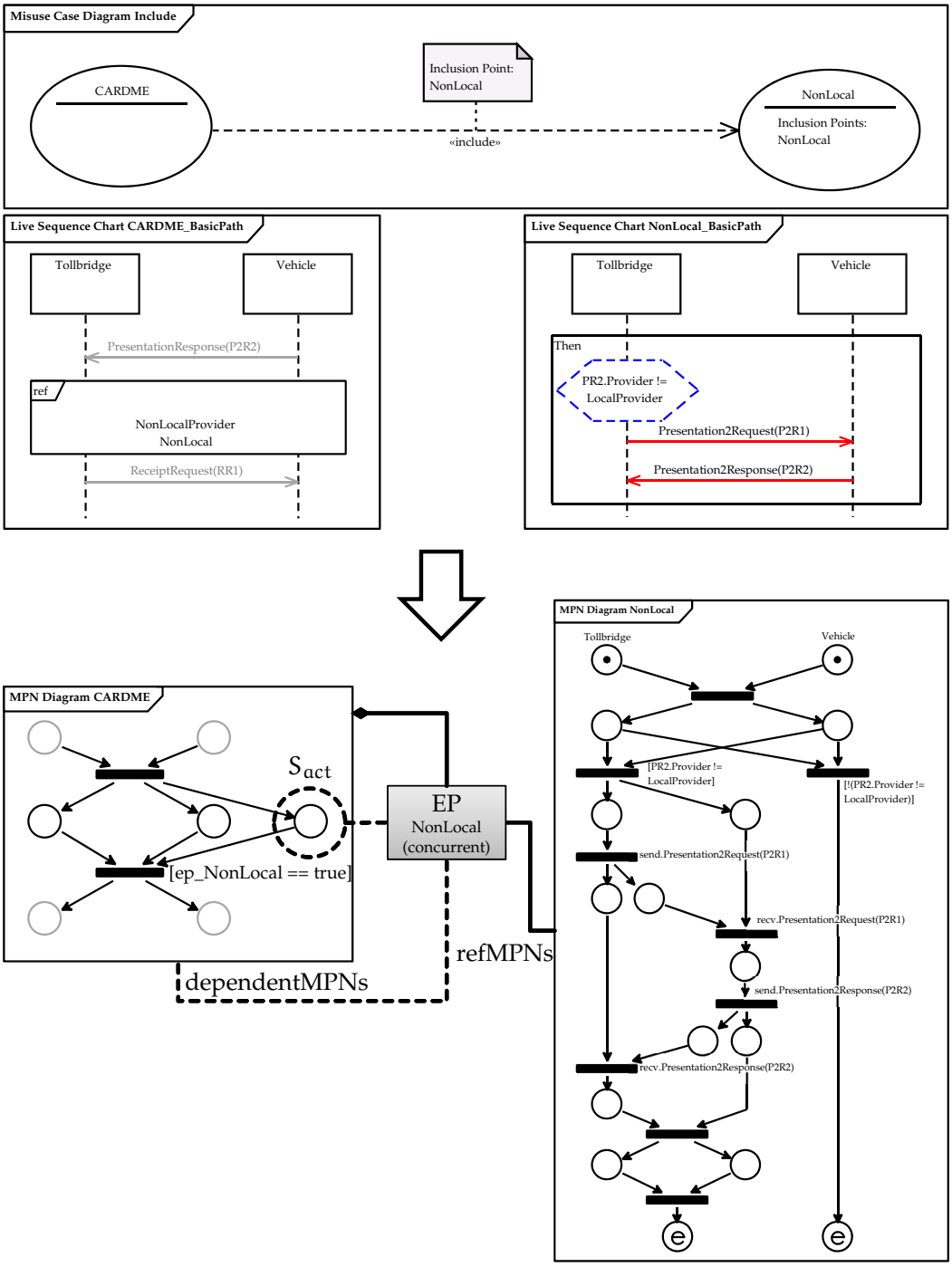


Abbildung 8.19: Übersetzung der «include»-Beziehung in das Referenzsystem der MPNs

Die oberen Hälfte der Abbildung 8.19 zeigt einen Ausschnitt der CARDME-Spezifikation in der MBSecMonSL-Sprache, die Fall 1 entspricht. Die «include»-Beziehung zwischen den Use-Cases `CARDME` und `NonLocal` stellt dem Basis-Use-Case `CARDME` über das Notizelement den Inklusionspunkt `NonLocal` zur Verfügung. Der Inklusionspunkt `NonLocal` repräsentiert das eLSC `NonLocal_BasicPath`. Dieser wird im Ausschnitt des eLSCs `CARDME_BasicPath` durch das Ref-Fragment `NonLocalProvider` referenziert.

Übersetzung der «include»-Beziehung: Die Übersetzung der einzelnen eLSCs erfolgt wie in Abschnitt 8 beschrieben, wobei das Ref-Fragment, wie jedes Chart in der MPN-Repräsentation, einen Chartplatz erhält. Als MPN-Repräsentation der «include»-Beziehung wird dem MPN, das aus dem eLSC des Basis-Use-Case entstanden ist, für jedes Ref-Fragment ein Erweiterungspunkt (EP) der MPN-Sprache hinzugefügt. In diesem Beispiel ist dies nur ein EP. Dieser wird als nebenläufig (*engl. concurrent*) markiert, wodurch gekennzeichnet wird, dass die Überwachung der Signatur `CARDME` nicht angehalten wird, wenn die referenzierten Signaturen (`refMPNs`) überwacht werden. Um das sofortige Verlassen der Plätze, die aus dem Ref-Fragment des Basis-Use-Cases entstanden sind, über die Epsilon-Transition zu blockieren, besitzt der EP eine boolesche Variable `ep_NonLocal`, die erst auf `true` gesetzt wird, wenn eine der referenzierten Teilsignaturen eingetreten ist. Hierfür wird diese Epsilon-Transition mit der Bedingung `[ep_NonLocal == true]` annotiert. Der EP referenziert die zu inkludierende Teilsignatur als MPN (`refMPNs`) und kennt alle MPNs (`dependentMPNs`), die für die Auswertung der Gesamtsignatur relevant sind. In diesem Beispiel wird der Use-Case `CARDME` von keinem weiteren Use-Case inkludiert, wodurch der EP nur `CARDME` als `dependentMPN` kennt. Zusätzlich zu den MPNs werden dem EP eine Menge von Aktivierungsplätzen (Chartplätze) $S_{\mathtt{act}}$ zugeordnet, die bei Belegung durch ein Token zu einer Aktivierung des MPNs in `refMPNs` führt. Im Beispiel existiert nur ein Aktivierungsplatz – der Chartplatz aus der Übersetzung des Ref-Fragments.

Sind mehr als zwei Lebenslinien in einem eLSC an der Signatur beteiligt, kann es vorkommen, dass das Ref-Fragment nicht alle Lebenslinien überspannt. In diesem Fall würde eine komplette Blockierung des Basis-MPNs, während die referenzierten MPNs überwacht werden, zu Fehlerkennungen führen, da das Basis-MPN nebenläufig weiter überwacht werden muss. Diese teilweise Blockierung des Basis-MPNs wird durch die im Beispiel vorgestellte Bedingung an der Endtransition des Ref-Fragments ermöglicht, da so nur der überlappte Teil der Signatur blockiert wird. Zur Auswertung, ob eine Signatur erkannt wurde oder fehlgeschlagen ist, werden bei kaskadierter Anwendung der «include»-Beziehung die referenzierten `dependentMPNs` zur Auswertung genutzt.

8.2.2 *«extend»-Beziehung*

Die Einbeziehung der «extend»-Beziehung in die MBSecMonSL bewirkt, dass alternative Teilabläufe nicht mehrfach komplett als eLSC mit allen ihren Vorbedingungen modelliert werden müssen. Mit dieser Beziehung können, wie in Abschnitt 5.2 in Abbildung 5.6 gezeigt, Teilsignaturen in Use-Cases die Signaturen in einem Basis-Use-Case und Teilsignaturen in Misuse-Cases die Signaturen des

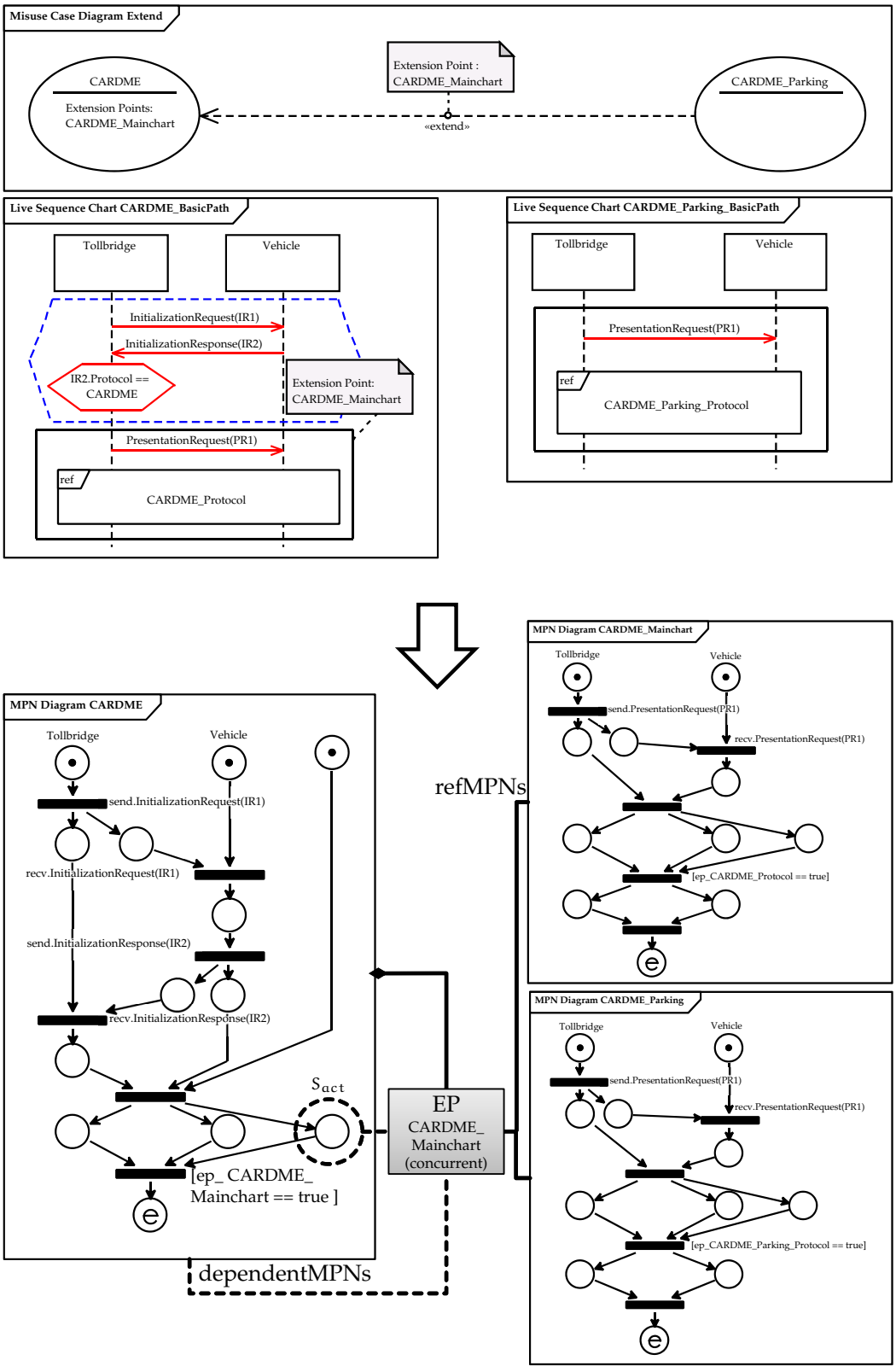


Abbildung 8.20: Übersetzung der «extend»-Beziehung in das Referenzsystem der MPNs

Basis-Misuse-Cases erweitern. Wie in Fall 3 der Transformation der inkludierten Anwendungsfälle in MPNs, müssen bei der Übersetzung der erweiternden Misuse-Cases die Fehlerplätze am Ende des Misuse-Case-MPNs durch Endplätze ersetzt werden, da es sich hier nur um eine alternative Teilsignatur handelt. Erst der vollständige Durchlauf durch die Signatur des Basis-Misuse-Cases führt zur Fehlerfallerkennung.

Die obere Hälfte der Abbildung 8.20 zeigt eine «extend»-Beziehung zwischen dem Basis-Use-Case `CARDME`, der durch den Use-Case `CARDME_Parking` erweitert wird. Das eLSC `CARDME_Parking_BasicPath` kann aufgrund der gleichen Vorbedingung wie das eLSC `CARDME_BasicPath` durch eine `extend`-Beziehung vereinfacht werden. Der Erweiterungspunkt `CARDME_Mainchart` des Use-Cases `CARDME` ist dem Mainchart des eLSCs zugeordnet und beschreibt hierdurch, dass das eLSC `CARDME_Parking_BasicPath` parallel zum Mainchart des eLSCs `CARDME_BasicPath` überwacht werden muss.

Übersetzung der «extend»-Beziehung: Die Abbildung der «extend»-Beziehung auf einen Erweiterungspunkt der MPNs verlangt die Auslagerung des Inhaltes des Erweiterungsfragments (hier das Mainchart des eLSCs `CARDME_BasicPath`) in ein eigenes MPN. Im Gegensatz hierzu, ist diese Auslagerung bei der «include»-Beziehung durch den Einsatz des Ref-Fragments schon explizit in der MBSecMonSL-Spezifikation enthalten. Somit entsteht im Basis-MPN eine äquivalente Struktur wie bei einer Übersetzung eines Ref-Fragments. Die entsprechende Transition im Basis-MPN, die erst nach erfolgreicher Beendigung eines der ausgelagerten MPNs schalten darf, wird mit der Bedingung `[ep_CARDME_Mainchart == true]` annotiert. Hierbei repräsentiert die boolesche Variable `ep_CARDME_Mainchart` die erfolgreiche Überwachung eines der beiden ausgelagerten MPNs. Der Menge S_{act} des EP wird der Chartplatz des Erweiterungsfragments zugeordnet. Das ausgelagerte MPN `CARDME_Mainchart` wird ebenso wie die alternative Signatur `CARDME_Parking` durch den EP referenziert.

Basierend auf dieser Extraktion des Erweiterungsfragments als eigenes MPN kann nun die «extend»-Beziehung in einen als nebenläufig markierten Erweiterungspunkt übersetzt werden. In diesem Beispiel sind somit das ausgelagerte Mainchart `CARDME_Mainchart` und die alternative Signatur `CARDME_Parking` die referenzierten MPNs des EP (`refMPNs`). Das MPN `CARDME` ist das Basis-MPN der ausgelagerten Signaturen und die ausgelagerten Signaturen sind somit für die Auswertung abhängig von `CARDME`. Dies wird durch das Hinzufügen des MPNs `CARDME` als Teilabhängigkeit des EP in die Menge `dependentMPNs` repräsentiert.

8.2.3 *«threaten»-Beziehung*

Die «threaten»-Beziehung der MUC-Sprache erlaubt es Misuse-Cases nur dann zu überwachen, wenn die Signatur des Misuse-Cases auch tatsächlich auftreten kann. Hierzu kann der bedrohende Misuse-Case über einen Erweiterungspunkt einem Basis-Use-Case zugeordnet werden. Der Erweiterungspunkt ist in den eLSCs des Basis-Misuse-Cases einem oder mehreren Erweiterungsfragmenten zugeordnet. Hierdurch kann die Laufzeit der generierten Monitore reduziert werden, da die Signatur des bedrohenden Misuse-Cases nicht die gesamte Vorbedingung, die der

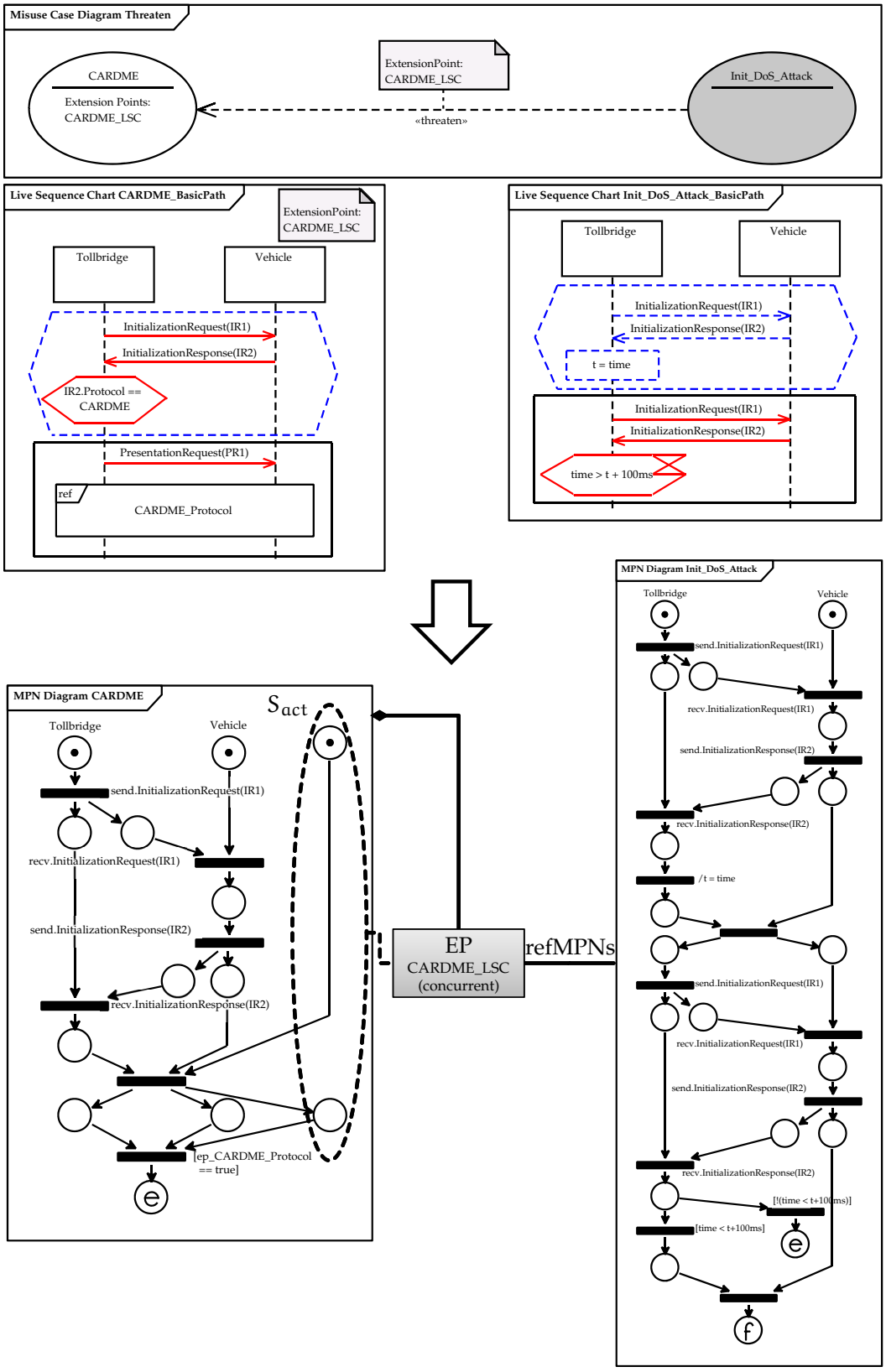


Abbildung 8.21: Übersetzung der «threaten»-Beziehung in das Referenzsystem der MPNs

Basis-Misuse-Case beschreibt, enthalten muss. Die Vorbedingung wird durch den Basis-Misuse-Case beschrieben. Die Übersetzung der eLSC findet in diesem Fall, wie in Abschnitt 8.1 beschrieben, statt.

Die obere Hälfte von Abbildung 8.21 zeigt einen Ausschnitt aus dem CARDME-Beispiel für eine «threaten»-Beziehung in der MBSecMonSL-Sprache. Der Use-Case `CARDME` wird durch den Misuse-Case `Init_DoS_Attack` bedroht. In diesem Fall ist das gesamte eLSC dem Erweiterungspunkt des Use-Cases zugeordnet und somit das Erweiterungsfragment. Dies bringt zwar im späteren Monitor keine Laufzeitvorteile, repräsentiert jedoch die Beziehung der beiden Signaturen auf einer höheren Abstraktionsebene – der MUC-Sprache.

Übersetzung der «threaten»-Beziehung: Die «threaten»-Beziehung wird im MPN-Referenzsystem ebenfalls durch einen nebenläufigen Erweiterungspunkt repräsentiert, der vom MPN `CARDME` besessen wird. Das MPN des Misuse-Cases wird in den EP als referenziertes MPN (`refMPNs`) eingehängt. Im Gegensatz zur Übersetzung der «extend»-Beziehung erhält in diesem Fall der EP keine Teilabhängigkeit zu den Basis-MPNs (`dependentMPNs`) zugeordnet, da der Misuse-Case eine vollständige Signatur repräsentiert, die einzeln ausgewertet werden kann. Zudem wird das MPN `CARDME` nicht durch eine Bedingung an der letzten Epsilon-Transition durch den Erweiterungspunkt blockiert. Die Aktivierungsplätze S_{act} sind in diesem Beispiel sowohl der Chartplatz des Precharts als auch des Maincharts des eLSCs, damit die Überwachung der Misuse-Case-Signatur mit der Überwachung des Precharts gestartet wird und auch im Mainchart aktiv bleibt. Erst nach der Erkennung der gesamten Signatur im Use-Case werden alle Chartplätze in S_{act} verlassen und somit auch die Überwachung des Misuse-Cases beendet.

8.2.4 *«mitigate»-Beziehung*

Durch die «mitigate»-Beziehung können bestimmten Teilen eines Basis-Use-Cases (Erweiterungsfragmenten der eLSCs) oder ganzen eLSC eines Basis-Misuse-Cases Gegenmaßnahmen zugeordnet werden. Diese Zuordnung erfolgt über die Erweiterungspunkte der MUC-Sprache, die den Erweiterungsfragmenten der eLSCs zugeordnet werden. Die Übersetzung der eLSCs beider Anwendungsfälle erfolgt, wie in Abschnitt 8.1 beschrieben.

In der oberen Hälfte von Abbildung 8.22 sind als Ausschnitt des CARDME-Protokolls der Basis-Misuse-Case `Init_DoS_Attack` mit dem abmildernden Use-Case `CloseConnection` dargestellt. Die «mitigate»-Beziehung referenziert über den Erweiterungspunkt `DoS_Attack_Detected` des Basis-Misuse-Cases das gesamte eLSC mit dem Namen `Init_DoS_Attack_BasicPath`. Wird der Misuse-Case vollständig erkannt, wird die Gegenmaßnahme ausgelöst.

Übersetzung der «mitigate»-Beziehung: Die Übersetzung der «mitigate»-Beziehung erfolgt im Gegensatz zu allen vorherigen Beziehungen in einen Erweiterungspunkt der MPNs, der als *nicht* nebenläufig markiert ist. Dies bewirkt, dass die zugeordnete Gegenmaßnahme nicht nebenläufig zu dem markierten Erweiterungsfragment des eLSCs überwacht wird, sondern nachfolgend ausgeführt

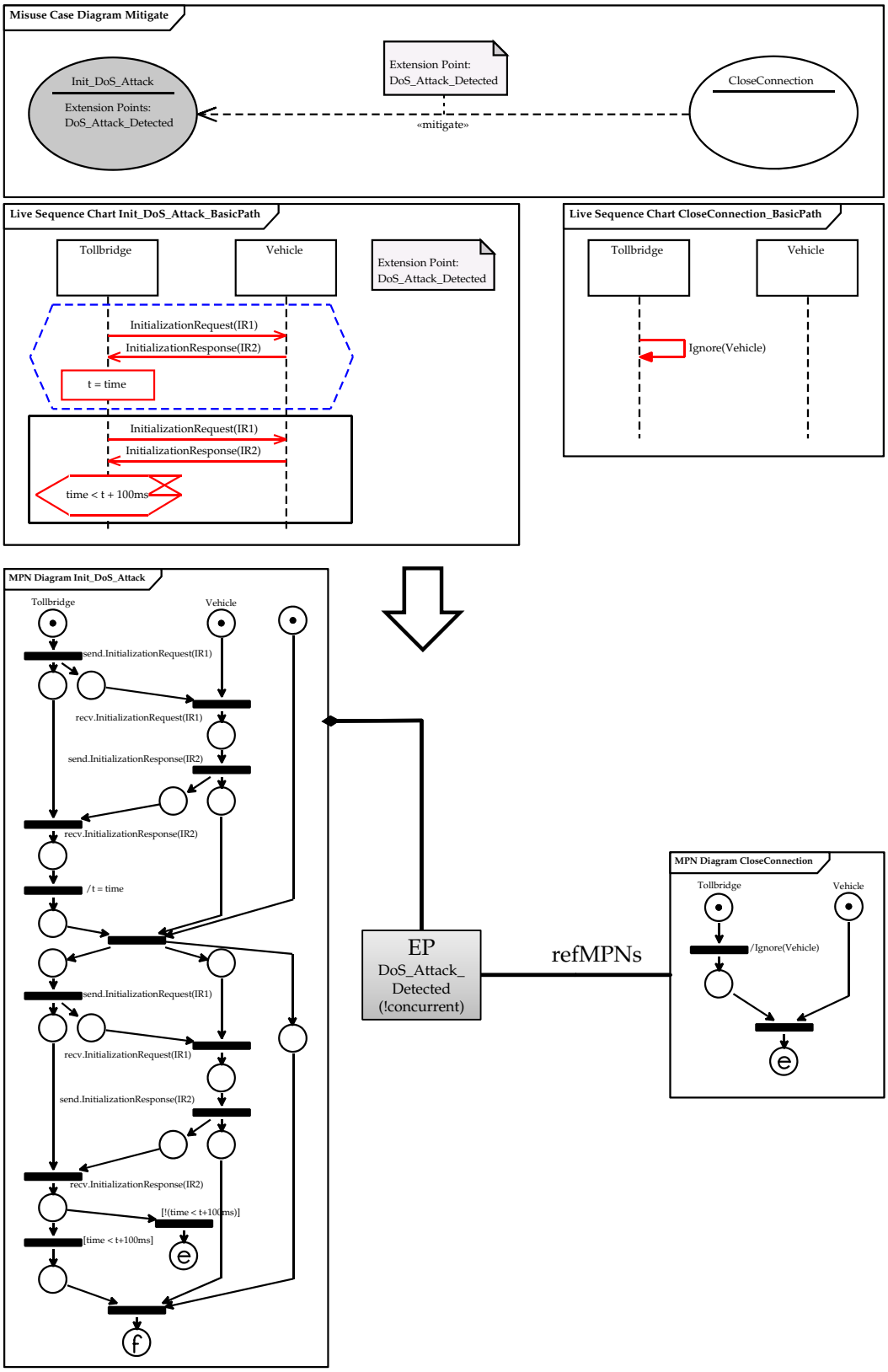


Abbildung 8.22: Übersetzung der «mitigate»-Beziehung in das Referenzsystem der MPNs

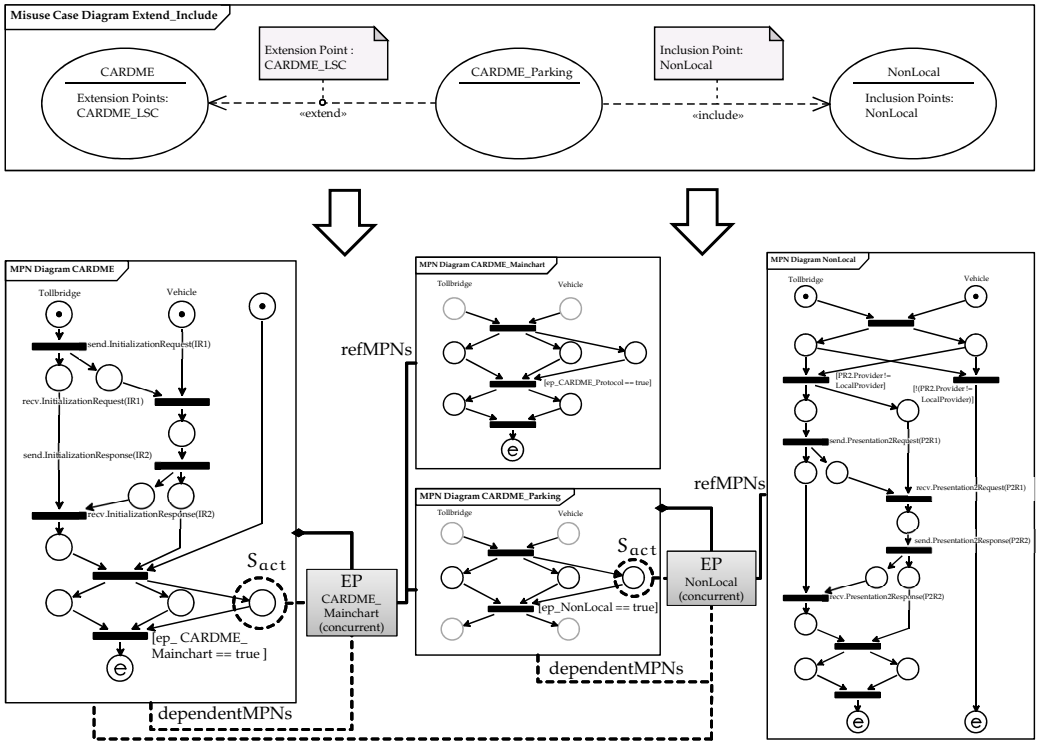


Abbildung 8.23: Kaskadierung von «include» und «extend»-Beziehung im Referenzsystem der MPNs

wird. In diesem Beispiel eines Misuse-Cases wird die Gegenmaßnahme erst ausgeführt, wenn der Fehlerplatz des MPNs `Init_DoS_Attack` erreicht wurde. Die Gegenmaßnahme ist dem EP als referenziertes MPN (`refMPNs`) bekannt. In einem Use-Case kann der EP auch einzelnen Chartplätzen über S_{act} zugeordnet werden. Ist ein Platz aus S_{act} belegt, wenn ein Fehlerplatz im MPN erreicht wird und das MPN fehlschlagen lässt, wird die entsprechende Gegenmaßnahme ausgeführt. S_{act} dient somit als zusätzliche Vorbedingung für die Auswahl des auszuführenden abmildernden Use-Cases. Da eine Gegenmaßnahme ausgeführt wird, und ihre Auswertung von keinen anderen MPNs abhängt, werden ihr keine `dependentMPNs` zugeordnet.

8.3 Kaskadierung der Beziehungen der MUC-Sprache in konkreter Syntax

Neben den einfachen Beziehungen zwischen zwei Use-Cases bzw. Use- und Misuse-Cases muss bei der Übersetzung in das MPN-Referenzsystem auch die Kaskadierung von Beziehungen beachtet werden. Während die «threaten»- und «mitigate»-Beziehung immer ohne Berücksichtigung der anderen Beziehungen, wie in Abschnitt 8.2 vorgestellt, übersetzt werden, müssen bei der Übersetzung der «include»- und «extend»-Beziehungen weitere Abhängigkeiten berücksichtigt werden, die aus der Kaskadierung von Use- und Misuse-Cases entstehen. Hierbei existiert in der Übersetzung kein Unterschied, ob nur «extend»-, nur «include»- oder eine Mischung von aufeinander folgenden «extend»- und «include»-Beziehungen in der MUC-Spezifikation auftreten.

Abbildung 8.23 zeigt ein MUC-Diagramm, in dem der erweiternde Use-Case `CARDME_Parking` ein Teilszenario aus dem Use-Case `NonLocal` inkludiert. Zunächst werden die einzelnen Beziehungen wie in Abschnitt 8.2 in einen Erweiterungspunkt der MPNs übersetzt, der durch den Basis-Use-Case besessen wird. Dieser Erweiterungspunkt referenziert die MPNs, die aus der Übersetzung des erweiternden bzw. inkludierten Use-Cases entstanden sind (`refMPNs`) und definiert über die Menge S_{act}, wann diese MPNs überwacht werden. Der einzige Unterschied zur einzelnen Übersetzung dieser Beziehungen besteht in der Bestimmung der für die Auswertung der Gesamtsignatur, die aus allen vier Teilsignaturen besteht, notwendigen Abhängigkeiten. Alle Basis-Use-Cases die über «threaten» und «include»-Beziehungen erreichbar sind, werden als `dependentMPNs` dem Erweiterungspunkt zugeordnet. Bei einem Fehlschlagen einer der Teilsignaturen beschreibt `dependentMPNs` die abhängig von diesem MPN auszuwertenden Signaturen. Der MPN-Erweiterungspunkt `CARDME_Mainchart`, der aus der «extend»-Beziehung zwischen `CARDME` und `CARDME_Parking` entsteht, besitzt als `dependentMPNs` nur das allgemeine `CARDME`-Protokoll. Der Erweiterungspunkt `NonLocal`, der aus der «include»-Beziehung entstanden ist, besitzt die beiden MPNs `CARDME` und `CARDME_Parking` als `dependentMPNs`.

8.4 Modellierungsrichtlinien der MBSecMonSL

Nicht alle möglichen Signaturen, die als eLSC syntaktisch modellierbar sind, sind semantisch sinnvoll oder ohne die Beachtung vieler Sonderfälle in einen Monitor übersetzbar. In diesem Abschnitt werden sechs Modellierungsregeln für eLSCs aufgestellt, die absichern, dass semantisch sinnvolle Signaturen modelliert werden, die in Monitore überführbar sind.

Einschränkung für kalte Nachrichten: Bei der Modellierung der eLSC-Signaturen sollten keine Signaturen modelliert werden, die nicht überwachbar sind. So ist die Bedeutung einer kalten Nachricht unmittelbar vor Ende des Pre-, Main-, Subcharts oder eines der zusätzlichen Fragmente nicht klar definiert. Wird diese Nachricht durch den Monitor ignoriert? Wird auf diese Nachricht gewartet? Wie lange wird auf diese Nachricht gewartet? In der MBSecMonSL werden diese nicht bzw. sehr aufwendig zu übersetzende Situationen explizit durch *Modellierungsregel 1* ausgeschlossen.

Modellierungsregel 1: Der Inhalt eines Pre-, Main- oder Subcharts darf nicht durch eine kalte Nachricht abgeschlossen werden, wenn auf den beteiligten Lebenslinien außerhalb des Subcharts als Nächstes keine heiße Location folgt.

Einschränkung für das *Loop*-Fragment: Ein weiteres Problem in der Übersetzung in die MPN-Repräsentation tritt beim Einsatz eines unbeschränkten *Loop*-Fragments (Abb. 8.14) ein, das durch die auf das *Loop*-Fragment folgende Nachricht abgebrochen wird. Wie in Abschnitt 8.1.3 in Abbildung 8.14 gezeigt, wird das Sendeereignis der nachfolgenden Nachricht an der Synchronisierungstransition annotiert, um nur bei Auftreten des Sendeereignisses die Schleife zu verlassen. Ist

jedoch der Typ der ersten Nachricht des *Loop*-Fragments und der nachfolgenden Nachricht identisch, kann bei Auftreten des Ereignisses nicht eindeutig entschieden werden, ob die Schleife erneut betreten wird oder verlassen werden muss. Dieses Problem ist jedoch nicht nur auf die MPN-Sprache begrenzt. So zeigt auch [KM09] bei der Übersetzung von LSCs in Automaten dasselbe Problem auf. Aus diesem Grund wird dieser Fall in der MBSecMonSL durch *Modellierungsregel 2* ausgeschlossen.

Modellierungsregel 2: Der Inhalt eines unbeschränkten *Loop*-Fragments darf nicht mit einer Nachricht desselben Nachrichtentyps beginnen, mit dem auch der Abbruch diese *Loop*-Fragments modelliert ist.

Einschränkung für das *Alt*-Fragment: Die in Abschnitt 8.1 vorgestellte Übersetzung des *Alt*-Fragments in die MPN-Repräsentation, erfordert eine aufwendige Analyse des Inhaltes der modellierten Alternativen. Beginnen mehrere Alternativen mit denselben Nachrichtentypen oder Bedingungen, müssen diese aufgrund der impliziten Auswertung der MPN-Semantik bei Nicht-Schalten des MPNs, wie in Abbildung 8.17 gezeigt zusammengefasst werden. Durch *Modellierungsregel 4* wird eine Vereinfachung des Übersetzungsprozesses erreicht, die jedoch durch eine Erweiterung der in Abschnitt 9 vorgestellten Transformationsimplementierung aufgehoben werden kann.

Modellierungsregel 3 (optional): Zur Transformationsvereinfachung des *Alt*-Fragments müssen die Bedingungen bzw. Nachrichten am Anfang jeder Alternative zueinander disjunkt sein.

Für die in den Abschnitten 8.1 und 8.2 beschriebene Abbildung der MBSecMonSL in die MPN-Sprache, existieren für die Auslagerung von Teilsignaturen in einen Anwendungsfall zwei Einschränkungen für die Modellierung in der MBSecMonSL.

Einschränkung für Instanzen: Eine weitere Modellierungsregel ist zur Absicherung der Übersetzbarkeit in Laufzeitmonitore notwendig. Werden Teile einer eLSC-Signatur über eine «include»-Beziehung in Teilsignaturen ausgelagert, könnten in ihnen zusätzliche Kommunikationspartner als Lebenslinien modelliert werden, die nicht an der Kommunikation der Hauptsignatur im Basis-Use-Case teilnehmen. Zur Umsetzung dieser Signaturen in einen Laufzeitmonitor müssten diese Kommunikationspartner dynamisch bei Aktivierung der Überwachung der Teilsignatur gebunden werden. Dies ist jedoch nicht auf jeder Zielplattform realisierbar. Da die Spezifikation in der MBSecMonSL unabhängig von der Zielplattform sein soll, wird auf dieses dynamische Binden verzichtet.

Modellierungsregel 4: In einem inkludierten Anwendungsfall dürfen nur Lebenslinien modelliert werden, die auch im Basis-Anwendungsfall modelliert sind.

Einschränkung für verschränkte Überwachung: Diese Einschränkung betrifft den Modus der MPNs, in dem eine Use-Case-Signatur ohne Prechart oder eine ausgelagerte Teilsignatur, wie Abschnitt 2.3.2 beschrieben, verschränkt überwacht wird. In diesem verschränkten Modus werden in den MPNs, sobald eine Marke aus einem der Initialplätze der Signatur entfernt wurde, eine neue Marke mit einer inkrementierten Subgeneration erzeugt. Beginnt die eLSC-Signatur zusätzlich mit einem Element, das die Lebenslinien synchronisiert (eLSC-Fragment oder Bedingung), und nicht mit einem eindeutigen Ereignis (z. B. Nachricht), führt dies zu fehlerhaften Laufzeitmonitoren. Während die Modellierung einer solchen verschränkt überwachten Use-Case-Signatur ohne Prechart einen Modellierungsfehler darstellt, lässt sich die Auslagerung einer unabhängig von der Basis-Signatur verschränkt überwachten Teilsignatur semantisch nicht auf die MPN-Sprache abbilden.

Beispiel *Ausgelagerte Teilsignatur*

Der Eintritt in ein Fragment der eLSC-Sprache bildet immer einen Synchronisationspunkt, der in eine synchronisierende *Epsilon*-Transition übersetzt wird. Das eLSC `NonLocal` aus Abbildung 8.19 zeigt diese Situation. Bevor die Bedingung des *If-Then-Else*-Fragments ausgewertet wird, muss in der MPN-Repräsentation eine *Epsilon*-Transition geschaltet werden. Die MPN-Semantik löst im Referenzsystem mit der Aktivierung der Überwachung der Teilsignatur durch die Belegung eines Platzes in der Menge S_{act} *Epsilon*-Ereignisse aus. Hierdurch schaltet die *Epsilon*-Transition sofort und die Bedingungen der folgenden Transitionen werden ausgewertet. Probleme treten bei dieser Übersetzung auf, wenn abweichend vom CARDME-Protokoll die ausgelagerte Teilsignatur verschränkt überwacht wird. In diesem Fall werden nach dem ersten Schalten der *Epsilon*-Transition und der Auswertung der Bedingung durch die Marken der Subgeneration 1 neue Marken der Subgeneration 2 in den Initialplätzen erzeugt. Für diese wird jedoch kein *Epsilon*-Ereignis mehr ausgelöst, da dieses in der MPN-Semantik an die Aktivierung der Teilsignaturüberwachung gekoppelt ist und die Subgeneration 2 erst am Ende des Makroschrittes erstellt wird. Wird dann das Ereignis `send.Presentation2Request(P2R1)` an die Teilsignatur übergeben, wird die Marke der Subgeneration 1 in den Vorplätzen der Sendetransition gelöscht und es werden zwei Marken in den Nachplätzen erzeugt. Da die Marken der Subgeneration 2 noch in den Vorplätzen der *Epsilon*-Transition liegen, führt das Sendeereignis für diese Subgeneration zu einer Fehlererkennung. Dieses Problem lässt sich auch im allgemeinen nicht durch eine Anpassung der Transformation lösen, indem wie bei der Übersetzung des Precharts die initiale Synchronisierungstransition weggelassen wird, da hierdurch bei jedem beliebigen Ereignis, das der Signatur übergeben wird, eine neue Subgeneration in den Initialplätzen angelegt werden würde.

Auch wenn in gewissen Sonderfällen dieses Problem durch Weglassen der initialen Synchronisierungstransition gelöst werden kann, ist dies in den meisten Fällen nicht möglich oder bedeutet eine deutlich komplexere Transformation. Somit ergibt sich *Modellierungsregel 5*, die diese Situation ausschließt.

Modellierungsregel 5: Eine eLSC-Signatur eines Use-Cases ohne Prechart oder eine ausgelagerte Teilsignatur, die verschränkt überwacht werden soll, darf nicht mit einem eLSC-Fragment (*If-Then-Else*, *Par*, *Alt*, *Loop*) beginnen.

In der Semantik der MPNs führt auch die Modellierung von Bedingungen als erstes Element einer verschränkt überwachten Signatur zu ähnlichen Problemen. Arbeitet die Bedingung auf globalen Variablen und die Bedingung ist sofort bei Aktivierung der Signaturüberwachung erfüllt, werden aus den Startplätzen der beteiligten Instanzen die Marken entfernt und in ihnen jeweils eine neue Marke mit inkrementierter Subgeneration erzeugt. Mit dem nächsten übergebenen Ereignis kommt es zu demselben fehlerhaften Fehlschlagen der Signatur wie im Fall der Fragmente. Dieses fehlerhafte Verhalten der übersetzten MPNs verhindert *Modellierungsregel 6*.

Modellierungsregel 6: Eine verschränkt überwachte eLSC-Signatur eines Use-Cases ohne Prechart oder eine ausgelagerte Teilsignatur darf nicht mit einer Bedingung beginnen.

9

KONKRETE REALISIERUNG DER TRANSFORMATION

Basierend auf der in Abschnitt 8.1 vorgestellten Übersetzung in konkreter Syntax, wird die komplette Transformation aller eLSC-Elemente in die MPN-Syntax in einer formalisierten Transformationssprache spezifiziert. Zur Umsetzung dieser komplexen Transformation soll eine regelbasierte Modell-zu-Modell-Transformationssprache eingesetzt werden, da ein regelbasierter Ansatz weniger fehleranfällig scheint als eine manuelle Implementierung des Mustererkennungsprozesses in einer universellen Programmiersprache (GPL) (*engl. General Purpose Language*). Des Weiteren dient eine in einer formalisierten Transformationssprache verfasste Übersetzung der eLSCs in die formalisierten MPNs als Definition der Semantik der eLSCs.

Im Folgenden wird in Abschnitt 9.1 basierend auf der Fallstudie [PP12], eine passende Transformationssprache ausgewählt, die zur Spezifikation der Transformation geeignet ist. Hierzu muss aufgrund der Vielfältigkeit der existierenden Transformationssprachen in ihrem Anwendungsgebiet, Funktionsumfang (*engl. feature set*) und ihrer formalen Grundlage die wichtigen Anforderungen an die Sprache aufgestellt und mit den unterstützten Konzepten der Sprachen verglichen werden. In Abschnitt 9.2 werden anschließend die Implementierungen der Transformation in den beiden Transformationssprachen ATL und SDM vorgestellt und anschließend in Kapitel 11 die Einsetzbarkeit der Implementierungen im MBSecMon-Prozess evaluiert.

9.1 Auswahl der Transformationssprache für die Transformation zwischen eLSCs und MPNs

Trotz vielfältiger Publikationen zu Vergleichen von Transformationssprachen im Laufe der letzten Jahre reichen diese Informationen nicht aus die passende Sprache zur Spezifikation der Übersetzung auszuwählen. Dies resultiert aus den gewählten Schwerpunkten der Publikationen. Einerseits wird seit 2007 der jährliche Transformation Tool Contest (TTC)[1] ausgerichtet, zu dem Lösungen zu speziellen Transformationsproblemen eingereicht und verglichen werden. Diese Publikationen liefern jedoch nur Informationen zu den Stärken und Schwächen der einzelnen Transformationswerkzeuge, fokussieren sich jedoch nicht, wie [ABKP11] festgestellt hat, auf die Vergleichbarkeit der Ergebnisse. Andererseits behandeln andere Publikationen nur kleine klassische Beispiele wie UML2RDBMS [CH06], fokussieren sich nur auf graphbasierte Transformationssprachen [TEG+05] oder

[1] TTC 2014: http://www.transformation-tool-contest.eu (besucht am 09.11.2014)

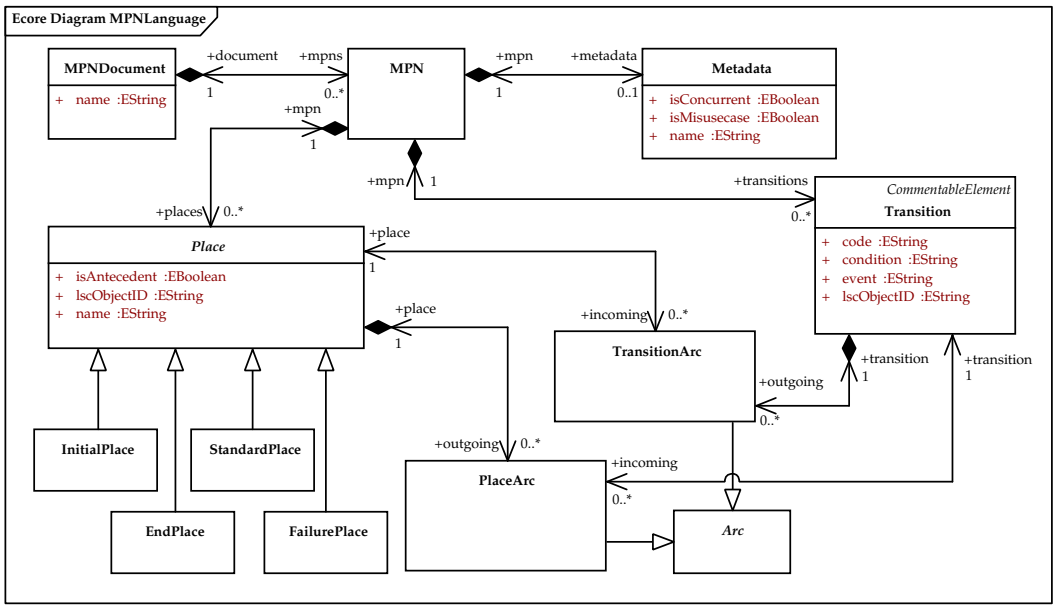


Abbildung 9.1: Metamodell der MPN-Sprache

beschränken sich auf eine kleine Menge an Eigenschaften wie die Vererbung [WKK+11]. Die Transformation von Spezifikationen in der MBSecMonSL in die entsprechende MPN-Repräsentation basiert im Gegensatz zu den vorher erwähnten Vergleichen auf sehr unterschiedlichen Modellen und komplexen Abbildungen, woraus vielfältige Anforderungen an die Transformationssprache entstehen.

9.1.1 *Anforderungen an die Transformationssprache*

Zur Ableitung der Anforderungen an die Transformationssprache wird in diesem Abschnitt zur kompakteren Darstellung nur eine Teilmenge der eLSC-Sprache betrachtet und für die MPN-Sprache in Abbildung 9.1 die abstrakte Syntax ohne das Referenzsystem eingeführt. Trotz der Einschränkung auf eine Teilmenge der eLSC-Konstrukte erlaubt die beispielhafte Darstellung anhand der wichtigsten Konzepte der Transformationen die Formulierung der aus ihnen resultierenden Anforderungen.

Für die im Folgenden betrachteten Transformationssprachen dienen Metamodelle, die die Struktur der zu transformierenden Modelle beschreiben, als Grundlage für das Quell- und Zielmodell und für die Spezifikation der Transformation. Das Metamodell der MPN-Sprache besitzt die Wurzelklasse `MPNDocument`, die als Behälter für MPN-Diagramme (`MPN`) dient. Zusätzliche Informationen über das MPN-Diagramm werden in der Klasse `Metadata` gespeichert. Diese beinhaltet drei Attribute: `name` ermöglicht die Identifikation des Diagramms, `isMisusecase` legt fest, ob das MPN einen Use- oder Misuse-Case beschreibt, und `isConcurrent` definiert, wie es interpretiert werden soll. Ein MPN besteht aus einer Menge von Plätzen, die durch die abstrakte Klasse `Place` und Transitionen, die durch die Klasse `Transition` repräsentiert werden. Plätze existieren in den vier Ausprägungen

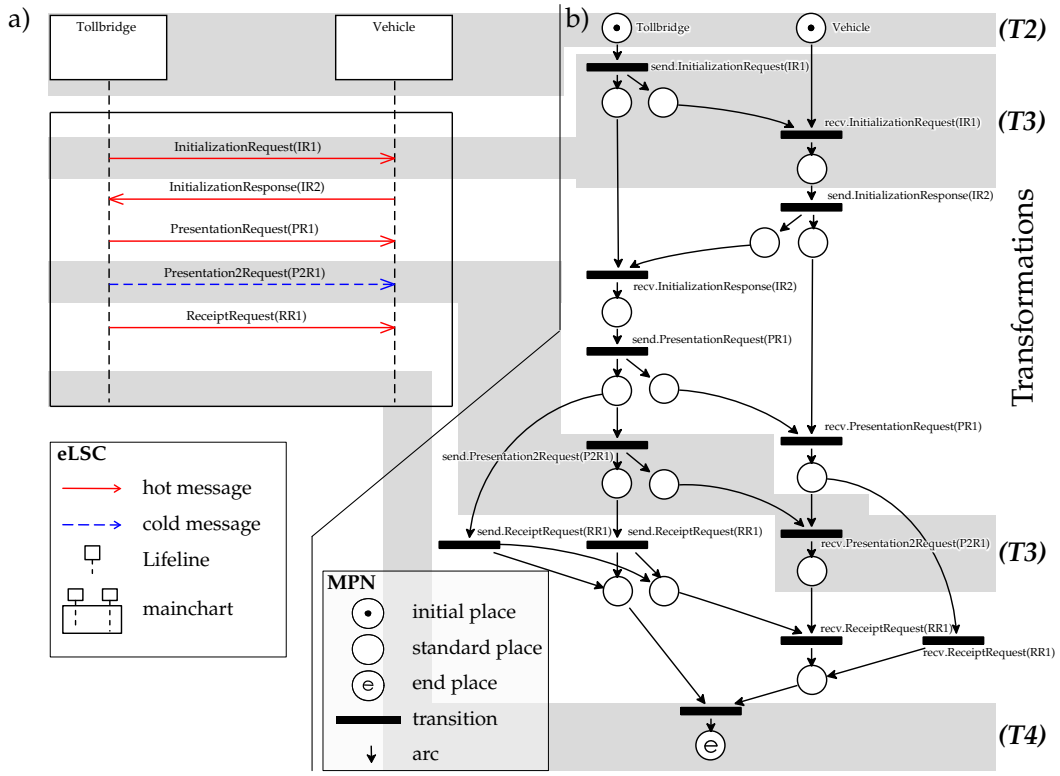


Abbildung 9.2: Konkrete Transformation asynchroner Nachrichten

`InitialPlace`, `StandardPlace`, `EndPlace` und `FailurePlace`, die in Abschnitt 2.3.2 vorgestellt wurden. Diese besitzen jeweils das Attribut `name` zur Identifikation, und die in Abschnitt 8.1 vorgestellte Zuordnung zu den eLSC-Lebenslinien realisiert durch das Attribut `lscObjectID`. Transitionen erhalten ebenfalls diese Zuordnung über das Attribut `lscObjectID` und besitzen zusätzlich drei weitere Attribute. Das Attribut `code` speichert die annotierte auszuführende Aktion, `condition` die annotierte Bedingung und `event` das annotierte Ereignis. Durch die Ausprägungen der abstrakten Klasse `Arc` werden Plätze mit Transitionen (`PlaceArc`) und Transitionen mit Plätzen (`TransitionArc`) verbunden.

Die Transformation eines Universal eLSCs ohne Prechart in seine MPN-Repräsentation ist in konkreter Syntax, basierend auf den Transformationsregeln in Abschnitt 8.1, in Abbildung 9.2 dargestellt. Abbildung 9.2 a) zeigt ein vereinfachtes Kommunikationsprotokoll (Use-Case) zwischen Fahrzeug und Mautbrücke als eLSC, während Abbildung 9.2 b) die zugehörige MPN-Repräsentation darstellt. Das Kommunikationsprotokoll zeigt einen erlaubten Austausch asynchroner Nachrichten zwischen zwei eLSC-Lebenslinien – der Mautbrücke (`Tollbridge`) und dem Fahrzeug (`Vehicle`). Hierbei wurden aus Gründen der Übersichtlichkeit Ignore-Fragmente weggelassen, die die Antworten des Fahrzeugs enthalten müssen.

Sobald das Fahrzeug die erste in Abbildung 9.2 a) modellierte asynchrone heiße Nachricht `InitalizationRequest(IR1)` empfängt, die durch die Mautbrücke ausgesendet wurde, antwortet es mit der Nachricht `InitialisationResponse(IR2)`. Daraufhin sendet die Mautbrücke eine Anfrage nach den Identifikationsdaten

des Fahrzeugs (`PresentationRequest(PR1)`). Optional kann die Mautbrücke nun weitere Daten vom Fahrzeug anfordern – modelliert durch die kalte asynchrone Nachricht `Presentation2Request(P2R1)`. Folgend auf eine Nachricht vom Typ `PresentationRequest` muss die Mautbrücke die Quittung durch die heiße Nachricht `ReceiptRequest(RR1)` übermitteln.

Die Transformation des eLSCs in die MPN-Repräsentation besteht für dieses Beispiel aus vier Transformationsschritten, die bis auf den übergeordneten ersten Schritt T1 in Abbildung 9.2 grau hinterlegt und nummeriert sind.

(T1) Für jedes eLSC, das in einem eLSC-Dokument gespeichert ist, wird zunächst ein MPN in einem MPN-Dokument erstellt.

(T2) Jede Lebenslinie wird in einen Initialplatz übersetzt, der mit dem Namen der Lebenslinie annotiert ist.

(T3) Eine heiße asynchrone Nachricht wie die Nachricht `InitalisationRequest (IR1)` wird in eine Transition für das Sendeereignis, eine für das Empfangsereignis und drei Standardplätze übersetzt. Diese werden über gerichtete Kanten (*engl. arcs*) verbunden, wobei ein Platz zur Sendeseite und ein Platz zur Empfangsseite gehört. Der dritte Platz sichert die Reihenfolge zwischen Sende- und Empfangsereignis. Dieses Muster wird auch für kalte Nachrichten verwendet und der optionale Charakter durch Übersprungstransitionen realisiert.

(T4) Das MPN wird am Ende der Übersetzung auf einen Endplatz synchronisiert. Hierbei wird eine Transition für alle möglichen Pfade durch das MPN generiert und zum Endplatz verbunden. Im Beispiel wird eine Transition erstellt, da die letzte modellierte Nachricht heiß ist und so je Lebenslinie im eLSC nur ein offener Platz existiert.

(T5) Am Ende des Transformationsprozesses wird das Zielmodell optimiert. In diesem Schritt werden redundante Plätze entfernt und Epsilon-Transitionen, die mit Aktionen annotiert sind, mit vorherigen Transitionen verschmolzen. Diese Optimierung taucht im Beispiel nicht explizit auf, da sie der Codegenerierung zugeordnet ist, die in [Pat14] betrachtet wird.

Aus diesen vier vorgestellten Transformationsschritten lassen sich neun Anforderungen an die Transformationssprache formulieren.

A1) 1-ZU-1 (MODELL) Jedes einzelne eLSC-Diagramm soll ein einzelnes MPN-Diagramm übersetzt werden. Für jedes Quellmodell muss somit ein einzelnes Zielmodell erzeugt werden. (BASIEREND AUF T1)

A2) M-ZU-N (ELEMENTE) Ein oder mehrere Elemente der Quellsprache müssen auf ein oder mehrere Elemente der Zielsprache abgebildet werden. Die Anzahl auf Quell- und Zielseite kann voneinander abweichen. (BASIEREND AUF T3)

A3) KORRESPONDENZLINKS Zur Verarbeitung der kalten Elemente muss die Rückverfolgbarkeit von Elementen der Quellseite zu bereits übersetzten Elementen der Zielseite ermöglicht werden, um Übersprungstransitionen an die entsprechenden vorher übersetzten Plätze im MPN anhängen zu können. (BASIEREND AUF T3)

A4) ATTRIBUTE Die Sprache muss Attribute der Klassen der Modellelemente behandeln können. Attribute müssen auf Quell- und Zielseite überprüft und darauf basierend auf der Zielseite generiert werden. (BASIEREND AUF T2)

A5) IN-PLACE Zur Optimierung des Zielmodells wird eine Art der *In-Place*-Transformation benötigt. (BASIEREND AUF T5)

A6) LÖSCHEN Während der Nachbearbeitung müssen Elemente aus dem Zielmodell der Übersetzung gelöscht werden, um redundante Plätze und nicht benötigte Transitionen zu entfernen. (BASIEREND AUF T5)

A7) REKURSIVE REGELN Für die Synchronisation am Ende des eLSCs müssen alle Kombinationen der offenen Plätze über eine eigene Transition zum Endplatz verbunden werden. Da auch die Anzahl der Lebenslinien variabel und bei Spezifikation der Transformation nicht bekannt ist, muss rekursiv bzw. iterativ auf den Modellelementen gearbeitet werden. (BASIEREND AUF T4)

A8) UNIDIREKTIONAL Einige Elemente besitzen keine eindeutige bijektive Abbildung. Eine Schleife mit einer festen Anzahl an n Durchläufen kann in n Repräsentationen seines Inhalts abgewickelt werden. Die Abbildung in die Rückrichtung wäre nicht eindeutig und wird in diesem Anwendungsfall nicht benötigt. (BASIEREND AUF T1)

A9) WERKZEUGUNTERSTÜTZUNG Zur Realisierung des Entwicklungsprozesses wird eine zuverlässige und funktionstüchtige Implementierung der Transformationssprache in einem Werkzeug benötigt.

9.1.2 *Vergleich der Transformationssprachen*

Zur Erfüllung der aufgestellten Anforderungen aus Abschnitt 9.1.1 findet ein Vergleich zwischen sieben regelbasierten Modelltransformationssprachen, die dem aktuellen Stand der Technik entsprechen, statt. Tabelle 9.1 zeigt, welche Anforderungen durch die Sprachen und ihre Implementierung erfüllt werden. Betrachtet werden die Atlas Transformation Language (ATL), die Epsilon Transformation Language (ETL), Operational QVT, PROGRES, Story Driven Modeling (SDM), Tripel-Graph-Grammatiken (TGG) und VIATRA2.

Erste Unterschiede ergeben sich durch die formale Grundlage der Transformationssprachen. Während PROGRES, SDM, TGG, und VIATRA2 auf den Grundlagen der Graphtransformationen (GT) basieren, sind ATL, ETL und QVT (QVTo) nicht formalisiert. Für ATL wurden nachträglich zur Sprachdefinition und Implementierung Ansätze zur Formalisierung durch Zustandsautomaten [RJK+06] und Termersetzungslogiken [TV10] veröffentlicht.

Bis auf TGGs sind die vorgestellten GT-Sprachen hybride und unterstützen die Modellierung des Kontrollflusses. Im Gegensatz zu den anderen Ansätzen sind TGGs vollständig deklarativ und erlauben die simultane Spezifikation einer bidirektionalen Transformation als Mapping zwischen Quell- und Zielmodellelementen. Diese Eigenschaft erschwert jedoch die Spezifikation der Transformationsregeln, da eine bijektive Abbildung nicht für alle Elemente existiert (A8) und die

	Anforderungen								
Transformationssprachen	A1	A2	A3	A4	A5	A6	A7	A8	A9
ATL 3.1 [JABK08]	✓	✓	✓	✓	o[4]	o[5]	o[7]	✓	✓
ETL [KPP08]	✓	–[2]	o[3]	✓	✓	✓[6]	✓	✓	✓
QVTo [OMG11a]	✓	✓	✓	✓	o[4]	✓	✓	✓	✓
PROGRES [Sch97]	✓[1]	✓	o[3]	✓	✓	✓	✓[8]	✓	–
SDM [FNTZ00]	✓[1]	✓	o[3]	✓	✓	✓	✓[8]	✓	✓
TGG [KRS09][AVS12]	✓	✓	✓	✓	–	–	–	o[9]	✓
VIATRA2 [VB07]	✓	✓	o[3]	✓	✓	✓	✓	✓	✓

✓unterstützt; o teilweise unterstützt; - nicht unterstützt; [1]in-place; [2]nur 1-zu-n; [3]manuell; [4]Verfeinerungsmodus; [5]neue Transformation; [6]mit EOL; [7]als Helper; [8]Kontrollfluss- und Pfadausdrücke; [9]bidirektional

Tabelle 9.1: Unterstützung der Anforderungen durch die Transformationssprache

Modellierung rekursiver Regeln (A7) nicht unterstützt wird. Hierdurch und die Tatsache, dass eine bidirektionale Transformation nicht benötigt wird, scheiden TGGs für die Modellierung der Transformation von eLSC zu MPNs aus.

Da viele Vergleiche in den beiden Gruppen der Transformationssprachen (z. B. [JK06a, TEG+05]) existieren und so die Auswahl in einer Gruppe selbst getroffen werden kann, wird je eine Sprache aus jeder Gruppe ausgewählt. ATL als häufig genutzte Modelltransformationssprache in der Eclipse-Community und SDM, das im Meta-CASE-Werkzeug eMoflon [ALPS11] eingesetzt wird. Anhand der eLSC-zu-MPN-Transformation werden die Unterschiede der Sprachen herausgearbeitet.

In SDM-Transformationen werden Regeln in einen explizit modellierten Kontrollfluss eingebettet, der als Aktivitäten eines Aktivitätsdiagramms modelliert ist. Hingegen werden Regeln einer ATL 3.1-Transformation nicht durch vorgelagerte Muster in einem Kontrollfluss bedingt, sondern durch OCL-Ausdrücke, die zur Ableitung der Reihenfolge der Anwendung durch die impliziten Beziehungen genutzt werden. Zusätzlich erlaubt es ATL Funktionalität in Hilfsfunktionen (*engl. helper functions*) auszulagern.

Eine Auswahl der Transformationssprache kann jedoch nicht alleine auf Basis der unterschiedlichen Syntax und der expliziten bzw. impliziten Modellierung des Kontrollflusses getroffen werden. Die Realisierung der eLSC-zu-MPN-Transformation in beiden Sprachen und deren Vergleich wird im folgenden Abschnitt genutzt, um eine Entscheidung für die passende Sprache zu treffen.

9.2 Konkrete Realisierung der Transformation mit SDM und ATL

Anhand der fünf identifizierten Regeln (T1 bis T5) in Abschnitt 9.1.1 wird auf die Anforderungen der Transformation, die nicht gleichermaßen von beiden Sprachen erfüllt werden (A3, A5, A6 und A7), genauer eingegangen.

Das SDM-Diagramm in Abbildung 9.3 visualisiert die grundlegenden Schritte der Transformation: (T1) *LSCDocument* zu *MPNDocument* und *LSC* zu *MPN*; (T2) *LSCObjects* zu *InitialPlaces*, (T3) *Messages* in ein *MPN*-Teilnetz und (T4) die Synchronisierung zu *EndPlaces* am Ende des *MPNs*.

ATL

```
rule LSCDocument2MPNDocument {
  from
    lscDocument: LSCMM!LSCDocument
  to
    mpnDocument: MPNMM!MPNDocument(
      name <- lscDocument.name
    )
}
```

```
rule LSC2MPN{
  from
    lsc: LSCMM!LSC
  to
    mpn: MPNMM!MPN(),
    metadata: MPNMM!METADATA(
      mpn <- mpn,
      isMisuseCase <- lsc.isMisuseCase,
      name <- lsc.name,
      document <- lsc.document
    )
}
```

SDM

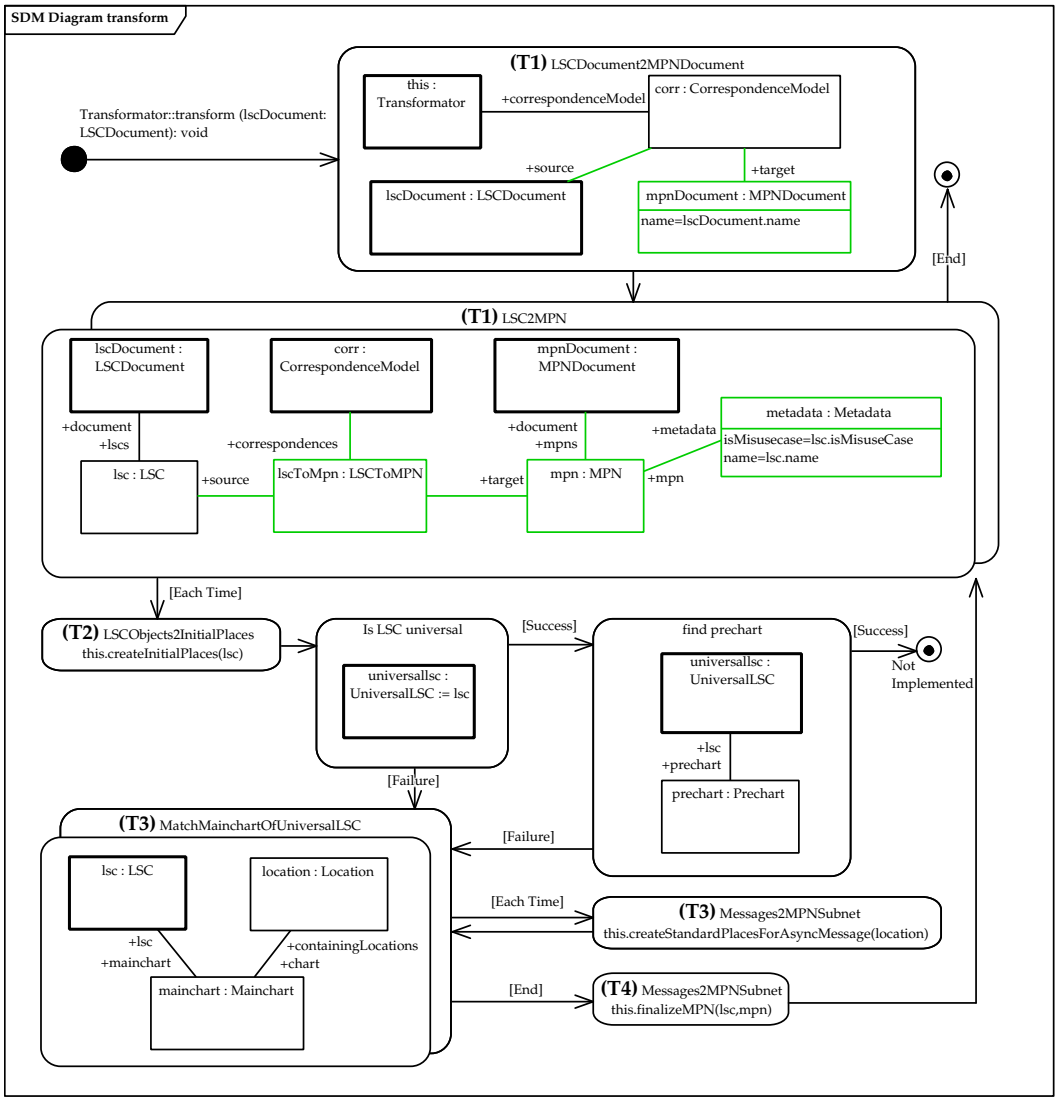


Abbildung 9.3: Initiale Regeln der Transformation (T1)

(T1) Die ersten Transformationsregeln in Abbildung 9.3 übersetzen zum einen das *LSCDocument* in ein *MPNDocument*. Hierbei werden jedes enthaltene *LSC*s in ein *MPN* übersetzt. Die Namen dieser neuen Elemente der MPN-Repräsentation werden auf denselben Namen des korrespondierenden eLSC-Elements gesetzt. Die ATL-Regel *LSCDocument2MPNDocument* entspricht der ersten gleichnamigen Aktivität der SDM-Regel. In der SDM-Regel ist der übergebende Parameter `lscDocument` bereits als gebundenes Objekt (*engl. bound object*) vorhanden und für dieses `LSCDocument` ein `MPNDocument` im Zielmodell erzeugt. Ausgehend vom `this`-Objekt des Typs `Transformator` wird durch das Muster eine im Modell vorhandene Instanz des *CorrespondenceModel* gesucht und die beiden Dokumente als Quelle (`source`) und Ziel (`target`) der Transformation verbunden. Im Gegensatz hierzu benötigt die ATL-Transformation keine bereits gebundenen Objekte zur Navigation. Während der Laufzeit der Transformation wird das `lscDocument` durch den *from*-Teil der Regel gefunden und das im *to*-Teil das beschriebene `mpnDocument` mit entsprechendem Namensattribut erstellt

Die zweite Regel *LSC2MPN* ist sehr ähnlich in beiden Sprachen. In der SDM-Regel ist eine *foreach*-Aktivität modelliert, in der das gebundene `lscDocument` zur Navigation zu allen von ihm besessenen `LSC`s genutzt wird. Für jedes gefundene `LSC` wird ein MPN erzeugt und an das durch die vorherige Aktivität angelegte `mpnDocument` gehängt. Die Attribute des LSCs werden in ein zum MPN gehöriges Objekt des Typs `Metadata` übertragen. Zur späteren Nachverfolgung wird ein Korrespondenzlink (*engl. traceability link*) zwischen dem LSC und dem zugehörigen MPN durch das Erstellen des `lscToMPN`-Objekts vom Typ `LSCToMPN` erzeugt und in das Korrespondenzmodell *CorrespondenceModel* eingehängt.

Diese beiden ersten Regeln decken schon die ersten Hauptunterschiede der beiden Sprachen auf. Während sich SDMs explizit auf der Modellierung des Kontrollflusses zwischen den deklarativen Mustern verlässt, sollte ATL so lange wie möglich rein deklarativ eingesetzt werden [JABK08]. Neben der Implementierung der SDMs in eMoflon, die für diesen Vergleich genutzt wird, existiert ebenfalls eine zu den Standard-SDMs rückwärts kompatible Variante der SDMs, die implizites Scheduling der deklarativen Regeln ermöglicht [MVG08]. Des weiteren erzeugen SDM-Regeln im Gegensatz zu ATL während der Laufzeit der Transformation nicht automatisch Korrespondenzlinks zwischen den Objekten des Quell- und den korrespondierenden Objekten des Zielmodells. Dieses Konzept muss in den SDMs als zusätzliches Metamodell (Abb. 9.4) zur Verfügung gestellt werden und explizit in jeder Regel modelliert werden.

(T2) Für jedes gebundene LSC in der Aktivität bzw. Regel *LSC2MPN*, werden die folgenden Transformationsschritte durchgeführt. Die nächste Aktivität der SDM-Transformation, die über einen *foreach*-Kante verbunden ist, ruft die Regel in Abbildung 9.5 auf, die für jede Lebenslinie des eLSCs (`LSCObject`) im Zielmodell einen Initialplatz (`InitialPlace`) anlegt. Zur Bestimmung des zum LSC gehörigen MPNs wird der manuell angelegte Korrespondenzlink `lscToMpn` zur Navigation genutzt. Zusätzlich werden für spätere Transformationsschritte zwischen dem `object` und dem neu erstellten Initialplatz jeweils zwei weitere Korrespondenzlinks angelegt. `ObjectToPlace` speichert dauerhaft die Korrespondenz zwi-

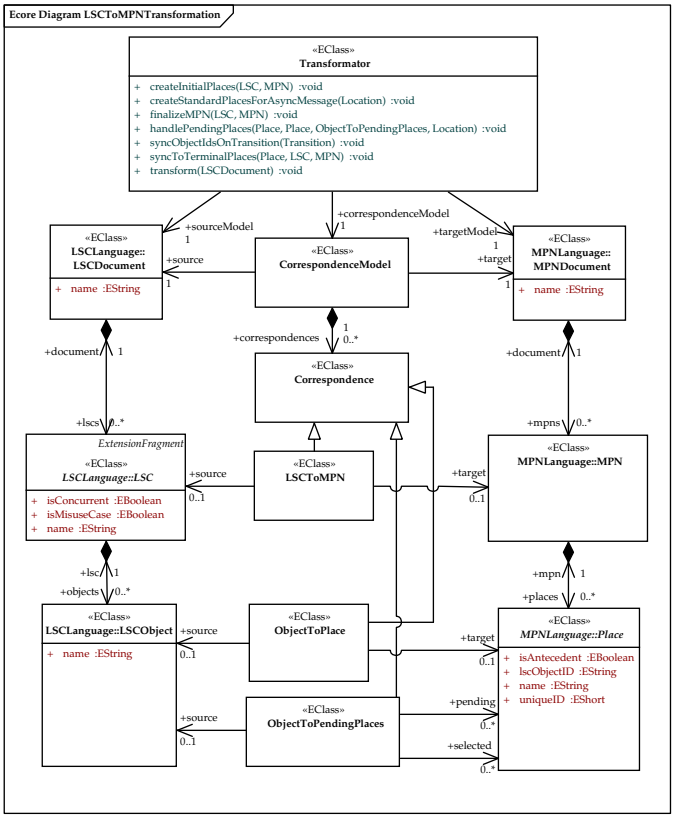


Abbildung 9.4: Korrespondenzmetamodell der SDM-Transformation

ATL

```
rule LSCObject2InitialPlace{

  from
    lscObject: LSCMM!LSCObject

  to
    initialPlace: MPNMM!InitalPlace(
      name <- lscObject.name,
      mpn <- thisModule.resolveTemp(
               lscObject.lsc, 'mpn')
    )
}
```

SDM

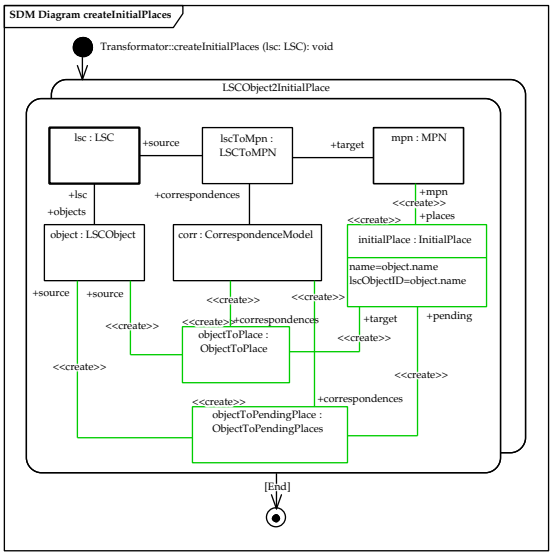


Abbildung 9.5: Erstellung der Initialplätze im MPN (T2)

schen Quell und Zielmodell ab, während `ObjectToPendingPlace` die *offenen Plätze* markiert, an die im Weiteren übersetzte Elemente angehängt werden.

Die ATL-Regel benötigt kein gebundenes Objekt zur Navigation und sucht übergeordnet in allen LSCs nach jeder Instanz von `LSCObject`, die im *from*-Teil der Regel spezifiziert ist. Der *to*-Teil der Regel erstellt wie die SDM-Regel für jedes gefundene `LSCObbject` einen Initialplatz, der an das entsprechende MPN gehängt wird. Die Bestimmung des entsprechenden MPNs erfolgt über die Korrespondenzlinks die ATL-Regeln automatisch erzeugen. Über die vordefinierte Hilfsmethode `resolveTemp`, werden durch Namensabgleich die Korrespondenzlinks ausgewertet. Als Parameter dienen die Elemente des Quellmodells und der Name des gesuchten Objekts im Zielmodell. Der Name ist durch die Regel, die das Element im Vorfeld erzeugt hat, festgelegt.

Diese Regel zeigt die Wichtigkeit der Korrespondenzlinks für eine deklarative Spezifikation der Transformationsregeln auf. In ATL erzeugt jede Anwendung einer Regel automatisch Korrespondenzlinks zwischen den gematchten Quellelementen (*from*-Teil) und den durch die Regel erzeugten Elementen des Zielmodells (*to*-Teil). Da ein solcher Mechanismus in SDMs fehlt, existieren zwei Varianten der Modellierung. Entweder werden alle benötigten gebundenen Objekte als Parameter über den Kontrollfluss weitergereicht, oder ein zusätzliches Metamodell wie durch [HWG11] vorgeschlagen und in der SDM-Transformation verwendet, spezifiziert. Dieses enthält die Transformationsregeln als Operationen der Klasse `Transformator` und stellt zusätzlich die bereits in den folgenden Aktivitäten verwendeten Korrespondenzlinks zur Verfügung.

(T3) Bisher waren die beiden Transformationen vom Konzept her sehr ähnlich. Die Definition komplexerer Regeln wie die Übersetzung asynchroner Nachrichten mit ihren Modalitäten (Temperaturen) fordert jedoch zwei unterschiedliche Ansätze in den beiden Sprachen. SDMs erlauben es durch die explizite Modellierung eines Kontrollflusses die Transformation in einer Regel zu formulieren, wohingegen in ATL drei einzelne Regeln spezifiziert wurden.

In Abbildung 9.6 finden die manuell in der SDM-Transformation hinzugefügten Korrespondenzlinks vom Typ `ObjectToPendingPlaces` Anwendung. Zunächst überprüft die erste Aktivität, ob die als Parameter übergebene *Location* die Location der Sendeseite einer asynchronen Nachricht ist, ermittelt das zugehörige LSC und über das Korrspondenzmodell (`CorrespondenceModel`) das zugehörige MPN. Zusätzlich werden Sende- und Empfangslebenslinie (`LSCObject`) gebunden. Basierend auf diesen gebundenen Objekten werden in der folgenden Aktivität `create standard places for message`, die drei benötigten Standardplätze der MPN-Repräsentation erstellt und über Objekte von Typ `ObjectToPlace` in das Korrespondenzmodell eingehängt. In den folgenden ausgelagerten zwei Mustern der *foreach*-Aktivitäten *handle send* bzw. *handle receive* werden die offenen Plätze (`pending paces`) der entsprechenden Lebenslinie auf der Sender- bzw. Empfangsseite genutzt, um sowohl heiße als auch kalte asynchrone Nachrichten zu übersetzen. Die durch die zwei *Statement*-Aktivitäten aufgerufene ausgelagerte Methode `this.handlePendingPlaces(...)` entscheidet, ob die gerade übersetzte Location der Sende- oder Empfangsseite heiß war, und entfernt in diesem Fall den Korrespondenzlink `ObjectToPendingPlaces`, der von der Lebenslinie zum gerade

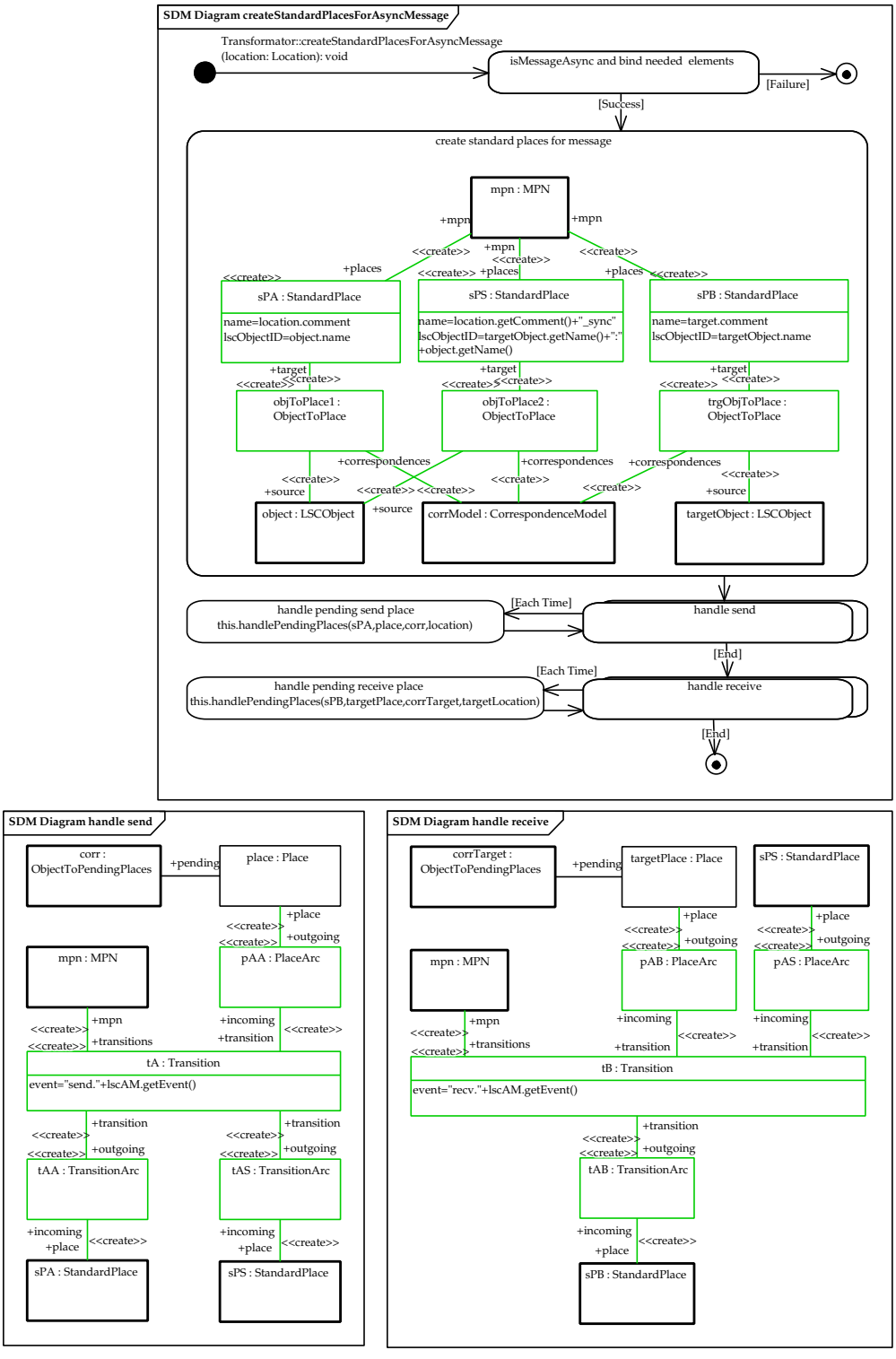


Abbildung 9.6: SDM-Regeln für asynchrone Nachrichten (T3)

```
rule AsyncMPNPattern{
  from
   lscAM : LSCMM!Message(lscAM.isMessageAsync)
  to
    sPA : MPNMM!StandardPlace(
      mpn <- thisModule.resolveTemp(lscAM.endLocation.object.lsc, 'mpn') ),
    sPS : MPNMM!StandardPlace( placeArcs <- tAS,
      mpn <- thisModule.resolveTemp(lscAM.endLocation.object.lsc, 'mpn') ),
    sPB : MPNMM!StandardPlace( placeArcs <- tAB,
      mpn <- thisModule.resolveTemp(lscAM.endLocation.object.lsc, 'mpn') ),
    tA : MPNMM!Transition( transitionArcs <- tAA, transitionArcs <- tAS,
      event <- 'send.'.concat(lscAM.content),
      mpn <- thisModule.resolveTemp(lscAM.endLocation.object.lsc, 'mpn') ),
    tAA : MPNMM!TransitionArc( place <- sPA ),
    tAS : MPNMM!TransitionArc( place <- sPS ),
    pAA : MPNMM!PlaceArc( transition <- tA, place <- thisModule.getPlaceOfPrevMsgSender(lscAM) ),
    ...
}
rule bypassTransition{
  from
    firstLoc : LSCMM!Location,
    secondLoc : LSCMM!Location(firstLoc.isBypassCombination(secondLoc))
  to
    pA : MPNMM!PlaceArc( transition <- t,
      place <- secondLoc.getPlaceForLocAndInst ),
    t : MPNMM!Transition( event <- secondLoc.getEventName, transitionArcs <- tA,
      mpn <- thisModule.resolveTemp(firstLoc.object.lsc, 'mpn') ),
    tA : MPNMM!TransitionArc( place <- thisModule.getPlaceForLocAndInst(secondLoc) )
}

rule bypassTransitionWithFollowingAsyncSend extends bypassTransition{
  from
    firstLoc : LSCMM!Location,
    secondLoc : LSCMM!Location(firstLoc.isBypassCombination(secondLoc)
      and secondLoc.isAsyncSendingLocation)
  to
    tAS : MPNMM!TransitionArc( transition <- t,
      place <- thisModule.resolveTemp(secondLoc.outgoingMessage, 'sPS')
    ),
    pAS : MPNMM!PlaceArc ( place <- thisModule.resolveTemp(secondLoc.outgoingMessage, 'sPS'),
      transition <- thisModule.resolveTemp(
        Tuple{fL = firstLoc.getOppositeLocation, sL = secondLoc.getOppositeLocation}, 't') )
}
```

Abbildung 9.7: ATL-Regeln für asynchrone Nachrichten (T3)

betrachteten offenen Platz (`place` bzw. `targetPlace`) zeigt. In beiden Fällen wird der neu erstellte Platz (`sPA` bzw. `sPB`) in den Korrespondenzlink als offener Platz eingehängt. Auf diese Art werden sowohl die Transitionen des zuletzt übersetzten eLSC-Elements als auch die Übersprungstransitionen im Fall einer vorher übersetzten kalten Nachricht erzeugt.

Die drei Regeln der ATL-Transformation in Abbildung 9.7 matchen im *from*-Teil der Regel verschiedene Elemente des Quellmodells. *AsyncMPNPattern* bindet alle asynchronen Nachrichten des eLSC-Modells und übersetzt diese in die MPN-Repräsentation. Die beiden Regeln `bypassTransition` und die erweiterte Variante `bypassTransitionWithFollowingAsyncSend` erzeugen die zusätzlichen Übersprungstransitionen für kalte Nachrichten. Da die erste Regel bereits alle asynchronen Nachrichten des Quellmodells in Elemente des Zielmodells übersetzt hat, kann keine weitere Regel derselben Transformation diesen gleichen Match auf der Quellseite (*from*-Teil) verwenden. Um dennoch dieselben Nachrichten des Quellmodells ein zweites Mal in der Transformation zu verwenden, kann die

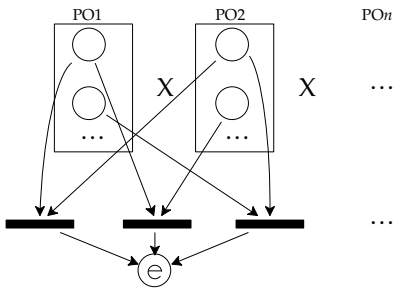


Abbildung 9.8: Synchronisation der offenen Plätze am Ende des MPNs (T4)

Transformation in zwei aufeinanderfolgende Transformationen aufgeteilt werden. Hierbei stehen jedoch alle automatisch erstellten Korrespondenzlinks der ersten Teiltransformation in der zweiten Transformation nicht mehr zur Verfügung, da das Zielmodell der ersten Transformation das Quellmodell der zweiten Transformation ist. Um dennoch eine zusätzliche Regel für die Übersprungstransitionen in derselben Transformation spezifizieren zu können, muss eine andere Kombination von Elementen des Quellmodells (Kombination von Locations) im *from*-Teil gebunden werden. Diese Kombinationen der Locations bilden eigene Korrespondenzlinks aus und können wie in der dritten Regel für die Zuweisung des Attributs `transition` von `pAS` genutzt werden. In der ATL-Transformation werden *helper*-Methoden eingesetzt, die die Matchings im *from*-Teil einschränken (`isBypassCombination` und `isAsyncSendingLocation`), die im Quellmodell navigieren (`getOppositeLocation`) und die bereits übersetzten Elemente im Zielmodell zur Wiederverwendung finden (`getPlaceOfPrevMsgSender` und `getPlaceForLocAndInst`). Diese *helper*-Methoden, die in OCL verfasst sind, verwenden extensiv die Korrespondenzlinks zwischen Quell- und Zielmodell zur Navigation.

(T4) Das in Abbildung 9.2 präsentierte eLSC endet mit einer heißen Nachricht, sodass alle offenen Plätze, die am Ende über eine Transition zum Endplatz synchronisiert werden müssen, zur letzten Nachricht gehören. Sind je Lebenslinie mehr als ein offener Platz vorhanden, z. B. durch eine kalte Nachricht wie in Abbildung 8.12, sind mehrere Transitionen zur Synchronisation aller möglichen Kombinationen verschiedener Plätze von jeweils verschiedenen Lebenslinien notwendig. Wie Abbildung 9.8 zeigt, muss für jede aus dem kartesischen Produkt in Formel 9.1 abgeleitete Menge an offenen Plätzen eine Transition zum Endplatz erzeugt werden. POn enthält hierbei alle offenen Plätze der n-ten Lebenslinie.

$$(\mathrm{po1}, \mathrm{po2}, \ldots, \mathrm{po}n) \in \mathrm{PO1} \times \mathrm{PO2} \times \ldots \times \mathrm{PO}n \;\text{ mit }\; n \in \mathbb{N} \tag{9.1}$$

Ein solches Muster mit zwei variablen Dimensionen, zum einen die Anzahl der Lebenslinien im eLSC und zum anderen die Anzahl der offenen Plätze, lässt sich nicht deklarativ in ATL und SDM beschreiben. Die hierfür benötigte rekursive Traversierung des Modells muss durch imperative Kontrollstrukturen realisiert werden. SDMs erlauben dies lokal in einer Regel durch den Einsatz des Kontrollflusses und den wiederholten rekursiven Aufruf dieser Regel. Im Gegensatz hierzu muss dieselbe Transformation in ATL komplett imperativ in globalen *helper*-Methoden implementiert werden.

9.2.1 *Ergebnisse für die Implementierung der Transformation*

Der Vergleich der beiden Transformationssprachen ATL und SDM zeigt, dass beide Sprachen ihre Vorteile und Nachteile haben. ATL wurde in der Version 3.1 betrachtet und wurde in den folgenden Versionen um einige Funktionen erweitert, die in der Fallstudie nicht berücksichtigt wurden, jedoch im Folgenden angesprochen werden.

Implizite Korrespondenzlinks (entspricht A3): Der Hauptvorteil der ATL liegt in der *automatischen Erstellung eines Korrespondenzmodells*, dass die Zuordnung bereits übersetzter Elemente zu ihren Quellelementen in anderen Regeln ermöglicht. In SDM-Transformationen muss dieses Korrespondenzmodell in den Transformationsregeln manuell modelliert werden, wodurch die Lesbarkeit der Muster leidet. Die hieraus resultierende Vergrößerung der Muster ist schon in den noch einfachen Regel T2 und T3 zu sehen, die mit deutlich weniger Objekten auskommen könnten. Dieses Problem könnte in SDMs durch die Umsetzung einer impliziten Erzeugung eines Korrespondenzenkonzepts behoben werden, wie es für den spezielleren Fall der Wiederverwendung von Teilgraphen (*engl. subgraph copying*) in [GSJ08] vorgeschlagen wurde.

In-Place-Transformationen (entspricht A5 und A6): Für die Nachverarbeitung der aus der Transformation entstanden MPNs ist SDM als *In-Place*-Transformationssprache im Gegensatz zu ATL 3.1 zu bevorzugen, da rekursives Löschen und Modifikationen mithilfe des Kontrollflusses modelliert werden können. ATL 3.1 bietet für die Modellierung einer solchen Nachverarbeitung auf ein und demselben Modell nur den Verfeinerungsmodus (*engl. refinement mode*) an. Dieser Verfeinerungsmodus basierend auf den Kopierregeln (*engl. copy rules*) versagt jedoch, wenn die Nachverarbeitungsschritte iterativ auf dem jeweils durch den vorherigen Schritt veränderten Modell durchgeführt werden müssen, bis keine neuen Matches mehr gefunden werden. Dies resultiert aus dem nur schreibbaren Zielmodell dieser Kopierregeln, dass extern immer wieder als Quellmodell an dieselbe Transformation übergeben werden müsste. Durch die Aufnahme der EMF Transformation Virtual Machine (EMFTVM) in die aktuelle ATL-Version 3.4 Anfang des Jahres, stehen in ATL nun u. a. eine stark erweiterte Unterstützung von *In-Place*-Transformationen zur Verfügung. Diese erlaubt nun auch das Löschen von Objekten, jedoch unterstützt ATL immer noch nicht die rekursive Anwendung von Regeln auf das veränderte Modell[2].

Muster beliebiger Breite (entspricht A7): Wie für T4 im vorherigen Abschnitt beschrieben, kann es vorkommen, dass zur Spezifikationszeit der Transformationsregel die Anzahl der Objekte auf der linken Seite der Regel noch nicht feststeht. Ein Templatekonzept zur deklarativen Beschreibung eines Musters, das zur Laufzeit an die Anzahl der Objekte im Modell angepasst wird, wird weder von ATL noch SDM unterstützt. In vielen Programmiersprachen werden solche Probleme durch Rekursion gelöst. Diese Rekursion kann mit SDM durch die explizite Modellierung des Kontrollflusses spezifiziert werden.

[2] ATL-Wiki: http://wiki.eclipse.org/ATL/EMFTVM#In-place_transformation (besucht am 30.01.2014)

Explizite Modellierung des Kontrollflusses: Die Beschreibung einer komplexen Transformation kann durch die explizite Modellierung des Kontrollflusses wie bei SDM erleichtert werden. In ATL kann während der Transformation jedes Objekt nur durch eine Regel gebunden werden, da die Anwendungsreihenfolge der Regeln implizit durch einen Algorithmus ermittelt wird. Hieraus resultiert der Entwurf von sehr komplexen und umfassenden Regeln durch den Entwickler und die Aufteilung in Teiltransformationen, die hintereinander ausgeführt werden müssen. Durch die Aufteilung in sequenzielle Teiltransformationen können die Korrespondenzlinks der vorherigen Transformation nicht mehr genutzt werden, wodurch der Transformationsregelentwurf zusätzlich erschwert wird.

Integration in den MBSecMon-Entwicklungsprozess: Die beiden *Eclipse*-Integrationen für ATL und SDM unterscheiden sich in der Bereitstellung der Transformation für das Softwareprodukt, in dem sie angewendet werden soll. eMOFLON verwendet SDM-Spezifikationen zur Generierung von nativem Java-Code und ATL übersetzt die Regeln in Byte-Code, der auf speziellen virtuellen Maschinen (ATL VM /EMFTVM) ausgeführt werden kann. Wie die Evaluation in Abschnitt 11.2 zeigen wird, skaliert der native Java-Code der SDM-Transformation für große eLSC-Modelle besser als die in einer angepassten virtuellen Maschine ausgeführte ATL-Transformation.

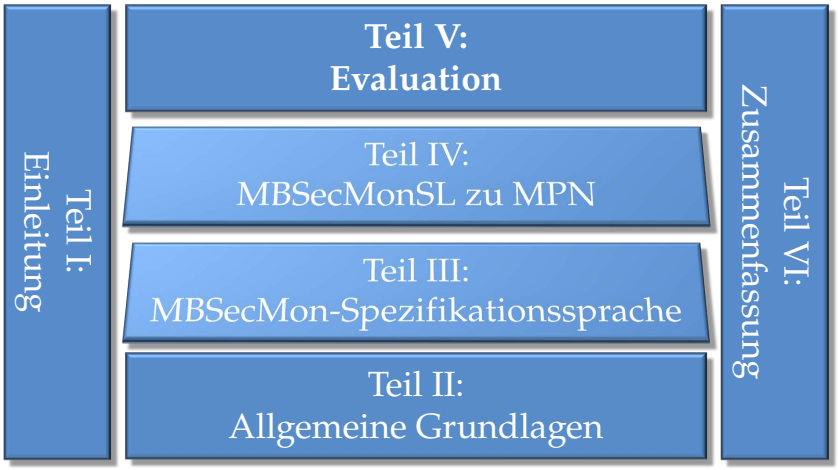
9.2.2 *Implementierung der MBSecMonSL-zu-MPN-Transformation*

Der Vergleich zwischen den häufig verwendeten Transformationssprachen und insbesondere zwischen ATL und SDM in Abschnitt 9.2 hat gezeigt, dass sowohl ATL als auch SDM grundsätzlich zur Modellierung der Transformation geeignet sind. Die Transformation kann weder mit den Mitteln von ATL noch SDM durch rein deklarative Regeln beschrieben werden, wodurch der sichergestellte Determinismus des Ergebnisses von ATL [JK06b] durch die Anwendung imperativer Konstrukte verloren geht. ATL spielt seine Vorteile bei der impliziten Erstellung der Korrespondenzlinks zwischen Quell- und Zielmodellelementen aus, während in aktuellen Implementierungen des SDM diese Korrespondenzlinks manuell in den Transformationsregeln angelegt werden müssen. Der Hauptvorteil für den Entwickler bei SDM und Hauptnachteil der ATL resultiert aus der expliziten Modellierung des Kontrollflusses in SDM. So erlaubt die Modellierung des Kontrollflusses zum einen rekursive *In-Place*-Transformationen auf einem Modell und fehlende Modellierungskonstrukte wie die im vorherigen Abschnitt erwähnten *Muster beliebiger Breite* durch eine Mischung aus Kontrollfluss und deklarativen Mustern zu spezifizieren. Zusätzlich erlaubt der Kontrollfluss der SDMs im Gegensatz zu Regeln der ATL-Elemente in einer Transformation mehrfach zu binden. Hierdurch wird die Erstellung der Transformation durch die Entwickler stark erleichtert. Auf Basis dieser Resultate des Vergleichs zwischen ATL 3.1 und der SDM-Implementierung im Werkzeug eMOFLON[3], wurde aus den oben genannten Gründen die komplette MBSecMonSL-zu-MPN-Transformation durch SDMs spezifiziert.

[3] eMoflon (Version: 1.0 build 20130803): http://www.emoflon.org

Teil V

EVALUATION



10

IMPLEMENTIERUNG UND EINBINDUNG IN EINEN ENTWICKLUNGSPROZESS

Nachdem in Teil III die verhaltensbeschreibende Signaturspezifikationssprache MBSecMonSL eingeführt und in Teil IV die Transformation in die Zwischensprache MPN vorgestellt wurde, wird in diesem Kapitel auf die Implementierung des MBSecMon-Prozesses eingegangen. Hierzu beschreibt dieses Kapitel in Abschnitt 10.1 die prototypische Implementierung des MBSecMon-Prozesses in der MBSecMon-Toolsuite. Abschnitt 10.2 zeigt anhand einer Integration der MBSecMon-Toolsuite in einen AUTOSAR-Entwicklungsprozess, die Anpassbarkeit des in dieser Arbeit entwickelten MBSecMon-Prozesses an bestehende Entwicklungsprozesse. Hierbei wird besonderes Augenmerk auf die Einbindung bestehender Entwicklungsartefakte in die Monitorsignaturmodellierung und die Nutzung dieser Informationen im Generierungsprozess gelegt.

10.1 MBSecMon-Toolsuite

Der modellbasierte Entwicklungsprozess für Security/Safety-Sicherheitsmonitore (MBSecMon) wurde als ein Prototyp, die MBSecMon-Toolsuite, implementiert. Diese Toolsuite (Abb. 10.1) enthält das in Kapitel 6 vorgestellte *MBSecMonSL-Add-In* für Sparx Systems Enterprise Architect (EA), das das UML 2.0-Modellierungswerkzeug um die Modellierung von MBSecMonSL-Spezifikationen erweitert. Das Add-In verwaltet den *MBSecMonSL-Editor*, der die zwei neuen Diagrammtypen unterstützt – die MUC-Diagramme mit den eingebetteten eLSC-Diagrammen. Zur Anpassung der existierenden Diagrammtypen werden die in Kapitel 6 vorgestellten UML2-Profile verwendet, die über die Model Driven Generation (MDG) Technologie in EA eingebunden werden. Der resultierende Editor bietet eine angepasste Toolbox, die neuen Diagrammtypen und die neuen syntaktischen Elemente, wie in Abbildung 10.2 dargestellt.

Die modellierten Spezifikationen werden durch die Komponente *XMI-Export* serialisiert. Sie extrahiert die benötigten Informationen aus dem EA-Repository und schreibt diese basierend auf dem Metamodell der MBSecMonSL in ein Ecore-XMI-kompatibles Format.

Diese exportierte Datei dient als Eingabe für das *MBSecMon-Tool*. Die Daten-Repositories – das *MBSecMonSL-Repository* und das *MPN-Repository* – werden durch das Meta-CASE-Werkzeug eMoflon[1] aus den Ecore-kompatiblen Metamodellen generiert.

[1] eMoflon CASE Tool: http://www.emoflon.org

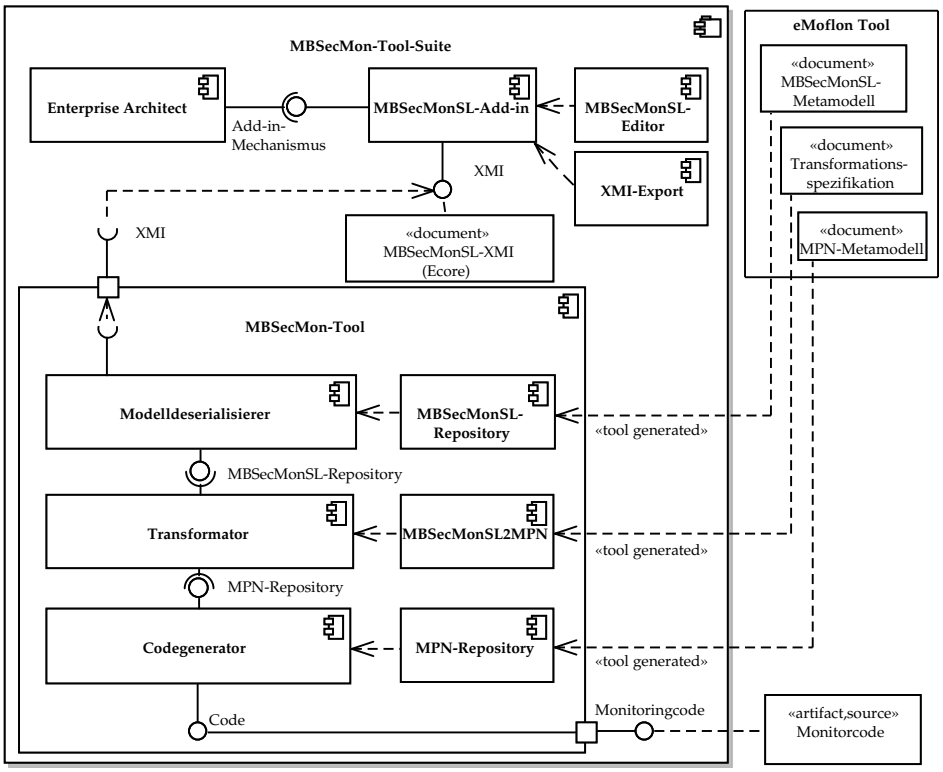


Abbildung 10.1: Die Struktur der MBSecMon-Toolsuite

Die in Kapitel 8 vorgestellte Transformation zwischen der MBSecMonSL und den MPNs ist in eMoflon mit Story Driven Modeling (SDM) [FNTZ00] spezifiziert. Der aus eMoflon generierte Transformationscode wird als Komponente *MBSecMonSL2MPN* in die Komponente *Transformator* eingebunden.

Das aus der Transformation resultierende MPN-Modell, das in dem *MPN-Repository* gespeichert wird, und zusätzliche plattformspezifische Informationen (PSI) dienen als Basis für den Codegenerierungsprozess [Pat14] der Komponente *Codegenerator*. Zur Codegenerierung wird ANTLR[2] und StringTemplate[3] eingesetzt, um die Annotationen in den MPN-Modellen wie Aktionen und Bedingung der Transitionen zu parsen und den Monitor-Code zu generieren. Der in Java implementierte Codegenerator füllt die Templates mit Informationen aus den MPN-Signaturen und aus den plattformspezifischen Informationen, die als XML-Dateien hinterlegt sind. Für jede Zielsprache können im Codegenerator verschiedene Templates spezifiziert werden. Aktuell sind dies C, Java und VHDL.

Die zusätzlichen Strukturierungsinformationen, die durch die MUC-Sprache zur eLSC-Spezifikation hinzugefügt wurden, werden, wie in Abschnitt 8.2 beschrieben, in die Kontrollstruktur des Monitors übersetzt. Hierdurch ist eine Signatur, die als eLSC beschrieben ist, nicht die gesamte Laufzeit des Monitors aktiv und belegt in Systemen mit dynamischer Speicherallokation nur Speicher, wenn die entsprechende Signatur überwacht werden muss. Durch den Einsatz

[2] ANTLR Parser Generator: http://www.antlr.org
[3] StringTemplate Engine: http://www.stringtemplate.org

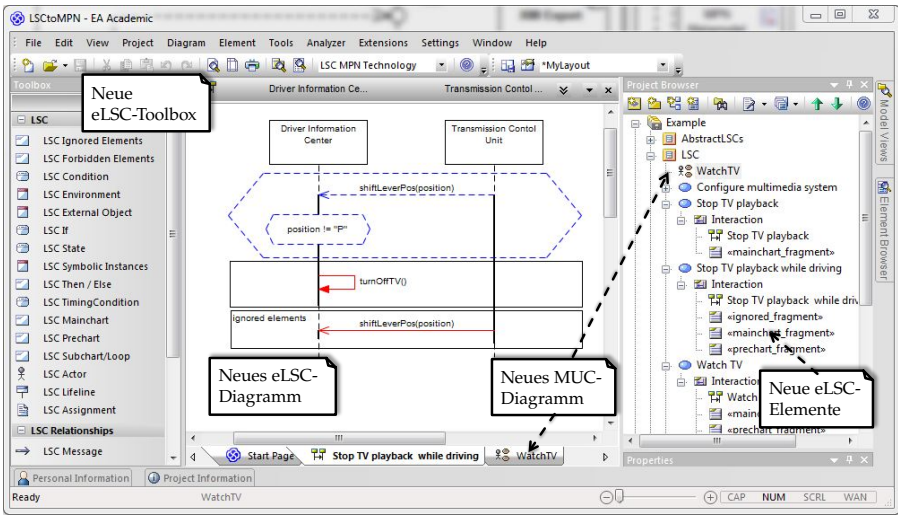


Abbildung 10.2: MBSecMon-Spezifikationssprache – Das EA-Add-In

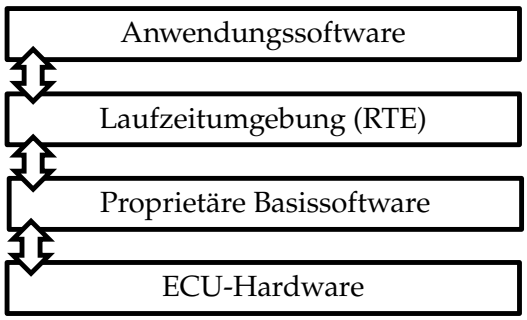


Abbildung 10.3: Übersicht über die AUTOSAR-Schichtenarchitektur

der MUC-Sprache zur Anpassung der Kontrollstruktur des Monitors kann zusätzlich die Menge der zu evaluierenden Vorbedingungen (Precharts) reduziert werden und so Rechenzeit eingespart werden. Dies erlaubt dem MBSecMon-Prozess Monitore zu generieren, die auch mit komplexeren Überwachungsaufgaben umgehen können, ohne Speicher- oder Rechenzeitgrenzen eingebetteter Systeme zu überschreiten.

10.2 AUTOSAR-Integration der MBSecMon-Toolsuite

Die Einsatzmöglichkeit der in Abschnitt 10.1 vorgestellten MBSecMon-Toolsuite wurde in einer Fallstudie [PPPM13] anhand der Integration in einen AUTOSAR-Entwicklungsprozesses evaluiert. Hierbei wird im Folgenden basierend auf der Fallstudie die Anpassung des MBSecMonSL-Add-ins und die Integration vorhandener Entwicklungsartefakte betrachtet.

10.2.1 *AUTOSAR*

Die AUTOSAR (AUTomotive Open System ARchitecture)-Plattform[4] [KF09] ist ein offener Industriestandard zur Entwicklung von Fahrzeugsystemen. Dieser Standard beschreibt eine modulare Softwarearchitektur und definiert hierfür standardisierte Schnittstellen, die über eine Laufzeitumgebung (*engl. run-time environment*) (RTE) kommunizieren.

Die RTE ist Teil einer Schichtenarchitektur (Abb. 10.3) und bildet die Schnittstelle zwischen der Basissoftware, die von der Hardware der ECU abstrahiert, und der Anwendungssoftwareebene, auf der die Softwarekomponenten (SW-C) ausgeführt werden. Auf Basis der System- bzw. Komponentensicht generiert ein RTE-Generator die RTE. Hierbei werden die abstrakt modellierten Kommunikationsbeziehungen zwischen den SW-Cs untereinander und zur Basissoftware in Kommunikationsmechanismen umgesetzt. Der AUTOSAR-Standard sieht zwei Kommunikationsmechanismen vor: Die Sender/Receiver-Kommunikation, die zum asynchronen Datenaustausch genutzt wird, und die Client/Server-Kommunikation, die Server-Dienste als Operationen für den Client zur Verfügung stellt. Durch diese Schichtenarchitektur und die damit verbundene Generierung der RTE ist die Verschiebbarkeit von Komponenten auf den verschiedenen ECUs gegeben.

Der AUTOSAR-Standard definiert zur strukturierten Spezifikation des Systems eine auf Komponentendiagrammen basierende grafische Notation, die in einem standardisierten AUTOSAR XML-Format (ARXML) serialisiert und als Austauschformat genutzt wird. Zusätzlich beschreibt eine Methodik die grundlegenden technischen Schritte, die durchgeführt werden müssen. Die darauf basierenden AUTOSAR-Entwicklungsprozesse unterstützen die Überwachung des Kontrollflusses und zeitlicher Eigenschaften auf einem niedrigen Abstraktionsniveau [AUT11a, AUT11b]. Eine Modellierung komplexer Überwachungsmuster auf einem höheren Abstraktionsniveau wie z. B. auf der Softwarekomponentenebene wird nicht zur Verfügung gestellt.

Für diesen Zweck wurde in einer Fallstudie [PPPM13] der generische Ansatz des MBSecMon-Entwicklungsprozesses (Abschn. 1.2) zur Monitorgenerierung in einen dem AUTOSAR-Standard entsprechenden Entwicklungsprozess eingebettet. Diese Methodik zur modellbasierten Entwicklung komplexer Laufzeitmonitore für AUTOSAR nutzt vorhandene Entwicklungsartefakte des AUTOSAR-Entwicklungsprozesses (u. a. das AUTOSAR-Systemmodell) zur Modellierung der zu überwachenden Signaturen.

10.2.2 *Der angepasste Monitorgenerierungsprozess für AUTOSAR*

Die MBSecMon-Werkzeugkette eignet sich zur nahtlosen Integration in bestehende modellbasierte Entwicklungsprozesse, die auf strukturbeschreibenden Spezifikationen beruhen. Die Integration dieser Werkzeugkette in einen vereinfachten Entwicklungsprozess, der der AUTOSAR-Methodik entspricht, ist in Abbildung 10.4 gezeigt. Hierfür sind nur geringfügige Anpassungen und Erweiterungen der MBSecMon-Werkzeugkette notwendig.

[4] AUTOSAR: http://www.autosar.org

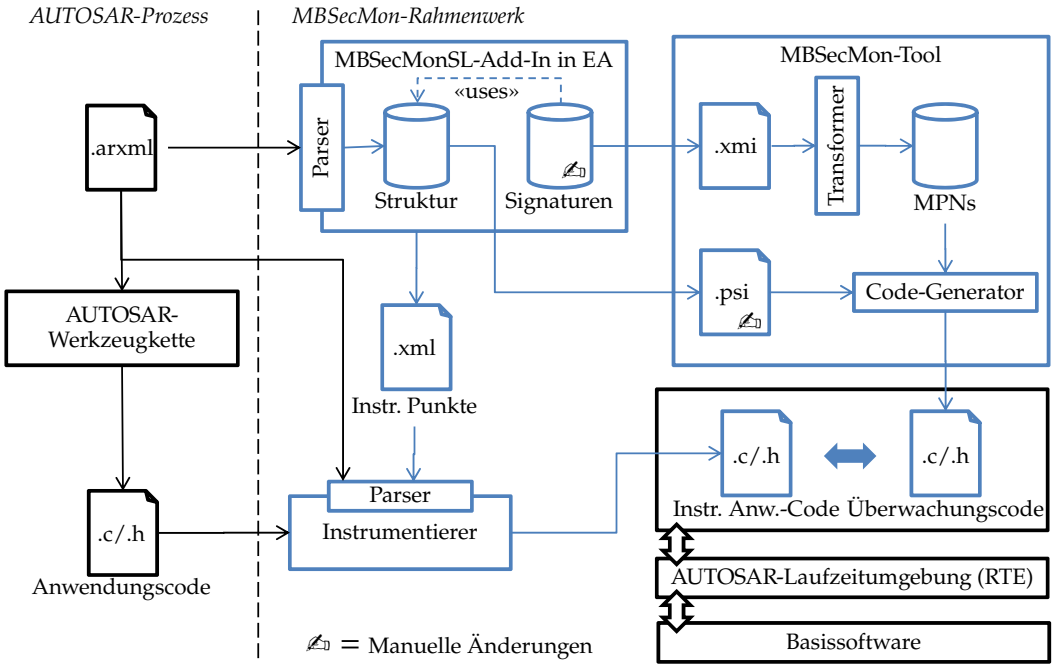


Abbildung 10.4: Das in den AUTOSAR-Entwicklungsprozess eingebettete MBSecMon-Rahmenwerk

Die linken Seite repräsentiert den vereinfachten AUTOSAR-Entwicklungsprozess, der mit der Strukturbeschreibung des Systems als ARXML-Datei beginnt. Diese Datei wird im AUTOSAR-Entwicklungsprozess verwendet, um die Skelette zur Implementierung der SW-Cs und die RTE, die durch den RTE-Generator konfiguriert wird, inklusive der Kommunikationsmechanismen zu generieren. Die Implementierung der SW-Cs erfolgt in diesem Entwicklungsprozess durch die Modellierung der SW-Cs als Simulink-Modelle, die durch den Simulink-Codegenerator in AUTOSAR-konformen C-Code übersetzt werden. Die so generierten SW-Cs und die generierte RTE sind in einer AUTOSAR-Simulationsumgebung oder direkt auf Steuergeräten eines Fahrzeuges, den Electronic Control Units (ECU), lauffähig.

Die rechte Seite der Abbildung zeigt das MBSecMon-Rahmenwerk, das die MBSecMon-Werkzeugkette in den AUTOSAR-Prozess eingebettet. Die Spezifikation der Monitorsignaturen erfolgt in dem UML2-Modellierungswerkzeug Enterprise Architect, das durch das MBSecMonSL-Add-In aus Abschnitt 6 an die Modellierung in der MBSecMonSL angepasst ist. Durch die Einbindung eines Parsers für den Import der Strukturbeschreibung und Kommunikationsbeziehungen aus der AUTOSAR-Spezifikation (ARXML-Datei) als Komponentendiagramme kann bei der Modellierung der Signaturen auf diese Informationen zurückgegriffen werden.

Die so modellierten Signaturen werden zusammen mit plattformspezifischen Informationen (PSI), die aus dem importierten AUTOSAR-Modell extrahiert werden, exportiert. Zusätzlich identifiziert das MBSecMonSL-Add-In anhand der mo-

dellierten Signaturen die Instrumentationspunkte des AUTOSAR-Anwendungscodes und persistiert diese Informationen in einer weiteren XML-Datei.

Durch die in Kapitel 8 vorgestellte graphbasierte Modell-zu-Modell-Transformation werden die exportierten Repräsentationen der Signaturen durch das MBSecMon-Tool in die formal definierten MPNs transformiert. Diese dienen als Zwischensprache zur Monitorgenerierung als AUTOSAR-konforme SW-C. Hierzu greift der Codegenerator auf die als MPN repräsentierten Signaturen und die als PSI-Datei exportierten zusätzlichen Informationen zurück. Da diese Überwachungs-SW-C durch Aufrufe ihres Interfaces stimuliert wird, müssen die AUTOSAR-Anwendungen instrumentiert werden. Hierzu kommt ein Instrumentierer [PWMS12] zum Einsatz, der wie auch die das MBSecMonSL-Add-In die AUTOSAR-Systemspezifikation zusammen mit den persistierten Instrumentierungspunkten einsetzt, um die notwendigen Schnittstellen zu instrumentieren. Durch diese Schnittstellenwrapper wird die Kommunikation zwischen den SW-Cs abgehört und diese Daten an die überwachende SW-C weitergeleitet.

Diese Anbindung an den AUTOSAR-Entwicklungsprozess ermöglicht die Spezifikation der Signaturen auf Basis der und auf derselben Abstraktionsebene wie die AUTOSAR-Systemspezifikation. Die Überwachungs-SW-Cs werden automatisch generiert, wobei nur die Spezifikationen der Monitore mit Hilfe des MBSecMonSL-Add-Ins erstellt und die PSI durch den Systementwickler leicht erweitert werden müssen.

Im Folgenden werden die notwendigen Anpassungen des MBSecMon-Entwicklungsprozesses, die im Rahmen dieser Arbeit betrachtet werden, anhand des Beispiels einer automatischen Getriebesteuerung genauer erläutert. Die Herausforderungen, auf denen diese Anpassungen beruhen, sind zusammengefasst:

H1 Die Integration der im AUTOSAR-Prozess existierenden Entwicklungsartefakte in den MBSecMon-Entwicklungsprozess.

H2 Die Absicherung der Typsicherheit während des gesamten Modellierungs- und Generierungsprozesses der Monitore.

H3 Die Modellierung der Signaturen auf demselben hohen Abstraktionsniveau wie die AUTOSAR-Spezifikationen.

H4 Die Abbildung der abstrakten Signaturen auf den plattformspezifischen Überwachungscode.

10.2.3 *Beispiel: Automatische Getriebesteuerung*

Die Modellierung der Signaturen und die Anpassungen des Entwicklungsprozesses betrachtet in diesem Abschnitt hauptsächlich die Kommunikation zwischen AUTOSAR SW-Cs. Zur Spezifikation des AUTOSAR-Systems inklusive seiner Interfaces wird das Werkzeug OptXware Embedded Architect[5] verwendet. Die Implementierung der SW-Cs wird durch die AUTOSAR-Codegenerierung von MathWork's Embedded Coder Plugin für Simulink generiert. Hierbei basieren die SW-

[5] OptXware: http://www.optxware.com

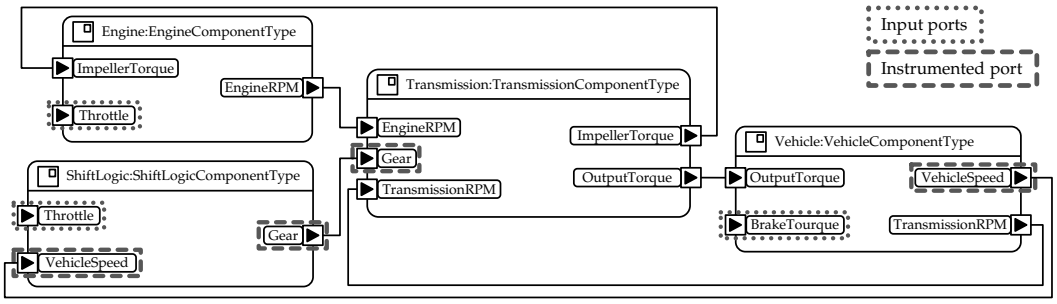


Abbildung 10.5: AUTOSAR-Komponentensicht eines automatischen Getriebes

Cs auf dem Beispielprojekt *Automatic Transmission Controller* [The12], das mit Matlab/Simulink ausgeliefert wird.

Dieses Modell beinhaltet, wie in Abbildung 10.5 als AUTOSAR-Komponentensicht dargestellt, drei SW-Cs, *ShiftLogic*, *Transmission* und *Engine*, die das interne Verhalten einer automatischen Getriebesteuerung beschreibt. Basierend auf den Eingangswerten *Throttle* und *BrakeTorque* und der Interaktion zwischen den SW-Cs werden die Komponenten stimuliert. Umwelteinflüsse auf das Fahrzeug wie die Aerodynamik des Fahrzeugs und Reibungswiderstände der Reifen werden durch die Komponente *Vehicle* repräsentiert. Die asynchrone Kommunikation zwischen den Komponenten wird durch Sender- und Receiver-Ports mit den entsprechenden Interfaces modelliert. Um den Vorgaben der AUTOSAR-Codegenerierung des Embedded Coders zu erfüllen, muss das verhaltensbeschreibende Simulink-Modell angepasst werden, da die Codegenerierung kontinuierliche Blöcke wie den Integrator nicht zulässt. Selbst bei Mikrocontrollern, die Fließkommaberechnungen unterstützen, führen kontinuierliche Blöcke zu einer unnötigen Auslastung des Systems. Nach dem Austausch aller kontinuierlichen Blöcke gegen ihre diskrete Variante wurden durch Testmessungen die Abweichungen des Ausgabeverhaltens bestimmt, um die Äquivalenz der beiden Modelle sicherzustellen.

Die Schnittstellen des generierten Codes werden durch die Konfigurationseinstellungen des Embedded Coders so angepasst, dass er direkt in das durch OptXware Embedded Architect generierte SW-C-Skelette eingefügt werden kann. Die generierte Monitor-SW-C wird gegen die Systemimplementierung in einer AUTOSAR-Simulationsumgebung ausgeführt und so die Funktionalität des MBSecMon-Prozesses getestet.

10.2.4 *Einsetzbarkeit des MBSecMon-Entwicklungsprozesses*

In diesem Abschnitt wird anhand der vorgestellten Herausforderungen H1-H4 gezeigt, dass der generische MBSecMon-Entwicklungsprozess in den AUTOSAR-Entwicklungsprozess integrierbar ist.

Herausforderung 1: INTEGRATION EXISTIERENDER ENTWICKLUNGSARTEFAKTE AUS DEM AUTOSAR-PROZESS. Der AUTOSAR-Standard und somit der AUTOSAR-Entwicklungsprozess sehen die Modellierung der Systemstruktur auf einer hohen Abstraktionsebene vor. Hier werden in der Komponentensicht

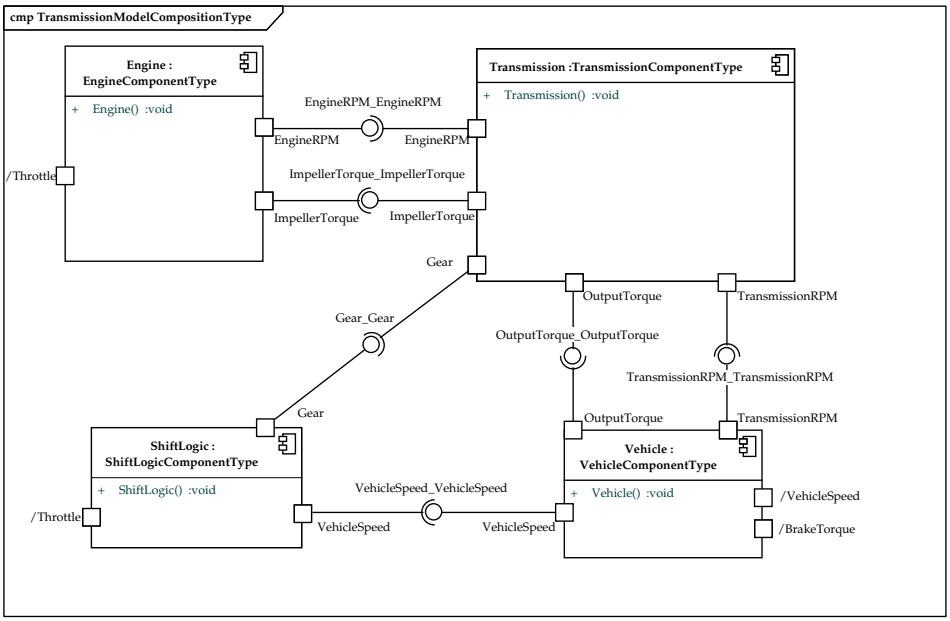


Abbildung 10.6: UML2-Komponentendiagramm des AUTOSAR-Systemmodells

SW-Cs, ihre Ports, deren Interface-Beschreibungen die Datentypen festlegen, und die Verbindungen zwischen den Komponenten spezifiziert. Ziel der Integration ist es, dass die generierten Monitore die Kommunikation zwischen den SW-Cs überwachen. Um dies zu erreichen, sollte die Modellierung der Signaturen auf dieser abstrakten Beschreibung, der Komponentensicht von AUTOSAR, basieren.

MBSecMon-Entwicklungsprozess: Das MBSecMonSL-Add-In verwendet eine AR-XML-Parser-Bibliothek, die die SW-C-Struktur, die in einem AUTOSAR-Modellierungswerkzeug wie OptXware Embedded Architect modelliert wurde, verarbeitet. Abgeleitet von diesen Informationen erstellt das Add-In ein UML2-Komponentendiagramm (Abb. 10.6), das die Komponenten, ihre Ports und die verbindenden Konnektoren beinhaltet. Hierbei werden die Namensgebungsrichtlinien von AUTOSAR berücksichtigt. Zur Steigerung der Übersichtlichkeit der Diagramme werden Systeminformationen wie der Namen der Konnektoren und der Ports in den Modellelementen abgekürzt, jedoch die vollständigen Bezeichner intern als Eigenschaftswerte (*engl. tagged values*) gespeichert.

Beispiel: Die SW-Cs in Abbildung 10.6 entsprechen der Komponentensicht in Abbildung 10.5 und somit den Blöcken des Simulink-Modells, aus dem der Anwendungscode generiert wird. Basierend auf den Komponenten und den Konnektoren zwischen den Komponenten werden im MBSecMon-Entwicklungsprozess die Signaturen modelliert, die die erlaubten und verbotenen Kommunikationssequenzen beschreiben.

Herausforderung 2: ABSICHERUNG DER TYPSICHERHEIT. Der AUTOSAR-Entwicklungsprozess setzt, wie auch allgemein in der Automobilindustrie üblich, auf spezialisierte Datentypen. Durch diese Spezialisierung können die Wertebereiche der Variablen über die Verwendung der Standarddatentypen hinaus eingeschränkt werden. Zuweisungen zwischen unterschiedlichen Datentypen können

Auflistung 10.1: Typsicherheit in AUTOSAR (ARXML-Datei)

```
<REAL-TYPE >
  <SHORT-NAME>VehicleSpeedDataType</SHORT-NAME>
  <LOWER-LIMIT INTERVAL-TYPE="CLOSED" >-1000.0</LOWER-LIMIT>
  <UPPER-LIMIT INTERVAL-TYPE="CLOSED" >1000.0</UPPER-LIMIT>
  <ALLOW-NAN>false</ALLOW-NAN>
  <ENCODING>DOUBLE</ENCODING>
</REAL-TYPE>
```

einfach durch Analysen erkannt werden. Auch wenn z. B. der Geschwindigkeitswert und der Drehmoment auf den gleichen Standarddatentyp basieren, wird hierdurch effektiv die Zuweisung eines Geschwindigkeitswertes auf eine Variable für den Drehmoment verhindert. Die generierten Monitore müssen diesen impliziten Safety-Mechanismus unterstützen, der im AUTOSAR-Standard vorgeschrieben ist.

MBSecMon-Entwicklungsprozess: Die speziellen Datentypen, die in der AUTOSAR-Systemspezifikation durch die Schnittstellen der Ports definierten sind, werden parallel mit der Komponentensicht in EA importiert und in den Ports der UML2-Komponentendiagramme gespeichert. Die Signatur-Modellierung verwendet diese Schnittstelleninformationen aus dem Komponentendiagramm und limitiert den Entwickler auf diese bei der Modellierung der Nachrichten in der MBSecMonSL. Im weiteren Verlauf des MBSecMon-Prozesses werden diese weiter verarbeitet (siehe Herausforderung 4) und der Codegenerator nutzt diese um eine typsichere Überwachungs-SW-C zu generieren.

Beispiel: Die ARXML-Datei enthält, wie in Auflistung 10.1 gezeigt, die Spezifikation der Datentypen. Der Datentyp *VehicleSpeedDataType*, der die Geschwindigkeit des Fahrzeugs repräsentiert, ist vom Basisdatentyp *Double* und wird in seinem Wertebereich auf das abgeschlossene Intervall `[-1000; +1000]` eingeschränkt.

Herausforderung 3: MODELLIERUNG AUF DER GLEICHEN ABSTRAKTIONSEBENE. Die Systemspezifikationen des AUTOSAR-Standards werden auf einer hohen Abstraktionsebene modelliert, wie in Herausforderung 1 zu sehen ist. Hierbei beschreiben die Ports, welche Kommunikationsmechanismen genutzt werden, um die Interaktion zwischen den SW-Cs über den virtuellen Funktionsbus (VFB) zu realisieren. Eine Überwachung dieser Kommunikation zwischen den SW-Cs ist Ziel der durch den MBSecMon-Prozess erstellten Monitore. Ein weitverbreiteter Ansatz Interaktionen zwischen Komponenten zu beschreiben sind Varianten der Sequenzdiagramme, die in Kapitel 3 vorgestellt wurden. Diese liegen auf derselben Abstraktionsebene wie die AUTOSAR-Spezifikation, bestehend aus der Komponentensicht und dem virtuellen Funktionsbus.

MBSecMon-Entwicklungsprozess: Die Modellierung der verhaltensbeschreibenden Signaturen erfolgt im MBSecMon-Prozess in der MBSecMonSL, die aus eLSCs und der strukturierenden MUC-Sprache besteht. Hierbei bildet der Import der der Komponentensicht des AUTOSAR-Werkzeugs in EA die Basis der Signaturmodellierung. So bietet das MBSecMonSL-Add-In (Fig. 10.7) die Modellierung der Lebenslinien auf Basis der SW-Cs und eine kontextsensitive Auswahl der Nachrichten und ihrer Parameter zwischen den SW-Cs. Diese Einschränkungen in der

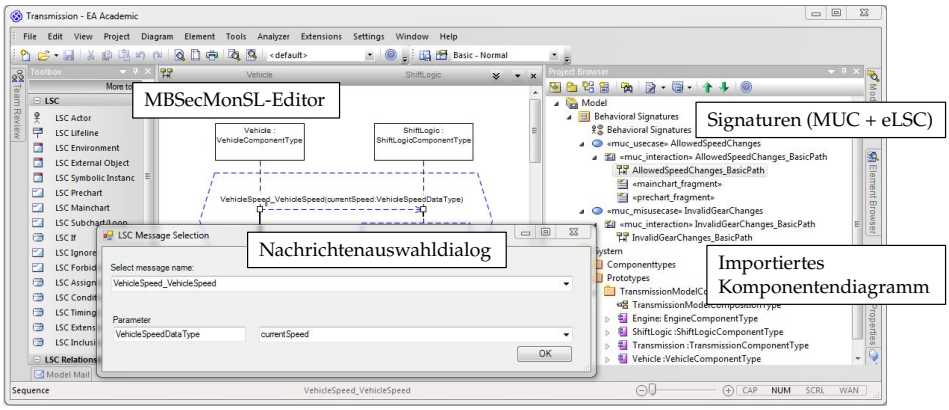


Abbildung 10.7: Angepasster Editor für den MBSecMon-Entwicklungsprozess für AUTOSAR

Modellierung der Signaturen sichert ihre Konformität zum modellierten AUTOSAR-System.

Beispiel: Abbildung 10.8 zeigt eine einfache nebenläufige Signatur, die die Kommunikation zwischen der SW-C *Vehicle* und der SW-C *ShiftLogic* überwacht. Hierbei ordnet der Editor die Typen *VehicleComponentType* und *ShiftLogicComponentType* der SW-Cs den Lebenslinien im eLSC zu. Zur Überwachung wird der Monitor, der aus der Signatur generiert wird, für jede Nachricht initialisiert, die über den Port *VehicleSpeed* übermittelt wird. Die Nachricht enthält einen Parameter *currentSpeed*, dessen Typ *VehicleSpeedDataType* ebenfalls durch die AUTOSAR-Spezifikation festgelegt ist. Dieser Parameter wird über die Zuweisung auf einer eLSC-spezifischen Variable *lastSpeed* gespeichert.

Das erste durch den Monitor verarbeitete Sendeereignis der Nachricht führt zu einer erneuten Instanziierung der Signatur, die dann nebenläufig zur vorherigen Signaturinstanz die nächste *VehicleSpeed*-Nachricht überwacht. Diese Nachricht wird durch das Mainchart der ersten Signaturinstanz verarbeitet und der übermittelte Wert in der folgenden Bedingung mit dem zuvor gespeicherten verglichen. Basierend auf der Auswertung der Bedingung, wird die Signatur positiv oder negativ beendet. Nachfolgend wird dieselbe Nachricht ebenfalls durch das Prechart der zweiten Signaturinstanz ausgewertet und der alte Wert der Variable *lastSpeed* durch den neuen Wert *currentSpeed* überschrieben.

Herausforderung 4: ABBILDUNG AUF PLATTFORMSPEZIFISCHEN MONITORINGCODE. Die Modellierung der Signaturen auf einer höheren Abstraktionsebene als der generierte Code für die Zielplattform und die Verwendung eine Annotationssprache erfordert zusätzliche Informationen über die Abbildung auf die Zielplattform. Viele dieser zur Codegenerierung notwendigen Informationen sind in der AUTOSAR-Spezifikation vorhanden.

MBSecMon-Entwicklungsprozess: Zur Sicherstellung von typsicheren, AUTOSAR-konformen Schnittstellen der generierten Überwachungs-SW-C werden beim Export zusätzliche Informationen in die PSI-Datei geschrieben. Sie enthält Abbildungen der AUTOSAR-Instrumentierungsschnittstellen auf die internen Ereignisse des Monitors, Datentypen der übertragenen Daten, eine Abbildung des Codes

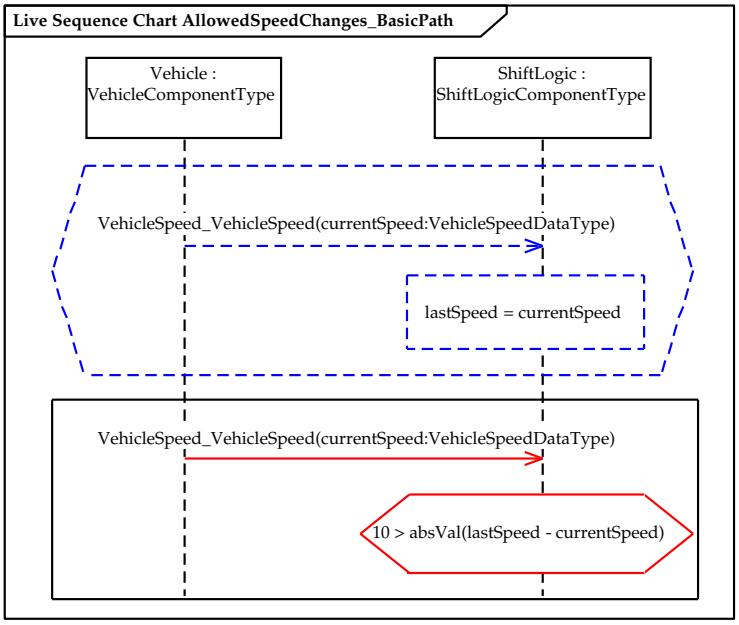


Abbildung 10.8: Signatur des Use-Cases „AllowedSpeedChanges“

Auflistung 10.2: eLSC-spezifische Informationen der PSI-Datei

```
<entry key="AllowedSpeedChanges_BasicPath.context_method.absVal">fabs
</entry>
```

der Signaturannotationen auf die Zieldomäne und Konfigurationsinformationen für die Codegenerierung. Fast alle diese Informationen lassen sich aus den modellierten Signaturen und der importierten Spezifikation des AUTOSAR-Systems extrahieren. Eine Ausnahme bildet die Übersetzung des annotierten Pseudocodes in den Signaturen, für die das MBSecMonSL-Add-In eine 1-zu-1-Abbildung generiert, die jedoch manuell an die Zielsprache angepasst werden muss. Diese Abbildung kann zur vereinfachten Anwendung des Generierungsprozesses in einer Abbildungsbibliothek, die für die Zielplattform oder Sprache gültig ist, hinterlegt werden.

Beispiel: Die für die Signatur in Abbildung 10.8 generierte PSI-Datei enthält u. a. eine Abbildung der in der Bedingung verwendeten Methode `absVal` auf die plattformkonforme Methode `fabs`. Die Methode *absVal* repräsentiert hierbei in einer von der Zielsprache unabhängigen Annotationssprache eine Methode, die den Absolutwert des übergebenen Arguments bestimmt. Auflistung 10.2 zeigt diesen XML-Konten, dessen ursprünglicher Wert `absVal` manuell durch den Modellierer mit `fabs` ersetzt wurde.

10.2.5 *Zusammenfassung*

Die vorgestellten Lösungen zur Bewältigung der Herausforderungen zeigen, dass die Modellierung von Signaturen und die Integration einer Laufzeitüberwachung

von komplexen Signaturen in den AUTOSAR-Entwicklungsprozess durch geringfügige Anpassungen möglich sind. Zudem wurde die Adaptierbarkeit des MBSecMon-Prozesses an einem und die Einbeziehung vorhandener Informationen aus einem existierenden Entwicklungsprozess nachgewiesen. Eine genaue Evaluation des Laufzeitverhaltens und des Speicherbedarfs der generierten Laufzeitmonitore im AUTOSAR-Kontext ist in [Pat14] zu finden. Hier wird gezeigt, dass diese Monitore auch auf eingebetteten Systemen zur Überwachung von AUTOSAR-SW-Cs und ihrer Kommunikation einsetzbar sind.

11

EVALUATION DER TRANSFORMATION

Nachdem in Kapitel 10 die Anpassbarkeit des MBSecMon-Prozesses und der MBSecMon-Toolsuite betrachtet wurde, wird in diesem Kapitel die Einsetzbarkeit der MBSecMonSL und der Transformation im MBSecMon-Prozess evaluiert. Hierzu werden in Abschnitt 11.1 die Vorteile der MBSecMonSL für die Modellierung der Signaturen im Gegensatz zur direkten Modellierung in der MPN-Sprache herausgestellt. In Abschnitt 11.2 wird die Skalierbarkeit der Transformation in der ATL- und SDM-Variante gegenübergestellt und gezeigt, warum die vollständige Implementierung der Transformation im MBSecMon-Prozess mit SDMs erfolgt ist.

11.1 Vergleich syntaktischer Elemente der eLSC- und der MPN-Repräsentation

Einen wichtigen Gesichtspunkt der Modellierung in einer Signatursprache stellt die Verständlichkeit und Übersichtlichkeit der Signaturen dar. Domänenspezifische Sprachen (DSLs) [Fow10], wie die hier vorgestellte MBSecMonSL und die Zielsprache der Transformation MPNs, haben zum Ziel, die Produktivität im Entwicklungsprozess zu steigern. Die DSL soll die Kommunizierbarkeit des beschriebenen Systems erhöhen. Durch die Einschränkung der Ausdrucksstärke einer DSL im Gegensatz zu eine General Purpose Language (GPL) und eine höhere Abstraktion über die Modellabstraktion hinaus, werden Fehler bei der Modellierung verhindert.

Hierbei muss ein Kompromiss zwischen der Spezialisierung der DSL (Anzahl syntaktischer Elemente und semantischer Komplexität der spezialisierten syntaktischen Elemente) und der allgemeinen Einsetzbarkeit der Sprache in der Domäne gefunden werden. Die Verständlichkeit der Spezifikation hängt zudem von der Anzahl der zur Modellierung eingesetzten syntaktischen Modellelemente ab.

Abbildung 11.1 zeigt anhand des CARDME-Protokolls die Unterschiede der grafischen Repräsentation zwischen den abstrakteren eLSCs und den MPNs, die den Kontrollfluss expliziter beschreiben. Während die eLSCs viele verschiedene Modellelementtypen zur Verfügung stellen und so eine klare Strukturierung der Diagramme ermöglichen, beschränkt sich die MPN-Sprache auf wenige Modellierungselemente. Schon an diesem Beispiel ohne optionales Verhalten zeigt sich, dass die direkte Modellierung der Signaturen in den MPN-Sprache durch fehlende Strukturierungselemente und dem generischen Einsatz von Plätzen und Transitionen zu umfangreichen und unübersichtlichen Spezifikationen führt. Im Folgenden wird anhand eines Vergleiches der notwendigen syntaktischen Elemente in der Signatur gezeigt, dass MPNs zwar als Zwischensprache für die Codegene-

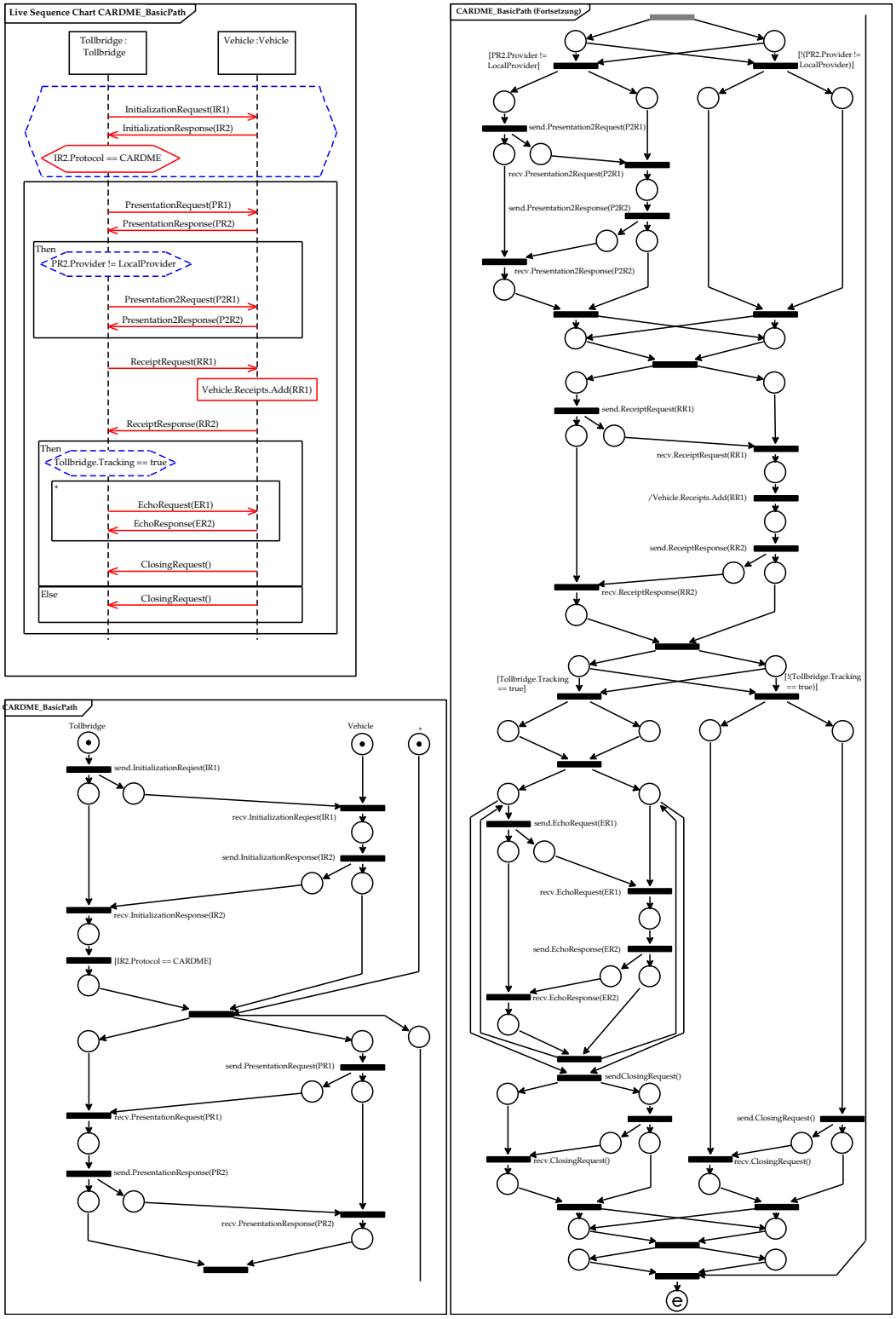


Abbildung 11.1: CARDME-eLSCs als MPN-Repräsentation

eLSC-Element	Instanz-anzahl	eLSC-Rep.		MPN-Repräsentation		Konkrete Trans-formation
		# *Elemente*	# *Locations*	# *Transitionen*	# *Plätze*	
Lebenslinien	l	l	0	0	l	siehe Abb. 8.1
Sync. heiße Nachrichten	n	n	$n \cdot 2$	n	$n \cdot 2$	siehe Abb. 8.2
Sync. heiße Eigennachr.	n	n	$n \cdot 2$	n	n	siehe Abb. 8.2
Async. heiße Nachrichten	n	n	$n \cdot 2$	$n \cdot 2$	$n \cdot 3$	siehe Abb. 8.2
Async. kalte Nachrichten	n	n	$n \cdot 2$	$n^2 + n \cdot 3$	$n \cdot 3$	siehe Abb. 8.8
heiße Bedingungen	b	b	$b \cdot l$	$b \cdot (1 + 1)$	$b \cdot (l + 1)$	siehe Abb. 8.3
kalte Bedingungen	b	b	$b \cdot l$	b	$b \cdot l$	siehe Abb. 8.3
kalte Bedingung im Subchart	b	b	$b \cdot l$	$b + a$	$b \cdot l$	–
heiße Zuweisung	z	z	$z \cdot l$	z	$z \cdot l$	siehe Abb. 8.4
Pre-/Mainchart-Übergang	1	1	$f \cdot l$	1	l	siehe Abb. 8.7
If-Then-Else-Fragmente	f	f	$f \cdot l$	$f \cdot 7$	$f \cdot l \cdot 6$	siehe Abb. 8.14
Schleifen	f	f	$f \cdot l$	$f \cdot 3$	$f \cdot l \cdot 2$	siehe Abb. 8.15
Par-Fragmente	f, c	f	$f \cdot l$	$f \cdot (2 + c \cdot 2)$	$f \cdot l \cdot (2 + c \cdot 2)$	siehe Abb. 8.16
Alt-Fragmente	f, c	f	$f \cdot l$	$f \cdot (1 + c \cdot 3)$	$f \cdot l \cdot (2 + c \cdot 2)$	siehe Abb. 8.17 (Worst-Case)

l = # Lebenslinien; n=# Nachrichten; b=# Bedingungen; z=# Zuweisungen; f=# Fragmente; c=# Subcharts; a=# Transitionen zum Abräumen im Fehlerfall

Tabelle 11.1: Vergleich der syntaktischen Elemente

rierung geeignet sind, jedoch die Modellierung in der abstrakteren eLSC-Sprache lesbarer und hierdurch weniger fehleranfällig ist.

Tabelle 11.1 stellt die Anzahl der eLSC-Konstrukte und die zu ihrer Repräsentation benötigten MPN-Modellierungselemente gegenüber. Die ermittelten Formeln für die Anzahl der Transitionen und Plätze im MPN basieren auf den konkreten Transformationsregeln aus Abschnitt 8.1. Die erste Spalte der Tabelle bezeichnet das *eLSC-Element*, das zu einem eLSC hinzugefügt wird, und die zweite Spalte die *Instanzanzahl* dieses Elementes als Variable, die in den folgenden Formeln verwendet wird. Die dritte und vierte Spalte geben die *Anzahl der Elemente* im eLSC und die benötigte *Anzahl der Locations* an. Hierbei sind bei der Modellierung der eLSCs die Locations und ihre Temperatur meist implizit durch die modellierten eLSC-Elemente und ihre Temperatur festgelegt. Eine Ausnahme bilden nur die beiden speziellen asynchronen Nachrichten (Fall 2 und Fall 7 in Tabelle 4.1), die verschiedene Temperaturen auf Sende- und Empfangsseite besitzen. Die fünfte und sechste Spalte beschreiben die *Anzahl der Transitionen* und die *Anzahl der benötigten Plätze* in der MPN-Repräsentation. Die siebte Spalte referenziert die *konkrete Transformation*, die als Referenz zur Herleitung der Formeln herangezogen wurde.

Die Elemente der eLSCs lassen sich in drei Gruppen einteilen, die von der Anzahl der in der MPN-Repräsentation benötigten Elemente abhängen. Die erste Gruppe bilden die drei verschiedenen Arten der Lebenslinien, deren Anzahl den erstellten Initialplätze im MPN entspricht.

Die zweite Gruppe bilden die synchronisierenden Elemente, deren MPN-Repräsentation von der Anzahl der Elemente und zusätzlich von der Anzahl der Locations im eLSC abhängt. Ihr mehrmaliges aufeinanderfolgendes Auftreten führt zu einem konstanten Wachstum der MPNs. Eine synchrone heiße Nachricht, die immer zwei Locations besitzt, wird in der MPN-Repräsentation als eine synchronisierende Transition und zwei nachfolgenden Plätzen dargestellt. Für synchrone heiße Eigennachrichten fällt ein Platz im Nachbereich der Transition weg, da nur eine Lebenslinie beteiligt ist. Heiße Bedingungen können zu mehreren Lebenslinien gehören und die entstehenden Plätze im MPN sind somit abhängig von ihrer der Anzahl (l). Zusätzlich führt eine zweite Transitionen auf einen entsprechenden Terminalplatz um die Verletzung der Bedingung zu repräsentieren. Kalte Bedingungen, die nicht in einem Subchart platziert sind, und heiße Zuweisungen, synchronisieren ebenfalls die ihnen zugehörigen Lebenslinien. Das hinzufügen eines Precharts (Pre-/Mainchart-Übergang) wird in eine Transition und in der Anzahl der beteiligten Lebenslinien entsprechenden Plätze übersetzt. Die weiteren kontrollflusssteuernden Elemente aus Abschnitt 8.1.3 fallen ebenfalls in diese Kategorie und sind in Tabelle 11.1 erfasst. Sie führen an ihrem Anfang und ihrem Ende eine Synchronisation im MPN durch und erzeugen hierfür eine konstante Anzahl an Modellelementen im MPN.

Die letzte Gruppe bilden eLSC-Element die keine synchronisierenden Eigenschaften haben, jedoch in Kombination mit kalter Temperatur zu einem starken Wachstum der Transitionen des MPNs führt. Asynchrone Nachrichten werden in zwei Transitionen und drei Plätze übersetzt. Ist die asynchrone Nachricht jedoch kalt, muss dieses optionale Verhalten im MPN explizit durch Übersprungstransitionen modelliert werden. Für die erste kalte Nachricht sind dies zwei zusätzliche Transitionen. Folgt direkt darauf eine weitere kalte Nachricht, muss sowohl die erste als auch die zweite kalte Nachricht auf Sender- und Empfangsseite im MPN übersprungen werden und es entstehen insgesamt vier zusätzliche Übersprungstransitionen. Jede weitere folgende kalte Nachricht erzeugt $2 * n$ zusätzliche Transitionen, wobei n der Anzahl der aufeinanderfolgenden kalten Nachrichten entspricht. Insgesamt werden somit nach Formel 11.1 durch die Transformation zusätzlich zur Übersetzung heißer Nachrichten #t Übersprungstransitionen im Ziel-MPN erzeugt.

$$\#t = \sum_{i=1}^{n} 2 * i = n * (n+1) = n^2 + n \tag{11.1}$$

Dieses Wachstum der Transitionsanzahl im MPN ist in Abbildung 11.2 visualisiert. Während heiße asynchrone Nachrichten zu einem linearen Wachstum des MPNs führen, ist dieses für aufeinanderfolgende kalte asynchrone Nachrichten polynomiell. Dieses polynomielle Wachstum tritt in realen Signaturen jedoch nur sehr eingeschränkt auf, da eine solche Anhäufung, bei der alle aufeinander fol-

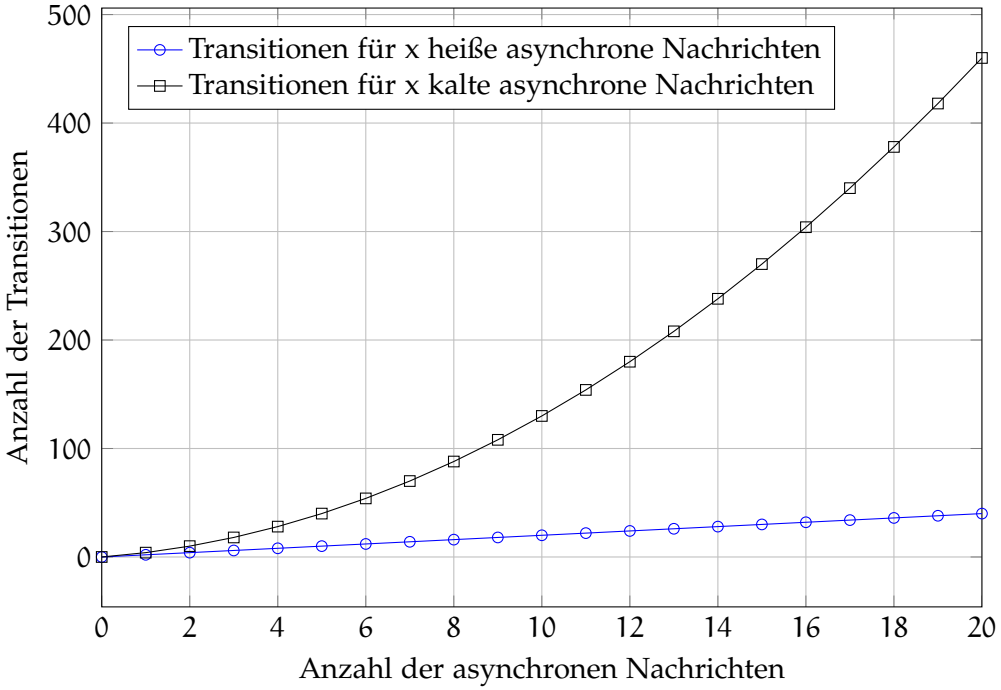


Abbildung 11.2: Wachstum der Transitionen durch kalte Nachrichten

genden Nachrichten optional sind, sich meist auf zwei oder drei Nachrichten beschränkt.

Neben den kalten asynchronen Nachrichten kann ein ähnliches Wachstum an Transitionen durch kalte Bedingungen, die an irgendeiner Stelle eines Subcharts vorkommen, verursacht werden. Wenn die entsprechende Bedingung ausgewertet wird, muss es für jede Kombination der möglichen belegten Plätze eine Epsilon-Transition mit der negierten Bedingung geben, die den Bereich des eLSC-Subcharts abräumen und die Plätze nach der Endsynchronisation des Subcharts belegen. Die Anzahl der benötigten Transitionen muss durch eine Analyse während der Transformation erfolgen und hängt stark vom Inhalt des Subcharts ab.

Der Vergleich der in eLSCs explizit zu modellierenden Elemente und den Elementen zur Repräsentation dieser eLSC-Elemente in der MPN-Sprache zeigt, dass in der zweiten Gruppe eine Abbildung auf mehrere Modellelemente der MPN-Sprache stattfindet. In der dritten Gruppe steigen zudem die Anzahl an Modellelementen – insbesondere die Transitionen – durch das optionale Verhalten der kalten Nachrichten und Bedingungen stark an. Dies zeigt, dass die Spezifikation der Signaturen als eLSCs für den Modellierer klarer und weniger fehleranfällig ist.

11.2 Vergleich der Laufzeit der Transformationen in ATL und SDM

Zur Untersuchung der Anwendbarkeit der spezifizierten Transformationen in ATL und SDM wird in diesem Kapitel die Skalierbarkeit der Transformation betrachtet. Hierbei wird nicht die im Funktionsumfang reduzierte Variante der

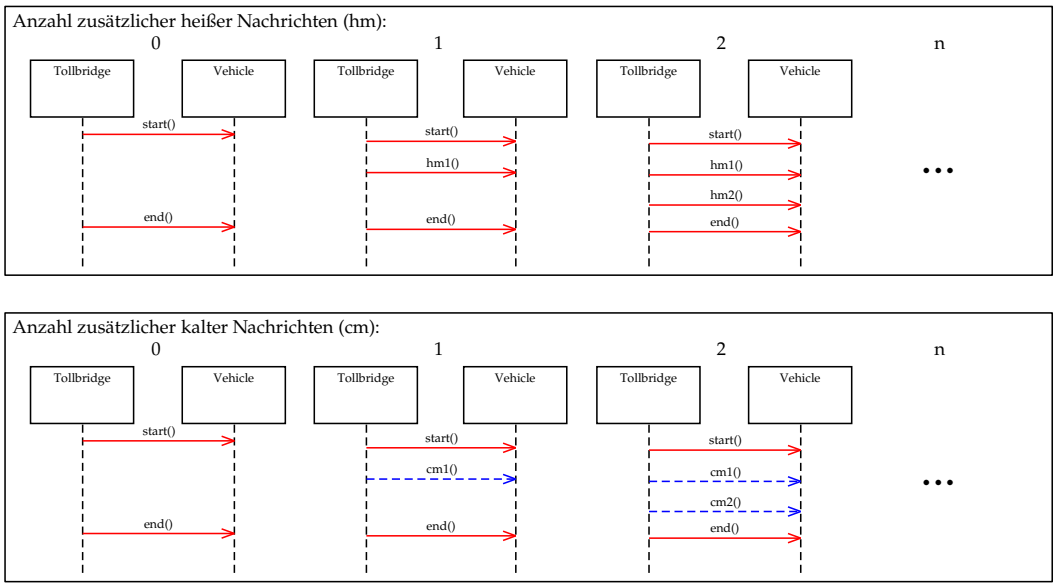


Abbildung 11.3: eLSCs zur Evaluation der Transformationszeiten

ATL-Transformation aus Abschnitt 9.2 verwendet, sondern eine, die zusätzlich die Transformation von Precharts der Universal eLSCs unterstützt. Zur Einschätzung der Laufzeit der Transformation werden zwei extreme Ausprägungen der eLSCs betrachtet. Beide basieren in ihrer grundlegenden Variante auf einem eLSC mit zwei Lebenslinien (`Tollbridge` und `Vehicle`) und zwei heißen asynchronen Nachrichten, wie in Abbildung 11.3 dargestellt. Schrittweise werden diese grundlegende eLSCs um zusätzliche asynchrone Nachrichten erweitert und die für die Transformation entsprechenden Transformationszeiten gemessen. Im ersten weniger aufwendigen Fall werden zusätzliche heiße Nachrichten zwischen die zwei vorhanden Nachrichten hinzugefügt. Im zweiten Fall werden statt der heißen Nachrichten kalte hinzugefügt und so ein polynomielles Wachstum des Zielmodells, wie in Abschnitt 11.1 gezeigt, hervorgerufen. Das polynomielle Wachstum resultiert aus den für kalte Nachrichten erzeugten Übersprungstransitionen im MPN.

Die Messungen wurden auf einem Lenovo ThinkPad T400 durchgeführt, das mit einem Intel Core 2 Duo P8600 2,40 GHz, 8 GB Arbeitsspeicher und Windows7 Professional SP1 (64 bit) ausgerüstet ist. Zur Ausführung der Transformationen wurde die Java-Plattform[1] eingesetzt, wobei der Virtuellen Maschine 2 GB Arbeitsspeicher zur Verfügung standen. Für die Ausführung der rein deklarativ gehaltenen ATL-Transformationsspezifikation wurde ATL 3.3.1 in einem Eclipse Juno (SR2) eingesetzt und als Virtuelle Maschine die performantere EMFVM verwendet. Die Spezifikation der SDM-Transformation und Generierung des Transformationscodes erfolgte mit dem Meta-Case-Werkzeug eMoflon[2].

Die beiden Transformationen wurden programmatisch ausgeführt und es wurden jeweils für jedes eLSC-Modell zehn getrennte Messungen durchgeführt. Die

[1] Oracle Java Platform (JDK) 7u25
[2] eMoflon (Version: 1.0 build 20130803): http://www.emoflon.org

```
rule bypassTransition {
    from
        firstLoc : LSCMM!Location,
        secondLoc : LSCMM!Location(firstLoc.isBypassCombination(secondLoc)
    to
        ...
}

helper context LSCMM!Location def: isBypassCombination(secondLoc:LSCMM!Location):Boolean =
    if not self.synchronizableElement.oclIsUndefined() or
        not secondLoc.synchronizableElement.oclIsUndefined() then false
    else
        let allLocs: Sequence(LSCMM!Location) = self.object.locations -> asSequence() ->
            select(relevantLoc | relevantLoc.chart.oclIsTypeOf(LSCMM!Mainchart) or
            relevantLoc.isLocOfLastMessageInPrechart) in
        if self.object = secondLoc.object and allLocs.indexOf(self) < allLocs.indexOf(secondLoc)
        then
            let locs: Sequence(LSCMM!Location) =
                allLocs.subSequence(allLocs.indexOf(self),
                allLocs.indexOf(secondLoc)).excluding(self).excluding(secondLoc) in
            if locs.size() < 1 then false
            else (not (locs-> select(loc | loc.isHot).size() > 0))
            endif
        else false
        endif
    endif
;
```

Abbildung 11.4: ATL-Regel zur Erstellung von Übersprungstransitionen im MPN

für die Transformation benötigten Zeiten wurden mithilfe des hochauflösenden Zählers (QueryPerformanceCounter) der Win32-API gemessen. Zur Ansteuerung des Zählers wurde die Methode `System.nanoTime()` der Java-API vor und nach der Transformation eingesetzt. Hierbei wurde sowohl die Transformationszeiten mit bzw. ohne das Laden des eLSCs als Quellmodell und die Serialisierung des MPN-Zielmodells gemessen.

Das Diagramm in Abbildung 11.5 stellt die Transformationszeiten für eine wachsende Anzahl an asynchronen heißen Nachrichten im Quellmodell dar. Die Fehlermarker des Diagramms zeigen die Streuung der Messwerte um den Mittelwert der Messungen. Trotz des linear wachsenden Quellmodells steigt die Laufzeit der ATL-Transformation polynomiell bzw. exponentiell an.[3] Der Grund hierfür liegt in der rein deklarativen ATL-Spezifikation, die den Kontrollfluss und damit die Anwendung der Regeln nur implizit über den *from*-Teil der Transformation definiert. Zur Generierung der Übersprungstransitionen werden durch die deklarative ATL-Regel `bypassTransition` in Abbildung 11.4 alle Paare der Locations im eLSC überprüft. Basierend auf den betrachteten Locations bestimmt nun der Helper `isBypassCombination`, ob eine Übersprungstransition zwischen den Plätzen des MPNs, die aus den Locations entstanden sind, erstellt wird.

Die ATL-Regel überprüft somit jede Möglichkeit zwei Locations ($k = 2$) aus einer Menge von n Locations auszuwählen. Die Locations dürfen sich in einem Versuch nicht wiederholen und die Reihenfolge der Auswahl ist nicht relevant.

[3] Die Messwerte der Laufzeiten lassen sich sowohl mit einer Polynom- als auch mit einer Exponentialfunktion annähern. Für eine sichere Aussage über das Laufzeitverhalten der ATL-Transformation müsste der Algorithmus der ATL-Transformation genauer analysiert werden.

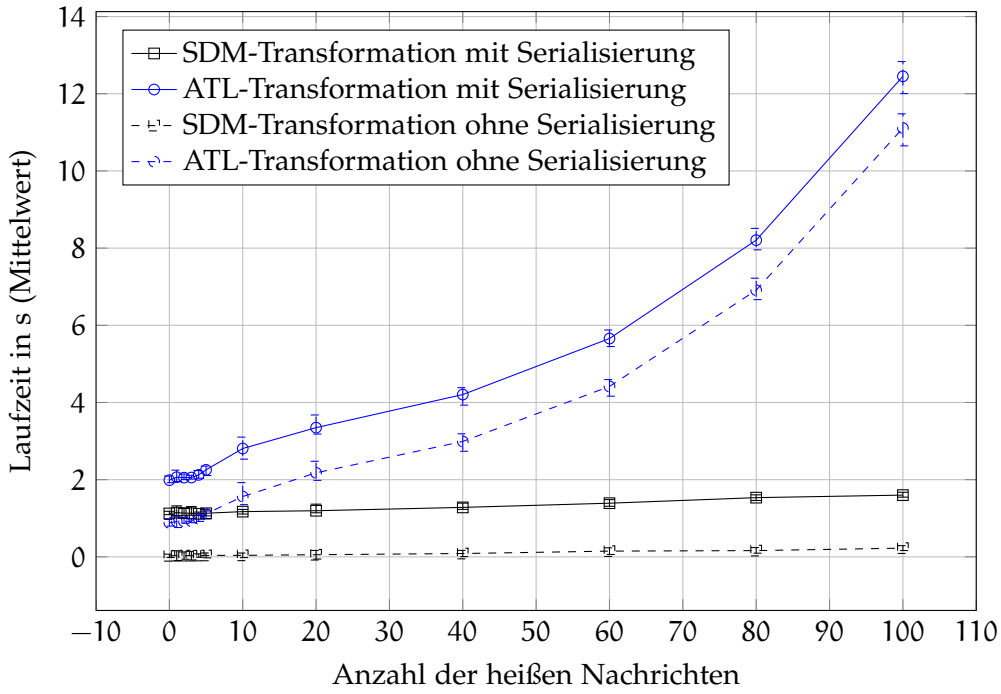


Abbildung 11.5: Laufzeiten der Transformationen für heiße Nachrichten

Somit ergibt sich die Anzahl aller möglichen Kombinationen (a) der Locations nach Formel 11.2.

$$a = \binom{n}{k} = \frac{n!}{k(n-k)!} \text{ mit } k = 2 \Rightarrow a = \binom{n}{2} = \frac{n!}{2(n-2)!} = \frac{n(n-1)}{2} \quad (11.2)$$

Bezogen auf die Anzahl der Nachrichten im betrachteten eLSC muss Formel 11.2 angepasst werden. Da jede Nachricht eine Sende- und eine Empfangslocation besitzt, werden durch jede zusätzliche Nachricht im eLSC zwei Locations ($n = 2 \cdot m$) hinzugefügt. Somit ergibt sich ein polynomielles Wachstum der Anzahl der Kombinationen (a_{ges}) nach Formel 11.3.

$$a_{ges} = \frac{n(n-1)}{2} \text{ mit } n = 2 \cdot m \Rightarrow a_{ges} = m \cdot (2m-1) \quad (11.3)$$

Abbildung 11.5 zeigt, dass sich dieses polynomielle Verhalten in der Laufzeit der ATL-Transformation widerspiegelt, wenn zu den anfänglich zwei heißen Nachrichten weitere Nachrichten hinzugefügt werden. Die SDM-Transformation skaliert für heiße Nachrichten deutlich besser, da die Bestimmung der Plätze im MPN, die für Übersprungstransitionen infrage kommen, wie in Abschnitt 9.2 vorgestellt, über ein Korrespondenzmodell erfolgt. Bei jeder zu übersetzenden Nachricht aus dem eLSC müssen von der entsprechenden Regel (`createStandardPlacesForAsyncMessage`) nur die im Korrespondenzmodell hinterlegten offenen Plätze betrachtet werden, woraus eine deutlich geringere Laufzeit resultiert.

Werden nun statt der heißen asynchronen Nachrichten kalte asynchrone Nachrichten, wie unten in Abbildung 11.3 gezeigt, hintereinander hinzugefügt, bedeutet dies für beide Transformationen einen polynomiellen Anstieg der erzeugten

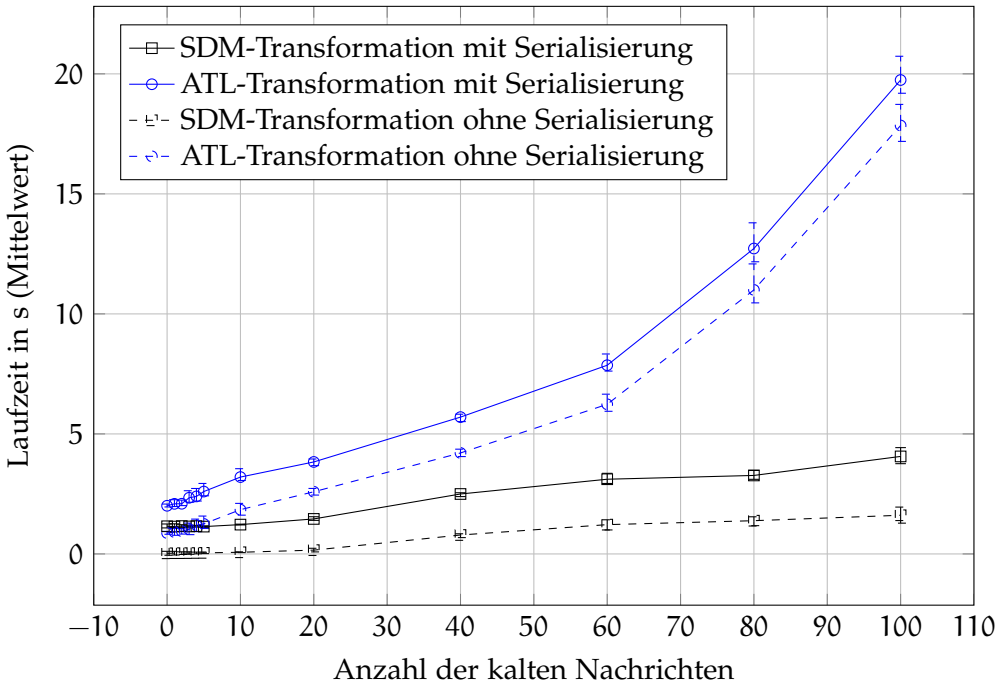


Abbildung 11.6: Laufzeiten der Transformationen für kalte Nachrichten

Transitionen im Ziel-MPN. Wie in Abschnitt 11.1 durch Formel 11.1 beschrieben, werden durch die Transformation zusätzlich zur Übersetzung heißer Nachrichten #t Übersprungstransitionen im Ziel-MPN erzeugt.

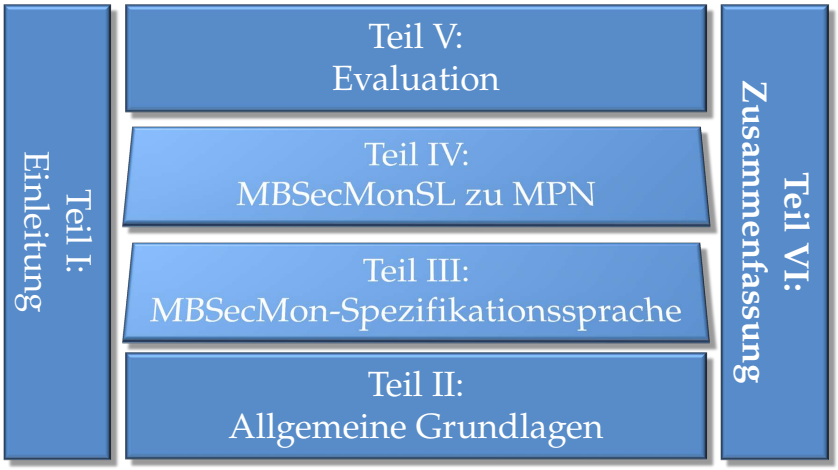
Die hieraus resultierenden Ergebnisse der Laufzeitmessungen sind in Abbildung 11.6 für eine wachsende Anzahl kalter Nachrichten im Quellmodell visualisiert. Sowohl für die ATL- als auch für die SDM-Transformation wächst die Laufzeit deutlich an.

Direkt aufeinanderfolgende kalte asynchrone Nachrichten stellen in der eLSC-zu-MPN-Transformation den Worst-Case für den Aufwand der Transformation dar. Die Laufzeitmessungen zeigen, dass die SDM-Transformation besser skaliert als die ATL-Variante. Der Grund hierfür liegt in der deklarativen Beschreibung der ATL-Regeln. Während in der SDM-Transformation explizit über die kalten Nachrichten Buch geführt wird, beschreibt die ATL-Regel nur global die Bedingung für zwei Locations im Quellmodell, unter der im Zielmodell Übersprungstransitionen erstellt werden sollen. Zusätzlich wird in eMoflon für die SDM-Transformation Java-Quelltext generiert, der durch den Java-Compiler optimiert in einer Java VM ausgeführt wird. Im Gegensatz hierzu verwendet ATL eine eigene Virtuelle Maschine (EMFVM), in der die Transformation durch einen Algorithmus gesteuert ausgeführt wird.

In der Praxis stellt die schlechte Skalierung der Laufzeit der Transformationen vieler asynchrone kalte Nachrichten keine Probleme dar, da in einer Signatur als eLSC nur wenige aufeinanderfolgende Elemente kalt sind. Sobald auf einer der Lebenslinien im eLSC eine heiße Location auftaucht, wird dieser polynomielle Anstieg an Übersprungstransitionen unterbrochen, da diese Location zwingend durch den Ablauf erreicht werden muss.

Teil VI

ZUSAMMENFASSUNG



12

ZUSAMMENFASSUNG

In dieser Arbeit wurde in Zusammenarbeit mit [Pat14] ein modellbasierter Entwicklungsprozess zur Generierung von Security/Safety-Monitoren (MBSecMon-Prozess) entwickelt. Der Fokus dieser Arbeit liegt hierbei auf dem ersten Teil des MBSecMon-Prozesses in Abbildung 12.1, der Monitorsignaturspezifikation und der Transformation in eine explizitere, besser für die Codegenerierung geeignete Repräsentation der Signaturen.

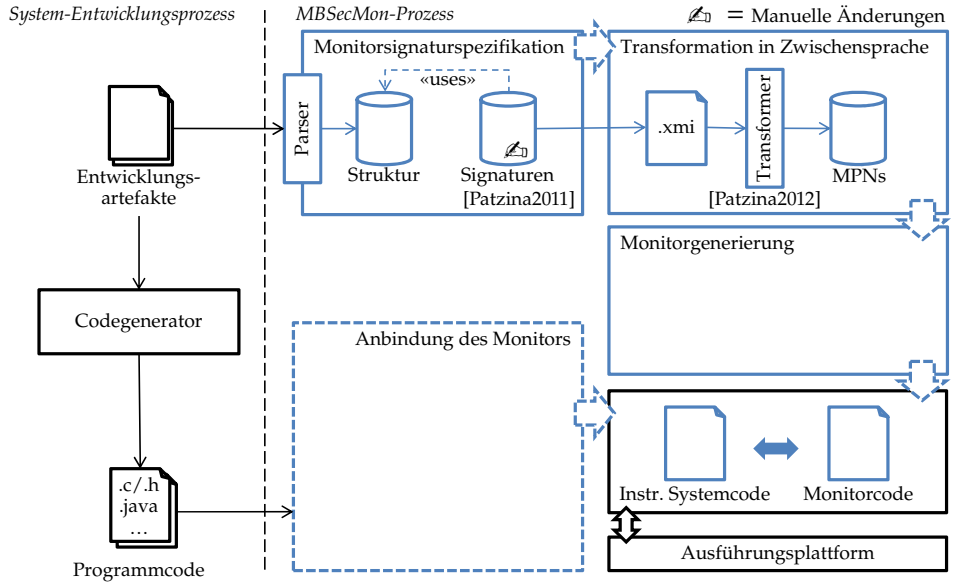


Abbildung 12.1: MBSecMon-Prozess: Übersetzung in die Zwischensprache

Hierfür wurde in Teil III eine neue auf LSCs und der Anwendungsfallmodellierung basierende verhaltensbeschreibende Signatursprache (MBSecMonSL) entwickelt, die aufgrund ihrer hohen Abstraktionsebene sowohl für den Systementwickler als auch für Nicht-Experten leicht verständlich ist. Diese aus der MUC-Sprache und der eLSC-Sprache bestehende MBSecMonSL unterstützt alle wichtigen Konzepte zur Modellierung komplexer nebenläufiger Kommunikationssignaturen. Durch die in Teil IV vorgestellte Transformation der MBSecMonSL-Spezifikationen in die formal definierte MPN-Sprache wird zum einen die Semantik der MBSecMonSL formalisiert und zum anderen die Komplexität der Generierung der Laufzeitmonitore für verschiedenste Zielplattformen und -sprachen reduziert.

12.1 Erreichte Ziele der Arbeit

An diesen MBSecMon-Prozess und an die in ihm eingesetzte Spezifikationssprache für Monitorsignaturen wurden in Abschnitt 1.3 sechs Anforderungen (① und ⑥) gestellt. Basierend auf diesen Anforderungen werden die in dieser Arbeit erreichten Ziele zusammengefasst und in Beziehung zur Fallstudie in Abschnitt 10.2 gesetzt. Für den MBSecMon-Prozess ist die ① *Eingliederung in einen bestehenden System-Entwicklungsprozess* und die ② *Einbeziehung bestehender Entwicklungsartefakte* in die Monitorsignaturspezifikation essentiell.

Anforderung ①: In dieser Arbeit wurde gezeigt, dass der MBSecMon-Prozess mit wenig Aufwand in bestehende Entwicklungsprozesse bzw. Vorgehensmodelle eingebunden werden kann. Durch den parallel zum bestehenden modellbasierten Systementwicklungsprozess angreifenden Ansatz lässt sich der MBSecMon-Prozess in beliebige Vorgehensmodelle eingliedern. Hierfür wird nur lesend auf bestehende Entwicklungsartefakte der Anforderungsanalyse und der Systemspezifikation zugegriffen und die Anbindung der generierten Monitore an das entwickelte System findet erst mit der Systemintegration statt.

Anforderung ②: Die Monitorsignaturen werden in der MBSecMonSL basierend auf den vorhandenen Entwicklungsartefakten spezifiziert. Hierfür wird eine Strukturspezifikation genutzt, die im Modellierungswerkzeug entweder als Komponentendiagramm oder als Klassendiagramm repräsentiert wird. Die Signaturen, die als eLSCs modelliert werden, verwenden diese hinterlegten Strukturinformationen um Lebenslinien auf Systemelemente abzubilden und Nachrichten auf Basis von Schnittstellen oder Methoden zu spezifizieren. Dies ermöglicht eine einfache Anbindung des generierten Monitors an das zu überwachende System.

Evaluation von ② und ③: Neben der theoretisch betrachteten Einbettung des MBSecMon-Prozesses in das V-Modell in Abschnitt 1.2, wurde der gesamte Prozess auf einen auf der AUTOSAR-Methodik basierenden Entwicklungsprozess angewendet. Hierfür fand die in Abschnitt 10.2 vorgestellte Integration in einen an die AUTOSAR-Methodik angelehnten Entwicklungsprozess statt, in dem Monitorsignaturen spezifiziert und aus ihnen Monitore generiert wurden. Diese Monitore wurden mithilfe einer Instrumentierung der zu überwachenden SW-Cs mit Ereignissen versorgt.

Hierfür musste in der MBSecMon-Toolchain ein Parser für ARXML an das MBSecMonSL-Add-in angebunden werden, der die intern verwendeten Komponentendiagramme erzeugt. So wurde durch das MBSecMon-Add-in für EA die ARXML-Datei des AUTOSAR-Entwicklungsprozess ausgelesen und Informationen wie zu überwachende Komponenten, deren Schnittstellen und Datentypen im MBSecMon-Add-in dem Modellierer der Signaturen zur Verfügung gestellt. Durch diese Anbindung konnten zusätzlich Namenskonventionen des System-Entwicklungsprozesses eingehalten werden, wodurch die automatische Generierung AUTOSAR-konformer SW-Cs und deren Anbindung an die zu überwachenden SW-Cs erreicht wurde.

Neben der Integration des MBSecMon-Prozesses in bestehende Systementwicklungsprozesse ist ein zentrales Konzept des MBSecMon-Prozesses die ③ *Unter-*

stützung verschiedenster Zielplattformen. Hieraus resultieren sowohl die Anforderung der universellen Einsetzbarkeit der verhaltensbeschreibenden Spezifikationssprache als auch die Anforderung nach der Unabhängigkeit der Codegenerierung von der gewählten Spezifikationssprache.

Anforderung ③: In dieser Arbeit wurde gezeigt, dass die MBSecMonSL zur kompakten Spezifikation von komplexen hoch nebenläufigen Kommunikationssequenzen geeignet ist. Da die direkte Codegenerierung für jede Zielsprache bzw. -plattform aus den ausdrucksstarken, kompakt modellierten Monitorsignaturen aufgrund der vielen impliziten Konzepte der eLSCs zu aufwendig ist, findet im MBSecMon-Prozess eine einmalige Übersetzung in die Zwischensprache MPN statt. Dieser in Teil IV vorgestellte Zwischenschritt der *Transformation in die Zwischensprache* MPN, formalisiert zum einen die MBSecMonSL und dient zum anderen als Vorbereitung zur Codegenerierung im MBSecMon-Prozess. Durch diese sehr umfangreiche Transformation wird eine Reduktion des Aufwands der Codegenerierung erreicht, die die einfache generische Generierung effizienter Monitore für verschiedenste Zielplattformen erlaubt.

Evaluation von ③: Neben der Generierung AUTOSAR-konformen C-Quelltextes in der Fallstudie in Abschnitt 10.2, zeigt die Arbeit von Lars Patzina [Pat14], dass durch die in dieser Arbeit vorgestellte Transformation in die Zwischensprache der MPN die einfache Codegenerierung für verschiedenste Zielplattformen unterstützt werden. Diese Zielsprachen- und Zielplattformunabhängigkeit wird zum einen durch die Spezifikation der verhaltensbeschreibenden Signaturen mit der MBSecMonSL erreicht und zum andern durch die ebenfalls unabhängige Übersetzung in die MPN-Repräsentation der Signaturen. Hierbei muss jedoch die Abbildbarkeit der gewählten Annotationssprache auf die Zielplattform gewährleistet sein.

Zur Monitorsignaturspezifikation wurde die in Teil III vorgestellte MBSecMon-Spezifikationssprache entwickelt. Diese erfüllt die drei in Abschnitt 1.3 aufgestellten Anforderungen: Eine ④ *Hohe Abstraktionsebene der Modellierung*, die ⑤ *Unterstützung aller wichtigen Konzepte zur Beschreibung komplexer Kommunikationssignaturen* und eine ⑥ *Hohe Verständlichkeit der Spezifikation* auch für Nicht-Experten.

Anforderung ④: Die Kommunikationssignaturen, die in der MBSecMonSL spezifiziert sind, liegen, wie in Abschnitt 3.3 gezeigt, auf einer *höheren Abstraktionsebene* als deren Spezifikation mit den häufig eingesetzten Expertensystemen oder Temporalen Logiken. Dies wird durch den Einsatz der Konzepte des szenariobasierten Designs in der MBSecMonSL erreicht. In der MUC-Sprache werden die Beziehungen zwischen den Signaturen explizit modelliert und Gegenmaßnahmen den Use- bzw. Misuse-Cases direkt zugeordnet. Die Signaturen werden als eLSCs auf Basis der Strukturbeschreibungen des Systementwicklungsprozesses und hierdurch auf derselben Abstraktionsebene wie die Modellierung des zu überwachenden Systems modelliert.

Anforderung ⑤: Die MBSecMonSL bestehend aus den eLSCs, einer zur verhaltensbeschreibenden Signaturmodellierung erweiterten Variante der LSCs, und der MUC-Sprache, die zur Strukturierung dieser Spezifikationen genutzt wird,

unterstützen alle wichtigen Konzepte zur Beschreibung komplexer Kommunikationssignaturen. Hierzu wurden in Abschnitt 4.3 die LSCs auf ihre Eigenschaften zur Modellierung verhaltensbeschreibender Signaturen untersucht und in Abschnitt 5.1 um die fehlenden Konzepte zu den eLSCs erweitert. Zudem vereinigt die MBSecMonSL die Vorteile der verschiedenen im Sicherheitsumfeld eingesetzten Formalismen zur Beschreibung funktionaler und nicht-funktionaler Anforderungen als Signaturen für Überwachungssysteme.

Anforderungen ⑥: Die grafische Notation der MBSecMonSL, die an die UML angelehnt ist, ist sowohl Systementwicklern bekannt als auch für Nicht-Experten verständlich, wie die häufig eingesetzte Use-Case-Modellierung des szenariobasierten Ansatzes in der Anforderungsanalyse von Entwicklungsprozessen zeigt. Zudem erhöht die *explizite Unterscheidung zwischen Use- und Misuse-Cases* der MUC-Sprache die Verständlichkeit der Spezifikationen und unterstützt in frühen Entwicklungsphasen die Berücksichtigung von verbotenen Verhalten bzw. von Angriffen auf das zu überwachende System. Durch das aus den LSCs in die eLSCs übernommene Referenzkonzept und die Repräsentation dieses Konzeptes auf der abstrakteren Ebene der MUC-Sprache wird eine *bessere Skalierbarkeit* der Spezifikationen erreicht. Des Weiteren werden Gegenmaßnahmen direkt in der Signaturbeschreibungssprache auf Ebene der MUC-Sprache durch die «mitigate»-Beziehung den Use- bzw. Misuse-Cases zugeordnet. Hierdurch werden die Zusammenhänge von positiven und negativen Signaturen und Gegenmaßnahmen auf einer hohen Abstraktionsebene verständlich und übersichtlich repräsentiert.

Evaluation von ④: Die Fallstudie zur Einbindung des MBSecMon-Entwicklungsprozesses in einen AUTOSAR-Entwicklungsprozess hat gezeigt, dass durch Toolunterstützung eine komfortable Modellierung der Monitorsignaturen möglich ist. Das MBSecMon-Add-in für EA liest die Strukturinformationen der in einer AUTOSAR-Entwicklungsumgebung modellierten SW-Cs und ihre Schnittstellen aus der persistierten Version (ARXML-Datei) aus und unterstützt den Modellierer bei der Spezifikation der Signaturen.

12.2 Zukünftige Arbeiten

In dieser Arbeit wurden zwar die Anforderungen, die in Abschnitt 1.3 aufgestellt wurden zum Großteil erfüllt, allerdings wurden einige erweiterte Konzepte aufgrund ihres Umfangs in dieser Arbeit nicht tiefergehend betrachtet. Diese Konzepte betreffen die Modellierung von Zeitbedingungen in eLSCs, eine erweiterte Variante der Modellierung von Gegenmaßnahmen, die Sicherstellung des Ergebnisses der Gegenmaßnahmen und die Erweiterung der Laufzeitüberwachung um die Überwachung von sich ändernden Systemkonfigurationen durch graphbasierte Mustersuche.

Erweiterte Zeitbehandlung: Die aktuelle Version der eLSC-Sprache unterstützt ausschließlich die in Abschnitt 4.1 vorgestellte Zeitbehandlung über eine Zeitvariable. So können Zeitpunkte durch Zuweisungen auf Variablen erfasst und durch Bedingungen an einem späteren Zeitpunkt mit der aktuellen Zeit verglichen werden. Mit diesem einfachen Konzept lassen sich vertikale Verzögerungen,

Nachrichtenverzögerungen und Timer modellieren. Diese werden in die MPN-Repräsentation in Aktionen und Bedingungen an Transitionen übersetzt.

Eine erweiterte Zeitmodellierung über diese einfachen Zeitkonstrukte hinaus sollte in Zukunft untersucht werden. Durch die Definition weiterer syntaktischer Elemente oder die Definition einer Annotationssprache für Zeitbedingungen, wie in dem UML2- und dem MSC-2000-Standard enthalten, kann evtl. die Modellierung von Zeitbedingungen erleichtert werden.

Erweiterte Modellierung von Gegenmaßnahmen: In der aktuellen Version der MBSecMonSL werden Gegenmaßnahmen, die als Existential eLSCs modelliert sind, durch *mitigate*-Beziehungen anderen Universal eLSC-Signaturen zugeordnet. Diese Universal eLSCs bilden die Vorbedingung für die Ausführung der spezifizierten Gegenmaßnahme.

Zur weiteren Steigerung der Modellierungsmöglichkeiten ist der Einsatz eines Universal eLSCs zur Spezifikation der Gegenmaßnahmen denkbar. Das Prechart kann eingesetzt werden, um zusätzliche Vorbedingungen für die Gültigkeit der Gegenmaßnahmen zu beschreiben und somit ebenfalls eine Auswahl an Gegenmaßnahmen spezifizieren zu können. Wird das Prechart des Universal eLSCs nicht benötigt, da die Vorbedingung für die Gegenmaßnahme durch die Überwachung des Basis-Anwendungsfalls gegeben ist, kann das Prechart einfach weggelassen werden.

Diese Verwendung eines Universal eLSCs zur Spezifikation von Gegenmaßnahmen stellt eine Erweiterung der MBSecMonSL dar. Allerdings muss deren sinnvolle Einsetzbarkeit zur Modellierung und zur Generierung effektiver Monitore in zukünftigen Arbeiten genauer evaluiert werden. Da im Prechart der Gegenmaßnahme auch Sequenzen von Nachrichten modelliert werden können, die zeitlich vor der Aktivierung der Gegenmaßnahme liegen, müssen diese Vorbedingungen dauerhaft während der Laufzeit des Monitors überwacht werden. Dies führt zu einem erhöhten Laufzeitoverhead der generierten Monitore und können so ihre Einsetzbarkeit einschränken.

Ergebnis der Gegenmaßnahme sicherstellen: Sind die Gegenmaßnahmen spezifiziert und werden diese im generierten Laufzeitmonitor ausgelöst, kann der Zustand des Systems durch Aufruf von Methoden oder Versenden von Nachrichten beeinflusst werden. Hierbei ist allerdings nicht sichergestellt, ob das System den kritischen Zustand durch die Ausführung der Gegenmaßnahme verlassen hat. Zusätzlich muss der Laufzeitmonitor auf diese Veränderungen des Systemzustands reagieren, laufende Überwachungen zurücksetzen und somit sicherstellen, dass der Zustand des Laufzeitmonitors mit dem System übereinstimmt. Diese Rekonfiguration des Systems und des Monitors zur Laufzeit stellt einen sehr umfangreichen Forschungsbereich dar, der weder in dieser noch in [Pat14] tiefergehend betrachtet wurde.

Neben diesen zukünftigen Erweiterungen der MBSecMonSL und der generierten Laufzeitmonitore zur Kommunikationsprotokollüberwachung kann der Entwicklungsprozess auch auf das Anwendungsgebiet der Systemkonfigurationsüberwachung erweitert werden.

Überwachung von sich ändernden Systemkonfigurationen: In dieser Arbeit wurden die Mustersuche auf Strukturen und Graphtransformationen nur im Rahmen der Übersetzung der MBSecMonSL in die MPN-Repräsentation in Teil IV eingesetzt. Diese beiden Konzepte können jedoch auch zur Überwachung von Systemkonfigurationen und Datenstrukturen eingesetzt werden [Dec13].

Durch diese Konzepte könnte der Laufzeitmonitor mit zusätzlichen Ereignissen über Veränderungen des zu überwachenden Systems oder dessen Daten informiert werden. Diese Ereignisse werden in den eLSCs als Nachrichten modelliert und so zusammen mit der Kommunikation zwischen den Komponenten überwacht. Wird eine verbotene Konfiguration für den aktuellen Zustand des Systems entdeckt, können als Gegenmaßnahmen Graphtransformationen auf den Strukturen ausgeführt werden.

Binden von Lebenslinien durch Mustersuche: In der in dieser Arbeit vorgestellten Version des MBSecMon-Prozesses, werden die Kommunikationspartner statisch bei der Instanziierung der Monitorsignaturen an die Lebenslinien in den eLSCs gebunden. Eine Zerstörung einer überwachten Instanz führt zu einem Fehlschlagen der Überwachung.

Eine weitere Erweiterungsmöglichkeit bildet der Einsatz der Mustersuche in den Strukturen des zu überwachenden Systems um *dynamisches Binden während der Laufzeit* zu realisieren. Hierzu könnten die symbolischen Instanzen der LSC in den eLSCs übernommen werden. Anhand von definierten Mustern werden während der Laufzeit Instanzen gebunden und die Überwachung sauber beendet, wenn Instanzen nicht mehr vorhanden sind.

Zudem lässt sich das Einsatzgebiet der Laufzeitmonitore mithilfe der Mustersuche ausdehnen. Wird *ein Muster von Kommunikationspartnern gefunden*, definiert dies die Instanziierung einer Monitorsignaturinstanz und somit den Beginn der Überwachung. Die durch die Mustersuche identifizierten Systemelemente werden an die entsprechenden Lebenslinien des eLSCs gebunden. Wird ein vorher *bestehendes Muster an gebundenen Systemelementen zerstört*, definiert dies das Beenden der Überwachung einer Monitorsignaturinstanz. Hierdurch wird die Überwachung kontrolliert beendet. Eine Teilsignatur (Subchart) kann durch eine eLSC-Bedingung bedingt werden. Durch die Einbindung eines Musters in eine Bedingung kann so die weitere *Überwachung einer Monitorsignatur abhängig von Konfigurationen oder Strukturen* durchgeführt werden. Findet die Mustersuche auf der Konfiguration der Kommunikationspartner statt und die Kanten des Graphen bilden die Kommunikationsbeziehungen, kann die Mustersuche zusätzlich zur *Identifikation von Kommunikationsverbindungen (Kommunikationskanälen)* genutzt werden.

Alle diese Erweiterungen der MBSecMonSL und somit der Laufzeitmonitore stellen ein interessantes Forschungsgebiet dar, da diese zu einer Steigerung des Speicherverbrauchs durch den Monitor führen und zusätzliche Rechenzeit benötigen. Hierbei ist insbesondere die Einsetzbarkeit dieser Konzepte auf ressourcenbeschränkten eingebetteten Systemen zu betrachten, die bei der Entwicklung des MBSecMon-Prozesses im Fokus stand.

[ABKP11] Amstel, Marcel van; Bosems, Steven; Kurtev, Ivan; Pires, Luís F.: Performance in Model Transformations: Experiments with ATL and QVT. Version: 2011. http://dx.doi.org/10.1007/978-3-642-21732-6_14. In: Cabot, Jordi; Visser, Eelco (Hrsg.): *Theory and Practice of Model Transformations* Bd. 6707. DOI. – 10.1007/978–3–642–21732– 6_14, S. 198–212 (Zitiert auf Seite 175.)

[Ale02] Alexander, Ian F.: Initial Industrial Experience of Misuse Cases in Trade-Off Analysis. In: *10th Anniversary IEEE Joint International Conference on Requirements Engineering (RE 2002)*. Washington, DC, USA : IEEE Computer Society, 2002. – ISBN 0–7695–1465–0, S. 61–70 (Zitiert auf Seite 7, 48, 53 und 54.)

[Ale03a] Alexander, Ian F.: Misuse Cases Help to Elicit Non-Functional Requirements. In: *Computing & Control Engineering Journal* 14 (2003), Februar, Nr. 1, S. 40–45. – ISSN 0956–3385 (Zitiert auf Seite 7, 48, 54 und 73.)

[Ale03b] Alexander, Ian F.: Misuse Cases: Use Cases with Hostile Intent. In: *IEEE Software* 20 (2003), Jan.–Feb., Nr. 1, S. 58–66. http://dx.doi.org/10.1109/MS.2003.1159030. – DOI 10.1109/MS.2003.1159030 (Zitiert auf Seite 48 und 53.)

[ALPS11] Anjorin, Anthony; Lauder, Marius; Patzina, Sven; Schürr, Andy: eMoflon: Leveraging EMF and Professional CASE Tools. In: *3. Workshop Methodische Entwicklung von Modellierungswerkzeugen (MEMWe 2011)*. Bonn : Gesellschaft für Informatik, 2011 (Lecture Notes in Informatics) (Zitiert auf Seite 180.)

[AUT11a] AUTOSAR: *Specification of Operating System (R4.0 Rev 3)*. Online. Version: 2011. http://www.autosar.org/download/R4.0/AUTOSAR_SWS_OS.pdf (Zitiert auf Seite 196.)

[AUT11b] AUTOSAR: *Specification of Watchdog Manager (R4.0 Rev 3)*. Online. Version: 2011. http://www.autosar.org/download/R4.0/AUTOSAR_SWS_WatchdogManager.pdf (Zitiert auf Seite 196.)

[AVS12] Anjorin, Anthony; Varró, Gergely; Schürr, Andy: Complex Attribute Manipulation in TGGs with Constraint-Based Programming Techniques. In: Margaria, Tiziana; Padberg, Julia; Taentzer, Gabriele; Hermann, F.; Voigtländer, J. (Hrsg.): *Bidirectional Transformations (BX 2012)* Bd. 49, 2012 (Electronic Communications of the EASST) (Zitiert auf Seite 180.)

[BDHR11] Becker, Steffen; Detten, Markus von; Heinzemann, Christian; Rieke, Jan: Structuring Complex Story Diagrams by Polymorphic Calls / Software Engineering Group, Heinz Nixdorf Institute, Universität Paderborn. 2011 (tr-ri-11-323). – Forschungsbericht (Zitiert auf Seite 135.)

[BDL06] Basin, David A.; Doser, Jürgen; Lodderstedt, Torsten: Model Driven Security: From UML Models to Access Control Infrastructures. In: *ACM Transactions on Software Engineering and Methodology* 15 (2006), Nr. 1, S. 39–91. http://dx.doi.org/10.1145/1125808.1125810. – DOI 10.1145/1125808.1125810. – ISSN 1049–331X (Zitiert auf Seite 11 und 68.)

[BG01] Bunker, Annette; Gopalakrishnan, Ganesh: Using Live Sequence Charts for Hardware Protocol Specification and Compliance Verification. In: *Sixth IEEE International High-Level Design Validation and Test Workshop (HLDVT 2001)*, IEEE, 2001. – ISBN 0–7695–1411–1, S. 95–100 (Zitiert auf Seite 11, 71 und 72.)

[BGS05] Bunker, Annette; Gopalakrishnan, Ganesh; Slind, Konrad: Live Sequence Charts Applied to Hardware Requirements Specification and Verification. In: *International Journal on Software Tools for Technology Transfer* 7 (2005), Nr. 4, S. 341–350. http://dx.doi.org/10.1007/s10009-004-0145-x. – DOI 10.1007/s10009–004–0145–x (Zitiert auf Seite 11 und 71.)

[BJ08] Bauer, Andreas; Jürjens, Jan: Security Protocols, Properties, and their Monitoring. In: *Proceedings of the 4th International Workshop on Software Engineering for Secure Systems (SecSE 2008)* ACM, 2008. – ISBN 978–1–60558–042–5, S. 33–40 (Zitiert auf Seite 11, 68 und 140.)

[BJ10] Bauer, Andreas; Jürjens, Jan: Runtime Verification of Cryptographic Protocols. In: *Computers & Security* 29 (2010), Nr. 3, S. 315 – 330. http://dx.doi.org/10.1016/j.cose.2009.09.003. – DOI 10.1016/j.cose.2009.09.003. – ISSN 0167–4048 (Zitiert auf Seite 6, 11, 68 und 140.)

[BJY11] Bauer, Andreas; Jürjens, Jan; Yu, Yijun: Run-Time Security Traceability for Evolving Systems. In: *The Computer Journal* 54 (2011), Nr. 1, S. 58–87 (Zitiert auf Seite 11, 68 und 140.)

[BLS06] Bauer, Andreas; Leucker, Martin; Schallhart, Christian: Monitoring of Real-Time Properties. Version: 2006. http://dx.doi.org/10.1007/11944836_25. In: Arun-Kumar, S.; Garg, Naveen (Hrsg.): *Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2006)* Bd. 4337. DOI. – 10.1007/1194483 6_25. – ISBN 978–3–540–49994–7, S. 260–272 (Zitiert auf Seite 140.)

[BLS07] Bauer, Andreas; Leucker, Martin; Schallhart, Christian: Runtime Verification for LTL and TLTL / Institut für Informatik, Technische

Universität München. 2007 (TUM-I0724). – Forschungsbericht (Zitiert auf Seite 140.)

[BMT11] Broy, Manfred; Mühleck, Klaus H.; Taubner, Dirk: Informatik in der Automobilindustrie. In: *Informatik Spektrum* 34 (2011), Nr. 1, S. 1–5. http://dx.doi.org/10.1007/s00287-010-0508-5. – DOI 10.1007/s00287–010–0508–5 (Zitiert auf Seite 5.)

[BS05] Bontemps, Yves; Schobbens, Pierre-Yves: The Complexity of Live Sequence Charts. Version: 2005. http://dx.doi.org/10.1007/978-3-540-31982-5_23. In: Sassone, Vladimiro (Hrsg.): *Foundations of Software Science and Computational Structures (FOSSACS 2005)* Bd. 3441. DOI. – 10.1007/978–3–540–31982– 5_23. – ISBN 978–3–540–25388–4, S. 364–378 (Zitiert auf Seite 86.)

[BTZ05] Bartelt, Christian; Ternité, Thomas; Zieger, Matthias: Modellbasierte Entwicklung mit dem V-Modell XT. In: *OBJEKTspektrum* 5 (2005), S. 53–64 (Zitiert auf Seite 7.)

[Bun12] Bundesstelle für Informationstechnik (BIT): *V-Modell XT*. Online. http://www.cio.bund.de/DE/Architekturen-und-Standards/V-Modell-XT/vmodell_xt_node.html. Version: Mai 2012, Abruf: 04.01.2014 (Zitiert auf Seite 7.)

[CF08] Cocchiarella, Nino B.; Freund, Max A.: *Modal Logic: An Introduction to its Syntax and Semantics*. Oxford University Press, 2008 (Zitiert auf Seite 70.)

[CH06] Czarnecki, Krzysztof; Helsen, Simon: Feature-Based Survey of Model Transformation Approaches. In: *IBM Systems Journal* 45 (2006), Nr. 3, S. 621–646. http://dx.doi.org/10.1147/sj.453.0621. – DOI 10.1147/sj.453.0621 (Zitiert auf Seite 175.)

[CMK+11] Checkoway, Stephen; McCoy, Damon; Kantor, Brian; Anderson, Danny; Shacham, Hovav; Savage, Stefan; Koscher, Karl; Czeskis, Alexei; Roesner, Franziska; Kohno, Tadayoshi: Comprehensive Experimental Analyses of Automotive Attack Surfaces. In: *Proceedings of the 20th USENIX Conference on Security (SEC 2011)* USENIX Association, 2011, S. 77–92 (Zitiert auf Seite 4 und 5.)

[Coc01] Cockburn, Alistair: *Writing Effective Use Cases*. Bd. 1. Addison-Wesley Reading, 2001 (Zitiert auf Seite 48.)

[Dec13] Deckwerth, Frederik: Generating Monitors for Usage Control. In: Wagner, Stefan; Lichter, Horst (Hrsg.): *Software Engineering 2013 - Workshopband* Bd. 215, Gesellschaft für Informatik, 2013 (LNI), S. 577–582 (Zitiert auf Seite 222.)

[DH01] Damm, Werner; Harel, David: LSCs: Breathing Life into Message Sequence Charts. In: *Formal Methods in System Design* 19 (2001), Nr. 1, S. 45–80 (Zitiert auf Seite 9, 71, 75 und 86.)

[DR98] Desel, Jörg; Reisig, Wolfgang: Place/Transition Petri Nets. Version: 1998. http://dx.doi.org/10.1007/3-540-65306-6_15. In: Reisig, Wolfgang; Rozenberg, Grzegorz (Hrsg.): *Lectures on Petri Nets I: Basic Models* Bd. 1491. DOI. – 10.1007/3–540–65306– 6_15. – ISBN 978–3–540–65306–6, S. 122–173 (Zitiert auf Seite 35.)

[Dru00] Drusinsky, Doron: The Temporal Rover and the ATG Rover. In: Havelund, Klaus; Penix, John; Visser, Willem (Hrsg.): *Proceedings of the 7th International SPIN Workshop on SPIN Model Checking and Software Verification* Bd. 1885, Springer, 2000 (LNCS). – ISBN 978–3–540–41030–0, S. 323–330 (Zitiert auf Seite 67.)

[DTW07] Damm, Werner; Toben, Tobe; Westphal, Bernd: On the Expressive Power of Live Sequence Charts. Version: 2007. http://dx.doi.org/10.1007/978-3-540-71322-7_11. In: Reps, Thomas; Sagiv, Mooly; Bauer, Jörg (Hrsg.): *Program Analysis and Compilation, Theory and Practice* Bd. 4444. DOI. – 10.1007/978–3–540–71322– 7_11. – ISBN 978–3–540–71315–9, S. 225–246 (Zitiert auf Seite 139.)

[EH86] Emerson, E. A.; Halpern, Joseph Y.: "Sometimes" and "Not Never" Revisited: On Branching Versus Linear Time Temporal Logic. In: *Journal of the ACM* 33 (1986), Januar, Nr. 1, S. 151–178. http://dx.doi.org/10.1145/4904.4999. – DOI 10.1145/4904.4999. – ISSN 0004–5411 (Zitiert auf Seite 139.)

[FGP09] Frankowiak, Marcos R.; Grosvenor, Roger I.; Prickett, Paul W.: Microcontroller-Based Process Monitoring Using Petri-Nets. In: *EURASIP Journal on Embedded Systems* 2009 (2009), S. 1–12. http://dx.doi.org/10.1155/2009/282708. – DOI 10.1155/2009/282708 (Zitiert auf Seite 69.)

[FM12] Flegel, Ulrich; Meier, Michael: Modeling and Describing Misuse Scenarios Using Signature-Nets and Event Description Language. In: *it - Information Technology* 54 (2012), März, Nr. 2, S. 71–81. http://dx.doi.org/10.1524/itit.2012.0666. – DOI 10.1524/itit.2012.0666 (Zitiert auf Seite 11 und 69.)

[FNTZ00] Fischer, Thorsten; Niere, Jörg; Torunski, Lars; Zündorf, Albert: Story Diagrams: A New Graph Rewrite Language Based on the Unified Modeling Language and Java. In: Ehrig, Hartmut; Engels, Gregor; Kreowski, Hans-Jörg; Rozenberg, Grzegorz (Hrsg.): *6th International Workshop on Ory and Application of Graph Transformations (TAGT 1998)* Bd. 1764, Springer, 2000 (LNCS). – ISBN 3–540–67203–6, S. 296–309 (Zitiert auf Seite 180 und 194.)

[Fow10] Fowler, Martin: *Domain-specific languages*. Pearson Education, 2010. – 640 S. – ISBN 978–0–321–71294–3 (Zitiert auf Seite 205.)

[GR98] GIARRATANO, Joseph C.; RILEY, Gary D.: *Expert Systems: Principles and Programming, Third Edition*. 3. Course Technology, 1998. – ISBN 978–0–534–95053–8 (Zitiert auf Seite 66.)

[GR09] GROLL, André; RULAND, Christoph: Secure and Authentic Communication on Existing In-Vehicle Networks. In: *IEEE Intelligent Vehicles Symposium (IV 2009)*, IEEE Computer Society, Juni 2009, S. 1093–1097 (Zitiert auf Seite 5.)

[Gre11] GREENYER, Joel: *Scenario-Based Design of Mechatronic Systems*, Universität Paderborn, Diss., 2011. http://nbn-resolving.de/urn/resolver.pl?urn=urn:nbn:de:hbz:466:2-7690. – URN urn:nbn:de:hbz:466:2–7690 (Zitiert auf Seite 72.)

[GRS01] GRABOWSKI, Jens; RUDOLPH, Ekkart; SCHMITT, Michael: Die Spezifikationssprachen MSC und SDL - Teil 1: Message Sequence Chart (MSC). In: *at - Automatisierungstechnik* 49 (2001), Dezember, Nr. 12, S. A19–A22. http://dx.doi.org/10.1524/auto.2001.49.12.a19. – DOI 10.1524/auto.2001.49.12.a19 (Zitiert auf Seite 55.)

[GSJ08] GORP, Pieter V.; SCHIPPERS, Hans; JANSSENS, Dirk: Copying Subgraphs Within Model Repositories. In: *Electronic Notes in Theoretical Computer Science* 211 (2008), S. 133–145. http://dx.doi.org/10.1016/j.entcs.2008.04.036. – DOI 10.1016/j.entcs.2008.04.036 (Zitiert auf Seite 188.)

[Har87] HAREL, David: Statecharts: A Visual Formalism for Complex Systems. In: *Science of Computer Programming* 8 (1987), Nr. 3, S. 231–274. http://dx.doi.org/10.1016/0167-6423(87)90035-9. – DOI 10.1016/0167–6423(87)90035–9 (Zitiert auf Seite 34.)

[Hau05] HAUGEN, Øystein: Comparing UML 2.0 Interactions and MSC-2000. In: *System Analysis and Modeling (SAM 2004)* Bd. 3319 Springer, 2005, S. 65–79 (Zitiert auf Seite 63 und 64.)

[HHRS05] HAUGEN, Østein; HUSA, Knut; RUNDE, Ragnhild; STØLEN, Ketil: STAIRS Towards Formal Design with Sequence Diagrams. In: *Software and Systems Modeling* 4 (2005), Nr. 4, S. 355–367. http://dx.doi.org/10.1007/s10270-005-0087-0. – DOI 10.1007/s10270–005–0087–0 (Zitiert auf Seite 70.)

[HK02] HAREL, David; KUGLER, Hillel: Synthesizing State-Based Object Systems from LSC Specifications. In: *International Journal of Foundations of Computer Science* 13 (2002), Nr. 1, S. 5–51. DOI. – 10.1007/3–540–44674–5_1 (Zitiert auf Seite 75.)

[HM02] HAREL, David; MARELLY, Rami: Playing with Time: On the Specification and Execution of Time-Enriched LSCs. In: *10th International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS 2002)*, IEEE Computer Society, 2002. – ISBN 0–7695–1840–0, S. 193–202 (Zitiert auf Seite 81 und 94.)

[HM03] Harel, David; Marelly, Rami: *Come, Let's Play: Scenario-Based Programming Using LSCs and the Play-Engine*. Springer, 2003. – 382 S. – ISBN 978–3–540–00787–6 (Zitiert auf Seite 10, 12, 75, 78, 82, 84 und 101.)

[HM08] Harel, David; Maoz, Shahar: Assert and Negate Revisited: Modal Semantics for UML Sequence Diagrams. In: *Software and Systems Modeling* 7 (2008), Mai, Nr. 2, S. 237–252. http://dx.doi.org/10.1007/s10270-007-0054-z. – DOI 10.1007/s10270–007–0054–z (Zitiert auf Seite 55, 64, 65 und 72.)

[Hoa81] Hoare, Charles A. R.: The Emperor's Old Clothes: The 1980 ACM Turing Award Lecture. In: *Communications of the ACM* 24 (1981), Nr. 4, S. 75–83 (Zitiert auf Seite iii.)

[HR01] Havelund, Klaus; Rosu, Grigore: Monitoring Java Programs with Java PathExplorer. In: *Electronic Notes in Theoretical Computer Science* 55 (2001), Nr. 2, S. 200–217. http://dx.doi.org/10.1016/S1571-0661(04)00253-1. – DOI 10.1016/S1571–0661(04)00253–1 (Zitiert auf Seite 67.)

[HWG11] Hildebrandt, Stephan; Wätzoldt, Sebastian; Giese, Holger: Executing Graph Transformations with the MDELab Story Diagram Interpreter. In: *Transformation Tool Contest (TTC 2011)*, 2011 (Zitiert auf Seite 184.)

[HZ06] Hussein, Mohammed; Zulkernine, Mohammad: UMLintr: A UML Profile for Specifying Intrusions. In: *Proc. 13th Annual IEEE International Symposium and Workshop on Engineering of Computer Based Systems (ECBS 2006)*, 2006, S. 279–288 (Zitiert auf Seite 68.)

[IT92] ITU-T: *Recommendation Z.120: Message Sequence Chart (MSC)*. Geneva, 1992 (SERIES Z: LANGUAGES AND GENERAL SOFTWARE ASPECTS FOR TELECOMMUNICATION SYSTEMS) (Zitiert auf Seite 54 und 55.)

[IT96] ITU-T: *Recommendation Z.120: Message Sequence Chart (MSC)*. Geneva, 1996 (SERIES Z: LANGUAGES AND GENERAL SOFTWARE ASPECTS FOR TELECOMMUNICATION SYSTEMS) (Zitiert auf Seite 75.)

[IT00] ITU-T: *Recommendation Z.120: Message Sequence Chart (MSC)*. Geneva, 2000 (SERIES Z: LANGUAGES AND GENERAL SOFTWARE ASPECTS FOR TELECOMMUNICATION SYSTEMS) (Zitiert auf Seite 59.)

[JABK08] Jouault, Frédéric; Allilaire, Freddy; Bézivin, Jean; Kurtev, Ivan: ATL: A Model Transformation Tool. In: *Science of Computer Programming* 72 (2008), Nr. 1-2, S. 31–39. http://dx.doi.org/10.1016/j.scico.2007.08.002. – DOI 10.1016/j.scico.2007.08.002 (Zitiert auf Seite 132, 180 und 182.)

[JCJ92] JACOBSON, Ivar; CHRISTERSON, Magnus; JONSSON, Patrik: *Object-Oriented Software Engineering: A Use Case Driven Approach*. Addison-Wesley Longman, 1992 (ACM Press). – 528 S. – ISBN 978–0–201.54435–0 (Zitiert auf Seite 6 und 48.)

[JK06a] JOUAULT, Frédéric; KURTEV, Ivan: On the Architectural Alignment of ATL and QVT. In: *ACM Symposium on Applied Computing (SAC 2006)*, ACM, 2006 (SAC). – ISBN 1–59593–108–2, S. 1188–1195 (Zitiert auf Seite 180.)

[JK06b] JOUAULT, Frédéric; KURTEV, Ivan: Transforming Models with ATL. Version: 2006. http://dx.doi.org/10.1007/11663430_14. In: BRUEL, Jean-Michel (Hrsg.): *Satellite Events at the MoDELS 2005 Conference, MoDELS 2005 International Workshops* Bd. 3844. DOI. – 10.1007/1166343 0_14. – ISBN 3–540–31780–5, S. 128–138 (Zitiert auf Seite 132 und 189.)

[Jür02] JÜRJENS, Jan: UMLsec: Extending UML for Secure Systems Development. In: JÉZÉQUEL, Jean-Marc; HUSSMANN, Heinrich; COOK, Stephen (Hrsg.): *«UML» 2002 – The Unified Modeling Language (UML 2002)* Bd. 2460, Springer, 2002 (LNCS). – ISBN 3–540–44254–5, S. 412–425 (Zitiert auf Seite 67.)

[Jür05] JÜRJENS, Jan: *Secure Systems Development with UML*. Springer, 2005. – 316 S. – ISBN 978–3–540–00701–2 (Zitiert auf Seite 11, 67 und 140.)

[Jür08] JÜRJENS, Jan: Model-Based Run-Time Checking of Security Permissions Using Guarded Objects. Version: 2008. http://dx.doi.org/10.1007/978-3-540-89247-2_3. In: LEUCKER, Martin (Hrsg.): *8th International Workshop on Runtime Verification (RV 2008)* Bd. 5289. DOI. – 10.1007/978–3–540–89247– 2_3. – ISBN 978–3–540–89246–5, S. 36–50 (Zitiert auf Seite 68.)

[JW07] JAYARAMAN, Praveen K.; WHITTLE, Jonathan: UCSIM: A Tool for Simulating Use Case Scenarios. In: *29th International Conference on Software Engineering (ICSE 2007)*, IEEE Computer Society, 2007, S. 43–44 (Zitiert auf Seite 73 und 140.)

[KCR+10] KOSCHER, Karl; CZESKIS, Alexei; ROESNER, Franziska; PATEL, Franziska; KOHNO, Tadayoshi; CHECKOWAY, Stephen; MCCOY, Damon; KANTOR, Brian; ANDERSON, Danny; SHACHAM, Hovav; SAVAGE, Stefan: Experimental Security Analysis of a Modern Automobile. In: *IEEE Symposium on Security and Privacy (SP 2010)*, IEEE Computer Society, 2010, S. 447–462 (Zitiert auf Seite 89.)

[KF09] KINDEL, Olaf; FRIEDRICH, Mario: *Softwareentwicklung mit AUTOSAR: Grundlagen, Engineering, Management in der Praxis*. dpunkt, 2009. – ISBN 978–3–898–64563–8 (Zitiert auf Seite 5 und 196.)

[KG03] Kulak, Daryl; Guiney, Eamonn: *Use Cases: Requirements in Context*. Addison-Wesley Professional, 2003. – ISBN 978–0–321–15498–9 (Zitiert auf Seite 48.)

[KHP+05] Kugler, Hillel; Harel, David; Pnueli, Amir; Lu, Yuan; Bontemps, Yves: Temporal Logic for Scenario-Based Specifications. In: Halbwachs, Nicolas; Zuck, Lenore D. (Hrsg.): *Tools and Algorithms for the Construction and Analysis of Systems* Bd. 3440, Springer, 2005 (LNCS), S. 445–460 (Zitiert auf Seite 67 und 139.)

[KM08] Kumar, Rahul; Mercer, Eric G.: Improving Live Sequence Chart to Automata Transformation for Verification. In: *ECEASST Graph Transformation and Visual Modeling Techniques (GT-VMT 2008)* 10 (2008). http://eceasst.cs.tu-berlin.de/index.php/eceasst/issue/view/19 (Zitiert auf Seite 139.)

[KM09] Kumar, Rahul; Mercer, Eric G.: Verifying Communication Protocols Using Live Sequence Chart Specifications. In: *Electronic Notes in Theoretical Computer Science* 250 (2009), Nr. 2, S. 33–48. http://dx.doi.org/10.1016/j.entcs.2009.08.016. – DOI 10.1016/j.entcs.2009.08.016 (Zitiert auf Seite 71, 139 und 172.)

[KMB09] Kumar, Rahul; Mercer, Eric G.; Bunker, Annette: Improving Translation of Live Sequence Charts to Temporal Logic. In: *Electronic Notes in Theoretical Computer Science* 250 (2009), Nr. 1, S. 137–152. http://dx.doi.org/10.1016/j.entcs.2009.08.010. – DOI 10.1016/j.entcs.2009.08.010 (Zitiert auf Seite 11 und 139.)

[KPP08] Kolovos, Dimitrios S.; Paige, Richard F.; Polack, Fiona A.: The Epsilon Transformation Language. In: Vallecillo, Antonio; Gray, Jeff; Pierantonio, Alfonso (Hrsg.): *First International Conference on the Ory and Practice of Model Transformations (ICMT 2008)* Bd. 5063, Springer, 2008 (LNCS). – ISBN 978–3–540–69926–2, S. 46–60 (Zitiert auf Seite 180.)

[KRS09] Klar, Felix; Rose, Sebastian; Schürr, Andy: TiE - A Tool Integration Environment. In: *Proceedings of the 5th ECMDA Traceability Workshop* Bd. WP09-09, 2009 (CTIT Workshop Proceedings), S. 39–48 (Zitiert auf Seite 180.)

[Kru04] Kruchten, Philippe: *The Rational Unified Process: An Introduction*. 3rd. Addison-Wesley Professional, 2004. – 336 S. – ISBN 978–0–321–19770–2 (Zitiert auf Seite 48.)

[Kum95] Kumar, Sandeep: *Classification and Detection of Computer Intrusions*, Purdue University, Diss., 1995 (Zitiert auf Seite 11 und 69.)

[LBD02] Lodderstedt, Torsten; Basin, David; Doser, Jürgen: SecureUML: A UML-Based Modeling Language for Model-Driven Security. In: *«UML» 2002 – The Unified Modeling Language (UML 2002)* Bd. 2460, Springer, 2002, S. 426–441 (Zitiert auf Seite 11 und 68.)

[LP99] LINDQVIST, Ulf; PORRAS, Phillip A.: Detecting Computer and Network Misuse Through the Production-Based Expert System Toolset (P-BEST). In: *IEEE Symposium on Security and Privacy (SP 1999)* IEEE, 1999. – ISBN 0–769–50176–1, S. 146–161 (Zitiert auf Seite 66.)

[LRS10] LUND, Mass S.; REFSDAL, Atle; STØLEN, Ketil: Semantics of UML Models for Dynamic Behavior: A Survey of Different Approaches. In: GIESE, Holger; KARSAI, Gabor; LEE, Edward; RUMPE, Bernhard; SCHÄTZ, Bernhard (Hrsg.): *Proceedings of the 2007 International Dagstuhl conference on Model-based engineering of embedded real-time systems (MBEERTS 2007)*, Springer, 2010, S. 77–103 (Zitiert auf Seite 55, 57 und 60.)

[Mei04] MEIER, Michael: A Model for the Semantics of Attack Signatures in Misuse Detection Systems. In: ZHANG, Kan; ZHENG, Yuliang (Hrsg.): *7th International Conference on Information Security (ISC 2004)* Bd. 3225, Springer, 2004 (LNCS). – ISBN 978–3–540–23208–7, S. 158–169 (Zitiert auf Seite 87, 88 und 93.)

[MF99] MCDERMOTT, John; FOX, Chris: Using Abuse Case Models for Security Requirements Analysis. In: *15th Annual Computer Security Applications Conference (ACSAC 1999)* IEEE, 1999, S. 55–64 (Zitiert auf Seite 54.)

[MJG+12] MEREDITH, Patrick O.; JIN, Dongyun; GRIFFITH, Dennis; CHEN, Feng; ROŞU, Grigore: An Overview of the MOP Runtime Verification Framework. In: *International Journal on Software Tools for Technology Transfer* 14 (2012), Nr. 3, S. 249–289. http://dx.doi.org/10.1007/s10009-011-0198-6. – DOI 10.1007/s10009–011–0198–6 (Zitiert auf Seite 67.)

[MN08] MASSACCI, Fabio; NALIUKA, Katsiaryna: Towards Practical Security Monitors of UML Policies for Mobile Applications. In: *Third International Conference on Availability, Reliability and Security (ARES 08)*, IEEE Computer Society, 2008, S. 1112–1119 (Zitiert auf Seite 11, 67 und 70.)

[MP84] MCMENAMIN, Stephen M.; PALMER, John F.: *Essential Systems Analysis*. Yourdon Press, 1984. – 408 S. – ISBN 978–0–917–07230–7 (Zitiert auf Seite 48.)

[MP99] MUSCHOLL, Anca; PELED, Doron: Message Sequence Graphs and Decision Problems on Mazurkiewicz Traces. In: KUTYŁOWSKI, Mirosław; PACHOLSKI, Leszek; WIERZBICKI, Tomasz (Hrsg.): *24th International Symposium on Mathematical Foundations of Computer Science (MFCS 1999)* Bd. 1672, Springer, 1999 (LNCS). – ISBN 978–3–540–66408–6, S. 81–91 (Zitiert auf Seite 57.)

[MS05] MEIER, Michael; SCHMERL, Sebastian: Effiziente Analyseverfahren für Intrusion-Detection-Systeme. In: FEDERRATH, Hannes (Hrsg.): *Sicherheit 2005: Sicherheit - Schutz und Zuverlässigkeit, Beiträge der 2. Jahrestagung des Fachbereichs Sicherheit der Gesellschaft für Informatik e.v. (GI)*

Bd. 62, GI, 2005 (LNI). – ISBN 3–885–79391–1, S. 209–220 (Zitiert auf Seite 11 und 69.)

[MVG08] Meyers, Bart; Van Gorp, Pieter: Towards A Hybrid Transformation Language: Implicit and Explicit Rule Scheduling in Story Diagrams. In: *Sixth International Fujaba Days* (2008), S. 15–18 (Zitiert auf Seite 182.)

[Oeh02] Oehry, Bernhard: *The CARDME Concept*. Online. www.uv.es/fsoriano/efc/cardme.pdf. Version: June 2002 (Zitiert auf Seite 19 und 20.)

[OMG06] OMG: *OMG Meta Object Facility (MOF) Core Specification*. online. Version: Januar 2006. http://www.omg.org/spec/MOF/2.0/ (Zitiert auf Seite 30, 31, 32 und 133.)

[OMG08] OMG: *A UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded systems*. online. Version: Juni 2008. http://www.omg.org/docs/ptc/08-06-08.pdf (Zitiert auf Seite 62.)

[OMG10] OMG: *OMG Object Constraint Language*. online. Version: Februar 2010. http://www.omg.org/spec/OCL/2.2/ (Zitiert auf Seite 32.)

[OMG11a] OMG: *Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification*. online. Version: Januar 2011. http://www.omg.org/spec/QVT/1.1/ (Zitiert auf Seite 180.)

[OMG11b] OMG: *OMG Unified Modeling LanguageTM (OMG UML), Infrastructure*. online. Version: August 2011. http://www.omg.org/spec/UML/2.4.1/Infrastructure/ (Zitiert auf Seite 31, 60 und 72.)

[OMG11c] OMG: *OMG Unified Modeling LanguageTM (OMG UML), Superstructure*. online. Version: August 2011. http://www.omg.org/spec/UML/2.4.1/Superstructure (Zitiert auf Seite 32, 60 und 65.)

[OMG11d] OMG: *A UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded systems*. online. Version: Juni 2011. http://www.omg.org/spec/MARTE/1.1/PDF/ (Zitiert auf Seite 62.)

[Pat14] Patzina, Lars: *Generierung von effizienten Security-/Safety-Monitoren aus modellbasierten Beschreibungen*, Technische Universität Darmstadt, Diss., 2014. http://nbn-resolving.de/urn/resolver.pl?urn=urn:nbn:de:tuda-tuprints-41334. – URN urn:nbn:de:tuda–tuprints–41334 (Zitiert auf Seite v, vii, viii, 7, 15, 17, 18, 37, 43, 86, 141, 152, 160, 162, 178, 194, 204, 217, 219 und 221.)

[Plo08] Plock, Cory: *Synthesizing executable programs from requirements*, Department of Computer Science, New York University, Diss., Mai 2008. – 172 S. (Zitiert auf Seite 84.)

[PM06] PETRASCH, Roland; MEIMBERG, Oliver: *Model Driven Architecture*. 1. Dpunkt Verlag, 2006. – 384 S. – ISBN 978–3–898–64343–6 (Zitiert auf Seite 25, 30 und 32.)

[PP12] PATZINA, Sven; PATZINA, Lars: A Case Study Based Comparison of ATL and SDM. In: SCHÜRR, Andy; VARRÓ, Dániel; VARRÓ, Gergely (Hrsg.): *Applications of Graph Transformations with Industrial Relevance (AGTIVE 2012)* Bd. 7233, Springer, 2012 (LNCS). – ISBN 978–3–642–34175–5, S. 210–221 (Zitiert auf Seite 175.)

[PPPM13] PATZINA, Lars; PATZINA, Sven; PIPER, Thorsten; MANNS, Paul: Model-Based Generation of Run-Time Monitors for AUTOSAR. Version: 2013. http://dx.doi.org/10.1007/978-3-642-39013-5_6. In: GORP, Pieter; RITTER, Tom; ROSE, Louis M. (Hrsg.): *Modelling Foundations and Applications (ECMFA 2013)* Bd. 7949. DOI. – 10.1007/978–3–642–39013– 5_6. – ISBN 978–3–642–39012–8, S. 70–85. – Best Paper Award (Zitiert auf Seite 195 und 196.)

[PPS11] PATZINA, Sven; PATZINA, Lars; SCHÜRR, Andy: Extending LSCs for Behavioral Signature Modeling. In: CAMENISCH, Jan; FISCHER-HÜBNER, Simone; MURAYAMA, Yuko; PORTMANN, Armand; RIEDER, Carlos (Hrsg.): *Future Challenges in Security and Privacy for Academia and Industry (IFIP SEC 2011)* Bd. 354, 2011 (IFIP Advances in Information and Communication Technology). – ISBN 978–3–642–21423–3, S. 293–304 (Zitiert auf Seite 74 und 97.)

[PWMS12] PIPER, Thorsten; WINTER, Stefan; MANN, Paul B.; SURI, Neeraj: Instrumenting AUTOSAR for Dependability Assessment: A Guidance Framework. In: *42nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2012)* IEEE, 2012, S. 1–12 (Zitiert auf Seite 198.)

[PX05] PAULI, Joshua J.; XU, Dianxiang: Trade-Off Analysis of Misuse Case-Based Secure Software Architectures: A Case Study. In: *3rd International Workshop on Modelling, Simulation, Verification and Validation of Enterprise Information Systems (MSVVEIS 2005)*, INSTICC Press, 2005, S. 89–95 (Zitiert auf Seite 54.)

[PX06] PAULI, Joshua J.; XU, Dianxiang: Ensuring Consistent Use/Misuse Case Decomposition for Secure Systems. In: *Eighteenth International Conference on Software Engineering & Knowledge Engineering (SEKE 2006)*, 2006, S. 392–397 (Zitiert auf Seite 54.)

[RGS12] ROYCHOUDHURY, Abhik; GOEL, Ankit; SENGUPTA, Bikram: Symbolic Message Sequence Charts. In: *ACM Transactions on Software Engineering and Methodology* 21 (2012), 3, Nr. 2. http://dx.doi.org/10.1145/2089116.2089122. – DOI 10.1145/2089116.2089122 (Zitiert auf Seite 55.)

[RJK+06] Ruscio, Davide D.; Jouault, Frédéric; Kurtev, Ivan; Bézivin, Jean; Pierantonio, Alfonso: Extending AMMA for Supporting Dynamic Semantics Specifications of DSLs / LINA, Université de Nantes. 2006 (06.02). – Forschungsbericht (Zitiert auf Seite 179.)

[RQJZ07] Rupp, Chris; Queins, Stefan; Jeckle, Mario; Zengler, Barbara: *UML2 glasklar: Praxiswissen für die UML-Modellierung*. 3. Hanser Verlag, 2007. – 554 S. – ISBN 978–3–446–41118–0 (Zitiert auf Seite 6, 26, 30, 34, 48, 52, 54, 60 und 64.)

[SBP08] Smith, Sandra; Beaulieu, Alain; Phillips, W. G.: Modeling Security Protocols Using UML 2. In: Whittle, Jon; Jürjens, Jan; Nuseibeh, Bashar; Dobson, Glen (Hrsg.): *Workshop on Modeling Security (MODSEC08)* Bd. 413, 2008 (CEUR Workshop Proceedings) (Zitiert auf Seite 85.)

[SBPM08] Steinberg, Dave; Budinsky, Frank; Paternostro, Marcelo; Merks, Ed: *EMF: Eclipse Modeling Framework (2nd Edition)*. 2nd Revised. Addison-Wesley Professional, 2008. – ISBN 978–0–321–33188–5 (Zitiert auf Seite 33.)

[Sch97] Schürr, A.: Programmed Graph Replacement Systems. In: Rozenberg, Grzegorz (Hrsg.): *Handbook of Graph Grammars and Computing by Graph Transformation: Foundations* Bd. 1. World Scientific, 1997, S. 479–546 (Zitiert auf Seite 180.)

[Sch04] Schmerl, Sebastian: *Entwurf und Entwicklung einer Effizienten Analyseeinheit für Intrusion-Detection-Systeme*, Lehrstuhl Rechnernetze, BTU Cottbus, Diplomarbeit, Oktober 2004 (Zitiert auf Seite 11, 69, 87 und 88.)

[SEJK08] Smith, Randy; Estan, Cristian; Jha, Somesh; Kong, Shijin: Deflating the Big Bang: Fast and Scalable Deep Packet Inspection with Extended Finite Automata. In: *SIGCOMM Computer Communication Review* 38 (2008), August, Nr. 4, S. 207–218. http://dx.doi.org/10.1145/1402946.1402983. – DOI 10.1145/1402946.1402983 (Zitiert auf Seite 69.)

[SES07] Solhaug, Bjornar; Elgesem, Dag; Stolen, Ketil: Specifying Policies Using UML Sequence Diagrams–An Evaluation Based on a Case Study. In: *8th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY 2007)*, IEEE Computer Society, 2007. – ISBN 0–7695–2767–1, S. 19–28 (Zitiert auf Seite 11, 70, 71, 86 und 87.)

[SL02] Sloman, M.; Lupu, E.: Security and Management Policy Specification. In: *IEEE Netw Mag Global Inform Exchange* 16 (2002), März, Nr. 2, S. 10–19. http://dx.doi.org/10.1109/65.993218. – DOI 10.1109/65.993218 (Zitiert auf Seite 86.)

[SO00] SINDRE, G.; OPDAHL, Andreas L.: Eliciting Security Requirements by Misuse Cases. In: *37th International Conference on Technology of Object-Oriented Languages and Systems (TOOLS 2000)*, 2000. – ISSN 1530–2067, S. 120–131 (Zitiert auf Seite 52.)

[SO05] SINDRE, Guttorm; OPDAHL, Andreas L.: Eliciting security requirements with misuse cases. In: *Requir. Eng.* 10 (2005), Nr. 1, S. 34–44. http://dx.doi.org/10.1007/s00766-004-0194-4. – DOI 10.1007/s00766–004–0194–4 (Zitiert auf Seite 7, 48, 54 und 73.)

[Spa13] SPARX SYSTEMS: *Enterprise Architect User Guide*. Online. http://www.sparxsystems.com/enterprise_architect_user_guide/10/index.html. Version: 10, 2013, Abruf: 04.01.2014 (Zitiert auf Seite 121.)

[Stö04] STÖRRLE, Harald: Trace Semantics of Interactions in UML 2.0 / IFI-PST, LMU München. 2004 (TR 0403). – Forschungsbericht (Zitiert auf Seite 65.)

[Sym13] SYMANTEC: *2013 Norton Report*. Online. http://www.symantec.com/about/news/resources/press_kits/detail.jsp?pkid=norton-report-2013. Version: 2013, Abruf: 07.11.2013 (Zitiert auf Seite 3.)

[TEG+05] TAENTZER, Gabriele; EHRIG, Karsten; GUERRA, Esther; DE LARA, Juan; LENGYEL, Laszlo; LEVENDOVSZKY, Tihamer; PRANGE, Ulrike; VARRÓ, Dániel; VARRÓ-GYAPAY, Szilvia: Model Transformation by Graph Transformation: A Comparative Study. In: *International Workshop on Model Transformations in Practice (MTiP 2005)*, 2005 (Zitiert auf Seite 175 und 180.)

[The12] THE MATHWORKS, INC.: *Modeling an Automatic Transmission Controller*. Online. http://www.mathworks.de/de/help/simulink/examples/modeling-an-automatic-transmission-controller.html. Version: 2012, Abruf: 10.10.2013 (Zitiert auf Seite 199.)

[TV10] TROYA, Javier; VALLECILLO, Antonio: Towards a Rewriting Logic Semantics for ATL. Version: 2010. http://dx.doi.org/10.1007/978-3-642-13688-7_16. In: TRATT, Laurence; GOGOLLA, Martin (Hrsg.): *Theory and Practice of Model Transformations* Bd. 6142. DOI. – 10.1007/978–3–642–13688– 7_16. – ISBN 978–3–642–13687–0, S. 230–244 (Zitiert auf Seite 179.)

[UKM04] UCHITEL, Sebastián; KRAMER, Jeffrey; MAGEE, Jeff: Incremental Elaboration of Scenario-Based Specifications and Behavior Models Using Implied Scenarios. In: *ACM Transactions on Software Engineering and Methodology* 13 (2004), Nr. 1, S. 37–85. http://dx.doi.org/10.1145/1005561.1005563. – DOI 10.1145/1005561.1005563 (Zitiert auf Seite 72.)

[Var01] Vardi, Moshe Y.: Branching vs. Linear Time: Final Showdown. In: Margaria, Tiziana; Yi, Wang (Hrsg.): *7th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2001)* Bd. 2031, Springer, 2001 (LNCS). – ISBN 3–540–41865–2, S. 1–22 (Zitiert auf Seite 138.)

[VB07] Varró, Dániel; Balogh, András: The Model Transformation Language of the VIATRA2 Framework. In: *Science of Computer Programming* 68 (2007), Nr. 3, S. 214–234. http://dx.doi.org/10.1016/j.scico.2007.05.004. – DOI 10.1016/j.scico.2007.05.004 (Zitiert auf Seite 180.)

[VEK00] Vigna, Giovanni; Eckmann, Steve T.; Kemmerer, Richard A.: The STAT Tool Suite. In: *DARPA Information Survivability Conference and Exposition (DISCEX 2000)* Bd. 2, IEEE Computer Society, 2000, S. 46–55 (Zitiert auf Seite 11 und 68.)

[Whi07] Whittle, Jon: Precise Specification of Use Case Scenarios. Version: 2007. http://dx.doi.org/10.1007/978-3-540-71289-3_15. In: Dwyer, Matthew B.; Lopes, Antónia (Hrsg.): *Fundamental Approaches to Software Engineering* Bd. 4422. DOI. – 10.1007/978–3–540–71289– 3_15. – ISBN 978–3–540–71288–6, S. 170–184 (Zitiert auf Seite 73.)

[WJ06] Whittle, Jonathan; Jayaraman, Praveen K.: Generating Hierarchical State Machines from Use Case Charts. In: *14th IEEE International Conference on Requirements Engineering (RE 2006)*, 2006, S. 16–25 (Zitiert auf Seite 140.)

[WKK+11] Wimmer, M.; Kappel, G.; Kusel, A.; Retschitzegger, W.; Schönböck, J.; Schwinger, W.; Kolovos, D.; Paige, R.; Lauder, M.; Schürr, A.; Wagelaar, D.: A Comparison of Rule Inheritance in Model-to-Model Transformation Languages. Version: 2011. http://dx.doi.org/10.1007/978-3-642-21732-6_3. In: Cabot, Jordi; Visser, Eelco (Hrsg.): *4th International Conference on Model Transformation (ICMT 2011), Theory and Practice of Model Transformations* Bd. 6707. DOI. – 10.1007/978–3–642–21732– 6_3, S. 31–46 (Zitiert auf Seite 176.)

[WPJH98] Weidenhaupt, Klaus; Pohl, Klaus; Jarke, Matthias; Haumer, Peter: Scenario Usage in System Development: A Report on Current Practice. In: *IEEE Software* 15 (1998), Nr. 2, S. 34–45. http://dx.doi.org/10.1109/52.663783. – DOI 10.1109/52.663783 (Zitiert auf Seite 86.)

[WT06] Westphal, Bernd; Toben, Tobe: The Good, the Bad and the Ugly: Well-Formedness of Live Sequence Charts. In: Baresi, Luciano; Heckel, Reiko (Hrsg.): *9th International Conference on Fundamental Approaches to Software Engineering (FASE 2006)* Bd. 3922, Springer, 2006 (LNCS), S. 230–246 (Zitiert auf Seite 86.)

[Wu93] Wu, Guofei: *Visual Programming for Image Analysis in Object Oriented Style*. 215. Oldenbourg Wissenschaftsverlag, 1993 (Berichte der Gesell-

schaft für Mathematik und Datenverarbeitung / Gesellschaft für Mathematik und Datenverarbeitung). – 167 S. – ISBN 978–3–486–22777–2 (Zitiert auf Seite 67.)

[WWH08] WHITTLE, Jonathan; WIJESEKERA, Duminda; HARTONG, Mark: Executable Misuse Cases for Modeling Security Concerns. In: *30th International Conference on Software Engineering (ICSE 2008)*, ACM, 2008. – ISBN 978–1–60558–079–1, S. 121–130 (Zitiert auf Seite 54, 73 und 140.)

[ZCL+14] ZHANG, Liang; CHOFFNES, David; LEVIN, Dave; DUMITRAS, Tudor; MISLOVE, Alan; SCHULMAN, Aaron; WILSON, Christo: Analysis of SSL Certificate Reissues and Revocations in the Wake of Heartbleed. In: *Proceedings of the 14th Conference on Internet Measurement Conference (IMC 2014)*, ACM, 2014. – Akzeptiert zur Publikation (Zitiert auf Seite 14.)

CURRICULUM VITAE

Name	Sven Patzina
Geburtsdatum	01. September 1981
Geburtsort	Bad Soden, Deutschland
E-Mail	fges.svenpatzina@gmail.com

2008 - 2013	Wissenschaftlicher Mitarbeiter am Fachgebiet Echtzeitsysteme der TU Darmstadt
2008	Dipl.-Ing. Elektro- und Informationstechnik
	Diplomarbeit: *„Anpassung eines UML-Metamodellierungswerkzeuges für die Metamodellierung domänenspezifischer Sprachen"*
2007 -2008	Fachpraktikum und Studienarbeit bei BMW AG in München (10 Monate)
	Studienarbeit: *„Berücksichtigung von Infrastrukturinformationen zur Reduktion des Aufwandes einer Testfallgenerierung"*
2002 - 2008	Studium der Elektro- und Informationstechnik (Vertiefung Datentechnik) an der TU Darmstadt
1994 - 2001	Gymnasium Oberursel, Abitur