Vorwort zur Wissenschaftlichen Schriftenreihe ”Eingebettete, selbstorganisierende Systeme”

Mit Beginn des Jahres 2008 erscheint in der wissenschaftlichen Schriftenreihe *Eingebettete, selbstorganisierende Systeme* der Band sechs. Dieser widmet sich der Untersuchung von Kompatibilitätsfragen von Steuergeräten. Die stetig steigende Zahl von Steuergeräten in Kraftfahrzeugen erfordert neue Kompatibilitätskonzepte, insbesondere für aufeinanderfolgende Fahrzeuggenerationen und für modellreihenübergreifende Entwurfsansätze. Herr Glockner greift mit seiner Arbeit eine Thematik von hoher industrieller Relevanz und ausgesprochen hoher Komplexität auf und präsentiert hier Grundlagenarbeit.

Besonders umfassend wird der Stand der Technik auf diesem neuen Gebiet analysiert und zusammengefasst. Schwerpunkte dieser Arbeit sind in der Entwicklung eines abstrakten Spezifikationsansatzes für Steuergeräte und den darauf aufbauenden Vergleichsmethoden für statische Parameter und funktionale Beschreibungselemente zu sehen. Das funktionale Verhalten wird auf Basis der Spezifikation durch Message Sequence Diagrams analysiert. Systematisch werden Message Sequence Diagramme in endliche Automaten konvertiert und auf dieser Basis miteinander verglichen. Die formale und algorithmische Vorgehensweise zur Lösung der Kompatibilitätsfrage ist Herrn Glockner sehr gut gelungen und führt zu einem in sich geschlossenen Untersuchungsansatz.

Viele weitere Anwendungsmöglichkeiten regen den Leser an, die vorgestellten Methoden zu adaptieren und in neuen Bereichen zu nutzen.

Ich freue mich, Herrn Glockner für die Veröffentlichung der Ergebnisse seiner Arbeiten in dieser wissenschaftlichen Schriftenreihe gewonnen zu haben, und wünsche allen Lesern einen interessanten Einblick in die Kompatibilitätsanalyse von Steuergeräten.

Prof. Dr. Wolfram Hardt
Professur Technische Informatik
Dekan der Fakultät für Informatik
Wissenschaftlicher Leiter Universitätsrechenzentrum
Technische Universität Chemnitz
Januar 2008

TECHNISCHE UNIVERSITÄT CHEMNITZ

Fakultät für Informatik

Methoden zur Analyse von Rückwärtskompatibilität von Steuergeräten

Dissertation

zur Erlangung des akademischen Grades

Doktoringenieur
(Dr.-Ing.)

vorgelegt

der Fakultät für Informatik
Technische Universität Chemnitz

von:	Dipl.-Ing.(Univ.) Matthias Glockner
geboren am:	05.12.1973
geboren in:	Viechtach

Gutachter:	Prof. Dr. rer. nat. Wolfram Hardt Prof. Dr.-Ing. Ulrich Heinkel Prof. Dr. Christophe Bobda

Chemnitz, den 23.10.2007

Danksagung

Mein besonderer Dank gilt Prof. Dr. Wolfram Hardt für die hervorragende, konstruktive Betreuung meiner Arbeit. Er nahm sich stets Zeit für regelmäßige wissenschaftliche Diskussionen, die wesentlich zum Erfolg dieser Arbeit beitrugen. Des Weiteren danke ich ihm für die Bereitstellung der Arbeitsumgebung am Lehrstuhl.

Ein großer Dank gilt auch meinen Kollegen bei BMW, die für fachliche Fragen immer zur Verfügung standen. Dr. Michael Macht danke ich für seine Unterstützung bei der Initialisierung dieser Arbeit und der Ermöglichung eines Sabbaticals. Gleichzeitig lebte er aktiv die Rolle des Mentors. Dr. Maximilian Fuchs danke ich sehr für die intensive fachliche Betreuung meiner Arbeit. Die konstruktiven Diskussionen mit ihm gaben oft entscheidende Anstöße bei konzeptionellen Fragen und unterstützten die Forschungsarbeit optimal. Herrn Uwe Lüddeke danke ich für den fachlichen Input vor allem in der Anfangsphase dieser Arbeit.

Bei Prof. Dr. Ulrich Heinkel und Prof. Dr. Christophe Bobda bedanke ich mich für die gute Zusammenarbeit und die konstruktiven Rückmeldungen zu meiner Arbeit. Prof. Dr. Maximilian Eibl danke ich für die interessanten Einblicke in neue Forschungsgebiete.

Allen Mitarbeiter und studentischen Hilfskräften des Lehrstuhls „Technische Informatik“ danke ich für die gute kooperative Zusammenarbeit. Stephan Reichelt, Kay Timmermann, Zheng Ma und Marcel Karras danke ich für die eigenständige und konstruktive Implementationsarbeit.

Es ist mir ein Anliegen, alle jenen Dank zu sagen, die mich in Sachsen und insbesondere in Chemnitz so freundlich aufgenommen und begleitet haben.

Abschließend möchte ich mich auch bei meiner Familie und meinen Freunden bedanken, die mich stets unterstützten und trotz meiner zeitlichen Einschränkungen zu mir standen.

Kurzfassung

Der Elektrik/Elektronik- und der IT-Anteil steigt derzeit in den aktuellen Premium - Fahrzeugen stetig an. Durch den Verbau von immer mehr (hochvernetzten) Steuergeräten im Fahrzeug wird versucht, dem Wunsch der Kunden nach mehr Funktionalität, Sicherheit, etc. gerecht zu werden.
Aufgrund der Komplexität und der großen Entwicklungssprünge sind jedoch die neuen Steuergeräte meistens nicht mehr kompatibel mit den Vorgänger-Steuergeräten. Hier ist ein großes Einsparpotenzial vorhanden und dies ist auch der Ansatzpunkt des Forschungsthemas „CompA“ (Compatibility Analysis of Electronic Control Units), das in diesem Dokument beschrieben wird. Im Rahmen dieses Forschungsthemas wird eine Methode definiert, mit der zwei Steuergeräte auf Rückwärtskompatibilität untersucht werden können. Der Ansatz baut auf drei Schwerpunkten auf:

- Definition eines Spezifikationsansatzes zur hinreichenden Beschreibung von Steuergeräten auf Basis eines XML-Schemas. Es werden hierbei sowohl die statischen als auch die dynamischen Eigenschaften abgebildet. Dieser neuartige Spezifikationsansatz bildet die Basis für den nächsten Schwerpunkt.

- Definition einer Methode zur Analyse von Rückwärtskompatibilität von Steuergeräten auf Basis eines XML-Schemas. Die Rückwärtskompatibilität zweier Steuergeräte bzw. Systeme wird auf Basis der zugehörigen XML-Dokumente analysiert. Kern der Vergleichsmethode ist hierbei ein effizientes Mapping der XML-Dokumente und ein Experten-Regelwerk.

- Definition einer Methode zur Kompatibilitätsanalyse von Message Sequence Charts (MSC). MSCs werden eingesetzt, um dynamisches Verhalten an der Schnittstelle von Steuergeräten zu beschreiben. In diesem Dokument wird ein Ansatz definiert, mit dem MSCs zueinander auf Rückwärtskompatibilität geprüft werden können. Der Vergleich erfolgt auf Basis deterministischer Automaten.

Des Weiteren wird im vorliegenden Dokument ein Konzept für eine graphische Benutzeroberfläche (GUI) vorgestellt, die zur Spezifikation von Steuergeräten geeignet ist und sich adaptiv unterschiedlichen Schemata anpasst. Alle vorgestellten Konzepte wurden in einem Software-Tool implementiert und die Gültigkeit an mehreren Beispielen validiert.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Käufer von aktuellen Premium-Fahrzeugen erwarten, dass ihre Fahrzeuge auf dem neuesten Stand der Technik sind. Headup-Display, Night-Vision, Wankstabilisierung oder Hybrid-Technologien sind Beispiele für solche neue Technologien und können nur durch einen verstärkten Einsatz von eingebetteten Systemen bzw. Steuergeräten realisiert werden. Dies hat jedoch wiederum zur Folge, dass sich die Anzahl und die Varianten-Vielfalt der Steuergeräte in den Fahrzeugen vergrößert. Um Entwicklungs- und Gewährleistungskosten dennoch im Griff zu behalten, suchen Fahrzeughersteller nach neuen Ansätzen zur Reduktion der Anzahl der Steuergeräte-Varianten.

In einer Zusammenarbeit der TU-Chemnitz und der BMW Group wurde das Forschungsthema *Compatibility Analysis of Electronic Control Units (CompA)* definiert, das im Rahmen dieser Arbeit vollständig bearbeitet wurde. Hierbei wurde eine Methode entwickelt, mit der Steuergeräte bzgl. Rückwärtskompatibilität analysiert werden können. Auf Basis dieser Analyse können Steuergeräte u. U. in anderen Fahrzeugmodellen verbaut und somit die Steuergeräte-Varianten reduziert werden. In der vorliegenden Arbeit wird dieser durchgängige Ansatz detailliert beschrieben.

Dieses Kapitel stellt die Motivation für das Projekt „CompA“ vor. Es wird hierbei betrachtet, wie sich der Elektrik/Elektronik-Anteil im Fahrzeug entwickelt hat und voraussichtlich weiter entwickeln wird (vgl. Abschnitt 1.1). Anschließend werden die Folgen dieses Trends in Abschnitt 1.2 betrachtet. In Abschnitt 1.3 wird das genaue Ziel von „CompA“ bzw. dieser Arbeit zusammengefasst. Das Einleitungs-Kapitel schließt mit der Beschreibung der weiteren Strukturierung dieser Arbeit (Abschnitt 1.4).

1.1 Entwicklung der E/E-Umfänge im Fahrzeug

Eingebettete Systeme

Das Kerngeschäft im Fahrzeugbau ist nicht mehr ausschließlich der Maschinenbau. Seit der Entwicklung der *Eingebetteten Systeme* hält die Elektrik/Elektronik[1](E/E) sehr stark Einzug in das Fahrzeug. Im Rahmen von CompA fokussieren wir eine zentrale E/E-Komponente, das Steuergerät.

Steuergeräte sind *Eingebettete Systeme* und stellen die zentralen Rechen- und Steuereinheiten für elektrische und elektronische Systeme dar [140]. Die Grundaufgaben von Steuergeräten sind Sensorauswertungen, Berechnung von Algorithmen, Diagnose und Ansteuerung von Aktoren. Durch Kombination dieser Aufgaben können

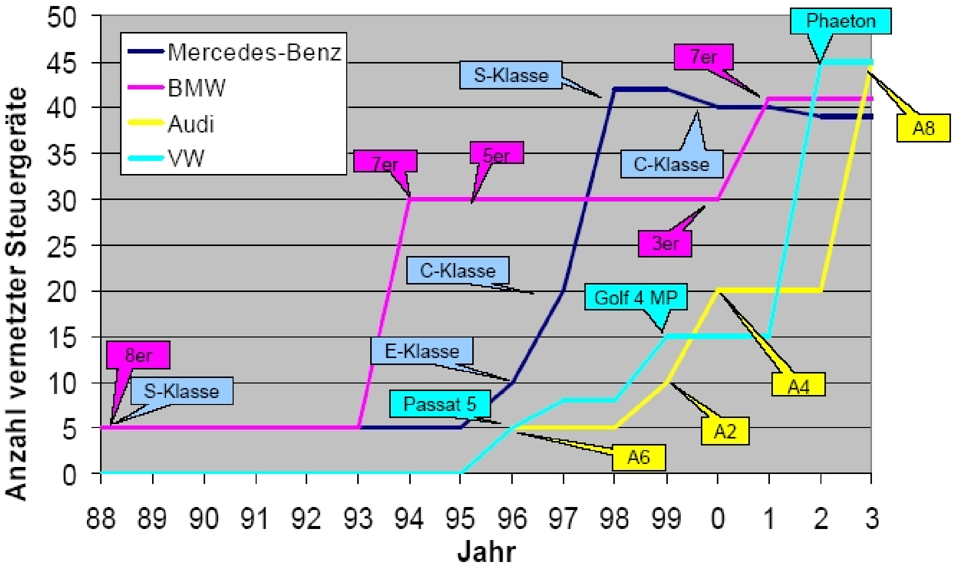
---

[1] Dem Elektrik-/Elektronik-Anteil werden hierbei alle elektrischen Komponenten wie Kabelbaum, elektrische Motoren, Bussysteme, Steuergeräte, etc. zugeordnet.

vielfältige Funktionen im Automobil realisiert werden. Solche Funktionen sind beispielsweise das Antiblockier-System (ABS) oder die Ansteuerung der Fensterheber inkl. des zugehörigen Einklemmschutzes.

Verbreitung der Steuergeräte im Fahrzeug

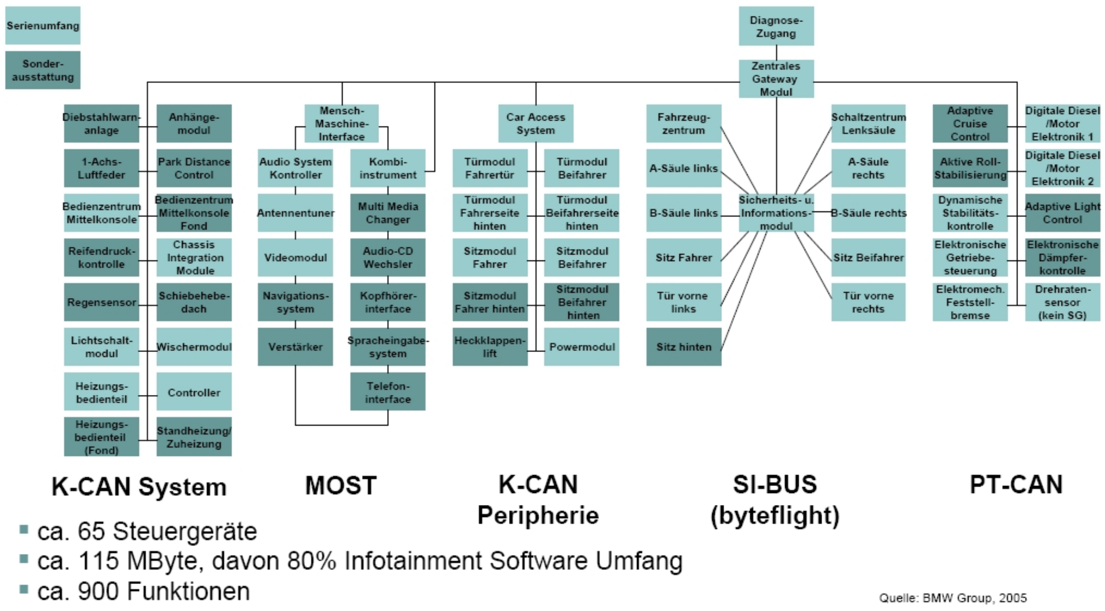
In Abbildung 1.1 ist die Entwicklung hin zu mehr Steuergeräten im Fahrzeug verdeutlicht. Im Jahr 1993 ist die Anzahl der Steuergeräte erstmals sprunghaft angestiegen und hat sich in den nächsten 10 Jahren um ca. das 8-fache vergrößert. Während Steuergeräte anfangs nicht oder nur sehr wenig vernetzt waren, sind in den heutigen Fahrzeugen fast alle Steuergeräte miteinander vernetzt und kommunizieren über verschiedene Bussysteme. Aufgrund unterschiedlicher Anforderungen (Kosten, Bandbreite, etc.) werden bei den Bussen verschiedene Technologien eingesetzt z.B. CAN-Bus, MOST-Bus oder Flex-Ray [140].



Quelle: Prof. Leohold, U Kassel

Abbildung 1.1: Entwicklung der Anzahl von Steuergeräten in verschiedenen Fahrzeugen [49]

Abbildung 1.2 zeigt den Systemverbund des BMW 7er (2001) und vermittelt einen Eindruck bzgl. des hohen Vernetzungsgrades aktueller Fahrzeuge. Im Ganzen sind ca. 65 Steuergeräte miteinander vernetzt und realisieren ca. 900 Funktionen. Die Funktionen reichen vom Fensterheber über Sitzheizung und Alarmanlage, bis zu Assistenz-Funktionen wie z.B. der *Dynamischen Stabilitätskontrolle (DSC)*. Die Steuergeräte enthalten hierbei Software von 115 MByte, wobei 80% der Software auf den Infotainment-Bereich entfallen.



- ca. 65 Steuergeräte
- ca. 115 MByte, davon 80% Infotainment Software Umfang
- ca. 900 Funktionen

Quelle: BMW Group, 2005

Abbildung 1.2: Systemverbund der Steuergeräte des BMW 7er [49]

Zukünftige Verbreitung der Steuergeräte

Bereits heute werden ca. 90% aller Innovationen im Kraftfahrzeug unter Nutzung elektronischer Systeme realisiert und künftige Innovationen werden vor allem über hoch vernetzte und komplexe Systeme realisiert werden [140]. Ein Indikator für diesen Trend ist, wenn man betrachtet, wie sich - angelehnt an das Moore'sche Gesetz - allein der Speicherbedarf in den Fahrzeugen entwickelt hat und voraussichtlich entwickeln wird. Das Moore'sche Gesetz besagt, dass sich der RAM-Speicher von PCs alle 4 Jahre ver-

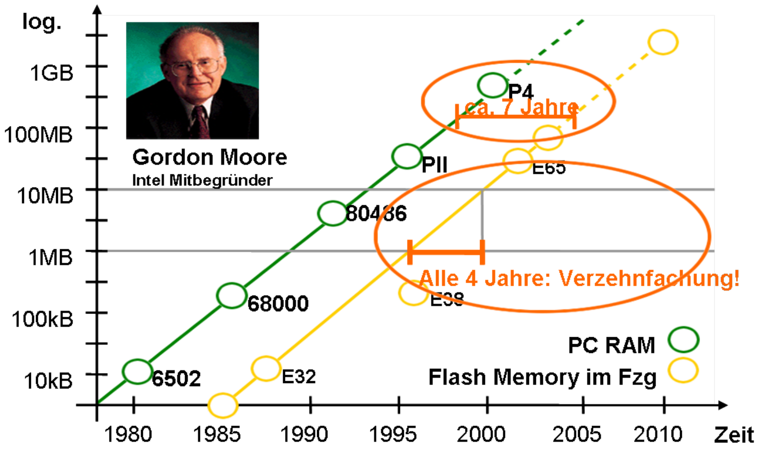


Abbildung 1.3: Voraussichtliche Entwicklung des Speicherbedarfs in Fahrzeugen [52]

zehnfacht. In Abbildung 1.3 ist die Entwicklung des PC RAMs (grüne Linie) der Entwicklung des Flash Memory im Fahrzeug (gelbe Linie) gegenübergestellt. Hier wird deutlich, dass die beiden Linien parallel verlaufen und um ca. 7 Jahre verschoben sind. Das bedeutet, dass sich in modernen Fahrzeugen der Flashspeicher alle vier Jahre verzehnfachen wird und Fahrzeuge mit dem Baujahr 2010 über 1GB Flashspeicher besitzen werden.

Der Anstieg der Steuergeräte-Anzahl und der hohe Vernetzungsgrad hat verschiedene Ursachen:

- *Kundenanforderungen:* Kunden erwarten, dass in Fahrzeugen der neueste Stand der Technik verbaut ist. Hierzu zählen Sicherheitssysteme wie Antiblockiersystem (ABS), Airbags (Knie-, Kopfairbag, etc.), aber auch neue Technologien wie Hybrid-Technologie, Motor-Start-Automatik. Diese Funktionalitäten sind nur durch einen verstärkten Einsatz von elektrischen Komponenten bzw. Steuergeräten umsetzbar.

- *Gesetzliche Anforderungen:* Die Erfüllung von gesetzlichen Normen kann z.T. nur durch den Einsatz von E/E-Komponenten erreicht werden. So muss beispielsweise zum Auslösen des Beifahrer-Airbags erkannt werden, ob dieser Sitz von einer Person oder einem Kindersitz belegt ist. Dies wird über „Sitzmatten“ automatisch erkannt und die Airbag-Auslöse-Freigabe an das Airbag-Steuergerät übermittelt.

- *Neue Technologien/Funktionen:* Neue Technologien/Funktionen sind nur durch eine hohe Vernetzung und Einsatz neuester Elektrik/Elektronik möglich. Beispiel hierfür sind Xenon-Lichter oder LED-Lichter kombiniert mit adaptivem Kurvenlicht.

- *Kosten:* Mittels Einsatz elektrischer Komponenten sind Funktionen realisierbar, die bei einer mechanischen Ausführung sehr kostenintensiv bzw. nicht möglich wären. Ein Beispiel hierfür ist die elektronische Einparkhilfe.

Dies ist nur ein Ausschnitt von Ursachen, die zu einer stetigen „Elektronifizierung“ von Fahrzeugen führen. Der steigende Innovationsgrad im E/E-Umfeld zwingt also die Autohersteller (OEM) in immer kürzeren Zeitabständen neue Steuergeräte mit mehr Rechenleistung, noch größerem Speicher und neuen Funktionen zu entwickeln. Es können zwar die komplexesten Funktionen durch Entwicklung neuer hochvernetzter Steuergeräte realisiert werden, für die Automobilhersteller birgt dieser Trend jedoch auch Risiken.

In Abschnitt 1.2 werden die Probleme bzw. Auswirkungen betrachtet, die die hohe Anzahl an Steuergeräten mit sich bringt.

1.2 Auswirkung auf Kostenfaktor

Durch die große Anzahl an Steuergeräten können die neuesten und komplexesten Funktionen realisiert und dem Kunden gewichtige Gründe für eine Kaufentscheidung geliefert werden. Für die Automobilhersteller ist dieser Trend jedoch mit hohen Kosten verbunden. Dies sei an den Entwicklungskosten und den Gewährleistungskosten verdeutlicht [124].

- *Entwicklungskosten:* In den Anfängen wurde jedes Steuergerät für jedes Fahrzeug-Modell neu entwickelt. Um Entwicklungskosten zu sparen, werden heute Steuergeräte für ein Lead-Projekt entwickelt und anschließend auf weitere Fahrzeugmodelle portiert. Kostenkritisch ist der Fall, inwiefern Steuergeräte, die aufgrund von neuen Funktionalitäten weiterentwickelt wurden, noch rückwärtskompatibel zu ihren Vorgänger-Steuergeräten sind.

- *Gewährleistungskosten:* Fahrzeughersteller geben z.T. eine Gewährleistungsgarantie auf Steuergeräte. So garantiert beispielsweise BMW, Steuergeräte bis zu 15 Jahre[2] nach Einstellung der Fahrzeug-Produktion weiterhin als Ersatzteile zur Verfügung zu stellen [94]. Da die Produktion der Steuergeräte auch mit der Produktion eines Fahrzeugs eingestellt wird, müssen daher diese Steuergeräte auf Halde gelagert werden. Aufgrund der hohen Anzahl an Steuergeräte-Varianten ist dies sehr kostenintensiv und stellt ferner „totes“ Kapital dar.

Die Kosten könnten in beiden Fällen gesenkt werden, wenn neu entwickelte Steuergeräte rückwärtskompatibel zu Vorgänger-Steuergeräten wären.

1.3 Ziel der Arbeit

Entwicklungskosten und Gewährleistungskosten für Steuergeräte können durch die Reduzierung der Anzahl an Steuergeräte-Varianten gesenkt werden. Ein Ansatz hierfür ist das Thema Rückwärtskompatibilität. Sind nämlich neu entwickelte Steuergeräte rückwärtskompatibel, so können diese die Vorgänger-Steuergeräte ersetzen und damit auch die Anzahl der zu verwaltenden Steuergeräte-Varianten senken. Aus diesem Grund fokussiert das Forschungsthema CompA bzw. diese Arbeit die Fragestellung, ob ein neu entwickeltes Steuergerät SG_2 ein Vorgänger-Steuergerät SG_1 im selben oder in anderen Fahrzeugmodellen ersetzen kann, d.h. ob SG_2 zu SG_1 rückwärtskompatibel ist.
Die Analyse der Rückwärtskompatibilität von Steuergerät SG_2 zu SG_1 soll dabei auf Basis der zugehörigen Spezifikationen erfolgen. In Abbildung 1.4 ist dies dargestellt. Fahrzeughersteller spezifizieren „nur“ die Anforderungen an Steuergeräte, da diese meist als *Black-Boxes* betrachtet werden. Funktionen der Steuergeräte werden dabei zum Teil durch Funktions-Modelle beschrieben, die als eine Erweiterung der Lastenhefte betrachtet werden können. Seitens der Zulieferer erfolgt der Entwurf der Architektur, die technische Realisierung und die technische Produktbeschreibung (TPB).
Durch einen Vergleich der Spezifikationen[3], also der Lastenhefte und der Funktionsmodelle, werden Aussagen bzgl. der Rückwärtskompatibilität aus Sicht der Fahrzeughersteller generiert. So kann entschieden werden, ob ein Steuergerät SG_2 zu Steuergerät

---

[2] Für Rolls-Royce gilt sogar lebenslange Versorgungssicherheit [94].

[3] Hierbei wird auf die Ebene der Fahrzeughersteller eingeschränkt.

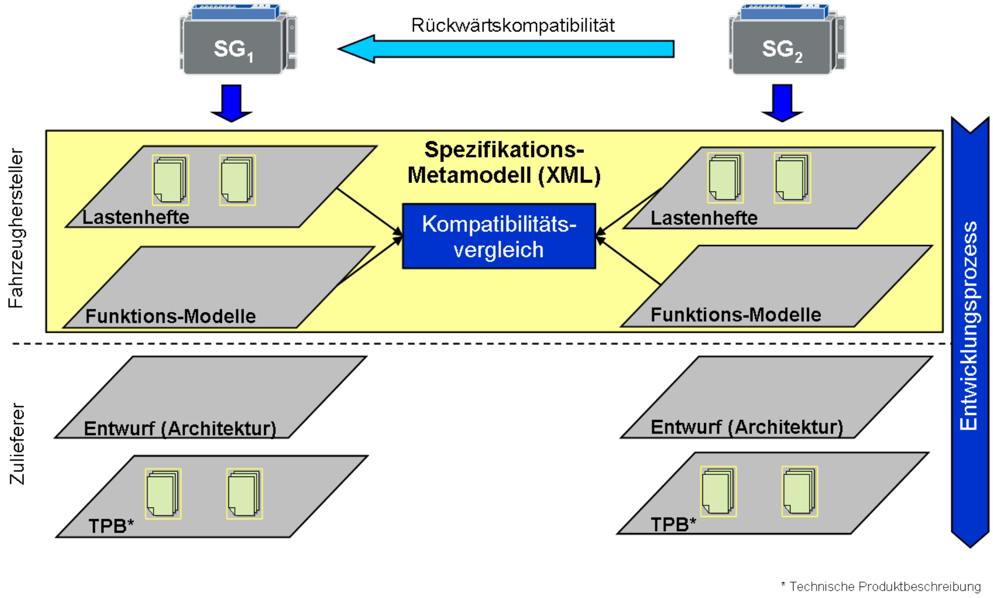


Abbildung 1.4: Rückwärtskompatibilität von Steuergeräten

SG_1, auf Basis der Anforderungen, rückwärtskompatibel ist. Die Voraussetzung für den Vergleich ist, dass die Spezifikationen auf einem gemeinsamen Metamodell basieren. In diesem Ansatz basiert das Metamodell auf einem XML-Schema.
Das Forschungsthema *CompA* hat den Anspruch, einen durchgängigen Ansatz für die Analyse von Rückwärtskompatibilität zur Verfügung zu stellen. Um dieses Ziel zu erreichen, wurden in dieser Arbeit mehrere Probleme bearbeitet, für die es bis dahin noch keine Lösungen gab.

Formalisierung: Spezifikationen von Steuergeräten sind großteils in Prosa-Text (Word, Doors,...) geschrieben und die Detaillierung und Formulierung ist „Autoren abhängig“. Des Weiteren gibt es kaum Vorgaben, wie detailliert welche Elemente von Steuergeräten beschrieben sein müssen. Aus diesem Grund ist ein automatisierter Vergleich von Spezifikationen nahezu unmöglich.
CompA hat zum Ziel, ein Steuergerät-Metamodell zu entwickeln, das festlegt, welche Elemente für eine hinreichende Spezifikation notwendig sind. Die Umsetzung soll auf Basis von XML-Schema erfolgen.
Zur Spezifikation von Funktionen werden als „formale“ Spezifikationsmethode sog. *Message Sequence Charts (MSC)* eingesetzt.

Vergleichsmethoden: Derzeit gibt es keine Methoden mit denen Spezifikationen (auf Basis von XML oder Message Sequence Charts) automatisiert bzgl. Rückwärtskompatibilität analysiert werden können.
Ziel von CompA ist, spezifische Vergleichsmethoden für XML-Spezifikationen und für

MSC-Modelle zu entwickeln, mit denen Aussagen bzgl. der Rückwärtskompatibilität der spezifizierten Elemente getroffen werden können.

Werkzeuge: Für die im Rahmen von CompA entwickelten Konzepte soll ein durchgängiges Tool zur Verfügung gestellt werden, das folgende Features bietet:

- Möglichkeit, Steuergeräte auf Basis eines Metamodells zu spezifizieren,
- automatisierte Vergleichsmethoden für XML basierte Spezifikationen
- automatisierte Vergleichsmethoden für MSCs.

Die Konzepte und die Implementierungen werden an mehreren Beispielen validiert.

1.4 Struktur der Arbeit

In Kapitel 2 werden zunächst die Grundlagen erläutert, die zum besseren Verständnis der, im Rahmen von CompA, entwickelten Methoden beitragen. Zuerst wird hierfür kurz der allgemeine Aufbau von Steuergeräten vorgestellt. Anschließend wird auf die Extensible Markup Language (XML) und deren spezifischen Elemente wie XML-Schema und XML-Dokumente eingegangen. Des Weiteren wird eine kurze Einführung zu Message Sequence Charts (MSC) und zu endlichen Automaten gegeben.
Nach der Erläuterung der Grundlagen wird in Kapitel 3 das Gesamtkonzept von CompA vorgestellt. Hierfür wird zuerst definiert, wann zwei Steuergeräte als zueinander rückwärtskompatibel gelten. Im Anschluss daran wird der durchgängige Ansatz selbst betrachtet und drei Schwerpunkte definiert:

- der Spezifikationsansatz,
- die XML-Vergleichsmethode (XML-CompA) und
- die MSC-Vergleichsmethode (MSC-CompA).

Bevor auf die einzelnen Schwerpunkte genauer eingegangen wird, wird in Kapitel 4 der Stand der Technik in Bezug auf die Schwerpunkte von CompA vorgestellt.
In Kapitel 5 wird der erste Schwerpunkt von CompA vorgestellt, der abstrakte Spezifikationsansatz. Hier wird der Entwurf eines abstrakten Steuergeräte-Metamodells und die konkrete Umsetzung beschrieben. Auf Basis des Steuergeräte-Metamodells (XML-SG-Schema) können Steuergeräte Spezifikationen (XML-SG-Instanzen) abgeleitet werden. Um Aussagen bzgl. der Rückwärtskompatibilität der zugehörigen Steuergeräte treffen zu können, werden die Spezifikationen miteinander verglichen.
In Kapitel 6 wird hierfür der zweite Schwerpunkt dieser Arbeit beschrieben, die spezifische XML-Vergleichsmethode (*XML-CompA*). Diese kann XML-SG-Instanzen unter Einbeziehung von Kompatibilitätsregeln vergleichen und Aussagen bzgl. der Rückwärtskompatibilität der XML-SG-Instanzen treffen.

Der dritte Schwerpunkt wird in Kapitel 7 beschrieben. Mittels Message Sequence Charts (MSCs) kann dynamisches Verhalten von Steuergeräten spezifiziert werden. Um eine Aussage bzgl. der Rückwärtskompatibilität des, auf diese Weise, spezifizierten Verhaltens zu generieren, wurde eine spezifische MSC-Vergleichsmethode (*MSC-CompA*) definiert.

In Kapitel 8 wird die Implementierung der Konzepte beschrieben. Dabei entstand pro Schwerpunkt ein Software-Modul. In den Kapiteln 9 und 10 werden die beiden Vergleichsmethoden *XML-CompA* und *MSC-CompA* und deren Implementierungen validiert.

Abschließend werden in Kapitel 11 die Ergebnisse der Arbeit noch einmal zusammengefasst und ein Ausblick auf mögliche Erweiterungen gegeben.

1.5 Zusammenfassung

In diesem Kapitel wurde motiviert, weshalb die Betrachtung bzw. die Analyse von Rückwärtskompatibilität von Steuergeräten ein wichtiger Bestandteil zur Reduzierung von Entwicklungs- und Gewährleistungskosten sein kann.

Hierzu wurde zuerst beschrieben, wie stark sich der Elektrik/Elektronik-Umfang im Fahrzeug vergrößert hat und noch vergrößern wird. Im Anschluss daran wurden kurz die Ursachen und die Auswirkungen der Technologisierung erläutert. Ein zentraler Punkt, der erläutert wurde, war die Erhöhung der Entwicklungs- und Gewährleistungskosten aufgrund der steigenden Anzahl an Steuergeräten bzw. deren Varianten.

Um diese Kosten, besser gesagt, die Anzahl der Steuergeräte-Varianten zu reduzieren, wurde in diesem Kapitel die Notwendigkeit der Rückwärtskompatibilität von Steuergeräten unterstrichen. Das Forschungsthema *CompA* hat zum Ziel, eine Methode zur Analyse von Rückwärtskompatibilität von Steuergeräten zu definieren, wobei die Analyse selbst auf Basis der zugehörigen Steuergeräte-Spezifikationen zu erfolgen hat. Die Herausforderungen für diesen Ansatz und die genauen Ziele wurden in diesem Kapitel erläutert.

2 Grundlagen

2.1 Aufbau von Steuergeräten

In der Einleitung (Kapitel 1.1) wurde die Verwendung von Steuergeräten bereits kurz erklärt. Steuergeräte sind eingebettete Systeme, die die zentralen Steuereinheiten im Fahrzeug repräsentieren und mit deren Hilfe eine Vielzahl an komplexen Funktionen realisiert werden.

In diesem Abschnitt wird kurz auf den allgemeinen technischen Aufbau von Steuergeräten eingegangen [121][140]. Im Kern eines jeden Steuergerätes befindet sich ein eingebetteter Micro-Controller. Neben dem Mikroprozessor (heute hauptsächlich 8-Bit-, 16-Bit- und 32-Bit-Prozessoren) befinden sich auf demselben Siliziumchip auch Speicher, Ein- und Ausgabe-Bausteine und z.T. auch Controller für Busse. Eingebettet bedeutet dabei, dass sich die Systeme in ein technisches Umfeld, ohne Bildschirmausgabe und Tastatureingabe, integrieren [61][65]. Auch FPGAs (Field-Programmable-Gate-Array), auf deren Gebiet noch breit geforscht wird [14][11][12][100][13], werden bereits in Steuergeräten eingesetzt [17]. In Abbildung 2.1 ist ein typischer Vertreter eines eingebetteten Steuergerätes dargestellt. Auf dem Steuergerät ist die elektronische Dämpferregelung implementiert. In dem Steuergerät befindet sich der eingebettete Prozessorkern mit einem statischen RAM und einem Flashspeicher. An das Steuergerät angeschlossen sind drei analoge Sensoren und vier Proportionalventile, die den Druck in den Dämpfern je nach Straßenlage ändern können. Zusätzlich besitzt das Steuergerät noch einen CAN-Anschluss mit einer Übertragungsrate von typischerweise 500kBaud [140].

Bei der Spezifikation der Steuergeräte können diese abstrakt betrachtet werden. Für Fahrzeughersteller stellen Steuergeräte Black-Boxes dar. Daher kann ein mögliches Ersatzmodell für Steuergeräte auf der Spezifikations-Ebene wie folgt aussehen. Die Schnittstelle des Steuergerätes (SG) wird hier als *Physikalisches Interface* bezeichnet. Über ein Interface werden Signale empfangen bzw. gesendet. Da SGe unterschiedliche Signale verarbeiten können müssen, wie z.B. Analog-/Digital-Signale, Treiber-Signale, etc. muss das Interface entsprechend spezifiziert werden. Die Signale selbst müssen zum einen bzgl. ihrer Syntax (z.B. Rechtecksignal) und ihrer Semantik (Bedeutung) beschrieben werden (z.B. Taster gedrückt == 0V; Taster nicht gedrückt ==5 V). Des Weiteren sind auch die Funktionen eines Steuergerätes zu definieren. Funktionen können auf unterschiedliche Weise spezifiziert werden, z.B. durch ASCET-, MSC-, UML-Modelle. Werden fertige Funktionsmodelle, z.B. in ASCET, an den Zulieferer übergeben, um diese mit in das Steuergerät zu integrieren, so ist es sinnvoll, auch das zum Funktionsmodell gehörige Funktions-Interface zu spezifizieren.

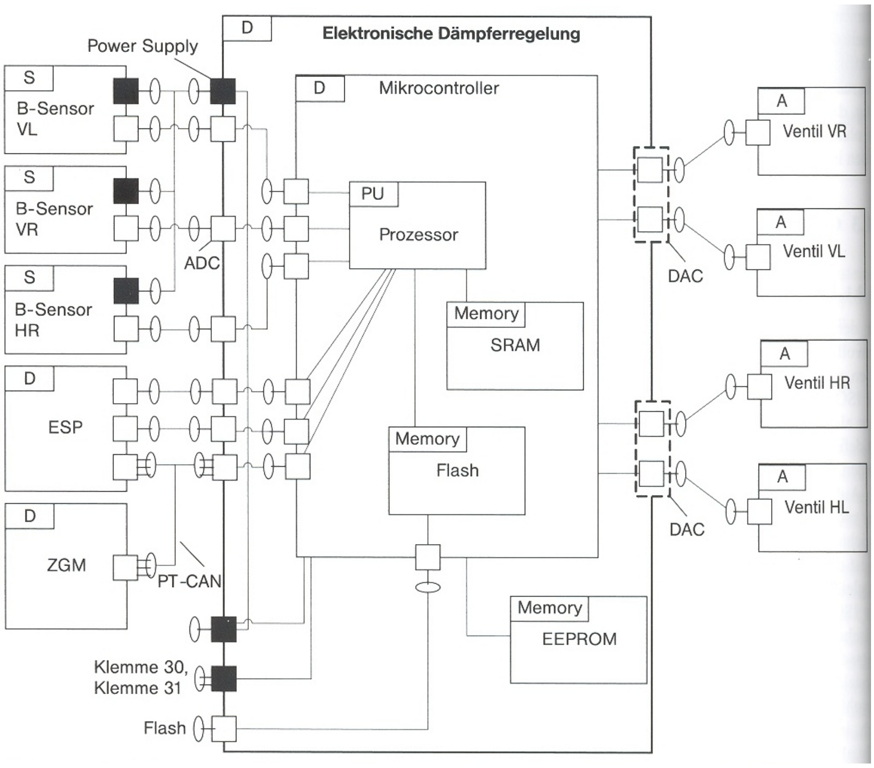


Abbildung 2.1: Aufbau eines Steuergerätes [140]

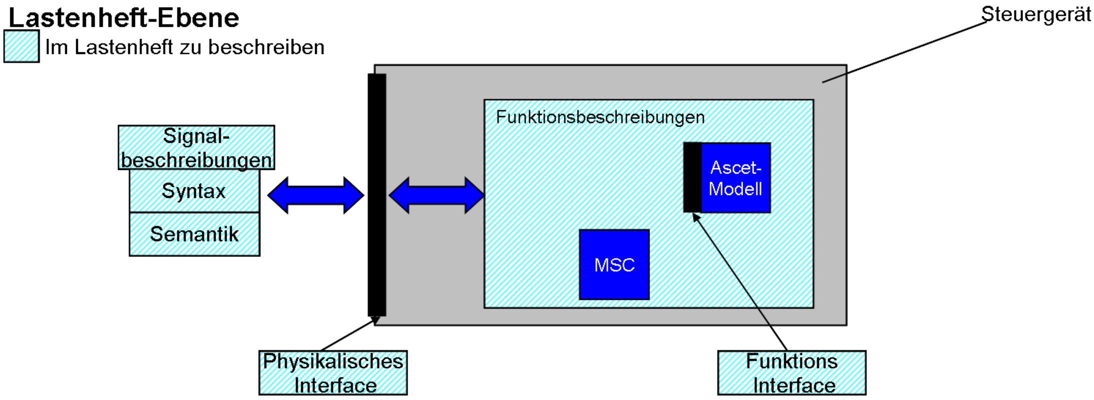


Abbildung 2.2: Abstraktion eines Steuergerätes

Dies soll nur einen kurzen Überblick geben, welche Elemente eines Steuergerätes bei der Spezifikation betrachtet werden müssen.

2.2 Extensible Markup Language (XML)

Auf der Auszeichnungssprache XML (Extensible Markup Language) baut ein Großteil der, im Rahmen von CompA, entwickelten Ansätze auf. Dieses Kapitel dient dazu, einen kurzen Überblick über die XML-Sprache zu geben. Das Grundkonzept von XML ist die Trennung von Inhalt, Struktur und Layout. Der Inhalt ist in den XML-Dokumenten (Kapitel 2.2.2) hinterlegt und die Struktur wird durch ein XML-Schema (Kapitel 2.2.3) beschrieben. Wie der Inhalt der Dokumente dargestellt werden soll, also das Layout, wird mit Hilfe der „Extensible Stylesheet Language Transformations“ (XSLT) [67] (Abschnitt 2.2.5) definiert. In Kapitel 2.2.5 werden noch weitere Methoden vorgestellt, die zum Arbeiten mit XML-Dokumenten und XML-Schema notwendig sind. Bevor auf die Einzelheiten eingegangen wird, wird in Kapitel 2.2.1 kurz die Entstehungsgeschichte von XML beschrieben [86].

2.2.1 Entstehungsgeschichte

Die ersten elektronischen Dokumente wurden auf Basis der sog. *Formatierungssprachen*[1] betrachtet oder ausgedruckt. Der Nachteil der Formatierungssprachen war, dass die *Formatierungscodes* programm- bzw. anwendungsfall-spezifisch waren, so dass Dokumente nur auf bestimmten Anwendungen ausgeführt werden konnten [142]. Dieses Problem wurde durch eine generische Kodierung gelöst, indem statt der spezifischen Formatierungscodes sog. deskriptive Tags verwendet wurden. Dies war der Beginn der *Auszeichnungssprache* (engl. Markup Language, Abk. ML).

Der erste bedeutende Schritt war die Entwicklung der *Generalized Markup Language* (GML), die von Charles Goldfarb, Edward Mosher und Raymond Lorie im Rahmen eines IBM-Projektes definiert wurde [6]. In dieser Auszeichnungssprache kodierte Dokumente konnten auf Grund ihrer inhaltsorientierten Tags von verschiedenen Programmen bearbeitet, formatiert und durchsucht werden.

Auf Basis von GML entstand die Metasprache *Standard Generalized Markup Language* (SGML) [1][34], mit deren Hilfe spezifische Auszeichnungssprachen für Dokumente definiert werden konnten. SGML ist sehr flexibel und sehr umfangreich. Daher war die Software zur Verarbeitung von SGML sehr komplex und so teuer, dass sich deren Anwendung auf große Organisationen beschränkte.

Der Durchbruch für die generische Kodierung erfolgte durch *Hypertext Markup Language* (HTML). HTML ist ein SGML-Dokumenttyp für Hypertext-Dokumente, für die Anwendungssoftware programmiert werden konnte. Allerdings mussten, zur Erreichung der notwendigen Unkompliziertheit, einige Prinzipien der generischen Kodierung „aufgegeben“ werden. So wird beispielsweise bei HTML für sämtliche Anwendungsfälle ein einziger Dokumenttyp benutzt und viele der Tags dienen ausschließlich Präsentationszwecken [101].

Um die Bandbreite der generischen Kodierung zu nutzen, wurde versucht, SGML an

---

[1] Formatierungssprachen können definieren, welche Farbe der Text hat oder was Fett gedruckt werden soll.

das Web oder auch das Web an SGML anzupassen. Dies erwies sich als schwierig. SGML war zu umfangreich, um in einen kleinen Webbrowser zu passen. Mit *Extensible Markup Language* (XML) wurde schließlich eine „kleinere“ Sprache definiert, bei der der verallgemeinernde Charakter von SGML bewahrt blieb [55][7]. Im Februar 1998 wurde XML in der Version 1.0 vom World Wide Web Consortium als Empfehlung verabschiedet [29] und weiterentwickelt [35]. Die aktuelle Version ist *XML 1.1* [33]. Obwohl XML nur ca. 20% der Komplexität von SGML besitzt, können damit ca. 80% der Anwendungsfälle abgedeckt werden [115].

2.2.2 Aufbau von XML-Dokumenten

In Listing 2.1 ist ein einfaches XML-Dokument dargestellt, das einzelne Daten eines Steuergerätes spezifiziert. Alle XML-Dokumente müssen mit einem sogenannten Prolog beginnen <*?xml version=“1.0“ encoding=“UTF-8?“*>. Dieser beinhaltet Metainformationen zum XML-Dokument wie z.B. die XML-Version (1.0) und den Zeichensatz (UTF-8). Optional kann angegeben werden, welches XML-Schema von dem XML-Dokument verwendet wird. XML-Dokumenten liegt eine Baumstruktur[2] zugrunde. So existiert ein Wurzel-Element (<*steuergeraet*>), das wiederum beliebig viele geschachtelte Elemente und Kind-Elemente enthalten kann. Der Inhalt eines XML-Dokuments wird mit Hilfe von deskriptiven Tags beschrieben. Tags sind dabei immer ein 2er-Tupel, bestehend aus einem öffnenden Part (<...>) und einem schließenden Part (< /... >). Zwischen den öffnenden und schließenden Tags (<*Steuergeraet*> ... <*/Steuergeraet*>) werden entweder weitere Kind-Elemente (<*Software*> ... <*/Software*>) oder der Inhalt eines Elements (*HardwareVersion=1.7*) hinterlegt. In Zeile 10 ist eine weitere Funktionalität zu erkennen - das XML-Attribut (*„nr“*). Mit dem XML-Attribut können weitere Informationen einem Element zugewiesen werden. Jedes Element kann ein oder mehrere Attribute, die als 2er-Tupel (Attributname und -wert) realisiert sind, enthalten. Der Attributwert steht dabei stets in Hochkommata.

Listing 2.1: Beispiel für ein XML-Dokument

```
1  <?xml version =”1.0” encoding=”UTF-8”?>
2  <Steuergeraet>
3      <Software>
4          <SoftwareVersion>1.1</SoftwareVersion>
5          <Boot-Software>2.0</Boot-Software>
6          <Applikationsdaten>210206</Applikationsdaten>
7      </Software>
8      <HardwareVersion>1.7</HardwareVersion>
9      <PINs>
10         <Pin nr=”1”>Klemme 15</Pin>
11         <Pin nr=”2”>Klemme 30</Pin>
12     </PINs>
```

---

[2] In XML wird als Synonym für Knoten die Bezeichnung Element verwendet.

```
</Steuergeraet>
```

Bei XML-Dokumenten werden zwei Arten von Korrektheit unterschieden, Wohlgeformtheit (well-formed) und Gültigkeit (valid) [129]:

- *Wohlgeformtheit (well-formed):* Ein XML-Dokument ist wohlgeformt, wenn die Syntax korrekt eingesetzt ist, d.h. dass alle öffnenden Tags über ein schließendes Gegenstück verfügen. Ferner müssen alle Tags geschachtelt angeordnet werden und es darf im gesamten XML-Dokument nur ein root-Element vorkommen.
- *Gültigkeit (valid):* Ein XML-Dokument ist gültig, wenn die benutzte Tag-Struktur der Anwendungslogik dem inhärenten Sinn des Dokuments genügt, d.h. soweit die Regelungen des Entwicklers erfüllt sind. Diese Regeln befinden sich im XML-Schema-Dokument.

Möchte man mehrere XML-Dokumente erstellen, die alle nach den gleichen Regeln aufgebaut sind, so definiert man dies mit Hilfe von XML-Schemata.

2.2.3 Aufbau von XML-Schemata

Im XML-Schema werden Vorgaben getroffen, welche Struktur XML-Dokumente haben und welche Elemente in den XML-Dokumenten auftauchen dürfen. Im Folgenden wird auf die, für diese Arbeit relevanten, Punkte eingegangen. Für spezifische Themen bzgl. XML-Schema sei auf [129][33][139] verwiesen.
In Listing 2.2 ist das zum XML-Dokument aus Listing 2.1 passende XML-Schema angegeben. Anhand dieses Beispiels werden zwei Schlüsseldefinitionen von XML-Schemata erläutert:

- Inhaltsmodell und
- Datentypen

Listing 2.2: Das zu Listing 2.1 gehörige XML-Schema

```
<xs:schema xmlns:xs=http://www.w3.org/2001/XMLSchema>
<xs:element name=”Steuergeraet”>
    <xs:complexType>
        <xs:sequence>
            <xs:element name=”Software”>
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name=”SoftwareVersion” type=”xs:string”>
                        <xs:element name=”Boot-Software” type=”xs:string”>
                        <xs:element name=”Applikationsdaten” type=”xs:string”>
```

```
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name=”HardwareVersion” type=”xs:string”/>
            <xs:element name=”PINs”>
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name=”Pin” minOccurs=”0” maxOccurs=”7”>
                            <xs:attribute name=”nr” type=”xs:string”/>
                        </xs:element>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

Inhaltsmodelle

Ein Inhaltsmodell ist die Beschreibung einer Struktur von Kindelementen und Textknoten (unabhängig von Attributen). Das Inhaltsmodell [139] heißt:

- *einfach*, wenn es einen Textknoten, aber keine Elemente gibt,
- *komplex*, wenn es Elementknoten, aber keinen Text gibt,
- *gemischt*, wenn es Text- und Elementknoten gibt,
- *leer*, wenn es weder Text- noch Elementknoten gibt.

Im Folgenden wird das einfache und komplexe Inhaltsmodell kurz erläutert:

Einfaches Inhaltsmodell Ein Element hat ein einfaches Inhaltsmodell, wenn es lediglich einen Textknoten als Kind enthält, also keine weiteren Unterelemente. In Listing 2.2 hat das Element *<xs:element name=”HardwareVersion” type=”xs:string”/>* ein einfaches Inhaltsmodell.

Komplexes Inhaltsmodell Das komplexe Inhaltsmodell beschreibt die Struktur des Inhalts eines komplexen Elements (*complextype*), indem es festlegt, ob und an welcher Position die untergeordneten Elemente auftreten dürfen oder müssen. Folgende Strukturelemente sind dafür vorgesehen:

- *all* erlaubt das Auftreten der Kind-Elemente höchstens einmal und in beliebiger Reihenfolge. Bei der Verwendung von *all* sind einige Einschränkungen zu beachten. So darf *all* nur auf der obersten Ebene des Inhaltsmodells verwendet werden, wobei *all* als einziges Kind am Anfang eines Inhaltsmodells auftritt. Außerdem dürfen als Kind-Elemente nur einzelne Elemente auftreten, weitere Inhaltsmodelle sind nicht zulässig.

- *choice* legt fest, dass ein Element aus der Auswahlgruppe genau einmal ausgewählt werden darf.

- *sequence* erzwingt das Auftreten der Kind-Elemente in der vorgegebenen Reihenfolge. Dabei sind neben einzelnen Elementen auch weitere Inhaltsmodelle als Kind-Elemente erlaubt.

- *group* ermöglicht, eine Gruppe von Elementen, also die drei vorhergehenden Modelle, mit einem Namen zu versehen und zu referenzieren. Dadurch ist es möglich, die Gruppe wieder zu verwenden.

Zusätzlich können den Inhaltsmodellen die Attribute *minOccurs* oder *maxOccurs* zugewiesen werden (z.B. *<xs:element name=”Pin” minOccurs=”0” maxOccurs=”7”/>*). Dadurch lässt sich im XML-Schema festlegen, wie oft Elemente und deren Unterbäume in einem XML-Dokument auftreten dürfen. Das Attribut $(min - /maxOccurs > 1)$ wird jedoch nicht mit in das XML-Dokument übernommen. Zur Unterscheidung der Elemente im XML-Dokument werden folgende Begrifflichkeiten eingeführt:

- *ordered-Elemente:* Elemente, deren Position in eine XML-Dokument durch das XML-Schema fest vorgegeben ist.

- *unordered-Elemente:* Elemente, die aufgrund des Attributs $minOccurs > 1$ und/oder $maxOccurs > 1$ mehrfach in einer XML-SG-Instanz instanziiert werden dürfen und bei denen eine Links-Rechts-Vertauschung valide ist.

Datentypen

XML-Schema bietet eine Vielzahl an Basisdatentypen an. Es besteht aber auch die Möglichkeit, eigene Datentypen zu definieren. Datentyp ist ein von *W3C XML Schema* verwendeter Ausdruck, mit dem sowohl der Inhalt als auch die Struktur eines Elements oder eines Attributs beschrieben werden kann. Datentypen werden als

- *einfach* bezeichnet, wenn sie ein Attribut oder ein Element ohne eingebettetes Element oder Attribut beschreiben oder als

- *komplex* bezeichnet, wenn sie Elemente mit eingebetteten Kindelementen oder Attributen beschreiben [139].

Einfacher Datentyp Einfache Datentypen sind beispielsweise String, Real, etc. (Bsp.:*<xs:element name=”HardwareVersion” type=”xs:string”/>* (vgl. Listing 2.2).

Komplexe Datentypen Komplexe Datentypen werden im XML-Schema durch *<complexType>* gekennzeichnet. In Listing 2.2 ist *<Software>* vom Typ *<complexType>* und wird durch die Elemente *<Softwareversion>*, *<Boot-Software>*, *<Applikationsdaten>* genauer spezifiziert.

Spezifische Besonderheiten

Namespaces XML-Schema unterstützt Namespaces. Der Namespace wird in der ersten Zeile durch das Präfix *xmlns* definiert *<xs:schema xmlns:xs=http://www.w3.org/2001/XMLSchema>*. Der Vorteil von Namespaces ist, dass dadurch Variablen mit gleichem Namen über ihren Namensraum eindeutig identifiziert und qualifiziert referenziert werden können.

ID/IDREF Elemente, die an verschiedenen Stellen im Schema definiert sind, können trotzdem in Beziehung stehen. Ein Beispiel ist, dass Signale und Pins von Steuergeräten an unterschiedlichen Stellen im XML-Schema definiert sind. Um nun abzubilden, dass ein bestimmtes Signal an einem bestimmten Pin anliegt, gibt es im XML-Schema die Möglichkeit, diese Abhängigkeit durch ID und $IDREF$ darzustellen.

include Durch den include-Befehl können weitere XML-Schemata, die in den Datentypen definiert sind, eingebunden werden.

restriction Das restriction-Element wird verwendet, um einen bestehenden Datentyp auf eine spezifische Wertemenge einzuschränken.

Listing 2.3: Beispiel für restriction-Element

```
....
    <xsd:simpleType name=”Kabel”>
        <xsd:restriction base=”ecu:stringMax64”>
            <xsd:enumeration value=”Kupfer”/>
            <xsd:enumeration value=”Glasfaser”/>
        </xsd:restriction>
    </xsd:simpleType>
...
```

In dem Beispiel ist das Element $Kabel$ vom Typ $StringMax64$. Die mögliche Wertemenge von $StringMax64$ wird durch *restriction* auf die Wertemenge $Kupfer$ und $Glasfaser$ eingeschränkt.

2.2.4 Rekursives- und Nicht-Rekursives XML-Schema

Ein XML-Schema kann *rekursiv* oder *nicht rekursiv* aufgebaut werden. Beide Strukturierungen haben gewisse Vor- und Nachteile. Der Aufbau und die Eigenschaften der beiden Schema-Arten wird im Folgenden genauer erläutert.

Nicht-Rekursives XML-Schema In Abbildung 2.3 ist ein nicht-rekursives XML-Schema dargestellt. Das XML-Schema besitzt ein Vater-Element $< steuergeraet >$ und vier Kind-Elemente, die wiederum Kind-Elemente besitzen. Ein nicht-rekursives XML-Schema ist dadurch gekennzeichnet, dass keine Schleifen innerhalb der Baumstruktur auftreten, d.h. dass die Kind-Elemente immer direkte „Nachfolger“ eines Vater-Elements sind und nicht „Nachfolger“ von anderen Elementen in der Baumstruktur. Ein nicht-rekursives XML-Schema bietet verschiedene Vorteile:

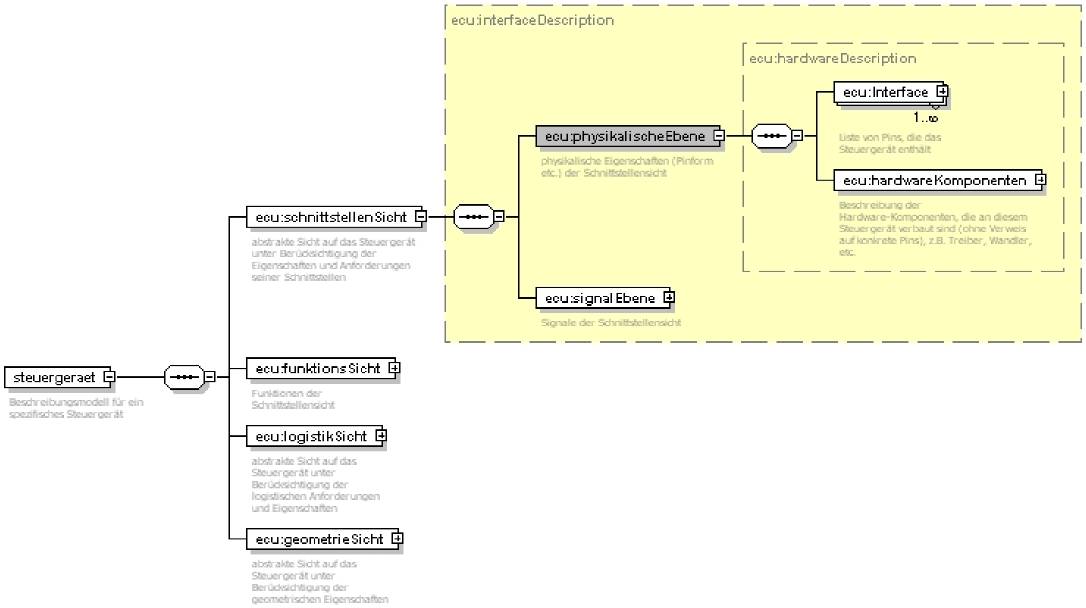


Abbildung 2.3: Beispiel für ein Nicht-Rekursives XML-Schema

- Hierarchische Struktur: Ein nicht-rekursives Schema ist leicht verständlich und übersichtlich, da die Kind-Elemente als logische Folgerungen der Vater-Elemente angeordnet werden können.
- Ordnungsstruktur: Die Struktur der zugehörigen Instanzen ist eindeutig.

Rekursives XML-Schema Ein rekursives Schema weist Schleifen innerhalb des XML-Schemas auf. Rekursive XML-Schemata sind sehr mächtig, da sie auf effiziente Weise einen großen Raum der Abbildungen, durch Kaskadierung von Elementen ermöglichen. Nachteil der rekursiven Schemata ist, dass die Strukturen der XML-Dokumente nicht eindeutig sind. Dies sei am Beispiel eines XML-Schemas für „Message Sequence Charts“ erklärt. In Abbildung 2.4 ist ein Ausschnitt des XML-Metamodells bzw. -Schemas von MSC (Message Sequence Charts) dargestellt. Man sieht, dass mehrere Elemente direkt unter dem Wurzelknoten angeordnet sind, d.h. das Schema besitzt mehrere Vater-Elemente. Das XML-Schema ist so konstruiert, dass

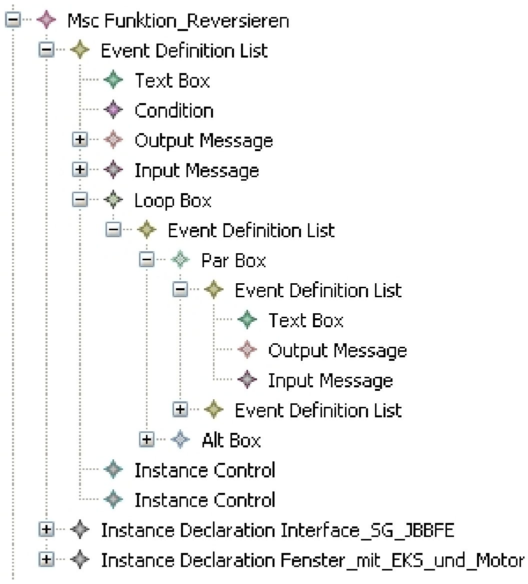
element	ActionBox
element	AltBox
element	BoxDefinition
element	EventDefinition
element	Condition
element	EventDefinitionList
element	ExcBox
element	Gate
element	InputMessage
element	Instance
element	InstanceControl
element	InstanceDeclaration
element	InstanceDescription
element	LoopBox
element	Msc
element	MscParameter
element	MscTextualFile
element	Message
element	OptBox
element	OutputMessage
element	Parameter
element	ParBox
element	Reference
element	ReferenceDeclaration
element	SeqBox
element	TextBox
element	TimerEvent
element	TimeConstraint
complexType	ActionBox
complexType	AltBox
complexType	BoxDefinition
complexType	Condition
complexType	EventDefinition
complexType	EventDefinitionList
complexType	ExcBox

Abbildung 2.4: Beispiel eines rekursiven Schemas für Messages Sequence Charts (MSC)

Vater-Elemente andere Vater-Elemente als Kind-Elemente referenzieren können. Deutlich wird dies, wenn man ein zugehöriges XML-Dokument betrachtet. In Abbildung 2.5 ist zu sehen, dass unter dem Knoten „Loop Box“ als Kindknoten das Element „Par Box“ liegt und dieses wiederum das Kind-Element „Text Box“ enthält. Hier wird deutlich, wie mächtig rekursive XML-Schemata sind und wie durch Kaskadierung unendlich viele Darstellungsmöglichkeiten entstehen können.
Jedes XML-Dokument eines rekursiven XML-Schemas ist immer nicht-rekursiv.

2.2.5 Operationen auf XML-Dokumenten

Im ersten Schritt muss ein bestehendes Dokument durch ein Programm eingelesen und ein Dokument-Objekt (DOM) erzeugt werden [95]. Anhand dieses Objekts kann mittels Methoden einer API auf Inhalte, Struktur und Darstellung zugegriffen werden.



- Msc Funktion_Reversieren
  - Event Definition List
    - Text Box
    - Condition
    - Output Message
    - Input Message
    - Loop Box
      - Event Definition List
        - Par Box
          - Event Definition List
            - Text Box
            - Output Message
            - Input Message
          - Event Definition List
        - Alt Box
    - Instance Control
    - Instance Control
  - Instance Declaration Interface_SG_JBBFE
  - Instance Declaration Fenster_mit_EKS_und_Motor

Abbildung 2.5: Beispiel für ein XML-Dokument eines MSCs, abgeleitet aus dem rekursiven XML-Schema von Abbildung 2.4 (dargestellt mit EMF-Eclipse-Editor)

DOM *Document Object Model (DOM)* [32] ist ein objektorientiertes Modell für XML-Dokumente einschließlich einer Definition einer API zur Verarbeitung des Document Object Models. DOM erlaubt die Navigation zwischen einzelnen Knoten eines Dokuments, das Erzeugen, Verschieben und Löschen von Knoten sowie das Auslesen, Ändern und Löschen von Textinhalten. Am Ende der Verarbeitung kann durch sogenannte Serialisierung aus DOM heraus ein neues XML- oder HTML-Dokument generiert werden.

XSLT *XML Stylesheet Language Transformations* (XSLT) [31] ist ein XML-Standard und ermöglicht die Definition von Regeln zur Umwandlung eines XML-Dokumentes in ein beliebiges anderes Format [66].

XPath XML-Dokumente sind baumartig strukturiert. Mit der *XML Path Language* (XPath) [30] steht eine Methode zur Verfügung, mit der einzelne Elemente über Ihre Pfade identifiziert und angesprochen werden können. Haupteinsatzgebiet von XPath ist XSLT, da es eine einfache Navigation innerhalb von XML-Dokumenten und einen flexiblen Umbau dieser Dokumente ermöglicht.

XSD-API Die Eclipse XSD API [76] umfasst eine vollständige Java-Bibliothek zur Bearbeitung, Erzeugung, Auswertung und Überprüfung der Korrektheit von Schemata (nach W3C). Die Bibliothek ist unabhängig von Eclipse und kann daher auch ohne diese Plattform verwendet werden [126].

2.3 Message Sequence Chart (MSC)

Message Sequence Charts (MSC) dienen der Beschreibung bzw. Spezifikation von Interaktionen zwischen Systemkomponenten auf Basis von „Message Flows“ (Datenflussdiagrammen). Das Hauptanwendungsgebiet von MSCs war anfangs die Spezifikation von Datenaustausch auf Basis von Protokollen. MSCs werden derzeit auch verwendet, um dynamisches Verhalten am Interface von Steuergeräten zu spezifizieren. In Kapitel 2.3.1 wird eine kurze Einführung zu MSCs gegeben. Anschließend wird auf einzelne Beschreibungs-Elemente von MSCs genauer eingegangen (vgl. Kapitel 2.3.2 und 2.3.3). Abschließend wird in Kapitel 2.3.4 die Formalisierung der Message Sequence Charts vorgestellt.

2.3.1 Einführung in MSCs

Message Sequence Charts (MSC) wurden auf Basis der OSI-Time Sequence Diagrams [56][77] entwickelt. Das Ergebnis war eine vollständige graphische Sprache mit formaler Syntax und Semantik, die erstmals in der ITU-T Empfehlung 1993 [78] beschrieben wurde. Es folgten weitere Überarbeitungen 1996 [79] und 2000 [80]. Mittels MSCs kann immer nur ein Scenario eines System-Verhaltens beschrieben werden. Das Gesamt-Verhalten eines MSCs kann jedoch über ein Set von MSCs beschrieben werden.

In den folgenden Kapiteln wird ein kurzer Überblick über die MSC-Sprachkonstrukte gegeben und einzelne davon detailliert vorgestellt[3]. MSC-Sprachkonstrukte werden in die Bereiche *Basic-MSC* und *Strukturelle Sprachkonstrukte* gegliedert [58] (vgl. Abbildung 2.6). Einen guten Überblick über die Erstellung von MSCs bietet das MOST Cookbook [104].

2.3.2 Basic-MSCs

Basic-MSCs umfassen alle Sprachkonstrukte, die notwendig sind, um Nachrichtenflüsse zu spezifizieren. Diese Sprachkonstrukte sind: $Instanz$, $Message$, $Environment$, $Action$, $Timer - Start$, $Timeout$, $Timer - Stop$ und $Condition$.

In Abbildung 2.7 ist ein MSC dargestellt, das eine fiktive Ampelschaltung spezifiziert: Steht eine $Fussgaenger\_Ampel$ auf Rot, so kann ein $Fussgaenger$ das Umschalten der Ampel anfordern $Request\_umschalten$. Ist dies der Fall, wird die $Auto\_Ampel$ auf Rot geschaltet $Request\_Umschalten\_AutoAmpel\_Rot$. Anschließend wird 10 sec gewartet und dann wird die $Fussgaenger\_Ampel$ auf $Gruen$ geschaltet.

Die einzelnen Elemente werden in den nächsten Abschnitten erklärt.

---

[3] Die restlichen Definitionen für die MSC-Sprachkonstrukte sind in [80] nachzulesen.

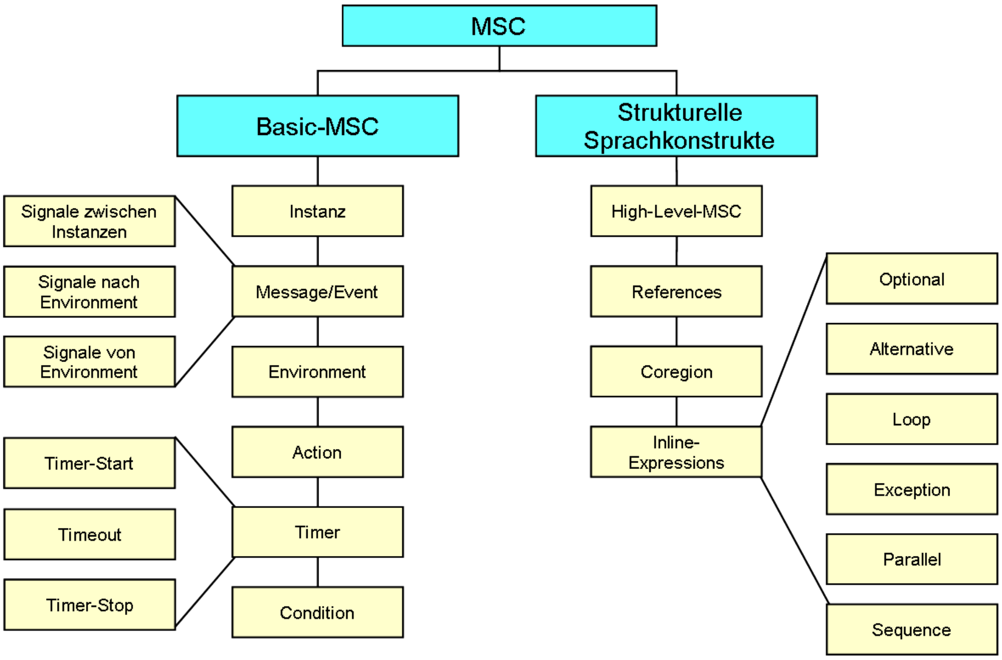


Abbildung 2.6: Überblick über MSC-Sprachkonstrukte

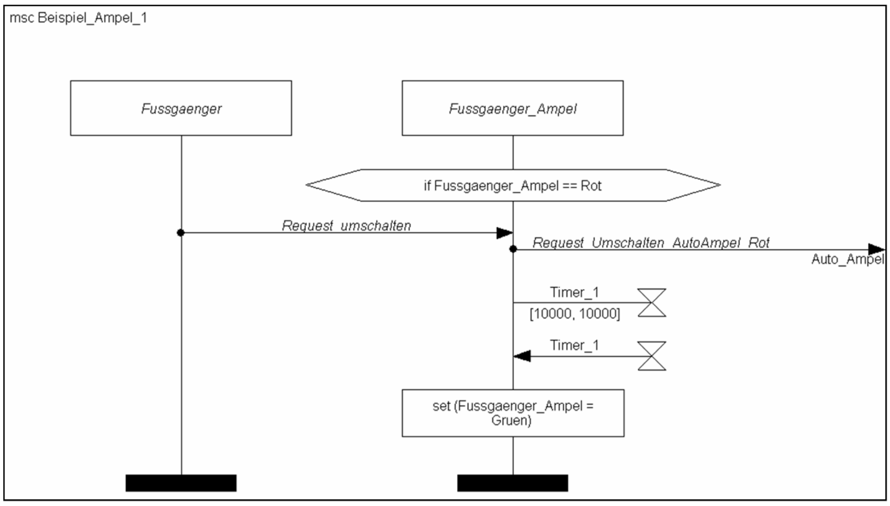


Abbildung 2.7: Beispiel für eine fiktive Ampelschaltung, aufgebaut mit Basic-MSCs

Instanzen $Instanzen$ repräsentieren Systeme, Komponenten, Steuergeräte, etc., die untereinander oder mit der Systemumgebung asynchron Messages (Nachrichten) austauschen. In der graphischen Form werden Instanzen als vertikale Zeitachsen dargestellt, an der die Zeit von oben nach unten fließt. Es wird also eine Totalordnung bzgl. der spezifizierten Message-Sende- und Message-Empfangsereignisse angenommen. Innerhalb des Instanzkopfes wird der Instanzname spezifiziert. Das Ende einer Instanz wird durch ein Instanz-Ende-Symbol (schwarzer Balken) beschrieben. Ein Instanz-Ende bedeutet nicht, dass die Instanz gestoppt ist, sondern lediglich, dass keine weiteren Ereignisse mehr beschrieben werden. Instanzen in Abbildung 2.7 sind $Fussgaenger$ und $Fussgaenger\_Ampel$.

Messages $Messages$ werden als gerichtete Kanten bzw. Pfeile dargestellt. Jede Kante definiert dabei zwei Ereignisse. Der Kantenanfang beschreibt das Senden und das Kantenende das Empfangen einer Message. Messages können zwischen verschiedenen MSC-Elementen ausgetauscht werden:

- Messages zwischen Instanzen (z.B. $Request\_umschalten$, Abbildung 2.7)
- Messages, die von/an Environment (Systemumgebung) empfangen/gesendet werden (Bsp:$Request\_Umschalten\_AutoAmpel\_Rot$, Abbildung 2.7)
- Messages, die von/an Gates empfangen/gesendet werden

Environment Ein MSC wird durch einen rechteckigen Rahmen begrenzt. Dieser Rahmen definiert die Systemumgebung und wird als $Environment$ bezeichnet. Messages, die vom/an das Environment empfangen/gesendet werden, beginnen bzw. enden auf dem $Environment$. Im Gegensatz zur Totalordnung entlang der Instanzachsen ist für die Sender- und Empfangsereignisse auf dem $Environment$ keine Ordnung definiert.

Action Mit $Action$ können Aktionen auf Instanzen spezifiziert werden, die zum betreffenden Zeitpunkt ausgeführt werden sollen. Die graphische Notation ist ein Rechtecksymbol, das beliebige Ausführungsanweisungen enthalten kann (vgl. $set(Fussgaenger\_Ampel = Gruen)$, Abbildung 2.7)

Timer Zur Beschreibung von Zeitabhängigkeiten bieten MSCs bspw. die Sprachkonstrukte *Timer-Start*, *Timeout* und *Timer-Stop* an. *Timer-Start* spezifiziert das Setzen, $Timeout$ den Ablauf und *Timer-Stop* das Zurücksetzen eines Timers. Ein $Timer$-Konstrukt ist dabei immer einer Instanz zugeordnet (vgl. $Timer\_umschalten$, Abbildung 2.7).

Condition Eine $Condition$ dient zur Spezifikation von Bedingungen, unter denen ein Durchlaufen auf der Instanz „erlaubt“ ist. Graphisch wird eine $Condition$ als Sechseck dargestellt (vgl. *if Fussgaenger_Ampel == Rot*, Abbildung 2.7).

Comment Ein $Comment$ ist in [80] definiert, zählt aber nicht zu den Basic-MSC-Sprachkonstrukten. Da er in der Praxis häufig benutzt wird, wird er hier mit aufgeführt. Er hat keinen Einfluss auf die Aussage der Spezifikation. Die graphische Darstellung ist ein Rechteck mit einer eingeknickten Ecke.

2.3.3 Strukturelle Sprachkonstrukte

Strukturelle Sprachkonstrukte bezeichnen Sprachelemente, mit denen sich komplexe Sequenzen abbilden lassen. In Abbildung 2.8 ist ein MSC mit strukturellen Sprachelementen dargestellt. Ein $Fussgaenger$ fordert die Umschaltung der Ampel $Request\_umschalten$. Nun werden zwei Alternativen unterschieden.
1. Alternative: Gilt *Fussgaenger_Ampel==Rot*, so wird das Umschalten der $Auto\_Ampel\_1$ und $Auto\_Ampel\_2$ durch $Request\_Umschalten\_AutoAmpel\_Rot$ angefordert und anschließend wird die *Fussgaenger_Ampel=Gruen* geschaltet.
2. Alternative: Ist die *Fussgaenger_Ampel==Gruen*, so wird dem $Fussgaenger$ eine Rückmeldung $Umschalten\_nicht\_notwendig$ gegeben.
In den folgenden Abschnitten werden einzelne ausgewählte Sprachkonstrukte vorgestellt.

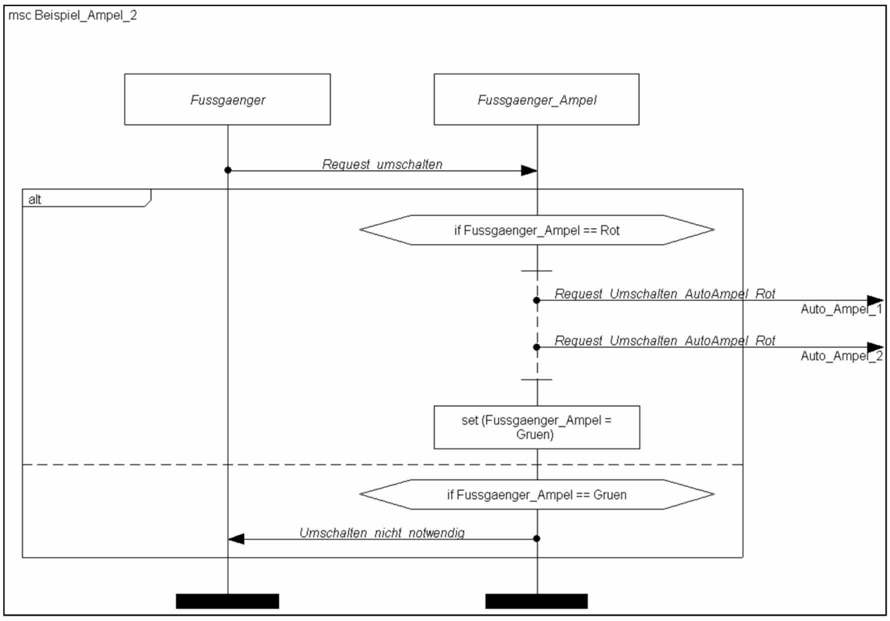


Abbildung 2.8: Beispiel für eine fiktive Ampelschaltung, aufgebaut mit Basic-MSCs und einer Inline-Expression

Inline-Expressions

Mit $Inline-Expressions$ können innerhalb von MSCs komplexe Sequenzen definiert werden. Hierzu werden analog zu Programmiersprachen verschiedene Operatoren angeboten: $alt, par, loop, opt, exc$ und seq[4]. Graphisch werden diese *Inline-Expressions* als Rechtecke dargestellt. Teilabläufe innerhalb der Operatoren werden mit gestrichel-

---

[4] seq-Operator ist der Vollständigkeit halber dargestellt, findet aber in der Praxis kaum Anwendung. Daher wird er im Folgenden nicht weiter betrachtet

ten Linien separiert und als *Sections* bezeichnet. Der Operator selbst wird im linken oberen Eck spezifiziert.

Optional Mit Hilfe des *Optional*-Operators (kurz: *opt*) können Sequenzen als *Optional* dargestellt werden, d.h. diese können, müssen aber nicht ausgeführt werden. In der Praxis empfiehlt sich, bei einem *Optional*-Operator eine zusätzliche, eindeutige Eintrittsbedingung durch eine *Condition* anzugeben [104].

Alternative Der *Alternative*-Operator (kurz: *alt*) bietet die Möglichkeit, verschiedene alternative Sequenzen darzustellen. Die alternativen Sequenzen werden jeweils in einer einzelnen *Section* spezifiziert. Die *Sections* werden innerhalb eines Operators durch eine gestrichelte Linie getrennt. Um zu unterscheiden, welche alternative Sequenz ausgeführt werden soll, empfiehlt es sich, eine *Condition* z.B. *if Fussgaenger_Ampel == Rot* als Eintrittsbedingung zu verwenden (vgl. Abbildung 2.8).

Loop Innerhalb des *Loop*-Operators (kurz: *loop*) können Sequenzen definiert werden, die wiederholt ausgeführt werden sollen. Hierbei können folgende Wiederholungen spezifiziert werden:

1. $loop < l, u >$**:** $l, u \in \mathbb{N}; l \leq u$
   l ist hierbei die minimale Anzahl an Durchläufen und u die maximale Anzahl an Durchläufen.

2. $loop < l >$: $l \in \mathbb{N}$
   Diese Bezeichnung ist eine Abkürzung für $loop < l, l >$ und bedeutet, dass exakt l Wiederholungen stattfinden sollen.

3. $loop$ Dies ist eine Abkürzung für $loop < 1, inf >$, d.h. die Schleife tritt mindestens einmal bis maximal unendlich (inf) auf.

Exception Eine *Exception* (kurz: *exc*) bietet die Möglichkeit der Ausnahmebehandlung. Wird *Exception* durchlaufen, bricht der Durchlauf des MSC nach dem *exc*-Operator ab.

Parallel-Block Mit Hilfe des *Parallel*-Operators (kurz: *par*) können parallel laufende Sequenzen abgebildet werden. Die parallelen Abläufe stehen in einzelnen Sections.

Coregion

Bei Ereignissen/Messages, die innerhalb einer *Coregion* definiert werden, ist die Totalordnung der Ereignisse außer Kraft, d.h. innerhalb der *Coregion* ist die Reihenfolge, in der Messages empfangen/gesendet werden, nicht relevant.

References

$References$ ermöglichen es, MSCs in anderen MSCs wieder zu verwenden. Eine $Reference$ referenziert ein anderes MSC über dessen Namen.

High-Level-MSC

High-Level-MSC (*HMSC*) ermöglichen die Spezifikation einer Kombination von mehreren MSCs auf abstrakter Ebene. Da *HMSCs* in Rahmen von CompA keine große Rolle spielen, wird hier für weitere Details auf [80][104] verwiesen.

2.3.4 Formalisierung von MSCs

Ein MSC kann als gerichteter, nicht zyklischer Graph wie folgt definiert sein [3][128][40]:

- P ist eine endliche Menge von *Instanzen*.
- T ist eine endliche Menge von *Sende-Ereignissen* und R eine endliche Menge von *Empfangs-Ereignissen*, mit $T \cap R = \emptyset$. Die Menge $T \cup R$ wird mit E bezeichnet und repräsentiert die Menge aller Ereignisse *e*.
- $o : E \to P$ ordnet jedem Ereignis e eine eindeutige Instanz $o(e) \in P$ zu. Die Menge der Ereignisse, die einer Instanz p zugeordnet werden, sei mit E_p bezeichnet.
- $c : S \to R$ ist eine bijektive Abbildung, die jedem Sende-Ereignis t genau ein Empfangs-Ereignis $c(t)$ und jedem Empfangs-Ereignis r genau ein Sender-Ereignis $c^{-1}(r)$ zuordnet.
- $<_p$ ist eine lokale Totalordnung, die für alle $p \in P$ die Ereignisse E_p linear ordnet. Die Ordnung korrespondiert mit der im MSC abgebildeten Ordnung.

An dieser Stelle sei auch der Begriff Ereignissequenz definiert:

- Eine Ereignissequenz s sei die Reihenfolge $<_p$ in der Ereignisse *e* gesendet bzw. empfangen werden. Die Menge der möglichen Ereignissequenzen eines MSCs bezeichnen wir künftig mit S.

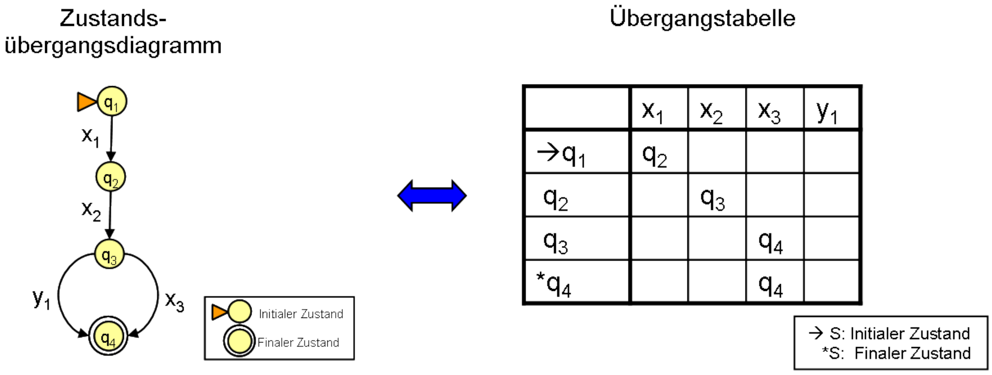
2.4 Endliche Automaten

Endliche Automaten (EA) beschreiben ein Verhaltens-Modell bestehend aus Zuständen, Zustandsübergängen und Aktionen. Automaten werden als endlich bezeichnet, wenn die Menge der Zustände, die angenommen werden können, endlich ist. Zustände speichern Informationen über die Vergangenheit, d.h. sie reflektieren

die Änderungen der Eingabe seit dem Systemstart bis zum aktuellen Zeitpunkt. Zustandsübergänge stellen die Verbindungen zwischen Zuständen her und werden durch logische Bedingungen beschrieben, die erfüllt sein müssen, um einen Übergang zu ermöglichen.
Endliche Automaten können als

- Zustandsübergangsdiagramm oder als
- Übergangstabelle

visualisiert werden (vgl. Abbildung 2.9).



	x_1	x_2	x_3	y_1
→ q_1	q_2			
q_2		q_3		
q_3			q_4	
*q_4			q_4	

→ S: Initialer Zustand
*S: Finaler Zustand

Abbildung 2.9: Gegenüberstellung von Übergangstabelle und Zustandsdiagramm

2.4.1 Klassifizierung

Generell werden bei endlichen Automaten zwei Gruppen von endlichen Automaten unterschieden:

- *Transduktoren* generieren Ausgaben in Abhängigkeit von Zustand und/oder Eingabe. Hierbei werden zwei Typen unterschieden.
  - *Moore-Automat*: Im Moore-Automaten ist die Ausgabe abhängig vom Zustand
  - *Mealy-Automat*: Die Ausgabe ist hier abhängig vom Zustand und von der Eingabe
- *Akzeptoren* akzeptieren und erkennen die Eingabe und signalisieren durch ihren Zustand das Ergebnis nach außen.

Im Rahmen dieser Arbeit werden Message Sequence Charts in Automaten transformiert. Bei den erzeugten Automaten spielt die Ausgabe der Automaten keine Rolle und daher kommen die Akzeptoren zum Einsatz. In der Automatentheorie können die endlichen Automaten noch weiter unterschieden werden:

- *Nicht-Deterministische Endliche Automaten (NEA):* Ein nicht-deterministischer Automat ist in der Lage, gleichzeitig über mehrere Zustände zu verfügen.
- *ϵ-Nicht-Deterministische Automaten (ϵ-NEA):* Das neue Leistungsmerkmal besteht darin, dass Übergänge für leere Zeichenreihen ϵ zugelassen sind. Dies bedeutet, dass ein NEA spontan in einen anderen Zustand wechseln kann, ohne ein Eingabesymbol empfangen zu haben, sofern ein ϵ-Übergang vorhanden ist.
- *Deterministische Endliche Automaten (DEA):* Bei einem deterministischen Automaten kann ein Zustand über eine Übergangsbedingung in genau einen Zustand übergehen.

2.4.2 Formalisierung

Alle endlichen Automaten können formal wie folgt beschrieben werden [127][74]:
$A = (\mathrm{Q}, \Sigma, \delta, q_0, F)$, wobei

- Q eine endliche Menge von Zuständen q_i ist,
- Σ eine endliche Menge von Eingabesymbolen σ ist,
- q_0, ein Element von Q, der initiale Zustand (Startzustand) ist,
- F, eine Teilmenge von Q, die Menge der finalen (oder akzeptierenden) Zustände ist,
- $\delta : Q \times \Sigma \rightarrow Q$, die Menge der Übergangsfunktionen ist.

2.4.3 Grundlagen Automatentheorie

Die Automatentheorie stellt eine Reihe von Algorithmen zur Verfügung, mit Hilfe derer endliche Automaten ineinander überführt werden können [18].

ϵ-Eliminierung ϵ-NEA enthalten ϵ-Übergänge, die spontan ihren Zustand wechseln können. Diese Übergänge können mit Hilfe der ϵ-Eliminierung „entfernt“ werden. Dies erfolgt, indem die ϵ-Übergänge zu ϵ-Hüllen zusammmengefasst und ausgehend von diesen die Übergänge neu berechnet werden. Der Algorithmus und der Beweis ist nachzulesen in [74].

Determinierung Mittels der Determinierung kann jeder NEA in einen DEA überführt werden.

Minimierung Zu jedem DEA existiert ein eindeutiger minimaler Automat, der dieselbe Sprache akzeptiert. Die Minimierung erfolgt durch eine fortwährende Verfeinerung der Äquivalenzklassen und ist in [74] detailliert beschrieben.

2.4.4 Verknüpfung von Automaten

Bei der Transformation der MSCs spielt das Konstruieren der erlaubten Sprache L bzw. das „Verbinden“ von Automaten eine große Rolle. Im Folgenden werden zwei Ansätze unterschieden [116][19]:

- Vereinigung von zwei Automaten A_1, A_2 zu Automat A
  Der vereinigte Automaten A akzeptiert alle Wörter bzw. die Sprache L des Automaten A_1 und des Automaten A_2. Es gilt $L(A) = L(A_1) \cup L(A_2)$.

- Konkatenation von zwei Automaten A_1, A_2 zu Automat A
  Der konkatenierte Automat akzeptiert Wörter bzw. die Sprache L bestehend aus einem Wort des ersten gefolgt von einem Wort des zweiten Automaten. Es gilt: $L(A) = L(A_1)L(A_2)$.

Die Formalisierung der Verknüpfung ist in [19][60][74] nachzulesen. In der Abbildung 2.10 sind die zwei Methoden dargestellt. Bei der Vereinigung wird ein neuer *initialer Zustand* erzeugt und mit den *initialen Zuständen* der Automaten A_1 und A_2 über ϵ-Transitionen verbunden. Die *finalen Zustände* der Automaten A_1 und A_2 werden ebenfalls auf einen neuen gemeinsamen *finalen Zustand* über eine ϵ-Transition zusammengeführt (vgl. Algorithmus 1).
Bei der Konkatenation werden die Automaten aneinander gehängt. Hierfür wird der *initiale Zustand* von Automat A_1 mit dem *initialen Zustand* von A_2 verbunden (vgl. Algorithmus 2).

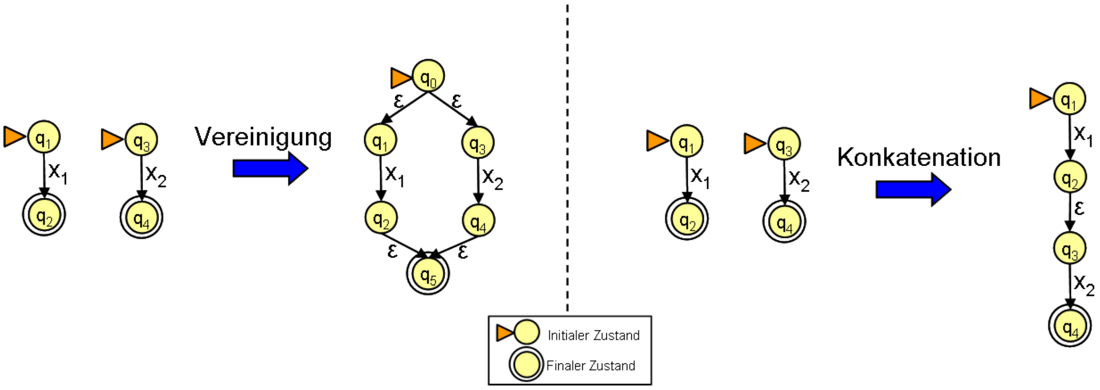


Abbildung 2.10: Vereinigung und Konkatenation von Automaten

2.5 Zusammenfassung

In diesem Kapitel wurden die theoretischen Grundlagen vermittelt, auf denen die im Rahmen dieser Arbeit entwickelten Konzepte aufbauen. Es wurden vier Themen schwerpunktmäßig behandelt.

Algorithm 1 (A,z)=Vereinigung_von_Automaten($z, k, \{A_k\}$)

Input: z // Index aktueller Zustand; k // Anzahl zu vereinigender Automaten; $\{A_k\}$ // Menge der zu vereinigenden Automaten mit $(Q_k, \Sigma_k, \delta_k, q_{0_k}, F_k)$;

Output: z // Anzahl an Zuständen; $A = (Q_a, \Sigma_A, \delta_A, q_{0_A}, F_A)$ // vereinigter Automat;

```
// Erzeuge zwei Zustände
z++; add $q_z$;
z++; add $q_z$;
// Hinzufügen zusätzlicher $\epsilon$-Transitionen
for l=1 to k do
    add $\delta(q_{z-1}, \epsilon) = q_{k_0}$; // erzeuge $\epsilon$-Übergang von Zustand $q_{z-1}$ auf $q_{k_0}$
    for $x = 1$ to $Anzahl\_Elemente(F_l)$ do
        add $\delta_A(F_l(x), \epsilon) = q_z$;
    end for
end for
// Erzeuge Automat A
$A = (Q_A, \Sigma_A, \delta_A, q_{0_A}, F_A)$;
$Q_A \leftarrow Q_1 \cup ... \cup Q_k \cup \{q_z, q_{z-1}\}$;
$\Sigma_A \leftarrow \Sigma_1 \cup ... \cup \Sigma_k \cup \epsilon$;
$\delta_A \leftarrow \delta_1 \cup ... \cup \delta_k$;
$q_{0_A} \leftarrow q_{z-1}$ // markiere $q_{z-1}$ als Anfangs-Zustand von A
$F_A \leftarrow q_z$ // markiere $q_z$ als akzeptierenden Zustand von A
// Hinzufügen zusätzlicher $\epsilon$-Transitionen
for l=1 to k do
    add $\delta(q_{z-1}, \epsilon) = q_{k_0}$; // erzeuge $\epsilon$-Übergang von Zustand $q_{z-1}$ auf $q_{k_0}$
    for $x = 1$ to $Anzahl\_Elemente(F_l)$ do
        add $\delta_A(F_l(x), \epsilon) = q_z$;
    end for
end for
return (A,z)
```

```
Algorithm 2 A=Konkatenation_von_Automaten($A_1, A_2$)
Input: $A_1 = (Q_1, \Sigma_1, \delta_1, q_{1_0}, F_1)$ // Automat $A_1$; $A_2 = (Q_2, \Sigma_2, \delta_2, q_{2_0}, F_2)$ // Automat $A_2$; M (global) // Menge von finalen Zuständen, die bei Konkatenation nicht zu berücksichtigen sind
Output: $A = (Q, \Sigma, \delta, q_0, F)$ // zusammengefügter Automat
// Erzeuge Automat A
$A = (Q, \Sigma, \delta, q_0, F)$;
$Q \leftarrow Q_1 \cup Q_2$;
$\Sigma \leftarrow \Sigma_1 \cup \Sigma_2 \cup \epsilon$;
$\delta \leftarrow \delta_1 \cup \delta_2$;
$q_0 \leftarrow q_{1_0}$;// Initialen Zustand festlegen
$F \leftarrow F_2$;// Finale Zustände festlegen
// Erzeuge $\epsilon$-Übergang von allen Elementen $\in F_1$ zu $q_{2_0}$, die nicht in M enthalten sind. Dadurch können finale Zustände explizit von der Konkatenation ausgeschlossen werden, s. Exception (Algorithmus 10)
for $x = 1$ to $Anzahl\_Elemente(F_1)$ do
    if $(F_1(x) \notin M)$ then
        add $\delta(F_1(x), \epsilon) = q_{2_0}$;
    end if
end for
return $A$;
```

Im 1. Teil wurde kurz der Aufbau der Steuergeräte betrachtet und spezifische Besonderheiten hervorgehoben.
Im 2. Teil wurde ein Überblick über die Extensible Markup Language (XML) gegeben. Vor allem der Aufbau von XML-Dokumenten und XML-Schemata wurde ausführlich erläutert. So wurden bspw. die Struktur-Elemente vorgestellt, die zur Erstellung eines Schemas relevant sind. Des Weiteren wurden kurz die Methoden vorgestellt, mit denen XML-Dokumente bearbeitet werden können.
Der 3. Teil vermittelte die Grundlagen zu den *Message Sequence Charts (MSCs)*. Es wurde der generelle Aufbau von MSCs, als auch die einzelnen MSC-Konstrukte detailliert erläutert. Ferner wurde die formale Beschreibung von MSCs erläutert bzw. definiert.
Der 4. Teil konzentrierte sich auf die Grundlagen der endlichen Automaten (EA). Es wurde erläutert, wie diese aufgebaut sind, beschrieben werden können und über Konkatenation oder Vereinigung zusammengefügt werden können. Des Weiteren wurde die formale Beschreibung der Automaten erläutert.

3 Rückwärtskompatibilität und Gesamtkonzept

Ziel des Forschungsthemas „CompA“ ist ein durchgängiger Ansatz, mit dem Steuergeräte auf Basis ihrer Spezifikationen (Lastenhefte) auf Rückwärtskompatibilität hin analysiert werden können.
An dieser Stelle werden die Teilziele bzw. die Anforderungen nochmals zusammengefasst:

- *Durchgängiger Ansatz:* Der Ansatz muss durchgängige Methoden, angefangen bei den Spezifikationen von Steuergeräten bis hin zu spezifischen Kompatibilitäts-Vergleichsmethoden, zur Verfügung stellen.
- *Black-Box:* Fahrzeughersteller spezifizieren die Anforderungen an Steuergeräte mittels Lastenhefte, wobei die Steuergeräte hierbei als Black-Boxes betrachtet werden. Die Kompatibilitätsanalyse der Steuergeräte erfolgt auf Basis dieser Lastenhefte.
- *Generik:* Der im Rahmen von CompA entwickelte Ansatz muss eine Methode beschreiben und auf unterschiedliche Steuergeräte bzw. Systeme übertragbar sein.
- *Vergleichsmethoden:* Definition von spezifischen Vergleichsalgorithmen zur Analyse von Rückwärtskompatibilität.
- *Expertenwissen:* Integration/Adaption von Expertenwissen bzgl. Steuergeräte-Aufbau und Kompatibilitätsanalyse muss möglich.

Bevor das Gesamtkonzept in Abschnitt 3.2 detailliert vorgestellt wird, wird in Abschnitt 3.1 definiert, unter welchen Bedingungen zwei Steuergeräte als zueinander rückwärtskompatibel gelten.

3.1 Rückwärtskompatibilität

Der CompA-Ansatz fokussiert auf die Untersuchung von Rückwärtskompatibilität von Steuergeräten.

Begrifflichkeiten und Definitionen

Im Rahmen von Kompatibilitätsbetrachtungen tauchen wiederholt die Begriffe Äquivalenz und Kompatibilität auf. Diese Themen unterscheiden sich wie folgt:

- *Äquivalenz/Äquivalenzprüfung:* Äquivalenz[1] bezeichnet die Gleichwertigkeit von verschiedenen Elementen. In der Informatik gibt es z.B. die Programmäquivalenz. Diese besagt, dass zwei Computerprogramme P_1, P_2 für dieselbe Eingabe auch immer dieselbe Ausgabe liefern bzw., dass sie dieselbe Sprache akzeptieren: $P_1(E_i) = A_i \Leftrightarrow P_2(E_i) = A_i; A_i \in Ausgabe, E_i \in Eingabe.$
  Übertragen auf den Vergleich von Lastenheften (LH) bedeutet eine Prüfung auf Äquivalenz, dass LH_1 und LH_2 äquivalente Anforderungen spezifizieren müssen, d.h. $\forall i, j : LH_1(A_i) = LH_2(A_j); i, j \in \mathbb{N}; A_i, A_j = Anforderungen.$

- *Kompatibilität:* Kompatibilität bedeutet, dass zwei Steuergeräte SG_1 und SG_2 die gleiche Funktionalität bzw. Anforderungen erfüllen, obwohl sie nicht identisch sind. Das *Deutsche Institut für Normung* definiert in einer Norm [37] Kompatibilität als *„Eignung einer Einheit unter spezifischen Bedingungen zusammen benutzt zu werden, um relevante Forderungen zu erfüllen“*. Mit dem Begriff *„zusammen benutzt“* bezieht sich die Norm auf die Definition von Verträglichkeit von Komponenten.

Definition Rückwärtskompatibilität

Um Kompatibilität formal zu beschreiben, wird ein neues Symbol eingeführt $\Leftrightarrow_c$ (Anlehnung an Äquivalenz-Symbol). Somit können die Varianten der Kompatibilität folgendermaßen definiert werden:

- $SG_1 \Leftrightarrow_c SG_2$ bedeutet, dass die Steuergeräte SG_1 und SG_2 zueinander kompatibel sind.

- $SG_1 \Leftarrow_c SG_2$ bedeutet, dass das Steuergerät SG_2 rückwärtskompatibel zu SG_2 ist.

- $SG_1 \Rightarrow_c SG_2$ bedeutet, dass das Steuergerät SG_1 vorwärtskompatibel zu SG_2 ist.

Die Kriterien, wann welche Elemente kompatibel sind, obwohl sie unterschiedlich sind, sind nicht generisch festlegbar. Daher müssen diese Kriterien von Experten definiert werden.
Im Rahmen von CompA wird auf die Analyse von Rückwärtskompatibilität fokussiert. Rückwärtskompatibilität sei im Rahmen von CompA wie folgt definiert [51]:

Definition 1 *Ein Steuergerät SG_2 ist rückwärtskompatibel zu SG_1 ($SG_1 \Leftarrow_c SG_2$), wenn folgende Forderungen erfüllt sind:*

1. *SG_2 kann an der identischen Stelle von SG_1 ohne Zusatzaufwände im Bauraum verbaut werden.*
2. *SG_2 kann zu identischen Rand- und Umgebungsbedingungen wie SG_1 störungsfrei arbeiten.*

---

[1] Vom Lateinischen: aequus „gleich“ und valere „wert sein“.

3. *SG_2 und SG_1 bedienen physikalisch die gleichen Schnittstellen.*

4. *Die Menge M_1 der Funktionen f_1 von SG_1 stellt eine Submenge der Menge M_2 von SG_2 dar: $M_1 \subseteq M_2; M_1 = \{f_1 \mid f_1 \; realisiert \; in \; SG_1\}$, wobei die Schnittstellen identisch bedient werden.*

5. *SG_2 und SG_1 weisen das gleiche dynamische funktionale Verhalten auf, bezogen auf M_1.*

Die Analyse, ob $SG_1 \Leftarrow_c SG_2$ gilt, erfolgt auf Basis der zugehörigen Spezifikationen bzw. Lastenhefte. Im Rahmen von CompA werden die Lastenhefte zur Analyse zu Grunde gelegt.

Definition 2 *Ein Steuergerät SG_2 ist zu SG_1 genau dann rückwärtskompatibel, wenn die Anforderungen des zugehörigen Lastenhefts $LH_2(SG_2)$ rückwärtskompatibel zu $LH_1(SG_1)$ sind.*
$(SG_1 \Leftarrow_c SG_2) \Leftarrow (LH_1 \Leftarrow_c LH_2)$

3.2 Gesamtkonzept

CompA hat sich zum Ziel gesetzt, eine durchgängige Methode zur Analyse von Rückwärtskompatibilität von Steuergeräten zu definieren. Grundgedanke ist, dass die Analyse aus Sicht der Fahrzeughersteller und somit auf Basis von Lastenheften erfolgen soll (vgl. Definition 2). Die Durchgängigkeit des Ansatzes, der im Rahmen dieser Arbeit bzw. CompA entwickelt wurde, basiert auf den Schritten *Formalisierung*, *Instanziierung* und *Kompatibilitätsanalyse*. In Abbildung 3.1 ist das Gesamtkonzept graphisch dargestellt und wird in den nachfolgenden Abschnitten erläutert.

Formalisierung

Um einen Vergleich von zwei Lastenheften $LH_1 \Leftarrow_c LH_2$ durchführen zu können, müssen diese die gleiche „Sprache sprechen“. Dies könnte entweder durch Transformation/Normalisierung der Lastenhefte erfolgen oder indem Lastenhefte auf einer gemeinsamen formalen Basis (Metamodell) erstellt werden.
Im Rahmen von CompA wurde ein Metamodell für Steuergeräte-Spezifikationen entworfen. Das Steuergeräte-Metamodell (SG-Metamodell) legt fest, welche Elemente eines Steuergerätes zu spezifizieren sind. Hierbei weist das SG-Metamodell folgende Besonderheiten auf:

- *Scope:* Das SG-Metamodell beschränkt sich nicht nur auf einzelne spezifische SG-Umfänge (z.B. Pin), sondern umfasst Vorgaben für das „ganze“ Steuergerät.

- *strukturierter hierarchischer Aufbau:* Zur intuitiven Unterstützung des Spezifikationsprozesses ist das SG-Metamodell als Entscheidungsbaum konstruiert.

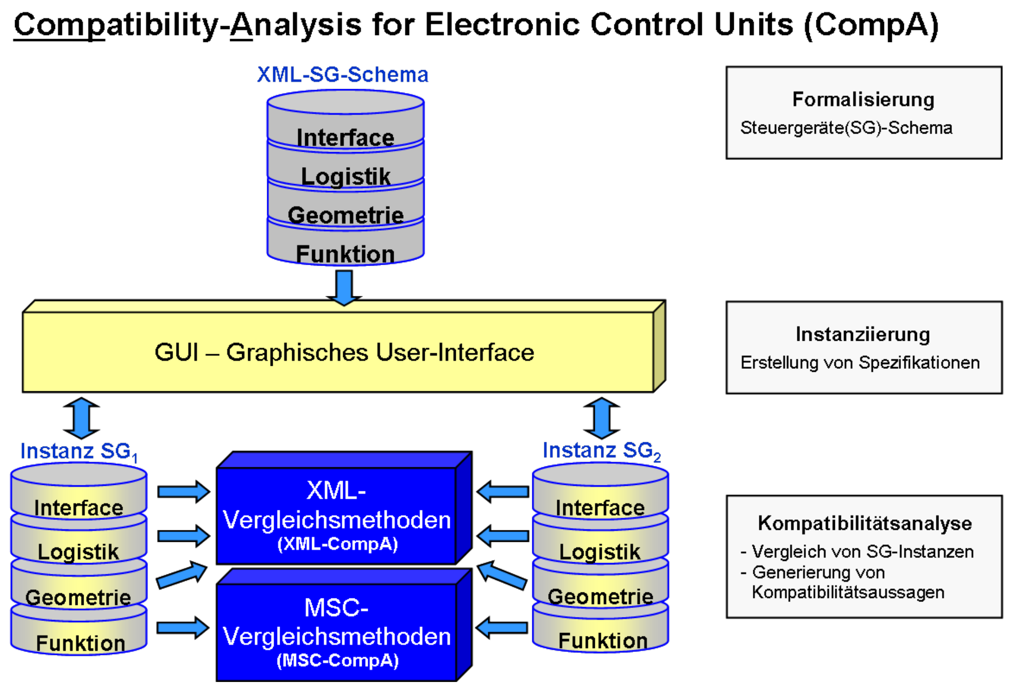


Abbildung 3.1: Gesamtkonzept von Compatibility Analysis for Electronic Control Units (CompA)

- *Expertenwissen:* „Welche“ Elemente eines Steuergerätes für eine hinreichende Spezifikation „Wie“ beschrieben sein müssen, wurde durch Experten validiert.
- *Muss/Kann-Kriterien:* Im SG-Schema ist hinterlegt, welche Elemente eines SG beschrieben werden müssen und welche optional sind.
- Das SG-Metamodell spezifiziert Umfänge für die statischen Eigenschaften von Steuergeräten, d.h. Vorgaben für *Interface*, *Logistik* und *Geometrie*.
- Das SG-Metamodell umfasst auch die dynamischen Eigenschaften *Funktion*. Zur Spezifikation des dynamisches Verhaltens werden Message Sequence Charts eingesetzt.
- *XML-Schema:* Das SG-Metamodell wird auf Basis eines XML-Schemas umgesetzt (kurz: XML-SG-Schema).

Der Aufbau des SG-Metamodells wird ausführlich in Kapitel 5.2 behandelt.

Instanziierung

Auf Basis des XML-SG-Schemas erfolgt die Spezifikation der Steuergeräte. Hierfür wurde in CompA eine graphische Bedienoberfläche (GUI) entwickelt. Die GUI bietet eine graphische und eine tabellarische Notation, um Steuergeräte-Elemente effizient

spezifizieren zu können. Des Weiteren wird die GUI auf Basis des XML-SG-Schemas generiert, so dass diese auf beliebige Schemata anwendbar ist. Alle spezifizierten Elemente werden systematisch gegen das XML-SG-Schema validiert und so wird sichergestellt, dass die SG-Spezifikationen gültig sind.
Das Ergebnis der Instanziierung sind XML-Dokumente, die ein Steuergerät hinreichend spezifizieren und gültig gegenüber dem XML-Schema sind. Durch den formalen Aufbau können XML-Vergleichsmethoden auf die XML-Dokumente angewendet werden. XML-Dokumente, die aus einem XML-SG-Schema abgeleitet wurden, werden künftig auch als XML-SG-Instanzen bezeichnet.

Kompatibilitätsanalyse

Auf Basis der erstellten XML-SG-Instanzen können Vergleichsmethoden zur Analyse von Rückwärtskompatibilität der Steuergeräte ausgeführt werden. Da die statischen Eigenschaften auf Basis eines XML-SG-Schemas und die dynamischen Eigenschaften der Steuergeräte mittels Message Sequence Charts spezifiziert werden, wurden im Rahmen von CompA zwei spezifische Vergleichsmethoden entwickelt, die XML-Vergleichsmethode (*XML-CompA*) und die MSC-Vergleichsmethode (*MSC-CompA*).

XML-Vergleichsmethode (*XML-CompA*) Für Analysen bzgl. Rückwärtskompatibilität werden die XML-SG-Instanzen miteinander verglichen werden. Da die herkömmlichen Change-Detection-Algorithmen hierfür nicht ausreichen (s. Kapitel 4.3), wurde im Rahmen von CompA ein 4-Schritt Vergleichsansatz entworfen. Dieser Ansatz wird in Kapitel 6 genauer ausgeführt wird und bietet folgende Besonderheiten:

- Interaktive Auswahl von Elementen, die bei einer Kompatibilitätsanalyse nicht berücksichtigt werden sollen.
- XML-SG-Instanzen werden strukturell verglichen. Hierfür werden die Elemente, die gleiche/ähnliche Eigenschaften von Steuergeräten beschreiben, einander zugeordnet.
- Die einander zugeordneten Elemente werden anschließend bzgl. Rückwärtskompatibilität verglichen. Bei dem Vergleich können zusätzlich Kompatibilitätsregeln mit hinzugezogen werden.
- Generierung einer Aussage über die Rückwärtskompatibilität von XML-SG-Instanz(SG_2) zu XML-SG-Instanz(SG_1).

Des Weiteren kann mit Hilfe der Methode analysiert werden, welche Elemente eine Rückwärtskompatibilität ($SG_1 \Leftarrow_c SG_2$) verhindern.

MSC-Vergleichsmethode (*MSC-CompA*) Message Sequence Charts (MSCs) werden zur Spezifikation von dynamischem Verhalten verwendet. Um eine Aussage bzgl. der Rückwärtskompatibilität des spezifizierten dynamischen Verhaltens zu treffen, müssen die zugehörigen MSCs miteinander verglichen werden. Eine Herausforderung hierbei ist, dass mittels MSCs gleiches dynamisches Verhalten unterschiedlich spezifiziert sein kann. Um diese Problematik zu lösen, werden die MSCs in endliche Automaten transformiert und die Kompatibilitätsanalyse der MSCs auf Basis der korrespondieren Automaten durchgeführt.

3.3 Zusammenfassung

In diesem Kapitel wurde ein Überblick über das Gesamtkonzept des Forschungsthemas *Compatibility Analysis for Electronic Control Units (CompA)* vorgestellt.
Zuerst wurde definiert, wann ein Steuergerät SG_2 als rückwärtskompatibel zu SG_1 betrachtet werden kann und welche Bedeutung dies für die Spezifikations-Ebene der Steuergeräte hat. Im Anschluss wurde der Aufbau des CompA-Konzepts vorgestellt und die zugehörigen Schwerpunkte vorgestellt. Diese waren zum einen die Formalisierung von Steuergeräte-Spezifikation auf Basis eines XML-Metamodells und zum anderen die Vergleichsmethoden zur Analyse von Rückwärtskompatibilität. Bei den Vergleichsmethoden wurden zwei Methoden angesprochen: die XML-Vergleichsmethode (*XML-CompA*) und die MSC-Vergleichsmethode (*MSC-CompA*).

4 Stand der Technik

4.1 Spezifikationsansätze

In diesem Abschnitt wird ein Überblick über die in der Industrie verwendeten Spezifikationsansätze für Steuergeräte gegeben. Häufig basieren diese Ansätze noch auf nicht formalen Ansätzen (Doors/Word). Verschiedene Forschungsprojekte (MSR, MOSES) beschäftigen sich daher mit Methoden zur Formalisierung der Spezifikationen durch Verwendung von Datenmodellen.

Im Folgenden werden nun verschiedene Spezifikationsansätze genauer betrachtet und bzgl. der Verwendbarkeit für *CompA* bewertet. Auf weitere interessante Spezifikationsansätze sei verwiesen [70][119][125][117].

4.1.1 Word/Doors

Die meisten Lastenhefte werden derzeit noch mit einem Textverarbeitungsprogramm z.B. *Word* geschrieben. Hierbei werden die Anforderungen an ein Steuergerät bzgl. Funktionalität, Abmessungen, etc. spezifiziert. Als Vorlage dienen hierfür Musterlastenhefte, die entweder von den Firmen selbst [10] oder von Organisationen z.B. dem deutschen Automobilverband (VDA) zur Verfügung gestellt werden [135].

Im Rahmen von Prozessverbesserungs-Maßnahmen fällt immer öfter der Begriff „durchgängiges Anforderungsmanagement“. Hiermit wird ein Top-Down Anforderungsmanagement von der Ziele-Ebene bis zur technischen Anfordungs-Ebene bezeichnet. Ziel ist, dass die Anforderungen Top-Down verfeinert werden. Um dies umzusetzen, wird verstärkt auf Anforderungsmanagement-Tools (z.B. *Doors*) zurückgegriffen. Das Tool *Doors* [131] ist so aufgebaut, dass jede Anforderung atomar formuliert wird und eine eigene ID erhält. Somit ist später leichter nachzuvollziehen, welche Anforderungen umgesetzt sind und welche nicht.

Bewertung Word/Doors sind weit verbreitet. *Doors* bietet gegenüber *Word* den Vorteil, dass die Anforderungen atomar formuliert sind. Jedoch werden die Anforderungen weiterhin in Prosa-Text spezifiziert und müssen keinen semantischen und/oder syntaktischen Beschreibungsvorgaben genügen. Daher ist ein automatisierter Kompatibilitätsvergleich auf Basis dieser Spezifikationen nicht sinnvoll.

4.1.2 MSR-Systems

Das MSR-Konsortium (Manufacturer-Supplier-Relationship) [106] hat sich zum Ziel gesetzt, Methoden und Schnittstellen zu definieren, durch deren Nutzung Synergieeffekte bei der Durchführung gemeinsamer Projekte zwischen Zulieferern und Fahrzeugherstellern entstehen sollen. Es wurden vier Schwerpunkte bzw. Gruppen definiert:

- *MEGMA* Erarbeitung eines Bibliothek-Standards für modellbasierten Entwurf von Steuergerätefunktionen für Kraftfahrzeuge [144][145]. Ziel ist der Austausch von Funktionsmodellen zwischen unterschiedlichen Tools z.B. ASCET und Matlab-Simulink.

- *MEDOC* Erarbeitung von Methoden, Standards und Werkzeugen zum Austausch von Informationen im Entwicklungsprozess. MEDOC stellt einheitliche Anwendungsprofile für den Daten-/Dokumentenaustausch auf Basis von SGML/XML-Technologie bereit [109]. Folgende Anwendungsgebiete sind beispielsweise hierbei betrachtet worden: System-Spezifikationen (*MSR System*) [108], Software-Spezifikationen (MSR Software) [112], etc..

- *MEPRO* Untersuchung von Synchronisationsproblemen in Entwicklungsprozessen und Schaffung eines Instrumentariums zur Vereinbarung von Synchronisationspunkten.

- *VHDL-AMS* Austausch von Funktionsmodellen auf Basis von VHDL-AMS [113].

Interessant für diese Arbeit ist das, im Schwerpunkt MEDOC, erarbeitete Datenmodell *MSR SYSTEM* [108][111][110][107], das mechanische und elektrische Komponenten umschreibt. Es handelt sich dabei um eine Vorlage für Lasten- und Pflichtenhefte. Dabei können administrative Daten wie Kapitelstrukturen oder Verweise auf externe Dokumente und Spezifikationen eingefügt werden. Ebenso können physikalische Eigenschaften wie Grenzwerte oder geometrische Abmessungen, aber auch abstrakte Anforderungen (z.B. überspannungsfest) formuliert werden. Der MSR-Ansatz ist sehr detailliert und umfangreich.

Bewertung MSR SYSTEM ist ein Hilfsmittel, um Lastenhefte nach einer spezifischen Syntax zu beschreiben. Die Sprache bietet sehr viele bereits definierte Beschreibungskonstrukte-/vorschriften für einzelne Elemente an. Allerdings bietet die Beschreibung auch sehr viele Freiheiten. So wird nicht vorgegeben, welche Elemente eines Systems beschrieben sein müssen und welche nicht. Dadurch fehlt ein Metamodell für Steuergeräte. Das Metamodell ist für den CompA-Ansatz die Voraussetzung, um Vergleiche durchführen zu können.
Ausblick: Eine Zusammenführung der Spezifikationsansätze wäre interessant und sollte in einer weitergehenden Arbeit untersucht werden.

4.1.3 MOSES

Am Fraunhofer ISST wurde das Projekt *MO*dellba*S*iert*E* *S*ystementwicklung (MOSES) [84] von Elektrik- und Elektroniksystemen im Automobil vorangetrieben. Hierbei wurden Methoden und Notationen (analog dem MSR-Ansatz) für verschiedene Entwicklungsprozesse definiert. Zu diesen gehören Anforderungsmanagement, Systemdesign sowie Software- und Hardware-Entwicklung. Diese Systematik lässt sich u.a. auf die Ebene von Komponenten applizieren, dazu gehören bspw. spezifische Steuergeräte oder Sensor- und Aktuator-Baugruppen. Alle genannten Entwicklungsprozesse werden dabei auf einem abstrakten Level betrachtet, d.h. MOSES beschreibt Systeme größtenteils auf Verständnisebene. So können bspw. spezifische Eigenschaften von Schnittstellen (z.B. Spannungspegel) nicht modelliert werden.

Bewertung Aus den Systemmodellierungen lassen sich derzeit noch keine Systembeschreibungen generieren. Daher ist dieser Ansatz ungeeignet, um als Eingabegröße für algorithmische Vergleichsmethoden im Rahmen von CompA zu dienen.

4.1.4 CASE-Tools

Der Begriff Computer-Aided Software Engineering (CASE) bezeichnet den Einsatz IT-gestützter Werkzeuge für die Umsetzung von Software-Konzeptionen. CASE-Tools werden oft für Rapid-Prototyping Anwendungen eingesetzt. Typische Vertreter sind bspw. Matlab-Simulink [98] und ASCET [44].
Mit Hilfe dieser Tools können Funktionen von Steuergeräten modelliert und simuliert werden. Die Möglichkeiten hierbei reichen von der Spezifikation eines einfachen Automaten bis hin zu komplexen Regelungssystemen. Des Weiteren kann mit Hilfe der Tools ein ausführbarer Programmcode generiert werden, der z.T. in Steuergeräte 1:1 übernommen wird.

Bewertung Case-Tools sind mächtige Tools, mit denen komplexe Funktionen entworfen werden können. Zur Spezifikation von Steuergeräten werden diese Tools jedoch selten eingesetzt. Grund hierfür ist, dass Fahrzeughersteller Anforderungen an ein Steuergerät spezifizieren (Blackbox), jedoch das spezifische Know-How der einzelnen Funktionen die Kernkompetenz der Zulieferer ist.
Werden dennoch Funktionen mit Hilfe dieser Tools spezifiziert, so werden diese Funktionsmodelle, im Rahmen von CompA, als Erweiterung der Lastenheft-Ebene angesehen. Zum Vergleich von Funktions-Modellen existieren bereits verschiedene Ansätze zur formalen Verifikation [71][81] wie z.B. Model-Checking [26][8][148]. Diese Methoden stehen jedoch nicht im Fokus dieser Arbeit.

4.1.5 UML-Sequenz-Diagramme

Mit *Sequence Diagrams* wurde eine Variante von Message Sequence Charts (MSC) in Unified Modeling Language (UML) [15][16] integriert. Obwohl beide Sprachen auf gleichen Prinzipien beruhen, gibt es auf Grund der verschiedenen Anwendungsbereiche unterschiedliche Sprachkonstrukte und Unterschiede in der Darstellung. Durch eine Kombination der Stärken von MSC und den Sequence Diagrams bemüht man sich zur Zeit verstärkt um eine Harmonisierung der beiden Diagrammarten [120][59]. In UML 2.0 wurden nun einige Defizite, wie die Wiederverwendung und Kombinierbarkeit von Sequenzen, gelöst. Ein Vergleich zwischen UML 2.0 und MSC zeigt, dass die Mächtigkeit der Tools inzwischen sehr ähnlich ist [69]. Ob sich UML 2.0 gegenüber MSC durchsetzt, wird davon abhängen, wie effizient eine automatische Codegenerierung mit UML 2.0 ist.

Bewertung UML-Sequenz Diagramme und Message Sequence Charts umfassen inzwischen einen ähnlichen Leistungsumfang. Ihm Rahmen von CompA werden ausschließlich MSCs betrachtet. Die entwickelten Methoden können aber, nach Anpassung, auch auf UML-Sequenzdiagramme portiert werden.

4.1.6 IPQ

Im Bereich des IP (Intellectual Property) basierten Entwurfs sucht ein SoC (System-on-Chip) Entwickler üblicherweise nach IPs [64], die dann nach dem Kauf weiterverarbeitet werden müssen. Um diese Schritte zu unterstützen, ist ein einheitliches Datenaustauschformat wie das *IPQ* Format notwendig [136]. Basierend auf diesem Format wurde an der TU-Chemnitz ein Werkzeugsatz entwickelt, der aktuelle Web Service Technologien verwendet. Das Konzept unterstützt eine dynamische Menge von benutzbaren IP bezogenen Web Services [138][137]. Dies erlaubt die jederzeitige Erweiterung des Werkzeugsatzes ohne die Änderung von bestehendem Code der Werkzeuge. Der Werkzeugsatz kann benutzt werden zur Suche nach Parametrisierung, Auslieferung, Simulation, Austausch,etc. von IPs.

Bewertung Im Rahmen von *IPQ* wurden auch Ansätze definiert, um zu bestimmen, welche Daten der IPs im Datenaustauschformat IPQ hinterlegt werden müssen. Auf Basis dieser Ansätze wurde die Idee bzgl. des Sichtenkonzepts entwickelt, das in Kapitel 5.1.3 erläutert wird. Des Weiteren entstand in dem Projekt *IPQ* die Softwareumgebung *CAMP* (vgl. Kapitel 8.1), auf die in dieser Arbeit aufgesetzt wird.

4.2 Kompatibilitäts-Prüfmethoden

4.2.1 MFA

In einem Forschungsprojekt *Modellbasierte Funktionsentwicklung und -absicherung (MFA)* [91][38] der BMW Group werden Modellierungsrichtlinien für Schnittstellenspezifikationen und für Testspezifikationen entwickelt. Der Schwerpunkt von MFA liegt

derzeit auf der Beschreibung von Systemfunktionen im CAN-Bereich.
Der Grundgedanke ist,

- die Funktionalität einer Schnittstelle (Interface) durch Zustandsdiagramme bzw. durch Automaten zu beschreiben und
- die Interaktion der Schnittstellen mit Message Sequence Charts (MSC) abzubilden.

Das Ergebnis dieses Spezifikationsansatzes ist, abstrakt gesehen, somit eine Verhaltensmodellierung von vernetzten Funktionen.

Bewertung Der Ansatz MFA fokussiert die Erstellung von Testspezifikationen und die Modellierung der Kommunikation von Systemen bzw. Steuergeräten z.B. über CAN. CompA dagegen fokussiert die Spezifikation eines Steuergerätes auf Lastenheft-Ebene und den Vergleich dieser Spezifikationen. Beide Ansätze verwenden aber zur Spezifikation von dynamischem Verhalten Message Sequence Charts. Hier konnten Synergien genutzt werden. So wird im Forschungsthema *CompA* ein MSC-Metamodell verwendet, das im Rahmen von MFA entwickelt wurde (vgl. Kapitel 7.4.2). Im Gegenzug könnte die, im Rahmen von *CompA*, entwickelte MSC-Vergleichsmethode Anwendung in MFA finden.

4.2.2 Mokoma

Im Rahmen der Forschungsoffensive „Software Engineering 2006“ wird das Projekt *Modellbasiertes Kompatibilitätsmanagement von komplexen eingebetteten Softwarelösungen und Elektroniknetzwerken in mobilen Anlagen* durchgeführt. Ziel wird sein, mit Hilfe modellbasiertem Engineering und funktionaler Betrachtungsweise Komplexität beherrschbar zu machen und Kompatibilitätsmanagement zu betreiben [103].

4.3 XML-Vergleichsmethoden

Zum Vergleich von XML-Dokumenten gibt es unterschiedliche Ansätze. Die am häufigsten verwendeten bzw. bekanntesten werden im Folgenden beschrieben.

4.3.1 XML Diff Algorithmen

XML Diff Algorithmen werden zur Change Detection [114][149][123] eingesetzt. Change Detection bedeutet, dass ein XML Dokument in zwei verschiedenen Versionen vorliegt und die Unterschiede dieser beiden Versionen dargestellt werden sollen. Dabei kann je nach Algorithmus erkannt werden, ob sich gleiche Informationen innerhalb eines Dokumentes verschoben haben. Es existieren verschiedene XML Diff Algorithmen, die sich hauptsächlich in den Laufzeiten und Anwendungsgebieten unterscheiden

[27]. In der folgenden Tabelle sind die gebräuchlichsten Diff Algorithmen aufgeführt [27][24][75][152]:

Name	**Referenz**	**Kommentar**
DeltaXML	[88]	Lineare Laufzeit/Geordnete und Ungeordnete Bäume
MMDiff	[23]	Quadratische Laufzeit/Geordnete Bäume
XMDiff	[23]	Quadratische Laufzeit/Geordnete Bäume
XyDiff	[28]	Lineare Laufzeit/Geordnete Bäume
LaDiff	[25]	Lineare Laufzeit/Geordnete Bäume
XMLTreeDiff	[43]	Quadratische Laufzeit/Geordnete Bäume
XML Diff	[99]	Geordnete und Ungeordnete Bäume
X-Diff	[141]	Quadratische Laufzeit/Ungeordnete Bäume
MH-Diff	[24]	Quadratische Laufzeit/Ungeordnete Bäume
DiffXML	[105]	Lineare Laufzeit/Geordnete Bäume
KF-Diff+	[147]	Lineare Laufzeit/Geordnete und Ungeordnete Bäume
BioDIFF	[130]	Quadratische Laufzeit/Ungeordnete Bäume

Tabelle 4.1: Liste der gebräuchlichsten Diff Algorithmen

Die Funktionsweise der XML Diff Algorithmen wird am Beispiel des *X-Diff Algorithmus* vorgestellt [141].

X-Diff Algorithmus

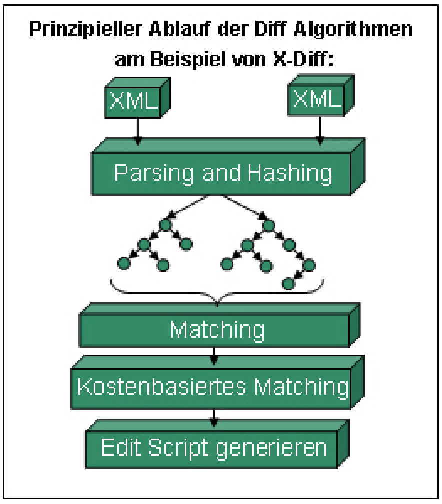


Abbildung 4.1: Konzept eines *X-Diff Algorithmus* [141]

Der *X-Diff Algorithmus* unterteilt sich in drei Hauptschritte (vgl. Abbildung 4.1):

1. *Parsing und Hashing:* Parsen von zwei XML Dokumenten und Überführung in eine Baumrepräsentation. Während des Parsens wird für jeden Knoten ein Hashwert [97] erzeugt. Da für die Hashwertbildung der gesamte Unterbaum mit einbezogen wird, kann daran abgelesen werden, welche Struktur sich unterhalb eines jeweiligen Knotens befindet.

2. *Matching*[1] [22][47]. Vergleich der zuvor erstellten Hashwerte beider Instanzen. Sind die Hashwerte gleich, so sind die Unterbäume identisch und müssen nicht mehr weiter betrachtet werden. Im weiteren Verlauf des *Matching* bleiben nur noch Unterbäume übrig, die nicht vollständig gleich sind. Diese werden durch ein *Minimales Kosten Matching* einander zugeordnet. Das Kosten Matching benutzt das Prinzip der Editierdistanz [151][150]. Die Editierdistanz zweier Unterbäume ist die Summe der Operationen, die notwendig sind, um beide Unterbäume aneinander anzugleichen bzw. um zwei XML Dokumente ineinander zu überführen. Im *Minimalen Kosten Matching* verwendet X-Diff die Editierdistanz, um die Unterbäume mit der geringsten Editierdistanz einander zuzuordnen.

3. Im dritten Schritt wird ein sogenanntes „Edit Script“ erstellt. Dieses Script enthält die Operationen, die notwendig sind, um die Dokumente aneinander anzugleichen. Die Operationen werden aus dem *Minimalen Kosten Matching* extrahiert.

Bewertung *X-Diff Algorithmen* wurden entwickelt, um Änderungen auf XML-Dokumenten zu detektieren (Change Detection). Hierbei findet ein *Äquivalenzvergleich* der einzelnen XML-Elemente statt.
X-Diff Algorithmen können zwar ungeordnete Strukturen, sog. „unordered trees“ in den XML-Dokumenten erkennen, aber zur Generierung von Kompatibilitätsaussagen fehlen mehrere relevante Features, wie in Kapitel 6 deutlich werden wird. Ein Beispiel ist, dass bei einem Vergleich von zwei XML-Elementen keine Vergleichsregeln (z.B. Kompatibilitätsregeln) definiert werden können. Grundsätzliche Ideen der *X-Diff Algorithmen* sind jedoch mit in das Konzept von *XML-CompA* eingeflossen, wie z.B. die Idee des Unterbaum-Matchings.
Der *X-Diff Algorithmus* wurde zur Validierung der Leistungsfähigkeit von *XML-CompA* eingesetzt (s. Kapitel 9.2).

4.3.2 Baum Isomorphismen

Baum Isomorphismen [134][82][48][102][21][96] sind ein allgemeiner Ansatz und nicht direkt auf XML zugeschnitten. Da XML-Dokumenten jedoch eine Baumstruktur zugrunde liegt, werden auch Baum Isomorphismen betrachtet.
Ein Baum Isomorphismus ist eine bijektive[2] Abbildung zwischen den Knoten beider

---

[1] In Abbildung 4.1 ist das Matching in „Matching“ und in „Kostenbasiertes Matching“ unterteilt.

[2] Eine Abbildung ist bijektiv, wenn sie injektiv und surjektiv ist. Injektiv bedeutet, dass keine zwei verschiedenen Elemente der Definitionsmenge auf ein und dasselbe Elemente der Zielmenge abgebildet werden. Surjektiv bedeutet, dass jedes Element der Definitionsmenge auf ein Element der Zielmenge

Bäume.
Es gibt Baum Isomorphismen für geordnete und ungeordnete Bäume. Weiterhin gibt es Algorithmen, die keinen exakten Isomorphismus zweier Bäume bestimmen, sondern nur einen maximalen[3] Isomorphismus. Eine weitere Untergliederung der Algorithmen lässt sich anhand der Art der Traversierung der Bäume treffen. Zum einen kann eine Top-Down Traversierung stattfinden und zum anderen eine Bottom-Up Traversierung. Die Isomorphismen werden am Beispiel des „Unordered Tree Isomorphism“ (Isomorphismus auf ungeordneten Bäumen) dargestellt werden.

Unordered Tree Isomorphism

Der Unordered Tree Isomorphism Algorithmus betrachtet nicht die Elementnamen der Baumknoten, sondern lediglich die zugrundeliegende Struktur des Baumes.
Im ersten Schritt des Algorithmus wird die Anzahl der Knoten überprüft, die beide Bäume enthalten. Bei nicht übereinstimmender Anzahl bricht der Algorithmus ab, ansonsten wird für jeden Knoten ein sogenannter „Isomorphismuscode“ generiert. Die Isomorphismuscodes werden in einer Bottom-Up Traversierung des Baumes pro Knoten erstellt. Alle Blattknoten erhalten den Isomorphismuscode „1“. Die Knoten der nächsthöheren Ebenen erhalten einen Isomorphismuscode, der folgendermaßen aufgebaut wird:
1. Anzahl der Knoten, die im Unterbaum des aktuellen Knotens enthalten sind, plus den Knoten selbst.
2. Die Isomorphismuscodes der Kindknoten, nachdem diese radixsortiert[4] wurden.
In Abbildung 4.2 sind die Isomorphismuscodes für einen Baum beispielhaft dargestellt.

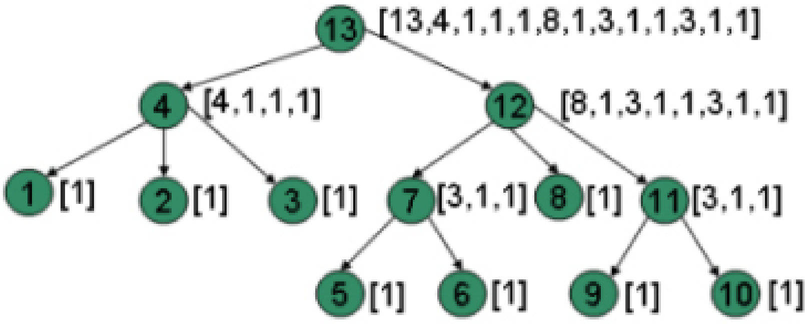


Abbildung 4.2: Unordered Tree Isomorphism

---

abgebildet werden kann.

[3] Maximal im Sinne von größtmöglichem Isomorphismus.

[4] Radixsort ist ein stabiles Sortierverfahren. Das bedeutet, wenn eine Liste alphabetisch sortierter Personennamen nach deren Geburtsdatum sortiert wird, bleibt die alphabetische Sortierung der Personen mit gleichem Geburtsdatum erhalten.

Anhand der Isomorphismuscodes der Wurzeln kann dann entschieden werden, ob ein Isomorphismus vorliegt. Dieser liegt genau dann vor, wenn die Isomorphismuscodes zweier Bäume übereinstimmen [134].

Bewertung Mit Hilfe von Baum Isomorphismen können zwei Baumstrukturen nur dahingehend analysiert werden, ob diese äquivalent sind.
Für die Kompatibilitätsanalyse ist jedoch relevant, ob die Element-Namen der Baumknoten äquivalent bzw. kompatibel sind und ob die Strukturen kompatibel sind, auch wenn diese nicht äquivalent sind. Die Baum Isomorphismen können diese Anforderungen nicht erfüllen und eignen sich daher nicht zur Kompatibilitätsanalyse.

4.3.3 Xandy

Xandy [39] wird ein Konzept genannt, das an den Universitäten Singapur und Missouri-Rolla entwickelt wurde. Das Ziel ist, wie schon bei den XML Diff Algorithmen, eine Änderungserkennung (Change Detection) für XML Dokumente durchzuführen.
Der Ansatz, der hier verfolgt wird, unterscheidet sich allerdings dahingehend, dass die XML Dokumente in relationale Tupel überführt und in eine Datenbank eingefügt werden. An diese Datenbank werden dann SQL Anfragen gestellt und mit Hilfe der Ergebnisse wird eine Change Detection durchgeführt. Xandy unterscheidet dabei Blattknoten und innere Knoten. Die Change Detection wird in zwei Phasen durchgeführt. In der ersten Phase werden die Unterbäume einander zugeordnet, die am besten übereinstimmen. Dazu werden SQL Anfragen generiert, die die Blattknoten vergleichen, Geschwisterreihenfolgen überprüfen und Übereinstimmungsgrade berechnen. Daraus können die Unterbäume richtig zugeordnet werden. In der zweiten Phase werden gelöschte, neu eingefügte oder veränderte Knoten gesucht, um die Change Detection für innere Knoten durchzuführen.

Bewertung Xandy wird ebenso wie die XML Diff Algorithmen für die Change Detection eingesetzt. Aus diesem Grund trifft die Bewertung bzgl. der Verwendbarkeit zur Kompatibilitätsanalyse auch hier zu.

4.4 Automaten-Transformation von MSCs

Message Sequence Charts (MSC) dienen zur Spezifikation von dynamischem Verhalten bzw. von Interaktionssequenzen. Analysen bzgl. des spezifizierten Verhaltens werden oft auf Basis von Automaten durchgeführt [73][87][89]. Ein weiteres Anwendungsgebiet für MSCs ist die Spezifikation von Testfällen bzw. die Erzeugung von Testfällen auf Basis von MSCs [57][122].
Im Rahmen von CompA wird der Ansatz von [87] erweitert. Im Folgenden seien zwei für CompA relevante Arbeiten vorgestellt.

Global Final State Machine

Ein Set an MSCs kann dynamisches Verhalten von Systemen spezifizieren. Nach [89] ist die Analyse des Verhaltens auf Basis von Automaten weit verbreitet [143][68]. Jedoch beachten diese Ansätze keine nebenläufigen Prozesse. Wenn ganze Sets von MSCs und nebenläufige Prozesse in einen Automaten transformiert werden, steigt die Zahl der Zustände exponentiell. In [89][90] wird mit *Global Finite State Machine (GFSM)* ein Ansatz beschrieben, mit dem die Größe eines globalen System-Models reduziert werden kann.

Bewertung In [89] wird die Transformation von einem Set an MSCs in einen Automaten betrachtet. Hierbei eignet sich die Darstellung in GFSM, um die Zahl der Zustände und Transitionen zu reduzieren. In folgenden Punkten kann der CompA-Ansatz abgegrenzt werden:

- In [89] werden keine Kasakdierungen von MSC-Elementen betrachtet und
- in CompA wird ferner ein Vergleich von MSC-Szenarien bzw. der zugehörigen Automaten bzgl. Rückwärtskompatibilität des beschriebenen Verhaltens durchgeführt.

Entwurf verteilter Systeme auf Basis von MSCs

In [87] wird eine Methode beschrieben, um aus Interaktionsszenarien von MSCs zu einer vollständigen Vorhabensbeschreibung zu gelangen. Hierfür werden zwei Transformationsverfahren vorgestellt:

- Schematische Extraktion von relationalen Assumption/Commitment-Spezifikationen aus MSCs.
- Syntaktische Transformation von MSCs in korrespondierende Zustandsautomaten.

Für das Forschungsthema CompA ist vor allem der 2. Ansatz relevant. Diesen Ansatz grenzt [87] in seiner Arbeit u.a. ausführlich von den folgenden Ansätzen ab: „Generating FSMs from interworkings“ [46], „Synthesizing ROOM Models from Message Sequence Chart Specifications “ [36] und von „Synthesizing Object Systems from LSC Specifications“ [68]. Da die Abgrenzungen vollständig auf CompA zutreffen, wird auf diese nicht weiter eingegangen, sondern auf [87] verwiesen.

Im Folgenden wird der Transformationsansatz genauer betrachtet. Ziel ist, durch Transformation mehrerer MSC-Szenarien zu einer vollständigen Zustandsbeschreibung einer Komponente zu kommen. Eine Komponente wird in [87] durch eine Instanz repräsentiert und die vollständige Zustandsbeschreibung soll auf Basis von Automaten erfolgen. Nachdem eine Instanz ausgewählt wurde, wird in 4 Schritten der korrespondierende Automat erzeugt (vgl. Abbildung 4.3):

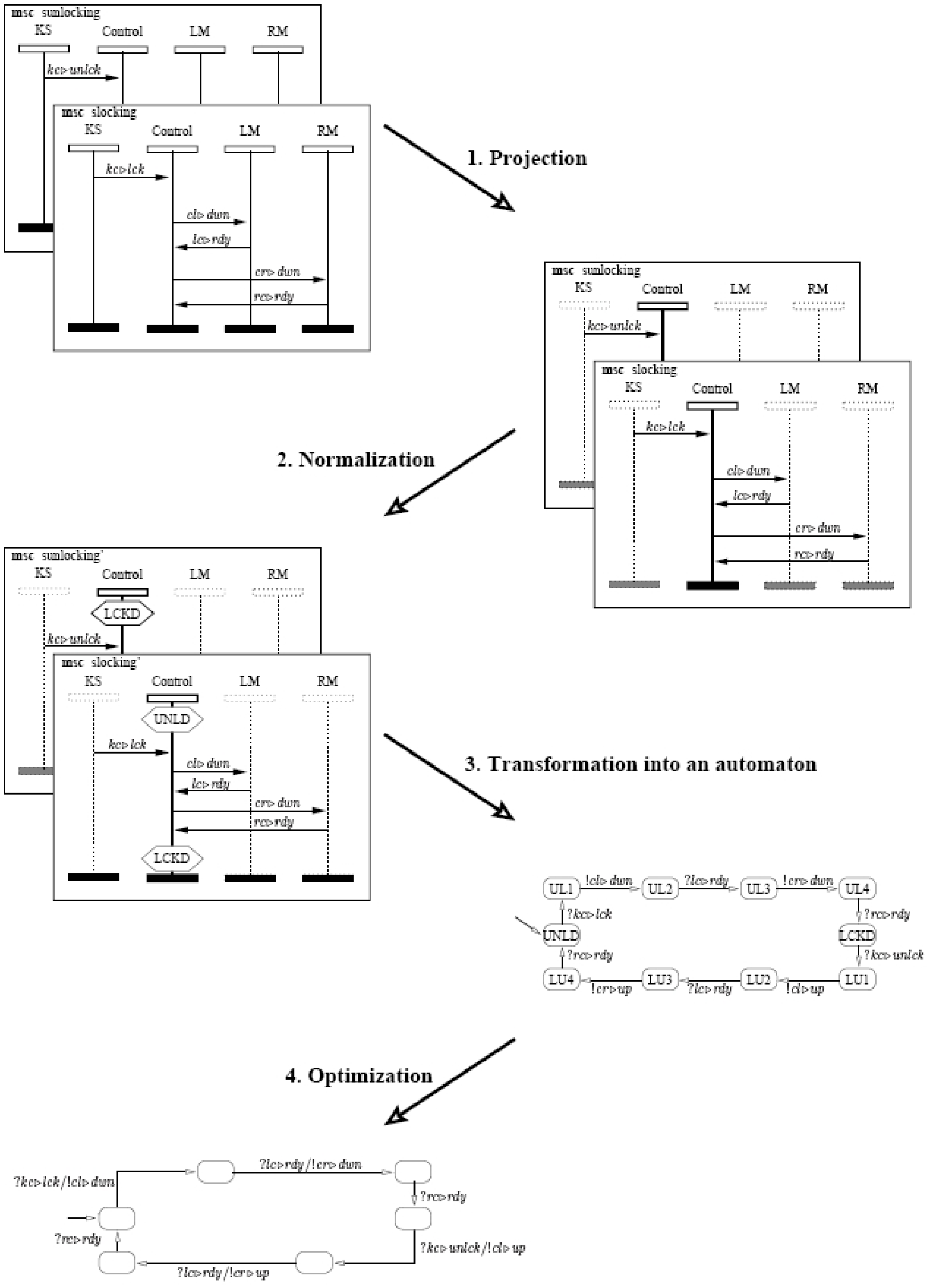


Abbildung 4.3: Prozess zur Transformation von MSCs in eine vollständige Verhaltensbeschreibung [87]

1. *Projection:* Nach der Auswahl einer Instanz werden die übrigen Instanzen ausgeblendet und auch die Messages, die weder auf der ausgewählten Instanz enden noch starten. Ergebnis sind *„projected MSCs“*.

2. *Normalization:* Jedes *„projected MSC“* beschreibt einen Teilzustand/Transitionspfad innerhalb des Automaten. Bei der Normalisierung wird, abstrakt gesehen, die Reihenfolge des Transitionspfades bzw. die Verknüpfung der *„projected MSCs“* festgelegt. Dies erfolgt über Anfangs- und Endbedingungen <*Guards*>, die in den MSCs spezifiziert werden. Sollten keine <*Guards*> spezifiziert sein, beschreibt [87] Regeln, wie und wo *Guards* eingefügt werden müssen (z.B. <*UNLD*>).

3. *Transformation into an Automaton:* Jede MSC-Message wird in eine Übergangsfunktion eines Automaten überführt. Zusätzlich werden Zustände (intermediate states z.B. $UNLD$) eingefügt, um die Transitionen so zu verbinden, dass diese zu der Sequenz der Messages innerhalb der normalisierten MSCs passen.

4. *Optimazation:* Das Ergebnis der Transformation sind nicht-deterministische Automaten. Auf diese Automaten werden anschließend aus der Automatentheorie bekannte Optimierungsverfahren angewendet.

Bewertung Die Ziele von [87] und von CompA sind vom Grundsatz her verschieden. Während [87] eine vollständige Verhaltensbeschreibung auf Basis mehrerer MSC-Szenarien generieren will, fokussiert CompA einen abstrakten Vergleich von MSCs. Des weiteren fokussiert CompA nicht die Verhaltensbeschreibung einer Komponenten, sondern betrachtet MSC-Szenarien immer als Ganzes. Dennoch können hier sehr große Synergien genutzt werden. Während die Schritte 1 und 2 für den CompA-Ansatz nicht notwendig sind, ist der 3. und 4. Schritt die Basis, auf die CompA für die Transformationen von MSCs in Automaten aufsetzt. CompA überträgt den Ansatz von [87] auf ein anderes Forschungsgebiet und erweitert diesen hierfür entsprechend. So werden im Rahmen von CompA beispielweise auch Transformationsregeln für verschachtelte Inline-Expressions eingeführt.

4.5 Zusammenfassung

In diesem Kapitel wurde der Stand der Technik zusammengefasst.
Im 1. Teil wurden hierzu verschiedene Spezifikationsansätze evaluiert, die zur Spezifikation von Steuergeräten eingesetzt werden bzw. eingesetzt werden sollen. Alle betrachteten Ansätze wurden bzgl. ihrer Verwendbarkeit für CompA analysiert. Es konnten zwar Synergien entdeckt werden, es gab aber keinen Ansatz, der ein Metamodell für eine Steuergeräte-Spezifikation zur Verfügung stellt.
Im 2. Teil wurden Ansätze betrachtet, die sich zur Kompatibilitätsprüfung von Steuergeräten eignen. Hier konnten vor allem mit dem Projekt MFA Synergien und Gemeinsamkeiten aufgezeigt werden.
Der 3. Teil evaluierte XML-Vergleichsmethoden und analysierte, inwiefern diese für einen XML-Kompatibilitätsvergleich eingesetzt werden können. Das Ergebnis ist, dass die vorhanden Methoden nicht über die notwendigen Erweiterungen zur Analyse von

Rückwärtskompatibilität verfügen.
Im 4. Teil wurden Methoden zum MSC-Vergleich betrachtet. Der Vergleich von MSCs erfolgt im Rahmen von CompA auf Basis von Automaten. Hierzu wurde vor allem ein Ansatz von [87] betrachtet, dessen Ideen für CompA zum Teil genutzt werden konnten. Die Ideen wurden jedoch für CompA um neue Features erweitert.

5 Abstrakter Spezifikationsansatz für Steuergeräte

Heutige Spezifikationstechniken (z.B. Doors, Word) basieren auf Prosa-Text-Spezifikationen und eignen sich daher nur bedingt als Basis für eine strukturierte Kompatibilitätsanalyse. Aus diesem Grunde wurde im Rahmen von CompA ein abstrakter Spezifikationsansatz für Steuergeräte definiert.
Dieser Spezifikationsansatz muss spezifische Voraussetzungen erfüllen, damit auf Basis der Steuergeräte-Spezifikationen (SG-Spezifikationen) eine Aussage bzgl. der Rückwärtskompatibilität von Steuergeräten getroffen werden kann:

- *hinreichende Spezifikation:* Alle Anforderungen, die für eine Kompatibilitätsbetrachtung relevant sind, müssen in Lastenheften spezifiziert sein.

- *formale Spezifikation:* Ein systematischer Vergleich von SG-Spezifikationen ist nur auf Basis von „formalen“ Spezifikationen möglich.

Die Basis des abstrakten Spezifikationsansatzes ist die Generierung eines Metamodells für Steuergeräte-Spezifikationen, das künftig als Steuergeräte-Metamodell (SG-Metamodell) bezeichnet wird. Das SG-Metamodell wird mit Hilfe von Expertenwissen aufgebaut und legt fest, welche Elemente eines Steuergerätes für eine hinreichende Spezifikation notwendig sind. Auf Basis des SG-Metamodells können dann spezifische Steuergeräte-Spezifikationen (Lastenhefte) abgeleitet werden.
In Kapitel 5.1 wird die Strukturierung des SG-Metamodells beschrieben. Anschließend wird in Kapitel 5.2 die Portierung des SG-Metamodells auf ein XML-Schema vorgestellt.

5.1 Abstraktionsansatz zur Erstellung eines Steuergeräte-Metamodells (SG-Metamodell)

Das im Rahmen von CompA definierte SG-Metamodell folgt einem spezifischen Aufbau, der von verschiedenen Aspekten beeinflusst wurde:

- *objektorientierte Gedankenwelt:* Definition von Klassen, Attributen, Typisierung und Strukturierung.

- *IPQ-Projekt:* Schema-basierte Definition (vgl. Kapitel 4.1.6).

- *Strukturierung:* Aufbau des SG-Metamodells als Entscheidungsbaum. Dies bedingt wiederum folgende Kriterien:
  - eine klare hierarchische Strukturierung
  - eine nicht-rekursive Strukturierung

Im Folgenden wird der spezifische Aufbau des SG-Metamodells genauer erläutert.

5.1.1 Strukturierungsregeln für SG-Metamodell

Für den Aufbau eines SG-Metamodell wurden im Rahmen von CompA spezifische Strukturierungsregeln definiert. Die Strukturierungsregeln bauen auf einem 4-Ebenen-Modell auf (vgl. Abbildung 5.1):

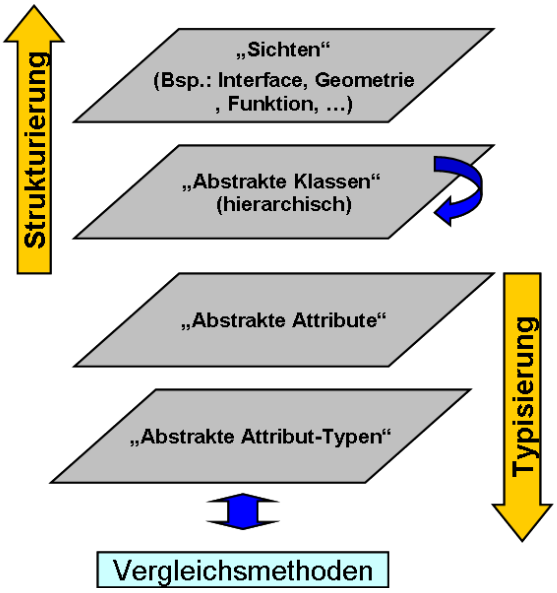


Abbildung 5.1: Abstrahierung der Spezifikations-Ebene auf ein 4-Ebenen-Modell

- ***Sichten***: Ein Steuergerät (SG) kann aus verschiedenen Blickwinkeln betrachtet werden. Hierzu zählen insbesondere die SG-Schnittstelle (Interface), die SG-Geometrie und die SG-Funktion. Weitere Sichten können definiert werden. Die Einführung von Sichten zur Entwurfsstrukturierung ist bereits aus [62][63] bekannt und erprobt.

- ***Abstrakte Klassen***: Durch *abstrakte Klassen* können Sichten strukturiert untergliedert und beschrieben werden. Hierbei kann eine Klasse, analog zur Objektorientierung, durch weitere Klassen definiert sein.

- ***Abstrakte Attribute***: *Abstrakte Attribute* beschreiben die Eigenschaften einer *abstrakten Klasse*.

- ***Abstrakte Attribut Typen***: Jedem *abstrakten Attribut* ist ein spezifischer Typ zuzuordnen z.B. *Integer*.
  Des Weiteren wird für jeden *abstrakten Attribut Typen* eine Regel zum Kompatibilitätsvergleich definiert. Dies wird in Abschnitt 5.1.2 näher erläutert werden.

Ein SG-Metamodell, das auf Basis dieser Strukturierungsregeln erstellt wird, besitzt eine nicht-rekursive Struktur. Bottom-Up betrachtet wird damit eine Strukturierung und Top-Down eine Typisierung erreicht. Bevor auf den Vorteil dieser Strukturierung näher eingegangen wird, sei kurz ein Beispiel eines SG-Metamodells vorgestellt.

Beispiel In Abbildung 5.2 ist links die Strukturierung der 4 Ebenen graphisch dargestellt und rechts daneben ein darauf aufbauendes SG-Metamodell. Eine spezifische

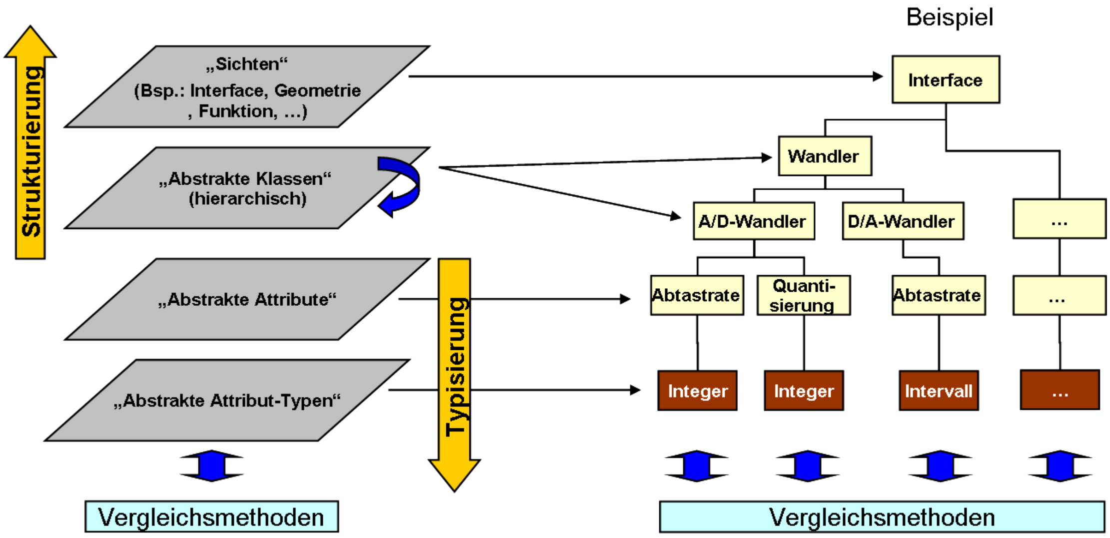


Abbildung 5.2: Beispiel für Anwendung des 4-Ebenen-Modell

Sicht auf ein Steuergerät ist beispielsweise die SG-Schnittstelle (*Interface*). Das *Interface* ist mit *Wandlern (Abstrakte Klasse)* beschaltet. Es gibt verschiedene *Abstrakte Klassen* von Wandler z.B. A/D-Wandler, D/A-Wandler. Die Wandler selbst werden durch verschiedene *Abstrakte Attribute* spezifiziert z.B. durch *Quantisierung* und *Abtastrate*. Der *Abstrakte Attribut Typ* der beiden *Attribute* für den A/D-Wandler ist jeweils *Integer* und für das *Attribut* des D/A-Wandler ist der Typ *Intervall*.

5.1.2 Ableitung von SG-Spezifikationen und abstrakter Typ-Vergleich

Ein SG-Metamodell gibt vor, welche Elemente eines Steuergerätes beschrieben werden müssen. Auf Basis des SG-Metamodells können anschließend SG-Spezifikationen abgeleitet werden. Um Aussagen bzgl. der Rückwärtskompatibilität zu treffen, müssen

die Steuergeräte-Spezifikationen verglichen werden. Aufgrund des gemeinsamen SG-Metamodells ist die Struktur der beiden zu vergleichenden SG-Spezifikationen festgelegt und damit auch, welche Elemente miteinander zu vergleichen sind. Damit kann ein Vergleich bei übereinstimmenden Strukturen auf einen *abstrakten Attribut Typ*-Vergleich beschränkt werden.
In Abbildung 5.3 ist dies beispielhaft dargestellt. Aus dem SG-Metamodell werden zwei Steuergeräte-Spezifikationen (SG_1-Spezifikation, SG_2-Spezifikation) abgeleitet. Bei einem Vergleich der beiden Spezifikationen ist durch die Struktur implizit festgelegt, welche Elemente zu vergleichen sind. In der SG_1-Spezifikation ist der A/D-Wandler durch das *abstrakte Attribut* Abtastrate spezifiziert und in der SG_2-Spezifikation durch das *abstrakte Attribut* Quantisierung. Der A/D-Wandler ist also in den beiden Spezifikation unterschiedlich spezifiziert. Dies äußert sich auch in der unterschiedlichen Struktur der SG-Spezifikationen. Daher bricht ein Vergleich der Spezifikationen bei dem Pfad-Element A/D-Wandler ab. Der D/A-Wandler hingegen weist in beiden Spezifikationen die gleiche Struktur auf, d.h. für den D/A-Wandler wurden dieselben Struktur-Elemente spezifiziert. Damit müssen nur die Blattknoten bzw. die Werte der *abstrakten Attribute* verglichen werden. Da die Werte einen spezifischen *abstrakten Attribut Typ* besitzen, erfolgt der Vergleich entsprechend dieses Typs. In dem Beispiel ist der *abstrakte Attribut Typ* ein Intervall. Da ein Intervall mindestens aus einem min- und einem max-Wert besteht, ist die vorgeschlagene Vergleichsmethode ein Strukturvergleich. Für jeden abstrakten Attribut-Typ ist also eine abstrakte Typ-Vergleichsmethode zu definieren.

5.1.3 Konkretisierung der hierarchischen Ebenen

Welche Elemente zur Spezifikation eines Steuergerätes hinreichend bzw. notwendig sind, kann abhängig vom Steuergeräte-Typ und vom Anwendungsgebiet sein. Im Rahmen von *CompA* wurde ein SG-Metamodell für Automotive-Steuergeräte entworfen. Die unter Abschnitt 5.1.1 definierten Strukturierungsregeln geben vor, wie ein SG-Metamodell aufzubauen ist. Die Sichten werden auf Basis der Definition 1 (Rückwärtskompatibilität) extrahiert.

Sichten

Um eine high-level Abstraktion zu ermöglichen, fordert der CompA-Ansatz, analog zu IPQ (vgl. Kapitel 4.1.6) eine Abstraktion durch verschiedene Blickwinkel bzw. Sichten. Bei dieser Abstraktion lassen sich für Steuergeräte vier Sichten definieren (s. Abbildung 5.4). Jede Sicht ist für die Analyse von Rückwärtskompatibilität relevant und muss daher hinreichend beschrieben werden. Dies wird deutlich, wenn die Definition für Rückwärtskompatibilität (Definition 1, Kapitel 3.1) mit den verschiedenen Sichten abgeglichen wird.

Sicht 1 ***Interface:*** Beschreibung von Schnittstellen bzgl. Stecker, Pin, Spannung, Protokolle, etc. (relevant für Punkt 1 und 3 von Definition 1)

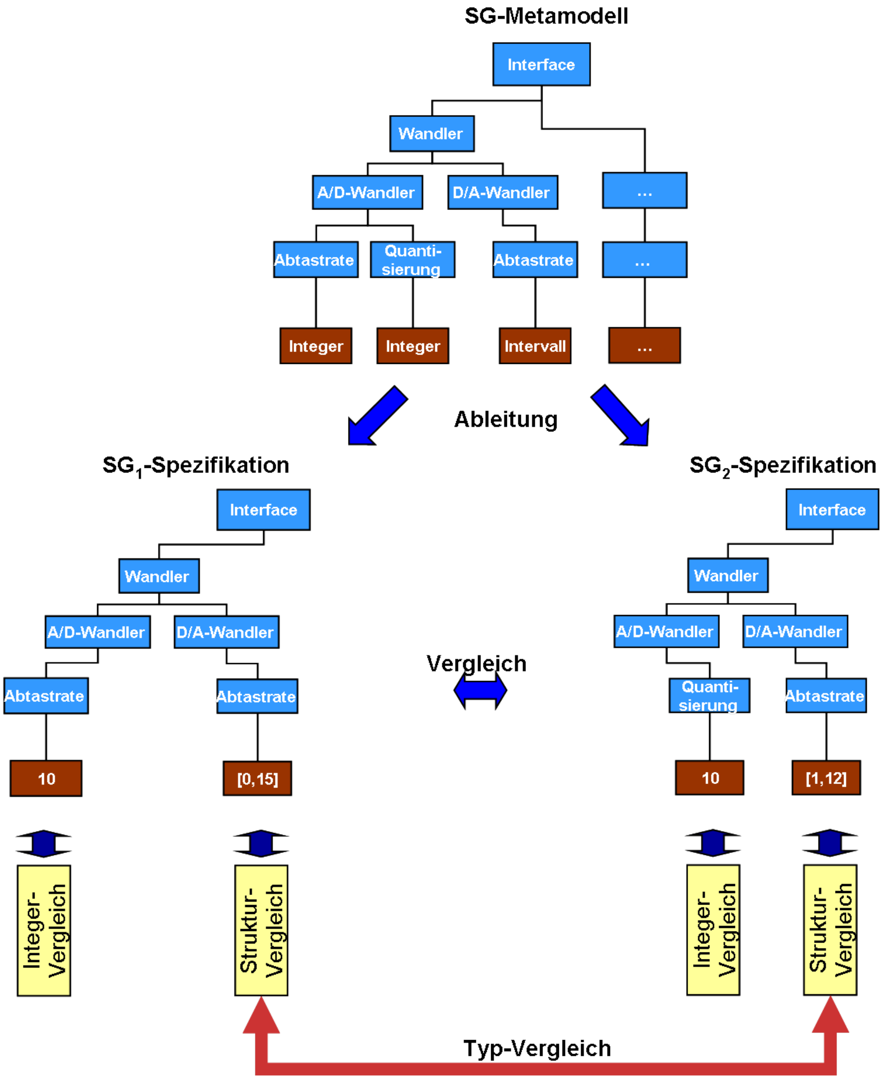


Abbildung 5.3: Ableitung und *abstrakter Attribut Typ*-Vergleich

Sicht 2 ***Funktion:*** Beschreibung des dynamischen Verhaltens von Steuergeräten (relevant für Punkt 4 und 5 von Definition 1).

Sicht 3 ***Geometrie:*** Beschreibung von Abmessungen, Gewicht, etc. eines Steuergerätes (relevant für Punkt 1 von Definition 1).

Sicht 4 ***Logistik:*** Beschreibung von Informationen, die für die Logistik relevant sind, wie z.B. Flashen, Diagnose, Identifikationsnummern, etc. (relevant für Punkt 1 von Definition 1).
Umgebung: Beschreibung eines Steuergerätes hinsichtlich äußerer Umgebungsbedingungen, wie beispielsweise Umgebungstemperatur, Elektromagnetische Verträglichkeit (EMV) etc. (relevant für Punkt 2 der Definition 1).

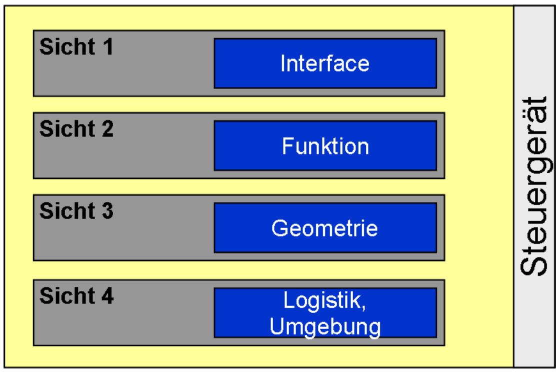


Abbildung 5.4: Abstraktion eines Steuergerätes auf verschiedene Sichten

Bei der Definition der Sichten zeigt sich, dass eine weitere Kategorisierung der Sichten in dynamische Sichten und statische Sichten sinnvoll ist (Abbildung 5.5).

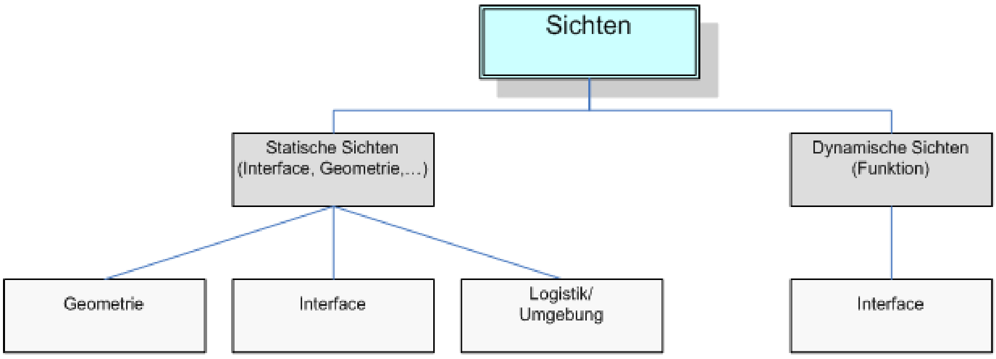


Abbildung 5.5: Kategorisierung der Sichten

Statische Sicht *Statische Sichten* sind dadurch gekennzeichnet, dass diese durch „nicht zeitabhängige“ Attribute beschrieben werden können. Zu den statischen Sichten zählen: *Interface* (Sicht 1), *Geometrie* (Sicht 3) und *Logistik, Umgebung* (Sicht 4).

Dynamische Sicht Zur Kategorie *dynamische Sichten* zählt die Sicht 2: *Funktion*. Die Funktionen der Steuergeräte korrespondieren mit dem dynamischen bzw. zeitabhängigem Verhalten der Steuergeräte. Im Rahmen von CompA erfolgt die Spezifikation dieser Sicht mit Hilfe von Message Sequence Charts.

Abstrakte Klassen und Attribute

Die Definition bzw. Extraktion der *abstrakten Klassen*, *abstrakten Attribute* erfolgt durch empirische Analysemethoden und Expertenwissen. Hierfür wurde auf folgendes

Wissen zurückgegriffen:

- Vorhandene Entwicklungslastenhefte z.B. [133][85][54]
- Muster-Lastenheft [10] und Standardvorlage Komponentenlastenheft (VDA) [135]
- Beschreibungsansätze wie z.B. MSR (vgl. Kapitel 4.1.2)
- Expertenbefragungen

Das Ergebnis ist eine Strukturierung der Elemente, die zur hinreichenden Spezifikation von Steuergeräten notwendig sind.

Ergebnis Strukturierung statischer Sicht In Abbildung 5.6 ist als Beispiel ein kleiner Ausschnitt aus der Strukturierung der statischen Sicht *Interface* durch *abstrakte Klassen* und *abstrakte Attribute* dargestellt. Eine SG-Schnittstelle (*Interface*) kann mit der *abstrakten Klasse* „Hardware“ beschaltet sein, die sich wiederum in die *abstrakten Klassen D/A-Wandler*, *A/D-Wandler* und *D/D-Wandler* untergliedert. Die abstrakten Attribute der Klassen sind *Messtoleranz*, *Abtastrate*, *Messgenauigkeit* und *Hardwareauflösung*. Die Schnittstellen können aber auch durch verschiedene Treiber realisiert sein. Die *abstrakte Klasse* „Art des Treibers“ untergliedert sich in *High-Side Treiber*, *Low-Side Treiber*, *Halb-Brücke* und *Voll-Brücke*. Die *Treiber* und *Brücken* werden in

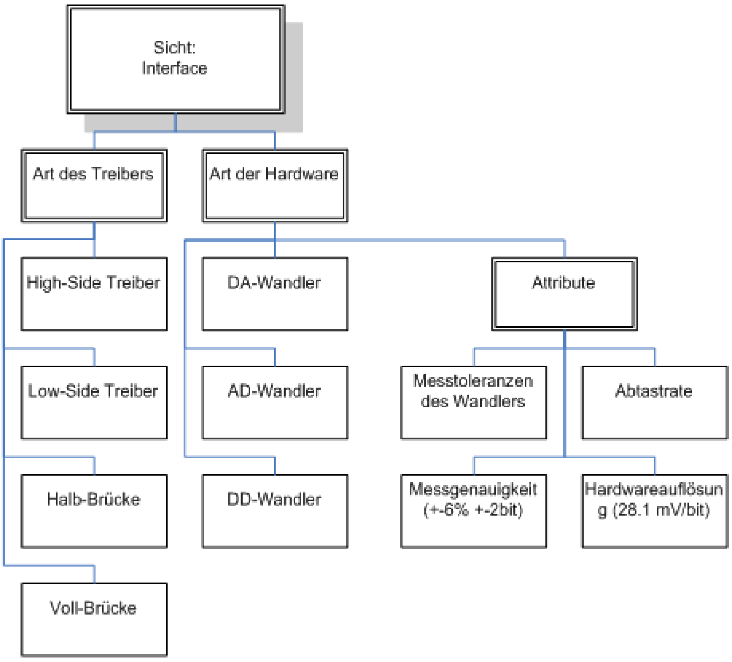


Abbildung 5.6: Strukturierung der Sicht *Interface* durch *abstrakte Klassen* und *abstrakte Attribute*

diesem Beispiel nicht genauer spezifiziert, sondern dienen einer genaueren Spezifikation der „Art des Treibers“ und stellen somit die *abstrakten Attribute* dar. Der *abstrakte Typ* wäre in diesem Fall *String*.

Ergebnis Strukturierung dynamischer Sicht In Abbildung 5.7 ist ein weiterer Ausschnitt aus dem SG-Metamodell angegeben. Er zeigt beispielsweise die Strukturierung der dynamischen Sicht und die Zugehörigkeit zu den hierarchischen Ebenen. Die dynamische Sicht wird beschrieben durch eine Vielzahl an Funktionen (Funktionsliste). Hierbei werden *vollständige Beschreibungen* und *Use-Case Beschreibungen* unterschieden. Bei den *Use-Case Beschreibungen* gibt es zwei Möglichkeiten:

- Message Sequence Charts (vgl. Kapitel 2.3)
- Sequenzdiagramme (vgl. Kapitel 4.1.5)

Bei den *vollständigen Beschreibungen* werden zeitdiskrete und kontinuierliche Modelle unterschieden. Die Modelle können durch die Methoden *Stateflow* oder *Ascet/Simulink* realisiert werden (vgl. Kapitel 4.1.4). Ein *abstraktes Attribut* ist z.B. *Source-Code*. Der zugehörige *abstrakte Attribut-Typ* ist vom Typ *Struktur I*. Für den *abstrakten Attribut-Typ* muss eine spezifische Vergleichsmethode verwendet werden, eine Möglichkeit wäre hier z.B ein Modelchecker [26]. Weitere Details zur Extraktion des SG-Metamodell sind in [118] zu finden. Im Rahmen von CompA wird auf die Spezifikation von dynamischem Verhalten mittels MSCs fokussiert. Weitere Spezifikationsmethoden können im SG-Metamodell einfach ergänzt werden.

5.2 Konzeption eines XML-SG-Schemas zur Spezifikation von Steuergeräten

In Abschnitt 5.1 wurde eine Vorgehensweise zum Entwurf eines SG-Metamodells vorgestellt. Die konkrete Umsetzung des SG-Metamodells erfolgt auf Basis der formalen Sprache *XML (Extensible Markup Language)*.
In Abschnitt 5.2.1 wird beschrieben, weshalb im *CompA*-Ansatz die *Extensible Markup Language (XML)* zur Spezifikation von Steuergeräten verwendet wird. Anschließend wird in Abschnitt 5.2.2 die Portierung des SG-Metamodells nach XML bzw. nach XML-Schema ausgeführt.

5.2.1 Entscheidung für XML-Schema

Das SG-Metamodell ist ein abstraktes Modell zur Spezifikation von Steuergeräten. Allerdings existieren hierfür keine formalen Sprachelemente und keine Methoden, um IT-technisch mit diesem Modell zu arbeiten. Dieses Defizit kann durch die Portierung des SG-Metamodells auf XML ausgeglichen werden. XML bringt hierfür bereits verschiedene Features mit (vgl. Kapitel 2.2):

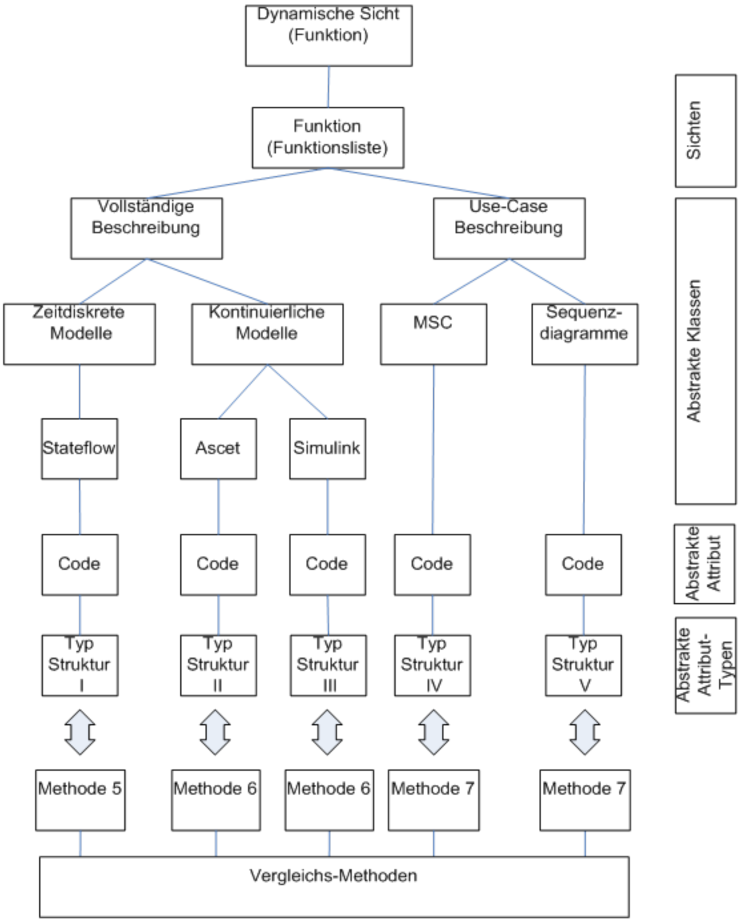


Abbildung 5.7: Abstrakte Übersicht über ein SG-Metamodell

- *Formal:* XML ist eine von W3C standardisierte und formal beschriebene Sprache.
- *Trennung von Inhalt und Struktur:* Das SG-Metamodell wird im XML-Schema hinterlegt und die SG-Spezifikationen werden in XML-Dokumenten beschrieben.
- *Methoden-Unterstützung:* Für XML existieren bereits eine Vielzahl an Methoden:
  - *Prüfmethoden:* Spezifische Methoden zur Prüfung, ob XML-Dokumente (Steuergeräte-Spezifikationen) *wohlgeformt* und *valide* gegenüber dem SG-Metamodell (XML-Schema) sind.
  - *DOM/Parser/API:* Spezifische Methoden zum Bearbeiten/Traversieren der XML-Dokumente/Schemata.

Für XML-Schemata sind des Weiteren eine Vielzahl an komplexen Beschreibungskonstrukten definiert, die eine Strukturierung eines XML-Schemas gemäß den Anforde-

rungen des SG-Metamodells unterstützen. Als Beispiel seien die Konstrukte *optionale Pfade/Elemente*, *einfache/komplexe Inhaltsmodelle* und *restriction* genannt, die bereits in Kapitel 2.2.3 erläutert wurden.
Die Portierung des SG-Metamodells nach XML wird dadurch unterstützt, dass beide Ansätze eine hierarchische Strukturierung bzw. die Baumstruktur unterstützen.

5.2.2 Portierung eines SG-Metamodells auf XML-SG-Schema

XML-SG-Schema In diesem Abschnitt wird die Portierung eines SG-Metamodells auf ein XML-Schema für Steuergeräte (XML-SG-Schema) vorgestellt. In Abbildung 5.8 ist dies dargestellt. Links ist ein SG-Metamodell abgebildet, das auf Basis des 4 Ebenen-Modells aufgebaut worden ist. So teilt sich die „Sicht“ *Interface* in die „Abstrakten Klassen“ *Pins* und *Signale*. Pins werden wiederum beschrieben durch „Abstrakte Attribute“ *Spannung* und *Strom* mit den „Abstrakten Attribut Typen“ *Intervall* und *Integer*.

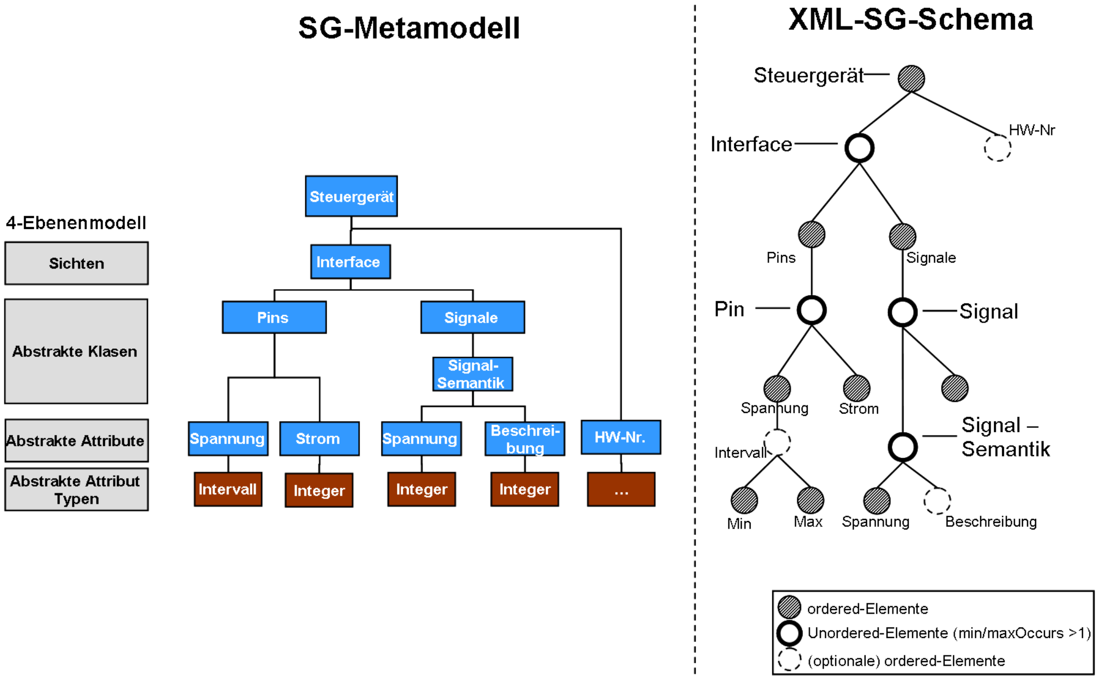


Abbildung 5.8: Portierung eines SG-Metamodells auf ein XML-Schema

Rechts in Abbildung 5.8 ist das zugehörige XML-Schema des Steuergerätes (kurz: XML-SG-Schema) dargestellt. Das XML-SG-Schema stellt die Konkretisierung des SG-Metamodells dar. Bei der Portierung auf XML-SG-Schema wird zwar die Strukturierung großteils übernommen, jedoch nicht die Begrifflichkeiten wie *abstrakte Klassen*, *abstrakte Attribute*. Grund hierfür ist, dass die Begrifflichkeiten zwar für die Strukturierung des SG-Metamodells benötigt wurden, aber bei der Konkretisierung in XML

zu Begriffs-Konflikten führen.
In XML-SG-Schema werden drei Elemente unterschieden:

- ordered-Elemente
- unordered-Elemente
- optionale-Elemente

Ordered-Elemente sind bzgl. ihrer Position fest verankert. *Unordered*-Elemente sind vom Typ *min/maxOccurs>1* und dürfen daher in dem zugehörigen XML-Dokument bzw. in der XML-SG-Instanz mehrfach auftreten. Optionale Elemente dürfen, müssen aber nicht in der XML-SG-Instanz auftreten.
Die Struktur des SG-Metamodells ist im XML-SG-Schema sofort wieder zu erkennen. Der Unterschied sei am Beispiel von *Pins* erläutert. Im SG-Metamodell wird lediglich die Strukturierung festgelegt. Im XML-SG-Schema kann nun hinterlegt werden, wie viele *Pins* in einer SG-Spezifikation definiert werden können. Daher wird im XML-SG-Schema *Pins* ein *unordered*-Element *Pin* untergeordnet, das mit der Spezifikation eines einzelnen Pins korrespondiert. Durch die Kennzeichnung mit *min/maxOccurs>1* kann das Element zusammen mit dessen Kind-Elementen mehrfach in einer SG-Spezifikation definiert werden.

Ableitung In Abbildung 5.9 ist die Ableitung einer SG-Spezifikation aus dem XML-SG-Schema dargestellt. Die Ableitung bzw. die Instanz wird künftig als XML-SG-Instanz bezeichnet. *Pin* wurde im Schema mit *min/maxOccurs>1* spezifiziert. Daher können in der XML-SG-Instanz mehrere Pins (Pin P_1, P_2) mit jeweils eigenen Unterbäumen definiert sein. Die Unterscheidung zwischen *unordered-/ordered*-Elemente wird in der XML-SG-Instanz nicht mehr getroffen. Die Markierung für die *unordered*-Elemente dient lediglich der Übersichtlichkeit.

5.2.3 Beispiele für Konkretisierung XML-SG-Schema

Beispiel: Sichten In Abbildung 5.10 ist die oberste Ebene des XML-SG-Schemas dargestellt, die verschieden Sichten. In Listing 5.1 ist der zugehörige Ausschnitt in XML beschrieben. Der Wurzelknoten des XML-SG-Schemas ist das Element *<steuergeraet>*. Die direkten Kindelemente von *<steuergeraet>* sind die, für die Steuergeräte, definierten Sichten: Interface-, Funktions-, Geometrie- und Logistik-/Umgebungs- Sicht.

Listing 5.1: Umsetzung der Sichten in XML-Schema

```
<xsd:schema targetNamespace="http://bmw.com/2006/development/
    ecuinterfacescheme" .... > <xsd:include schemaLocation="
    interfaceDescription.xsd"/> <xsd:include schemaLocation="
    logisticsDescription.xsd"/> <xsd:include schemaLocation="
    geometryDescription.xsd"/> <xsd:include schemaLocation="
    functionDescription.xsd"/> <xsd:element name="steuergeraet">
```

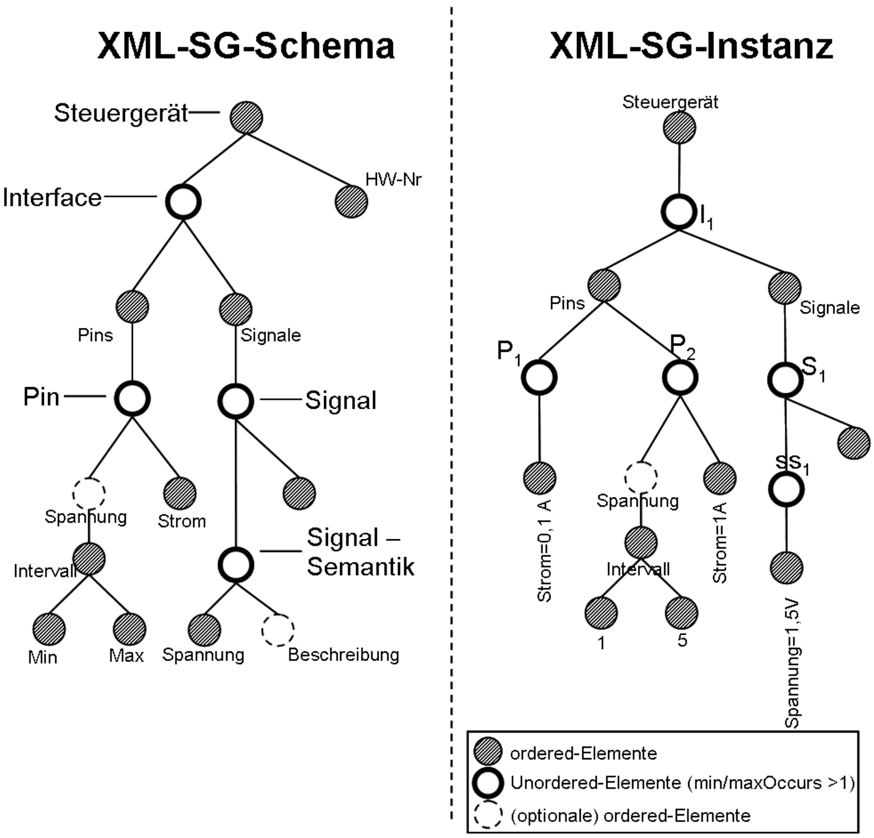


Abbildung 5.9: Beispiel für Ableitung einer XML-SG-Instanz aus einem XML-SG-Schema

```
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name=”schnittstellenSicht” type=”ecu:
        interfaceDescription”/>
      <xsd:element name=”funktionsSicht” type=”ecu:
        functionDescription”/>
      <xsd:element name=”logistikSicht” type=”ecu:
        logisticsDescription”/>
      <xsd:element name=”geometrieSicht” type=”ecu:
        geometryDescription/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element> </xsd:schema>
```

Die Sichten besitzen jeweils einen spezifischen Datentyp, der durch *include* weitere Unterbäume einbindet.

Beispiel: Baumstruktur Bei der Umsetzung der Baumstruktur (also der Strukturierung durch die *abstrakten Klassen*, *abstrakten Attribute*) kommt meist das komplexe Inhaltsmodell zum Einsatz. In Abbildung 5.11 ist ein Pfad der Baumstruktur des XML-SG-Schemas graphisch dargestellt. Die <*SchnittstellenSicht*> teilt sich auf

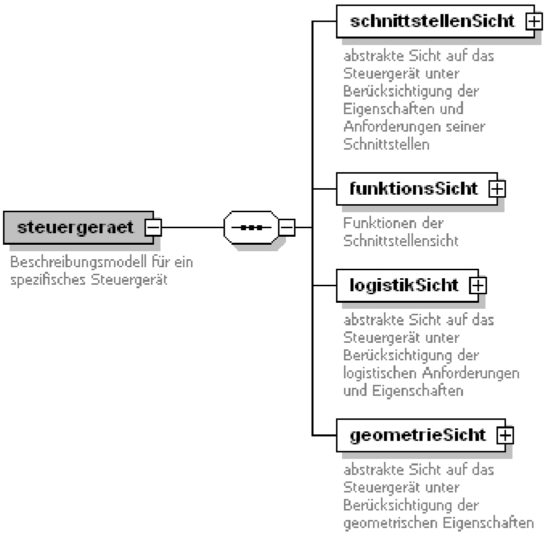


Abbildung 5.10: Beispiel für Konkretisierung Sicht

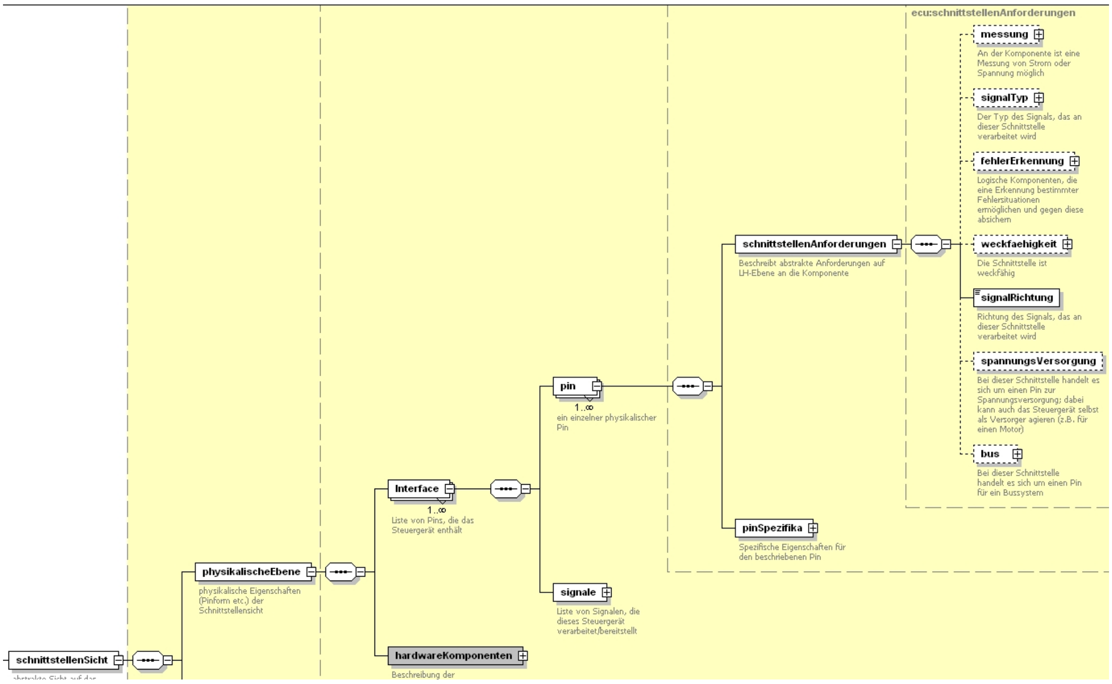


Abbildung 5.11: Beispiel für Kaskadierung von *abstrakten Klassen*

in <*physikalische Ebene*> und <*SignalEbene*>. Die <*physikalische Ebene*> kann wiederum in <*Interface*> und <*HardwareKomponenten*> gegliedert werden. Da ein Steuergerät mehrere Schnittstellen bzw. Interfaces haben kann, ist *Interface* vom Typ $maxoccurs > 1$ definiert und kann somit in der XML-SG-Instanz mehrfach instanziiert werden. Jedes Interface besitzt in der Regel auch mehrere Pins. Aus diesem Grund ist das Element Pin ebenfalls vom Typ $maxoccurs > 1$ definiert. Diese Strukturierung kann mit weiteren Elementen entsprechend weitergeführt werden.

Beispiel: Intervall In Abbildung 5.12 ist die Umsetzung eines Intervalls in XML-SG-Schema abgebildet. In Listing 5.2 ist die dazugehörige Umsetzung in *XML* dargestellt. Die Intervall-Struktur ist wie folgt aufgebaut: Man hat die Wahl, entweder einen Fix-Wert oder ein Intervall anzugeben. Die Auswahl ist durch das komplexe Inhaltsmodell <*choice*> umgesetzt. Das Intervall selbst besitzt ebenfalls ein komplexes Inhaltsmodell <*sequence*> und definiert, dass für ein *Intervall* ein *min*-, ein *max*- und ein *typischer* Wert angegeben werden kann. Die Werte müssen jedoch nicht angegeben werden, da alle Elemente das Attribut *minOccurs=”0”* besitzen.

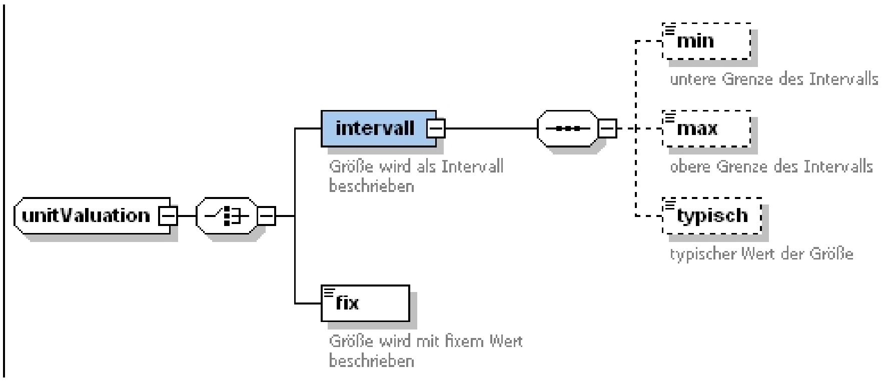


Abbildung 5.12: Beispiel für Struktur eines Intervalls

Listing 5.2: Beispiel für Umsetzung eines Intervalls

```
<xsd:complexType name=”unitValuation”>
    <xsd:choice>
        <xsd:element name=”intervall”>
            <xsd:complexType>
                <xsd:sequence>
                    <xsd:element name=”min” type=”xsd:float” minOccurs=”0”/>
                    <xsd:element name=”max” type=”xsd:float” minOccurs=”0”/>
                    <xsd:element name=”typisch” type=”xsd:float” minOccurs=”0”/>
                </xsd:sequence>
```

```
                </xsd:complexType>
            </xsd:element>
            <xsd:element name=”fix” type=”xsd:float”/>
        </xsd:choice>
        <xsd:attribute name=”siVorsatz” type=”ecu:siUnitPrefix” use=”required”>
        </xsd:attribute>
    </xsd:complexType>
```

5.3 Zusammenfassung

In diesem Kapitel wurde der abstrakte Spezifikationsansatz für Steuergeräte beschrieben. Hierzu wurde ein 4-Ebenen-Modell, bestehend aus *Sichten-Ebene, Abstrakte Klassen, Abstrakte Attribute, Abstrakte Attribut Typen*, vorgestellt. Dieser abstrakte Spezifikationsansatz diente der Erstellung eines Steuergeräte-Metamodells (SG-Metamodells). Die konkrete Umsetzung des SG-Metamodells erfolgt auf Basis eines XML-SG-Schemas. Hierzu wurden kurz die Vorteile von XML, die Portierung des SG-Metamodells auf XML-SG-Schema und die Ableitung einer XML-SG-Instanz vorgestellt.

6 XML-Vergleichsmethode zur Analyse von Rückwärtskompatibilität (*XML-CompA*)

In Kapitel 5 wurde ein XML-SG-Metamodell vorgestellt, auf dessen Basis Steuergeräte-Spezifikationen (XML-SG-Instanzen) erstellt werden können. Um Aussagen bzgl. der Rückwärtskompatibilität von Steuergerät SG_2 zu Steuergerät SG_1 treffen zu können, müssen die zugehörigen XML-SG-Instanzen miteinander verglichen und entsprechende Aussagen generiert werden.
Zu diesem Zweck wurde im Rahmen von CompA eine Kompatibilitäts-Vergleichsmethode für XML-SG-Instanzen definiert. Diese Methode sei künftig als *XML-CompA* (XML-Compatibility Analysis) bezeichnet.
Im nächsten Abschnitt 6.1 werden An- bzw. Herausforderungen dargestellt, die an *XML-CompA* gestellt werden. Anschließend wird in Abschnitt 6.2 das Konzept von *XML-CompA* erläutert und in den Abschnitten 6.3 bis 6.6 weiter vertieft. In Abschnitt 6.7 erfolgt dann die Zusammenfassung dieses Kapitels.

6.1 Motivation - Herausforderungen

Um Änderungen in XML-Dokumenten zu detektieren (Change Detection) gibt es unterschiedliche Ansätze, wie z.B. *XML Diff Algorithmen* (vgl. Kapitel 4.3). Diese könnten auch für XML-SG-Instanzen verwendet werden. Um jedoch Aussagen bzgl. Rückwärtskompatibilität von Steuergeräten auf Basis der XML-SG-Instanzen treffen zu können, werden spezifische Anforderungen an eine XML-Vergleichsmethode gestellt, die in den nachfolgenden Abschnitten beleuchtet werden.

6.1.1 Ordnungsstruktur - Zuordnung

Ordnungsstruktur

In Abbildung 6.1 ist ein XML-SG-Schema mit einer zugehörigen XML-SG-Instanz dargestellt. In der XML-SG-Instanz treten mehrere Interfaces (I_1, I_2), mehrere Pins $(P_1, P_2, P_3, P_4, P_5)$, etc. auf. Dies ist valide, wenn im XML-SG-Schema für diese Elemente das Attribut $(minOccurs/maxOccurs) > 1$ definiert wurde. Die Ordnungs-

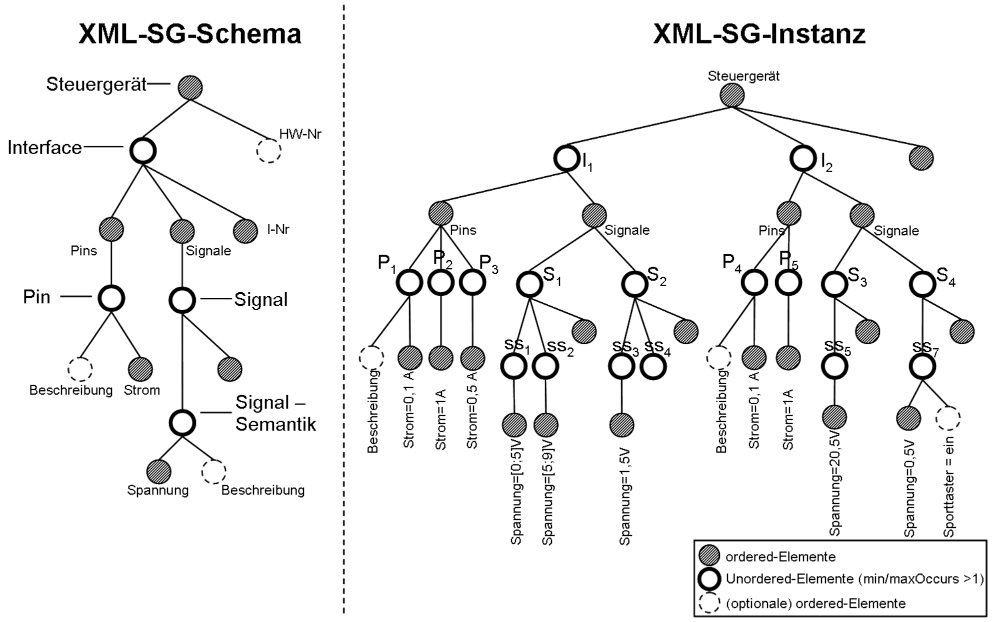


Abbildung 6.1: Ableitung einer XML-SG-Instanz aus einem XML-SG-Schema

struktur, d.h. an welche Position in der XML-SG-Instanz die Elemente gehängt werden, ist durch das XML-SG-Schema vorgegeben. Bei einer Mehrfach-Instanziierung von Elementen werden diese sequentiell, nach dem FIFO-Prinzip, angelegt.

Bei einer Mehrfach-Instanziierung sind zwei Ordnungsstrukturen zu unterscheiden: „unordered und ordered trees“ [141], also geordnete und ungeordnete Strukturen. Bei geordneten Strukturen ist die Anordnung relevant, also eine Links-Rechts-Vertauschung der mehrfach instanziierten Elemente (inkl. deren Unterbäume) ist damit nicht erlaubt. Bei ungeordneten Strukturen ist eine Links-Rechts-Vertauschung dagegen nicht von Bedeutung. Da XML-SG-Instanzen auf ungeordneten Strukturen basieren, muss diese erlaubte Permutation von XML-Vergleichsmethoden berücksichtigt werden können.

Dies sei anhand von Abbildung 6.1 kurz erläutert. Pins sind unordered-Elemente, besitzen also im XML-SG-Schema das Attribut $(minOccurs/maxOccurs) > 1$ und können damit mehrfach instanziiert werden. Die Pins von Interface (I_1) sind in der Reihenfolge (P_1, P_2, P_3) angeordnet. Wäre zuerst der *Pin 2* spezifiziert worden, wäre die Reihenfolge bspw. (P_2, P_1, P_3) gewesen. Die Reihenfolge ergibt sich also aus dem Arbeitsablauf des Designers und kann nicht vordefiniert werden. XML-Vergleichsmethoden müssen bei einem Vergleich nun erkennen, dass die XML-SG-Instanzen äquivalent sind, obwohl diese nicht identisch sind.

Zuordnung

In Abbildung 6.2 sind zwei XML-SG-Instanzen von SG_1 und SG_2 gegeben. SG_1 hat ein Interface und SG_2 hat zwei Interfaces, wobei das Interface $I_2(SG_2)$ iden-

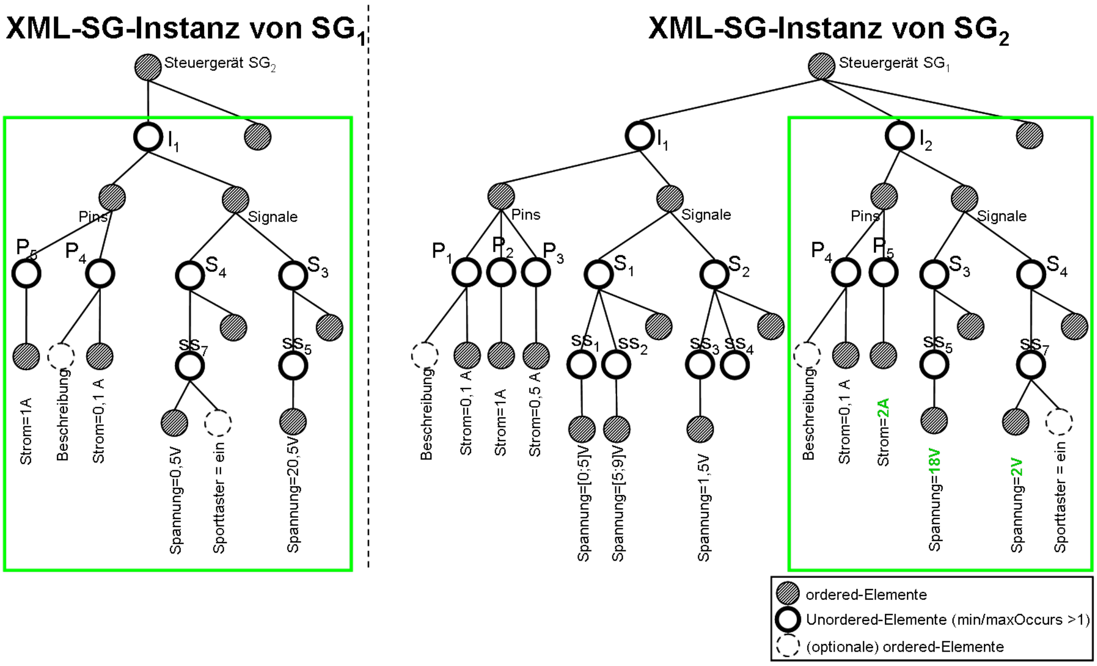


Abbildung 6.2: Vergleich von zwei XML-SG-Instanzen

tisch ist mit dem Interface $I_1(SG_1)$, bis auf einzelne Element-Werte (*Strom*(P_5)*=2A*, *Spannung*(ss_5)*=18V*, *Spannung*(ss_7)*=2V*).

Um eine Analyse auf Basis der beiden XML-SG-Instanzen durchzuführen, muss bekannt sein, welche Elemente von den beiden Instanzen einander zugeordnet werden können. Prinzipiell ist die Zuordnung der Elemente über das XML-SG-Schema gegeben. Eine Ausnahme sind die ungeordneten Strukturen. Diese Zuordnung wird umso komplexer, je unterschiedlicher die Steuergeräte-Instanzen spezifiziert sind. Eine korrekte Zuordnung bedeutet, dass genau die Interfaces zweier Steuergeräte einander zugeordnet werden sollen, welche die größte Ähnlichkeit aufweisen. Maximale Ähnlichkeit ist dann gegeben, wenn diese über die gleichen Elemente verfügen und der Wert/Inhalt der Elemente ebenfalls übereinstimmt.

Ein Zuordnungs-Mechanismus muss also Struktur- und String-Vergleiche durchführen können, um so Rückschlüsse auf die Ähnlichkeit zweier Komponenten zu erhalten. Anhand der so generierten Ähnlichkeitswerte kann dann eine Entscheidung getroffen werden, welche Komponenten einander zugeordnet werden müssen. Eine XML-Vergleichsmethode muss also Interface I_1 auf Interface I_2 zuordnen können, da diese die größte Ähnlichkeit besitzen.

Auf Basis der Zuordnung erfolgt dann eine Analyse bzgl. der Rückwärtskompatibilität der XML-SG-Instanzen.

6.1.2 Kompatibilitätsregeln

Bei einem Vergleich von Elementen können diese zwar unterschiedlich, aber aufgrund spezifischer Kompatibilitätsregeln dennoch rückwärtskompatibel sein. In Abbildung 6.2 ist zum Beispiel das Element *Strom* von Pin P_5 in den XML-SG-Instanzen unterschiedlich. Unter spezifischen Kompatibilitätskriterien kann *Pin* P_5 von SG_2 zu SG_1 dennoch als rückwärtskompatibel angesehen werden, wenn z.B. das Steuergerät SG_2 auch mit dem spezifizierten Strom von SG_1 korrekt funktioniert. In den folgenden Abschnitten werden verschiedene Kompatibilitätskriterien bzw. -szenarien vorgestellt.

Normierung

Elemente können kompatibel sein, wenn diese zwar nicht identisch, aber physikalisch äquivalent sind.
Beispiel: In Steuergerät SG_1 ist ein Signal S_3 mit der Spannung $U_1 = 0,613V$ und in SG_2 mit $U_1 = 613mV$ spezifiziert. Physikalisch gesehen, sind beide Signal identisch, basieren aber auf einer unterschiedlichen Normierung. Die Herausforderung für eine XML-Vergleichsmethode ist, dass die Normierungen mit in die Kompatibilitätsanalyse einbezogen werden.

Toleranzband

Elemente können zueinander rückwärtskompatibel sein, wenn sich die Elemente innerhalb bestimmter Toleranzbänder befinden.
Beispiel: Für ein CAN-Signal S_1 von SG_1 ist beispielsweise als Sende-Intervall $T_s = 10ms$ spezifiziert und für das gleiche Signal in SG_2 ist $T_s = 11ms$ spezifiziert. Angenommen, es ist bei dem Sende-Intervall ein Toleranzband von 10% definiert, dann kann das Signal von SG_2 als rückwärtskompatibel zu SG_1 betrachtet werden. Welche Toleranzbänder erlaubt sind, muss von Experten definiert werden.

Teilmenge

Bei der Spezifikation von Steuergeräten werden viele Elemente durch Intervalle beschrieben z.B. Spannung $U = [0, 12]V$. Bei der Kompatibilitätsanalyse müssen daher verschiedene Fälle betrachtet werden, z.B. ob ein spezifizierter Wert von Steuergerät SG_1 innerhalb des Intervalls von SG_2 liegt.
Im XML-SG-Schema sind verschiedene Typen von Intervallen definiert. In Listing 6.1 ist die XML-Struktur eines Intervalls dargestellt. Bei diesem Konstrukt besteht die Möglichkeit, einem Element entweder ein Intervall (*<xsd:element name="Intervall">*) oder einen festen Wert (*<xsd:element name="Fix" type="xsd:float"/>*) zuzuweisen. Das Intervall setzt sich aus drei Elementen zusammen:

- Min-Wert

- Max-Wert und
- Typischer Wert

Listing 6.1: XML-SG-Schema: Darstellung eines Intervalls

```
....
<xsd:choice>
   <xsd:element name="Intervall">
       <xsd:complexType>
           <xsd:sequence>
               <xsd:element name="Min" type="xsd:float"
                   minOccurs="0"/>
               <xsd:element name="Max" type="xsd:float"
                   minOccurs="0"/>
               <xsd:element name="Typisch" type="xsd:float"
                   minOccurs="0"/>
           </xsd:sequence>
       </xsd:complexType>
   </xsd:element>
   <xsd:element name="Fix" type="xsd:float"/>
</xsd:choice>
...
```

Dieses Konstrukt hat zur Folge, dass bei einem Vergleich von XML-SG-Instanzen Fix-Werte, Intervalle oder Fix-Werte mit Intervallen verglichen werden können. Hier sind mehrere kompatible Szenarien vorstellbar.

1. Beispiel: *Fix-Wert*$(SG_1) \subseteq Intervall(SG_2)$

In Steuergerät SG_1 ist die erlaubte Versorgungsspannung mit $U_v = 12V$ spezifiziert und in SG_2 mit $U_v = [10, 14]V$. Da SG_2 mit dem spezifizierten Wert von SG_1 betrieben werden kann, kann SG_2 als rückwärtskompatibel zu SG_1, in Bezug auf die Versorgungsspannung, betrachtet werden.

2. Beispiel: $Intervall(SG_1) \subseteq Intervall(SG_2)$

In Steuergerät SG_1 ist für Pin P_1 die Ausgangsspannung mit $U_{P_1} = [2..4]V$ und für SG_2 mit $U_{P_1} = [2..3]V$ definiert.

Bezogen auf Pin P_1 kann der Pin von SG_2 keine größere Ausgangsspannung liefern als SG_1 und würde daher auch die Anforderung von SG_1 erfüllen d.h., dass SG_2 rückwärtskompatibel zu SG_1 wäre. Allerdings könnte auch argumentiert werden, dass SG_2 die Anforderung von SG_1 nicht erfüllt und daher nicht rückwärtskompatibel wäre. Wann welche Elemente zueinander rückwärtskompatibel sind, ist nur mit Wissen von Experten entscheidbar. Daher muss bei einer XML-Vergleichsmethode die Möglichkeit bestehen, spezifisches „Expertenwissen“ für die Kompatibilitätsanalyse zu berücksichtigen.

Definition von Kompatibilitätsregeln

In den vorhergehenden Abschnitten sind verschiedene Kompatibilitätskriterien bzw. -szenarien vorgestellt worden. Für die einzelnen Szenarien werden spezifische Kompatibilitätsregeln benötigt, die besagen, wann ein Szenario bzw. die betroffenen Elemente als zueinander rückwärtskompatibel gelten. Ein Lösungsansatz wird hierfür in Kapitel 6.5 vorgestellt.

6.1.3 Ausblendung von Strukturinformationen

In Abbildung 6.3 sind zwei Steuergeräte (SG_1, SG_2) und die zugehörigen XML-SG-Instanzen skizziert. Es sei angenommen, dass beide Steuergeräte identisch sind, bis auf die Anzahl der Interfaces und der Zuordnung der Pins zu den Interfaces.
Es ist offensichtlich, dass die beiden Steuergeräte nicht kompatibel sind. Würde man jedoch das Interface und damit die Zuordnung der Pins zu den Interfaces ausblenden, wären beide Steuergeräte zueinander kompatibel. Technisch würde dies bedeuteten, dass die Rückwärtskompatibilität durch einen Steckeradapter hergestellt werden könnte.
Die Möglichkeit, spezifische Elemente bzw. Strukturinformationen definiert auszublenden, ist also für die Analyse von Rückwärtskompatibilität ein wichtiger Aspekt und muss daher in einer XML-Vergleichsmethode für Steuergeräte integriert sein.

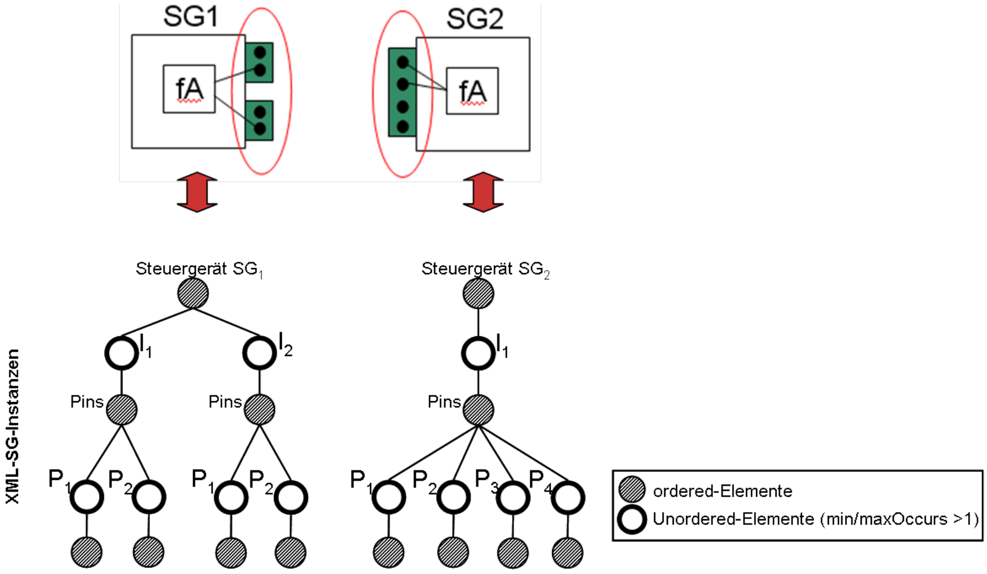


Abbildung 6.3: Vergleich zweier Steuergeräte mit unterschiedlichen Interfaces

6.1.4 Optionale oder nicht-relevante Informationen

Im XML-SG-Schema sind spezifische Elemente als $optional$ gekennzeichnet. Diese Elemente können, müssen aber nicht spezifiziert werden. Ein typisches Beispiel für ein $optionales$ Element ist ein Kommentarfeld. Für die Kompatibilitätsanalyse haben die als $optional$ gekennzeichneten Elemente keinen Einfluss und sollen daher zur Analyse ausgeblendet werden.

Des Weiteren können Elemente als $required$ definiert sein, aber dennoch keinen Beitrag, im Worst Case einen falschen Beitrag, zur Kompatibilitätsanalyse haben. Ein typisches Element ist das XML-Attribut ID. Dieses Element wird nur benötigt, um innere Abhängigkeiten zu beschreiben (vgl. Kapitel 2.2.3) und hat keinen direkten Einfluss auf die inhaltliche Aussage einer XML-SG-Instanz.

Dies bedeutet, dass eine XML-Vergleichsmethode benötigt wird, die optionale Informationen und spezifische nicht-relevante Informationen ausblenden kann.

6.1.5 Abhängigkeiten

In den XML-SG-Instanzen werden Elemente wie z.B. Signale oder Pins an unterschiedlichen Stellen im XML-Dokument spezifiziert. Die Kennzeichnung, dass z.B. $Signal_1$ an Pin_1 anliegt, erfolgt über Abhängigkeiten. Die Abhängigkeiten werden mit Hilfe der XML-Schema Attribute *ID* und *IDREF* hergestellt (vgl. Kapitel 2.2.3).

Diese Abhängigkeiten können Einfluss auf die Zuordnung der Elemente von XML-SG-Instanz SG_2 zur XML-SG-Instanz SG_1 haben und müssen daher mit berücksichtigt werden.

6.1.6 Zusammenfassung und Bewertung

In diesem Kapitel wurden Herausforderungen beschrieben, die eine XML-Vergleichsmethode zur Kompatibilitätsanalyse auf Basis von XML-SG-Instanzen „handeln“ können muss. Hierfür wurde im Rahmen von CompA die XML-Vergleichsmethode *XML-CompA* definiert.

In Tabelle 6.1 wurde analysiert, welche XML-Vergleichsmethode die zuvor aufgeführten Herausforderungen lösen kann. Es wird deutlich, dass *XML-CompA* speziell zur Analyse von Rückwärtskompatibilität entwickelt wurde und somit Lösungen für die spezifischen Herausforderungen anbietet.

6.2 Konzept XML-Vergleichsmethode (*XML-CompA*)

Im Rahmen des Forschungsthemas *CompA* bzw. dieser Arbeit wurde die spezifische XML-Vergleichsmethode „*XML-CompA*“ entwickelt, die die in Kapitel 6.1 beschrie-

		Vergleichsmethoden				
		Äquivalenz-vergleich	Baum Isomorphismen	Xandy	X-Diff Algorithmus	XML-CompA
Herausforderungen	Ordnungsstruktur -Zuordnung	×	×	✓	✓	✓
	Kompatibilitätsregeln	×	×	×	×	✓
	Ausblendung von Strukturinformation	×	×	×	×	✓
	Optionale nicht relevante Informationen	×	×	×	×	✓
	Abhängigkeiten	×	×	×	×	✓

Tabelle 6.1: Vergleich von XML-Vergleichsmethoden bzgl. Fähigkeiten zur Kompatibilitätsanalyse

benen Herausforderungen erfüllt. *XML-CompA* baut hierbei auf 4-Schritten auf (vgl. Abbildung 6.4):

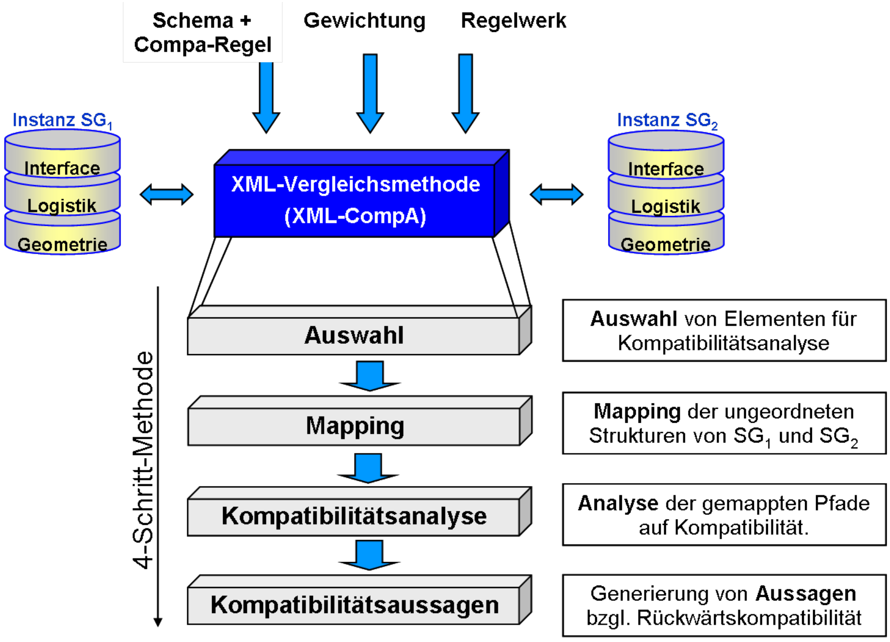


Abbildung 6.4: Prozess der XML-Vergleichsmethode (*XML-CompA*)

1. ***Auswahl:*** Festlegung, welche XML-Elemente bei einer Kompatibilitätsanalyse berücksichtigt bzw. nicht berücksichtigt werden sollen.
2. ***Mapping:*** Zuordnung der zu vergleichenden Elemente von *XML-SG-Instanz*(SG_1) zur *XML-SG-Instanz*(SG_2).
3. ***Kompatibilitätsanalyse:*** Analyse der gemappten Elemente bzgl. Rückwärtskompatibilität, unter Einbeziehung von Kompatibilitätsregeln.

4. ***Kompatibilitätsaussagen:*** Generierung von Aussagen bzgl. Rückwärtskompatibilität von Steuergerät SG_2 zu SG_1.

Für den Vergleich werden vier Eingangsgrößen benötigt bzw. hinzugezogen:

- *XML-SG-Instanzen:* Die XML-SG-Instanzen müssen vom gleichen XML-SG-Schema abgeleitet sein.
- *XML-SG-Schema + CompA-Regeln:* Im XML-SG-Schema werden die Kompatibilitätsregeln hinterlegt, also die Regeln, wann welche Elemente zueinander kompatibel sind. Ferner findet auf Basis des XML-SG-Schemas die Auswahl statt, welche Elemente bei der Kompatibilitätsanalyse ausgeblendet werden sollen.
- *Gewichtung:* Über die Gewichtungen wird angegeben, welche „Rolle“ einzelne Elemente bei der Kompatibilitätsanalyse „spielen“ sollen.
- *Regelwerk/Strukturen:* Hier sind spezifische Informationen für Kompatibilitätsregeln hinterlegt.

6.3 Auswahl

Im ersten Schritt des XML-Vergleichsalgorithmus können Elemente ausgewählt werden, die bei einer Kompatibilitätsanalyse nicht berücksichtigt werden sollen. Die Auswahl, welche Elemente bzw. Strukturinformationen ausgeblendet werden sollen, erfolgt auf Basis des XML-SG-Schemas.
In Abbildung 6.5 ist ein XML-SG-Schema dargestellt, bei dem das Element *Interface* ausgeblendet wird (schwarzer Balken). Dies hat zur Folge, dass auch die Interfaces in der XML-SG-Instanz ausgeblendet werden. Im Prinzip bedeutet dies, dass die Kind-Elemente *Pins* und *Signale* der *Interfaces* an dessen Vater-Element $Steuergeraet$ gehängt werden.

6.4 Mapping

Um Steuergeräte-Elemente bzgl. Kompatibilität analysieren zu können, müssen die entsprechenden XML-Elemente bestmöglich einander zugeordnet werden. Die Zuordnung der Elemente wird im Rahmen von CompA als „Mapping“ bezeichnet.
Prinzipiell ist die Position und damit die Zuordnung der XML-Elemente in der XML-SG-Instanz durch ein XML-SG-Schema fest vorgegeben. Eine Ausnahme sind hierbei die ungeordneten Strukturen (vgl. Abschnitt 6.1.1). Diese können bei der Instanziierung von Elementen mit dem Attribut ($min - /maxOccurs > 1$) auftreten.
Zur Erinnerung: Im Rahmen von CompA wurden folgende Begrifflichkeiten eingeführt:

- *ordered-Elemente:* Elemente, deren Position in einer XML-SG-Instanz durch das XML-SG-Schema fest vorgegeben ist.

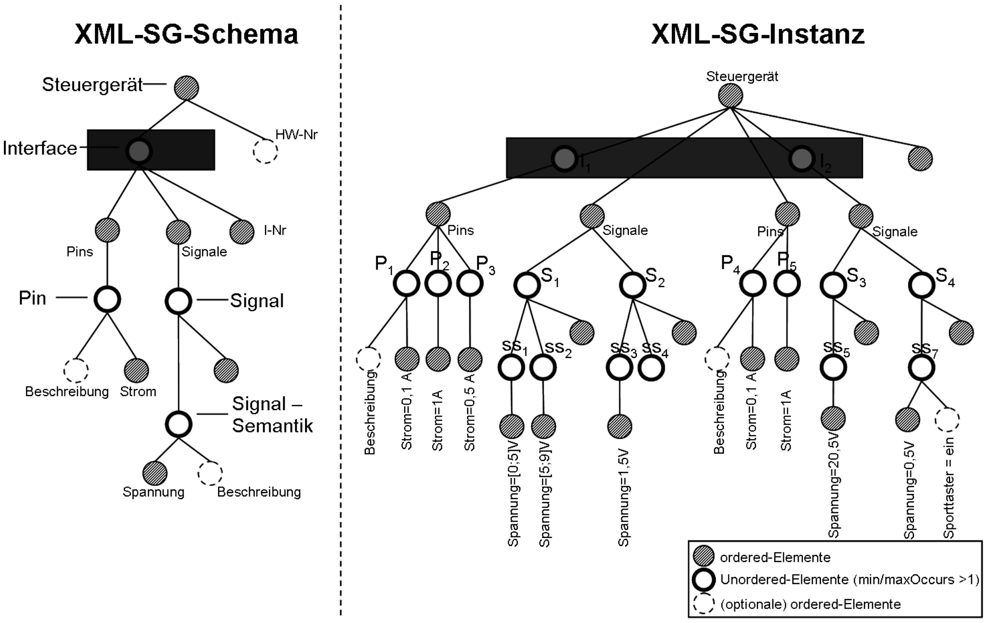


Abbildung 6.5: Auswahl von XML-Elementen, die bei der Kompatibilitätsanalyse nicht berücksichtigt werden sollen

- *unordered-Elemente:* Elemente, die aufgrund des Attributs $minOccurs > 1$ und/oder $maxOccurs > 1$ mehrfach in einer XML-SG-Instanz instanziiert werden dürfen und bei denen einen Links-Rechts-Vertauschung valide ist.

Eine explizite Zuordnung der Elemente von XML-SG-Instanzen ist also an der Stelle notwendig, an der eine ungeordnete Struktur auftreten kann, also bei unordered-Elementen. Um das Mapping der Elemente korrekt durchzuführen, müssen auch die Kind-Elemente bzw. der zugehörige Unterbaum beim Mapping mit betrachtet werden. Hierzu wird der Begriff *Cluster* eingeführt.

Definition 3 *Als Cluster wird künftig die Bündelung eines* unordered-*Element mit dessen Kind-Elementen bezeichnet.*

Somit kann das Mapping als eine Zuordnung von Clustern interpretiert werden. Eine Herausforderung hierbei ist, dass Unterbäume eines *unordered*-Elements wiederum *unordered*-Elemente besitzen können, also eine Verknüpfung von Clustern auftreten kann. Um die Cluster optimal einander zuzuordnen und die hierarchischen Strukturen zu berücksichtigen, wurde der Mapping-Algorithmus spezifisch erweitert. Der im Rahmen von CompA definierte Mapping-Algorithmus gliedert sich in 3 Schritte auf:

1. *Dekomposition & Clusterung:* Detektion der *unordered*-Elemente und Zusammenfassung der *unordered*-Elemente mit deren *ordered*-Elementen.
2. *Ähnlichkeits-Mapping:* Zuordnung der Cluster von *XML-SG-Instanz(*SG_1*)* zu *XML-SG-Instanz(*SG_2*)*.

3. *Erweitertes Mapping:* Berücksichtigung von inneren Abhängigkeiten (ID / IDREF) beim Mapping.

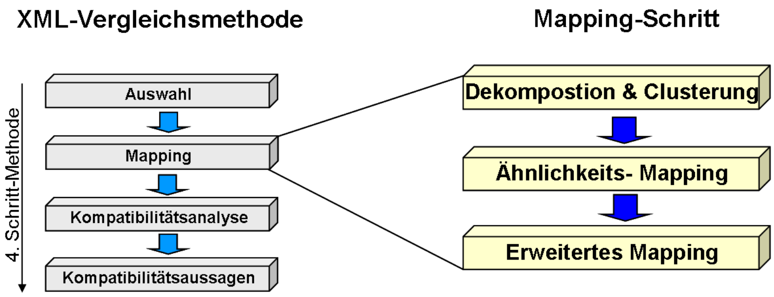


Abbildung 6.6: Mapping-Schritte

6.4.1 Dekomposition und Clusterung

Die Dekomposition dient zur Detektion derjenigen Elemente bzw. Unterbäume in der XML-SG-Instanz, bei denen ein Mapping notwendig ist. Dies sind die XML-Elemente, die in einer XML-SG-Instanz mehrfach instanziiert werden können und im XML-SG-Schema mit dem XML-Attribut $maxOccurs > 1$ gekennzeichnet sind, d.h. die *unordered*-Elemente.

Im nächsten Schritt erfolgt die Clusterung. Hier werden die *unordered*-Elemente mit den *ordered*-Elementen zusammengefasst. Cluster, die den gleichen Ursprung im XML-SG-Schema haben, werden dabei als „vom gleichen Typ“ bezeichnet. Da *unordered*-Elemente wiederum *unordered*-Kind-Elemente haben können, können die Cluster auf verschiedenen Ebenen auftreten.

Ein „Set“ an Cluster kann demnach durch die hierarchische Ebene und den Typ kategorisiert werden: Kategorie=[Ebene,Typ]. Für ein spezifisches $Cluster_x$ einer Kategorie schreiben wir: $Cluster_x = [Ebene, Typ][x]$.

In Abbildung 6.7 ist eine XML-SG-Instanz von SG_1 zu sehen. Das erste unordered-Element ist Interface I_1. Dieses Element wird mit den *ordered*-Elementen (z.B. Beschreibung) zu einem Cluster zusammengefasst und mit $[Ebene_1, Typ_1]$ kategorisiert. Formal geschrieben wäre $I_1 = [Ebene_1, Typ_1][1]$. Die Ebenen-Kennzeichnung wird hierarchisch, die Typ-Bezeichnung und Nummerierung wird sequentiell vergeben. Auf der zweiten Hierachie-Ebene kann Signal S_2 als $S_2 = [Ebene_2, Typ_2][2]$ beschrieben werden.

6.4.2 Ähnlichkeits-Mapping

Das Ergebnis der Dekomposition und der Clusterung sind Cluster, die nun aufeinander zu mappen sind. Durch die Kategorisierung der Cluster in [Ebene,Typ] kann die

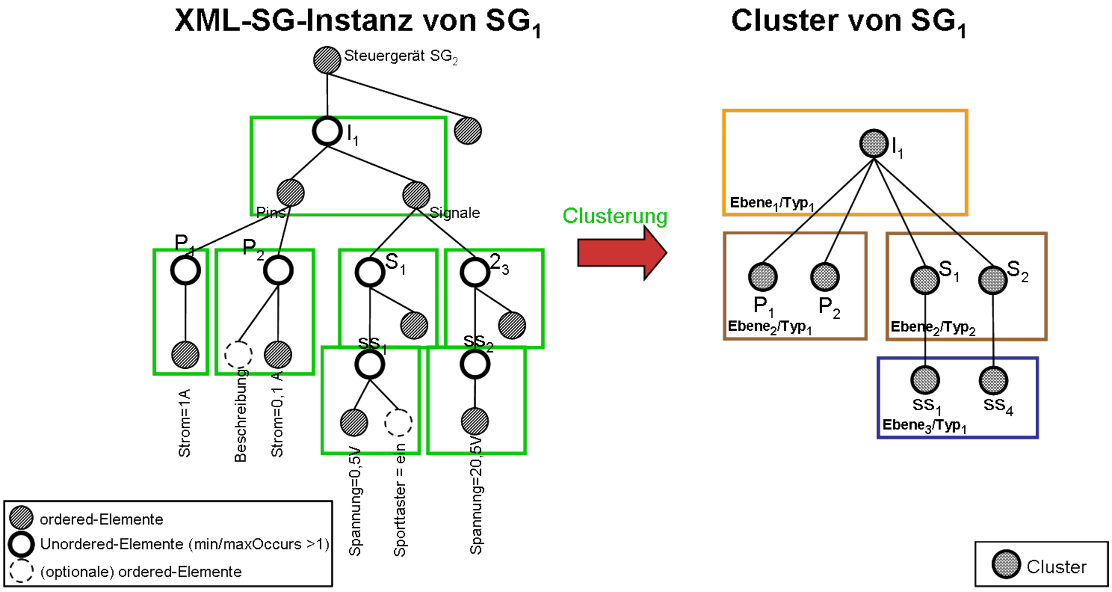


Abbildung 6.7: Clusterung

Zuordnung auf Cluster der jeweils gleichen Kategorie beschränkt werden (vgl. Abbildung 6.8). Das Ähnlichkeits-Mapping selbst erfolgt durch eine Äquivalenzprüfung der

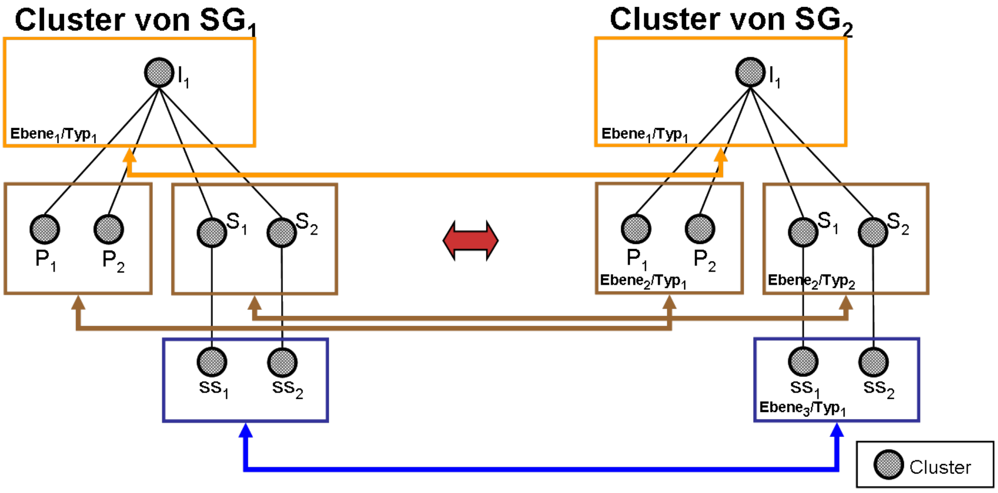


Abbildung 6.8: Vergleich von Clustern

Cluster-Elemente. Dies läuft wie folgt ab:
1. Schritt: Für jede Cluster-Kategorie [Ebene,Typ] wird eine Vergleichs-Matrix angelegt. Entlang der x-Achse werden die zugehörigen Cluster von SG_2 und entlang der y-Achse die Cluster von SG_1 angelegt. Im Anschluss werden alle Cluster einer Kategorie aus SG_1 und SG_2 miteinander verglichen.

Bei dem Vergleich eines Clusters x von SG_2 bspw. $Cluster_x = [Ebene, Typ][x]$ mit einem Cluster y von SG_2 z.B. $Cluster_y = [Ebene, Typ][y]$ wird die *Anzahl der äquivalenten XML-Elemente* bestimmt. Zusätzlich wird die maximal mögliche Anzahl der äquivalenten XML-Elemente anhand des XML-Schemas bestimmt. Die berechneten Zahlen werden als Mappingergebnis bezeichnet und in die Matrizen eingetragen.

Definition 4 *Ein Mappingergebnis setzt sich aus der* Anzahl der äquivalenten XML-Elemente *zweier Cluster ([Ebene,Typ][k] und [Ebene,Typ][l] mit $l, k \in \mathbb{N}$) und der maximal möglichen Anzahl äquivalenter XML-Elemente zusammen:*
Mappingergebnis$= \left(\frac{Anzahl\ aequivalenter\ Elemente}{Anzahl\ gesamter\ Elemente}\right)$
Der Mappinggrad beschreibt das Verhältnis:
Mappingergebnis[%]$= \left(\frac{Anzahl\ aequivalenter\ Elemente}{Anzahl\ gesamter\ Elemente}\right) \cdot 100$

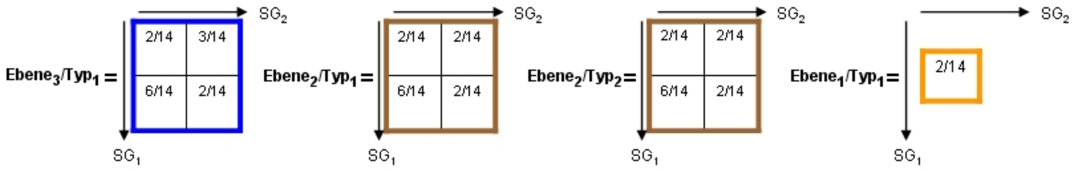


Abbildung 6.9: Ergebnismatrizen des Clustervergleichs

Für Abbildung 6.8 würden sich beispielsweise folgende Matrizen ergeben (vgl. Abbildung 6.9), unter der Annahme, dass alle Cluster jeweils 14 Elemente besitzen. In der Kategorie $[Ebene_3, Typ_1]$ besitzt SG_1 und SG_2 jeweils zwei Cluster. Für das Mapping zwischen $SS_1 = [Ebene_3, Typ_1][1]$ von SG_1 und $SS_2 = [Ebene_3, Typ_1][2]$ von SG_2 ergibt sich eine Überstimmung bei 3 von 14 Elementen. Die Berechnung für die übrigen Cluster erfolgt analog. Die Cluster mit dem höchsten Mappinggrad, also dem besten Verhältnis zwischen übereinstimmender und zu vergleichender Elemente werden einander zugeordnet. Das Ergebnis des ersten Schrittes ist also eine Aussage, welche Cluster am besten aufeinander mappen würden.

2. Schritt: Im 2. Schritt wird die hierarchische Zugehörigkeit der Cluster mit in die Berechnung integriert. Für die korrekte Zuordnung der Cluster ist dies relevant, da somit sichergestellt werden kann, dass z.B. die Pins entsprechend der Interface-Mappings zugeordnet werden.

Die Integration der hierarchischen Zugehörigkeit in das Mapping erfolgt Bottom-Up. Hierzu wird auf der untersten Ebene begonnen und das zugehörige Vater-Cluster jedes Clusters bestimmt (vgl. Abbildung 6.10). Hier z.B. $S_1(SG_1) = S_1 + SS_1$. Dieses Konstrukt wird nun wieder mit den gleichen Konstrukten aus SG_2, also $S_1(SG_2) = S_1 + SS_1\ und\ S_2(SG_2) = S_2 + SS_2$ verglichen. Das Ergebnis wird wieder in die korrespondierende Matrix des Vater-Clusters hinterlegt. Dieses Berechnungsvorgehen wird

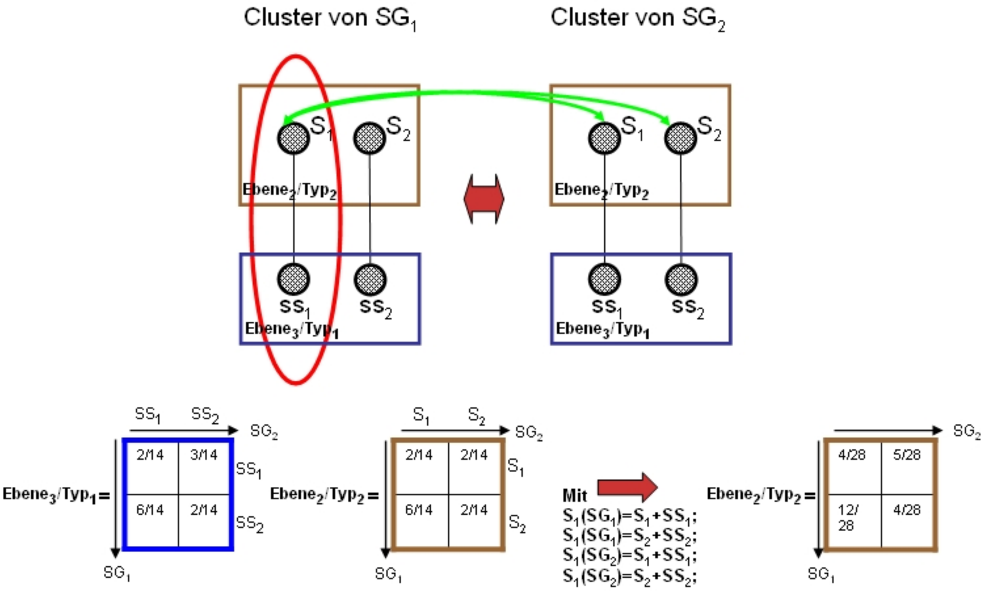


Abbildung 6.10: Bottom-Up Propagation der Mapping-Ergebnisse

nun so oft wiederholt, bis die oberste Hierarchie-Ebene mit berechnet worden ist.
Auf diese Weise erhält man auf der obersten Ebene ein Mapping-Ergebnis, das besagt, welche Cluster am besten aufeinander mappen. Durch die Bottom-Up Propagation der Mappingergebnisse und der Berechnungsvorschriften wird ermöglicht, dass auf der obersten Hierarchiestufe ein vollständiges Mappingergebnis vorliegt und eine Aussage getroffen werden kann, welche Cluster einander zugeordnet werden müssen.
Der Algorithmus für das Ähnlichkeits-Mapping ist in Algorithmus 3 nachzulesen.

6.4.3 Erweitertes Mapping

Beim Ähnlichkeits-Mapping wurde nun zwar die hierarchische Strukturierung berücksichtigt, aber die inneren Abhängigkeiten, also welches Signal an welchem Pin anliegt, wurde bis jetzt noch vernachlässigt. Dies wird durch das erweiterte Mapping gelöst.
Zur Umsetzung des erweiterten Mappings werden spezifische Annahmen bzw. Einschränkungen getroffen:

Annahmen/Einschränkungen

Propagation auf Cluster-Ebene Abhängigkeiten können zwischen allen Elementen auftreten, also bei *ordered*-Elementen und *unordered*-Elementen. Da ein Mapping, abstrakt gesehen, nur auf der Cluster-Ebene relevant ist, werden Abhängigkeiten von Elementen (*ordered* oder *unordered*) auf das zugehörige Cluster propagiert.

Algorithm 3 Aehnlichkeits-Mapping (Cluster(SG_1),Cluster(SG_2),*XML-SG-Schema*)

```
Input: Cluster($SG_1$), Cluster($SG_2$), XML-SG-Schema;
Output: $[x, y][i, l]$ // Mapping-Matrizen; $[2][x, y][i, l]$;
// Berechne Anzahl der Ebenen und Typen von $SG_1$
anz_Ebene= get_Anzahl_Ebenen($SG_1$);
anz_Typen= get_Anzahl_Typen($SG_1$);
// Alle Cluster von $SG_1$ werden $SG_2$ zugeordnet; Vergleiche Bottom-Up
for x=anz_Ebene to 1; anz_Ebene−− do
    for y=1 to anz_Typen; anz_Typen++ do
        // Berechne Anzahl der Cluster innerhalb einer Kategorie
        m=get_Anzahl_Cluster([x,y],$SG_2$); n=get_Anzahl_Cluster([x,y],$SG_1$);
        for i=1 to m; i++; do
            for l=1 to n, l++; do
                // Vergleich zweier Cluster; Ergebnis: a=Anzahl äquivalenter Elemente;
                u=Anzahl max. möglicher übereinstimmender Elemente; Speichern der
                Ergebnisse in Matrizen
                (a,u)=compare_Cluster(i,l,XML-SG-Schema);
                [x,y][i,l]=(a,u);
            end for
        end for
    end for
end for
// Einbeziehung hierarchischer Zugehörigkeiten
for x=(anz_Ebene - 1) to 1; anz_Ebene−−; do
    for y=0 to anz_Typen; anz_Typen++; do
        // Falls keine weiteren Kind-Elemente, dann springe zum nächsten Cluster
        if (child_Cluster([x,y][i])=$False$) then
            break;
        end if
        for i=1 to m; i++ do
            $kind_1$=get_child_Cluster_$SG_1$[x,y][i]
            for l=1 to n; i++ do
                $kind_2$=get_child_Cluster_$SG_2$[x,y][l]
                // Vergleiche Cluster unter Einbeziehung der Kind-Cluster z.B.
                $S_1(SG_1) = S_1 + SS_1$ gegen $S_1(SG_2) = S_1 + SS_1$
                (a,u)=compare_Cluster(i,l, $kind_1$, $kind_2$);
                // Speichern der Elemente in Mapping-Matrizen
                [x,y][i,l]=(a,u);
                // Speichern, der verglichenen Cluster z.B. $S_1, SS_1$ in Matrizen[2]
                [2][x,y][i,l]=add ($kind_2$;$kind_1$);
            end for
        end for
    end for
end for
// Zuordnung
Zuordnung der Cluster auf obersten Hierarchie-Ebene [1,y] nach dem höchsten
Mappinggrad;
Zuordnung der Kind-Cluster erfolgt über [2][x,y][i,l];
```

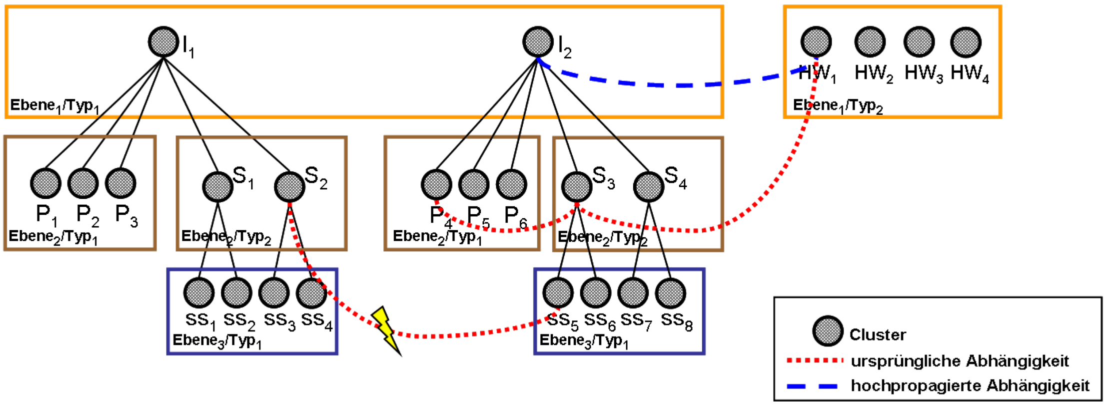


Abbildung 6.11: Erweitertes Mapping

Einschränkung Abhängigkeiten Prinzipiell können Abhängigkeiten zwischen allen Clustern definiert werden. Dies kann zu derartigen Verflechtungen ausarten, dass eine sinnvolle Zuordnung nicht mehr möglich ist. Daher werden im Rahmen von *CompA* nur Abhängigkeiten zwischen Clustern betrachtet, die sich im gleichen Vater-Cluster oder auf der höchsten Hierarchie-Ebene befinden. Eine Abhängigkeit zwischen ss_5 und S_2 wird daher nicht betrachtet (vgl. Abbildung 6.11), da sich die Abhängigkeit über mehrere Vater-Cluster (I_1, I_2) erstreckt.

Propagation auf höchste gemeinsame Hierarchie-Ebene Abhängigkeiten können zwischen Clustern verschiedener Hierarchie-Ebenen auftreten. Um die Abhängigkeiten sinnvoll in das Mapping einfließen zu lassen, sind die Abhängigkeiten auf eine gleiche Hierarchie-Ebene zu propagieren.
In Abbildung 6.11 sind die Cluster einer XML-SG-Instanz dargestellt. Zusätzlich ist darin die Abhängigkeit zwischen Pin P_4, dem Signal S_3 und einem Hardware-Element HW_1 eingetragen. Die Abhängigkeit besagt, dass das Signal S_3 am Pin P_4 anliegt und Pin P_4 ein Hardware-Element HW_1 besitzt. Um nun die Abhängigkeiten in den Vergleich mit einer anderen Instanz einfließen zu lassen, müssen diese „hochpropagiert“ werden. Die gestrichelte (blaue) Linie gekennzeichnet die hochpropagierte Abhängigkeit.

Festlegung von Prioritäten Für die Abhängigkeiten können Prioritäten festgelegt werden. Damit kann der Nutzer festlegen, welches Element den Master bei der Zuordnung spielt. In Abbildung 6.11 muss einmal das Interface I_1 zugeordnet werden und das Element HW_1. Da diese in Abhängigkeit stehen, muss nun entschieden werden, ob eine korrekte Zuordnung des Interfaces I_1 oder der Hardware HW_1 im Vordergrund steht. Wird die Priorität für das Interface gewählt, so werden zuerst die Interfaces bestmöglich zugeordnet und im Anschluss wird die Hardware bestmöglich, entsprechend der Abhängigkeiten, zugeordnet.

Konzept - Erweitertes Mapping

Das erweiterte Mapping läuft in mehreren Schritten ab:

1. **Festlegung Priorität**: Zuerst wird die Priorität der Abhängigkeiten festgelegt, d.h. welche Abhängigkeiten zuerst berücksichtigt werden.

2. **Hoch-Propagation der Abhängigkeiten**: Besteht eine Abhängigkeit zwischen zwei Hierarchie-Ebenen, so werden diese auf die höchste gemeinsame Hierarchie-Ebene propagiert. In Abbildung 6.12 ist beispielhaft eine Abhängigkeits-Tabelle dargestellt. P_2 und SS_2 befinden sich bspw. in Clustern, die auf unterschiedlichen Ebenen angesiedelt sind. Daher wird die Abhängigkeit auf SS_2 in ein Cluster propagiert, das auf der gleichen Ebene wie P_2 ist, und zwar auf S_2.

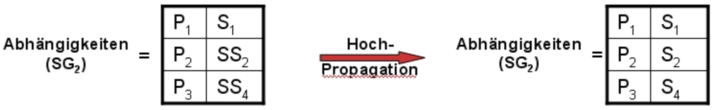


Abbildung 6.12: Propagation von Abhängigkeiten auf die oberste gemeinsame Ebene

3. **Berücksichtigung der Abhängigkeiten**: Abhängigkeiten fließen Top-Down in das Ähnlichkeits-Mapping ein. Das heißt, eine Abhängigkeit auf der obersten Ebene kann Einfluss auf das Mapping auf tieferen Ebenen haben, aber nicht umgekehrt.

6.4.4 Mapping-Ergebnis

Das Ergebnis des Mappings ist eine eindeutige Zuordnung aller Elemente der zu vergleichenden XML-SG-Instanzen. Jedes Element der XML-SG-Instanz von SG_1 wurde, soweit möglich, einem Element von XML-SG-Instanz von SG_2 zugeordnet. Konnten Elemente nicht gemappt werden, werden dies als „nicht mappbar“ gekennzeichnet. Auf Basis des Mappings ist bereits näherungsweise eine Aussage möglich, wie groß die Teilmenge von SG_1 zu SG_2 ist.
Bisweilen kann es erforderlich sein, dass ein Mapping von bestimmten Elementen erzwungen oder Konflikte aufgrund eines gleichen Mapping-Ergebnisses aufgelöst werden sollen. Diese Möglichkeit ist in *XML-CompA* zusätzlich mit integriert worden. Hierzu kann das Mapping-Ergebnis in der entsprechenden Mapping-Matrix geändert. Die Auswirkungen auf andere Mappings wird durch eine Wiederholung des Mappings automatisch berechnet.

6.5 Kompatibilitätsanalyse

Durch das Mapping ist definiert, welche Cluster der XML-SG-Instanzen von SG_1 und SG_2 aufeinander gemappt und somit verglichen werden müssen, um eine Aussage bzgl. Rückwärtskompatibilität der Steuergeräte zu generieren. Bei der Kompatibilitätsanalyse werden die zugeordneten Cluster bzw. die XML-Elemente der Cluster unter Einbeziehung von Kompatibilitätsregeln verglichen. In Abbildung 6.13 ist dies konzeptionell dargestellt. Hierbei können zwei Kategorien unterschieden werden:

- attributbasierter Vergleich und
- strukturbasierter Vergleich

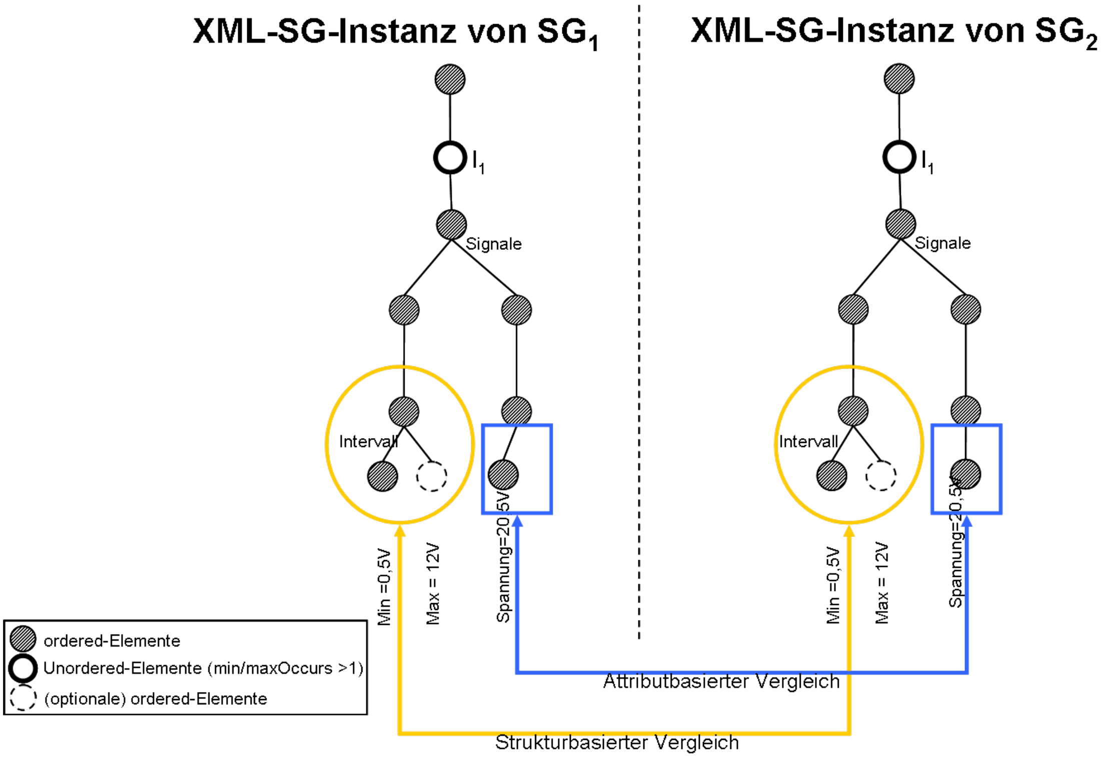


Abbildung 6.13: Kompatibilitätsanalyse

Zunächst werden die beiden Vergleichsmethoden vorgestellt und anschließend werden die Kompatibilitätsregeln eingeführt, die bei einer Kompatibilitätsanalyse herangezogen werden können.

6.5.1 Attributbasierter Vergleich

Beim attributbasierten Vergleich können die zugeordneten XML-Elemente direkt über Kompatibilitätsregeln miteinander verglichen werden, d.h. es ist immer nur ein XML-

Element aus einer XML-SG-Instanz beim Vergleich involviert. So können beispielsweise XML-Elemente vom Typ Integer-Werte gegen Elemente von Typ-Integer verglichen werden.

6.5.2 Strukturbasierter Vergleich

Der strukturbasierte Vergleich stellt eine Erweiterung des Attribut-Vergleichs dar. Jedoch sind bei diesem Vergleich mehrere Elemente mit involviert. Ein Beispiel ist der Vergleich von Werten, bei denen die Einheit bspw. [milli Volt] oder [Volt] mit betrachtet werden muss. Ein anderes typisches Beispiel ist das Intervall. Beim Vergleich von Intervallen muss bekannt sein, wo die Min-Werte, Max-Werte etc. angegeben sind.
Diese Strukturen müssen der XML-Vergleichsmethode bekannt gegeben werden. Hierzu wurde in *XML-CompA* die Möglichkeit geschaffen, definierte Strukturen zu hinterlegen.
Die Möglichkeit zur Beschreibung der Intervallstruktur ist notwendig, weil nicht alle Intervalle innerhalb eines XML-SG-Schemas gleich aufgebaut sind und die Intervallgrenzen an verschiedenen Positionen innerhalb der XML-SG-Instanzen stehen können.
Die Definition einer Intervallstruktur *IntervallX* ist beispielhaft in Listing 6.2 angegeben:

Listing 6.2: Beispiel für Strukturen

```
<intervallX>
<element1>a</element1>
<element2>b</element2>
<min>c</min>
<max>d</max>
.
.
.
</intervallX>
```

In dieser Struktur steht beispielsweise der $< min >$-Wert *c* an der 3. Position.

6.5.3 Kompatibilitätsregeln

Wann welche Elemente zueinander kompatibel sind, wird über Kompatibilitätsregeln festgelegt. Diese Regeln legen fest, wann zwei miteinander zu vergleichende Elemente als kompatibel gelten. Die Kompatibilitätsregeln werden als XML Attribut *CompAregel* im XML-SG-Schema hinterlegt und somit auch in den entsprechenden XML-SG-Instanzen instanziiert. Die Kompatibilitätsregeln in Form des XML Attributs bilden einen Mechanismus, der gleichermaßen beim strukturbasierten als auch beim attributbasierten Vergleich von Bedeutung ist.
Welche Kompatibilitätsregel bei welchem Element zur Anwendung kommt, muss von einem Experten entschieden werden. Verfügt ein Element über das Attribut *CompAregel*, so gilt der Wert des Attributs für dieses Element und alle nachfolgenden Elemente,

solange kein weiteres Element mit diesem Attribut auftritt. Der Wert des Attributs korrespondiert mit einer spezifischen Kompatibilitätsregel. Im Rahmen von CompA wurden 12 Kompatibilitätsregeln definiert, die sich aus der Praxiserfahrung heraus gezeigt haben. Diese sind jederzeit erweiterbar:

none Das betroffene Element soll mit der Default-Einstellung behandelt werden.

tabelle Das Element verfügt über Tabellen-Informationen mit kompatiblen Elementen.

ignorieren Das Element wird bei der Kompatibilitätsanalyse ignoriert.

I_SG1_innerhalb_SG2 Das nachfolgende Intervall des Elements ist kompatibel, wenn das Intervall von Steuergerät SG_1 innerhalb des Intervalls von Steuergerät SG_2 liegt: $(I(SG_1) \subseteq (I(SG_2))$.

I_SG2_innerhalb_SG1 Das nachfolgende Intervall des Elements ist kompatibel, wenn das Intervall von Steuergerät SG_2 innerhalb des Intervalls von Steuergerät SG_1 liegt: $(I(SG_2) \subseteq (I(SG_1))$.

I_maxSG2_kleinergleich_maxSG1 Das nachfolgende Intervall des Elements ist kompatibel, wenn der Maximalwert des Intervalls von Steuergerät SG_2 kleiner gleich dem Maximalwert des Intervalls von Steuergerät SG_1 ist.

I_minSG2_groessergleich_minSG1 Das nachfolgende Intervall des Elements ist kompatibel, wenn der Minimalwert des Intervalls von Steuergerät SG_2 größer gleich dem Minimalwert des Intervalls von Steuergerät SG_1 ist.

I_untereobereGrenze_plusminusX Das nachfolgende Intervall des Elements ist kompatibel, wenn die Minimal- und Maximalwerte der Intervalle beider Steuergeräte um maximal X% auseinander liegen.

F_SG2_greater_SG1 Der Fix-Wert, der zum Element gehört, ist kompatibel, wenn der Wert von Steuergerät SG_2 größer (gleich) dem Wert von Steuergerät SG_1 ist.

F_SG2_groessergleich_SG1_plusX Der Fix-Wert, der zum Element gehört, ist kompatibel, wenn der Wert von Steuergerät SG_2 größer gleich dem Wert von Steuergerät SG_1 ist, wobei der Wert von Steuergerät SG_2 nicht größer als X% sein darf.

F_SG2_kleinergleich_SG1_minusX Der Fix-Wert, der zum Element gehört, ist kompatibel, wenn der Wert von Steuergerät zwei kleiner gleich dem Wert von Steuergerät SG_1 ist, wobei der Wert von Steuergerät SG_2 nicht kleiner als X% sein darf.

F_SG2_innerhalb_SG1_plusminusX Der Fix-Wert, der zum Element gehört, ist kompatibel, wenn der Wert von Steuergerät SG_2 und der Wert von Steuergerät SG_1 nicht weiter auseinander liegen als X.

Regelwerk

Zusätzlich zu den Kompatibilitätsregeln, die besagen, ob XML-Elemente/-Strukturen aufgrund ihrer Werte kompatibel sind, wird ein Regelwerk eingeführt. Im Regelwerk wird für den strukturbasierten Vergleich hinterlegt, welche Strukturen wie gegeneinander zu vergleichen sind. Das Regelwerk basiert auf einem generischen Ansatz und ist für verschiedene Anwendungsgebiete übertragbar. Es enthält derzeit folgende Tags, die Regeln für Instanz-Elemente festlegen:

- $< doc1 >< /doc1 >$: Regel gilt für Instanz
- $< doc2 >< /doc2 >$: Regel gilt für Instanz 2
- $< case >< /case >$: Die Kind-Elemente von $< case >$ können mit deren Kind-Elementen über eine bestimmte Regel/Struktur kompatibel zueinander sein
- $< only >< /only >$: Das zugehörige Element ist nur kompatibel, wenn alle Kind-Elemente von $< only >$ kompatibel sind.
- $< count >< /count >$: Das zugehörige Element ist nur kompatibel, wenn die Anzahl der unter $< count >$ angegebenen Elemente in beiden Instanzen gleich ist.
- $< Elementname >< /Elementname >$: Ein Element einer Instanz, für das eine Regel gilt.
- $< /Elementname >$: Ein Element, das für eine Regel benötigt wird.

Durch Kombination der eben definierten Tags können geeignete Regeln definiert werden:

Listing 6.3: Beispiel für Regelwerk

```
<doc1>
    <intervallKodiert>
        <count>
            </intervall>
        </count>
        <only>
            </intervall>
        </only>
    </intervallKodiert>
</doc1>
```

Die in Listing 6.3 aufgeführte Regel besagt, dass das Element „intervallKodiert“ aus Instanz 1 nur kompatibel zum Element „intervallKodiert“ aus einer zweiten Instanz sein kann, wenn zum einen die Anzahl der Kind-Elemente „intervall“ übereinstimmt und wenn auch alle Elemente „intervall“ kompatibel zueinander sind.

6.6 Kompatibilitätsaussagen

Die Kompatibilitätsaussagen sind eine Auswertung der Ergebnisse der Kompatibilitätsanalyse und geben an, ob ein Steuergerät SG_2 rückwärtskompatibel zu SG_1 ist ($SG_2 \Leftarrow_c SG_2$). Zwei Steuergeräte SG_1 und SG_2 sind nach Definition 1 genau dann rückwärtskompatibel, wenn in *XML-SG-Instanz(SG_2)* alle Elemente von *XML-SG-Instanz(SG_1)* äquivalent bzw. rückwärtskompatibel spezifiziert wurden. Man kann also sagen, dass *XML-SG-Instanz(SG_1)* eine echte Teilmenge von *XML-SG-Instanz(SG_2)* (*XML-SG-Instanz(SG_1)$\subseteq$XML-SG-Instanz(SG_2)*) ist, unter Berücksichtigung der Kompatibilitätsregeln.
Aufbauend darauf setzen sich die Kompatibilitätsaussagen wie folgt zusammen:

- **Mapping-Aussage (MA)[%]**:
  Angabe, wie viel Cluster von SG_1 dem Steuergerät SG_2 zugeordnet werden konnten.
  MA[%]= $\left(\frac{Anzahl\ gemappter\ Cluster}{Anzahl\ aller\ Cluster\ aus\ SG_1}\right) \cdot 100$
  Konnten alle Cluster von SG_1 dem Steuergerät SG_2 zugeordnet werden, ist der Wert 100%. Nur wenn 100% zugeordnet werden konnten, kann die Teilmengenforderung *XML-SG-Instanz(SG_1)$\subseteq$XML-SG-Instanz(SG_2)* erfüllt werden.
  Die Mapping-Aussage werde künftig mit **MA** abgekürzt.

- **Rückwärtskompatibilität der gemappten XML-Elemente (RKA)[%]**:
  Durch das Mapping der Cluster können alle zugehörigen XML-Elemente ebenfalls sofort zugeordnet werden. Die Aussage der Rückwärtskompatibilität basiert auf den XML-Elementen. Sie besagt, zu wie viel Prozent die zugeordneten XML-Elemente der XML-SG-Instanzen rückwärtskompatibel sind. Die Aussage berechnet sich wie folgt:
  RKA[%]= $\left(\frac{Summe\ der\ kompatiblen\ XML-Elemente}{Anzahl\ aller\ gemappten\ XML-Elemente}\right) \cdot 100$

- **einfacher *Äquivalenzvergleich*[%]**:
  Angabe, zu wie viel Prozent SG_1 und SG_2 äquivalent sind. Der *Äquivalenzvergleich* verfügt über kein spezielles Expertenwissen oder andere intelligente Mechanismen. Er dient zum Vergleich der Effizienz des Konzepts mit einem reinen *Äquivalenzvergleich.*

Ein Steuergerät SG_2 ist zu Steuergerät SG_1 also genau dann rückwärtskompatibel, wenn alle Cluster zugeordnet werden konnten (Mapping-Aussage = 100%) und die Aussage bzgl. Rückwärtskompatibilität ebenfalls 100% ergibt.

6.7 Zusammenfassung

In diesem Kapitel wurde die XML-Vergleichsmethode *XML-CompA* ausführlich beschrieben.

Hierzu wurden zuerst die Herausforderungen aufgeführt, die an eine XML-Vergleichsmethode zur Generierung von Kompatibilitätsaussagen gestellt werden. Im Anschluss daran wurde das Konzept von *XML-CompA* erläutert. Der Ansatz baut auf einer 4-Schritt Methode auf. Die einzelnen Schritte wurden im Konzept detailliert beschrieben und werden im Folgenden kurz zusammengefasst.

Im ersten Schritt (Auswahl) besteht die Möglichkeit, XML-Elemente, die bei der Kompatibilitätsanalyse nicht berücksichtigt werden sollen, auszublenden.

Für den zweiten Schritt (Mapping) wurde ein Konzept vorgestellt, das die zu vergleichenden Elemente der beiden XML-SG-Instanzen aufgrund ihrer ähnlichen Bedeutung einander zuordnet und dabei auch innere Abhängigkeiten berücksichtigt. Das Mapping wurde hierfür in ein Ähnlichkeits-Mapping und ein Erweitertes-Mapping gegliedert. Zur Durchführung des Mappings wurde des Weiteren ein Cluster-Konzept eingeführt, das in diesem Kapitel ausführlich erläutert wurde.

Im dritten Schritt (Kompatibilitätsanalyse) wurden mehrere Kompatibilitätsregeln eingeführt. Die Kompatibilitätsregeln werden herangezogen, wenn die durch das Mapping zugeordneten Elemente bzgl. Rückwärtskompatibilität verglichen werden. Insgesamt wurden 12 Kompatibilitätsregeln definiert. Des Weiteren wurde in diesem Schritt das Vorgehen des attributbasierten und strukturbasierten Vergleichs beschrieben. Der strukturbasierte Vergleich wird herangezogen, wenn zur Analyse mehr als ein XML-Element herangezogen wird (z.B. Intervall und Fix-Wert Vergleich).

Im 4. Schritt (Kompatibilitätsaussagen) wurde erläutert, welche Aussagen aus der Kompatibilitätsanalyse generiert werden können und wie diese zu interpretieren sind.

Die Validierung der XML-Vergleichsmethode, die in Kapitel 9 vorgestellt wird, bestätigt die korrekte Funktion und Effizienz der hier entwickelten XML-Vergleichsmethode *XML-CompA*.

7 MSC-Vergleichsmethode zur Analyse von Rückwärtskompatibilität (*MSC-CompA*)

Dynamisches Verhalten von Steuergeräten kann mit unterschiedlichen Spezifikationstechniken wie ASCET, Message Sequence Charts, Matlab Simulink, etc. beschrieben werden (vgl. Kapitel 4.1). Das Projekt CompA fokussiert die Spezifikation von dynamischem Verhalten mittels Message Sequence Charts (MSC) (vgl. Kapitel 5). Um eine Aussage zu treffen, ob ein, von verschiedenen MSCs, spezifiziertes Verhalten zueinander rückwärtskompatibel ist, müssen die MSCs miteinander verglichen werden [53].
In Kapitel 7.1 werden die Schwierigkeiten bzw. Herausforderung eines MSC-Vergleichs dargestellt. Anschließend wird in Kapitel 7.2 definiert, wann MSCs als zueinander rückwärtskompatibel betrachtet werden können. In Kapitel 7.3 wird schließlich ein kurzer Überblick über die entwickelte MSC-Vergleichsmethode gegeben, die in Kapitel 7.4 detailliert ausgeführt wird. Die beschriebenen Ansätze werden abschließend anhand eines Beispiels, in Kapitel 7.5, veranschaulicht.

7.1 Motivation - Herausforderungen

Aufgrund der Mächtigkeit, der in [80][79][78] definierten MSC-Syntax, kann äquivalentes (dynamisches) Verhalten mit MSCs „strukturell unterschiedlich“ spezifiziert werden[1]. Dies wird im folgenden Beispiel genauer erläutert. In Abbildung 7.1

sind zwei MSCs (MSC_1, MSC_2) dargestellt, die die identische Instanz p_1 und die identischen Ereignisse E besitzen. E besteht aus den Sende-Ereignissen $T = y_1, y_2, y_3$ und Empfangs-Ereignissen $R = x_1, x_2, x_3$ (vgl. Kapitel 2.3.4). MSC_1 ist aus einem *Alternative-Operator* mit drei *Sections* aufgebaut. MSC_2 ist aus einem *Optional-Operator* aufgebaut, der wiederum einen *Alternative-Operator* mit zwei *Sections* enthält. MSC_1 und MSC_2 unterscheiden sich also in ihrem strukturellen Aufbau. Ein Vergleich der beiden graphischen Darstellungen führt zu keiner Aussage bzgl. deren Rückwärtskompatibilität.

---

[1] Der Aufbau und die formalen Definitionen von MSCs wurden in Kapitel 2.3 eingeführt.

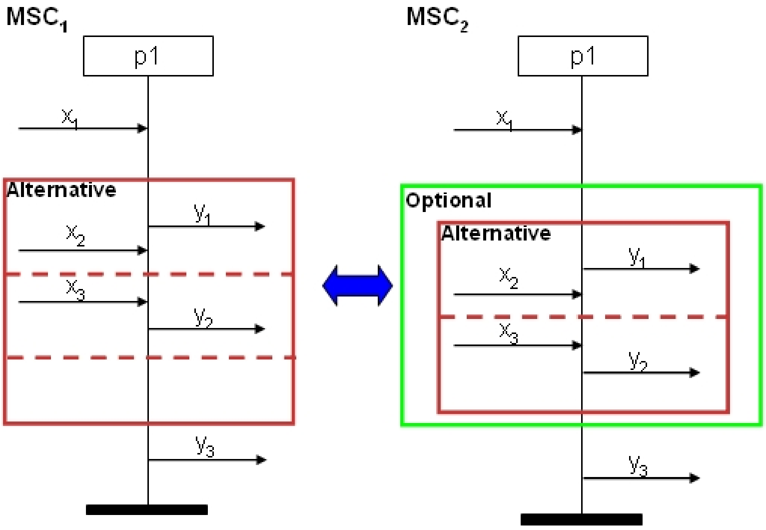


Abbildung 7.1: MSC_1 und MSC_2 mit unterschiedlicher struktureller Darstellung, aber mit äquivalentem Verhalten

Eine andere Vergleichsmöglichkeit ist der Vergleich der, durch die MSCs spezifizierten, Ereignissequenzen. Ereignissequenzen sind alle möglichen Sequenzen, in der die Sende-/Empfangsereignisse gesendet bzw. empfangen werden können. MSC_1 und MSC_2 besitzen jeweils drei mögliche Ereignissequenzen (vgl. Abbildung 7.2). MSC_1 hat einen *Alternative-Operator* mit drei $Sections$ und somit drei mögliche Ereignissequenzen. MSC_2 besitzt einen *Optional-Operator*, mit einem inneliegenden *Alternative-Operator*. Die 1. u. 2. mögliche Ereignissequenz tritt ein, wenn der *Optional-Operator* = *True* ist und so die beiden alternativen $Sections$ des *Alternative-Operator* durchlaufen werden können. Die 3. Ereignissequenz tritt ein, wenn der *Optional-Operator* = *False* ist, also nicht durchlaufen wird. Vergleicht man nun die möglichen Ereignissequenzen der beiden MSCs, so zeigt sich, dass diese identisch sind. Da MSC_1 und MSC_2 auch die identischen Ereignisse E_p und die Instanz p_1 beschreiben, spezifizieren beide MSC ein äquivalentes (dynamisches) Verhalten. Dieses Beispiel zeigt anschaulich die Problematik, dass MSCs trotz unterschiedlicher Darstellung ein äquivalentes bzw. kompatibles Verhalten spezifizieren können. Um dies zu untersuchen und Aussagen bzgl. der Rückwärtskompatibilität von MSC_2 zu MSC_1 zu treffen, gibt es derzeit noch keine Verfahren. Im Rahmen von CompA wurde eine MSC-Vergleichsmethode definiert, die genau diese Problematik löst. Bevor die Methode hierfür in Kapitel 7.3 genauer vorgestellt wird, wird im nächsten Kapitel 7.2 die Definition bzgl. der Rückwärtskompatibilität von MSCs betrachtet.

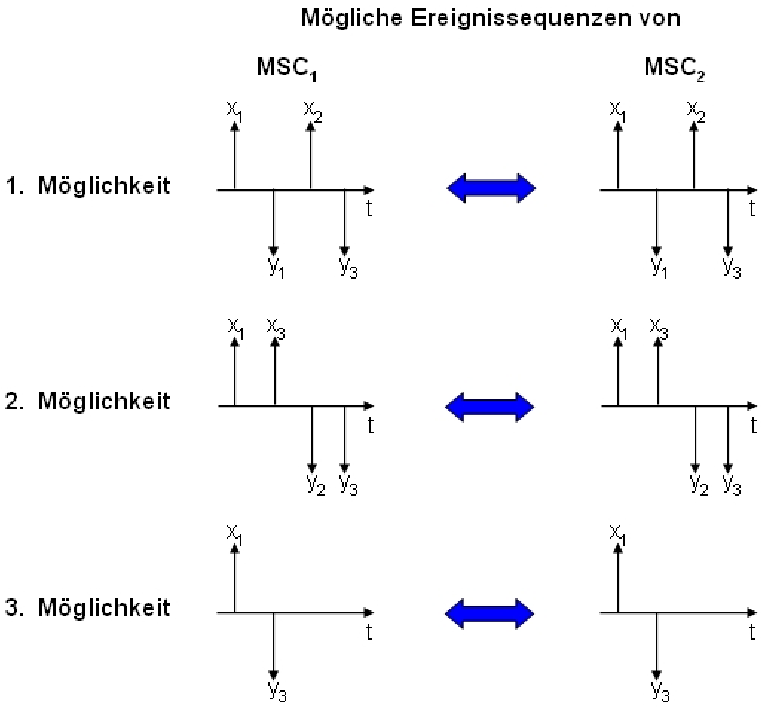


Abbildung 7.2: Ereignissequenzen von MSC_1 und MSC_2 aus Abbildung 7.1

7.2 Erweiterung der Rückwärtskompatibilitäts-Definition für Message Sequence Charts (MSC)

Rückblick - Grundlagen

In Kapitel 3.1 wurde in Definition 1 die Rückwärtskompatibilität von Automotive-Steuergeräten definiert. Der 3. und 4. Punkt dieser Definition fokussierten die Rückwärtskompatibilität von Steuergeräten bzgl. ihres dynamischen Verhaltens. Zur Erinnerung:

3. Die Menge M_1 der Funktionen f_1 von SG_1 stellt eine Submenge der Menge M_2 von SG_2 dar: $M_1 \subseteq M_2$, wobei die Schnittstellen identisch bedient werden.
4. SG_2 und SG_1 weisen das gleiche dynamische funktionale Verhalten auf, bezogen auf M_1.

M_1 und M_2 umfassen per definitionem alle spezifizierten Funktionen der Steuergeräte, auch die, die beispielsweise mit ASCET spezifiziert worden sind. In Rahmen dieser Arbeit beschränken wir uns aber auf die Spezifikation von dynamischen Verhaltens durch MSCs (vgl. Kapitel 1.3). Formal gesehen betrachteten wir also die Menge der Funktionen (M_1', M_2'), die mit MSCs spezifiziert worden sind. M_1' bzw. M_2' stellen hierbei eine

echte Teilmenge von M_1 und M_2 dar.
Im Rahmen von CompA wurde für Rückwärtskompatibilität bzgl. dynamischen Verhaltens folgende Definition getroffen:

Definition 5 *Ein spezifiziertes Verhalten von Steuergerät SG_2 $V_2(SG_2)$ gilt als rückwärtskompatibel zum Verhalten von Steuergerät SG_1 $V_1(SG_1)$, wenn das Verhalten $V_2(SG_2)$ das Verhalten von $V_1(SG_1)$ zulässt. Das heißt, dass $V_1(SG_1)$ eine echte Teilmenge von $V_2(SG_2)$ ist: $V_1(SG_1) \subseteq V_2(SG_2)$.*

Übertragen auf die Rückwärtskompatibilitäts-Definition von MSCs, wird im Rahmen von CompA, Rückwärtskompatibilität von MSCs wie folgt definiert:

Definition 6 *Ein MSC_2 ist zu einem MSC_1 genau dann rückwärtskompatibel, wenn die Menge der Ereignissequenzen S_1 von MSC_1 eine echte Teilmenge der Menge der Ereignissequenzen S_2 von MSC_2 ist und deren Ereignisse* E *und Instanzen* P *zueinander rückwärtskompatibel sind.*
$(MSC_1 \Leftarrow_c MSC_2) \Leftrightarrow (S_1 \subseteq S_2 \wedge E_1 \Leftarrow_c E_2 \wedge P_1 \Leftarrow_c P_2)$ *mit*
S_1, S_2*: Menge der Ereignissequenzen von* MSC_1 *und* MSC_2
E_1, E_2*: Menge der Ereignisse von* MSC_1 *und* MSC_2
P_1, P_2*: Menge der Instanzen von* MSC_1 *und* MSC_2
$\Leftarrow_c$*: rückwärtskompatibel*

Definition 6 wird im Folgenden an einem Beispiel veranschaulicht. In Abbildung 7.3 sind zwei MSCs, bestehend aus der identischen Instanz p_1 und den Ereignissen x_1, y_1,

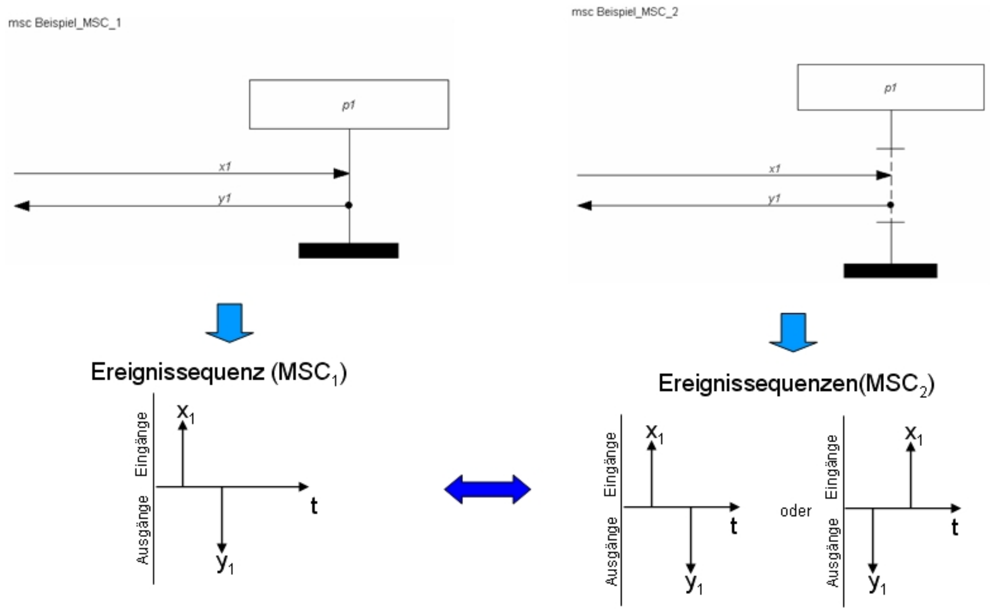


Abbildung 7.3: Ereignissequenzen von zwei MSCs

dargestellt. In MSC_1 wird an der Instanz p_1 zuerst das Ereignis x_1 und y_1 empfangen/gesendet. In MSC_2 dagegen werden dieselben Ereignisse x_1, y_1 innerhalb einer Co-Region empfangen bzw. gesendet. Eine *Coregion* hat zur Folge, dass die sequentielle Reihenfolge $<_p$, in der die Ereignisse auftreten können, beliebig ist. Betrachtet man nun die möglichen Ereignissequenzen S_1 von MSC_1 und S_2 von MSC_2, so erkennt man, dass MSC_1 eine und MSC_2 zwei mögliche Ereignissequenzen besitzt. Die Menge der Ereignissequenzen S_1 stellt dabei ein echte Teilmenge von S_2 dar. In der Praxis bedeutet dies, dass MSC_2 mit dem in MSC_1 spezifizierten Verhalten „umgehen“ kann und MSC_2 daher rückwärtskompatibel zu MSC_1 ist.
Ein MSC besteht in der Regel aus mehreren Instanzen. Dies kann wie folgt definiert werden:

Definition 7 *Ein MSC ist die Aggregation aus $n \in \mathbb{N}$ Einzel-Instanzen (p_i) mit einer eindeutigen Zuordnung der Ereignisse zu den Instanzen:*
$MSC = \sum_{i=0}^{n} p_i; \; mit \; o : E \rightarrow P$

Aus diesem Grund ist eine Dekomposition der MSCs in einzelne Instanzen möglich. Dies ist für die MSC-Vergleichsmethode notwendig, wie der nächste Abschnitt zeigen wird.

7.3 Konzept der MSC-Vergleichsmethode (*MSC-CompA*)

Um Aussagen bzgl. Rückwärtskompatibilität von MSCs treffen zu können, müssen alle MSCs von SG_1 und SG_2 miteinander verglichen werden. Zur Erläuterung der Methode konzentrieren wir uns auf den Vergleich von zwei MSC-Szenarien. Der Vergleich weiterer MSC-Szenarien erfolgt analog.
In Abbildung 7.1 wurde die Problematik der unterschiedlichen strukturellen Darstellung von äquivalentem dynamischen Verhalten bereits dargestellt. Um MSCs vergleichen zu können, müssten diese in eine normierte Darstellung überführt werden. Im Rahmen von CompA wird daher eine Transformation der MSCs in Automaten vorgeschlagen. Eine Transformation von MSCs in Automaten ist aufgrund der Beobachtung möglich, dass jede einzelne Instanz eines MSCs analog zu einem Automaten ein Ein-/Ausgangsverhalten beschreibt. Hierfür müssen die Instanzen der MSCs dekomponiert und anschließend in Automaten umgewandelt werden.
In Abbildung 7.4 sind zwei MSCs dargestellt, die zu vergleichen sind. Nach der Dekomposition der Instanzen wird jede Instanz in einen zugehörigen endlichen Automaten transformiert. Diese können bei Bedarf mit Methoden aus der Automatentheorie (ϵ-Eliminierung, Determinierung, Minimierung) reduziert werden [74]. Durch einen Vergleich der erzeugten Automaten von MSC_1 und MSC_2 kann dann eine Aussage bzgl. deren Rückwärtskompatibilität von MSC_2 zu MSC_1 getroffen werden.

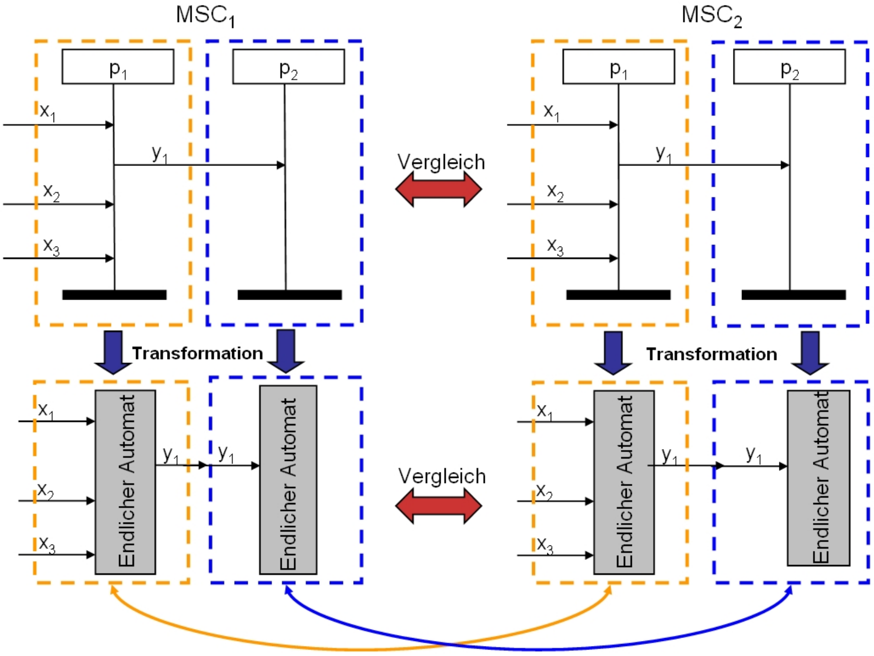


Abbildung 7.4: Vergleich von MSCs erfolgt auf Basis von Automaten

7.4 MSC-Vergleichsmethode

Die im CompA-Ansatz vorgeschlagene Methode gliedert sich konkret in folgende Teilschritte (vgl. Abbildung 7.5).

1. *Graphische Darstellung*: Message Sequence Charts werden mittels eines MSC-Editors graphisch erstellt und als ASCII-File exportiert.

2. *Transformation in XML-Darstellung*: Die ASCII-Files werden auf Basis eines MSC-Metamodells in MSC-XML-Instanzen überführt.

3. *Transformation in Automatendarstellung*: MSC-XML-Instanzen werden in Endliche Automaten (EA) transformiert.

4. *Reduktion der Automaten*: Endliche Automaten (EA) werden durch ϵ-Eliminierung und durch Determinierung in deterministische endliche Automaten überführt (DEA).

5. *Kompatibilitätsanalyse auf Basis von Automaten*: Vergleich der deterministischen Automaten $(DEA(MSC_1)$, $DEA(MSC_1))$ und Analyse bzgl. deren Rückwärtskompatibilität.

In den nächsten Abschnitten werden die einzelnen Schritte genauer erläutert.

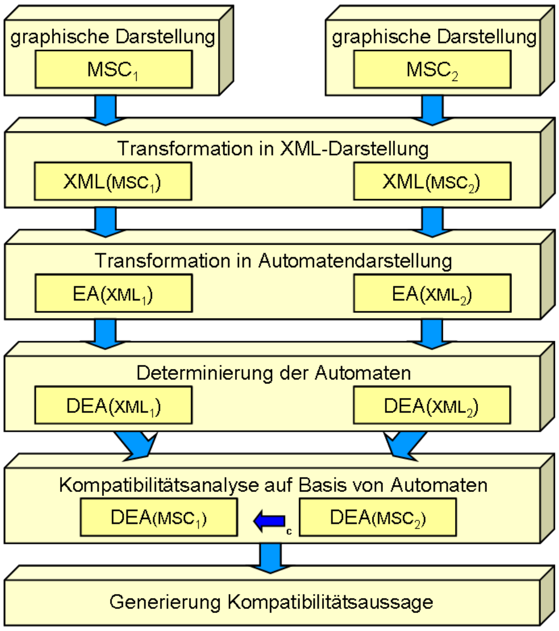


Abbildung 7.5: Schritte der MSC-Vergleichsmethode

7.4.1 Graphische Darstellung von MSCs

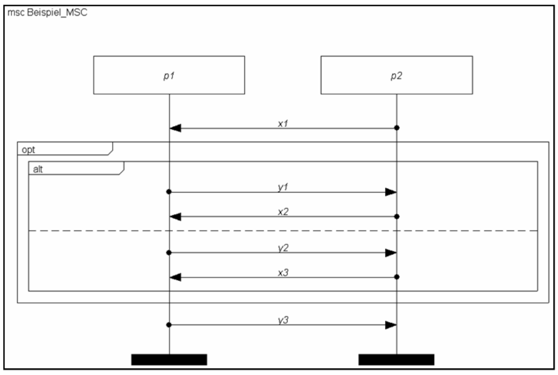


Abbildung 7.6: Beispiel für ein MSC, das mit MSC-Editor erstellt wurde

MSCs können mit unterschiedlichen, am Markt befindlichen, Tools erstellt werden. Die Beschreibung der MSCs basiert auf einer gemeinsamen Syntax, die in [80] festgelegt ist. Der Vorteil dieser Tools ist, dass diese die Einhaltung der Syntax sicherstellen. In Rahmen von CompA wurde ein von ESG[2] entwickelter MSC-Editor eingesetzt [72]. In Abbildung 7.6 ist ein damit erstelltes MSC dargestellt. Das MSC kann als ASCII-File exportiert (vgl. Listing 7.1) und dient als weitere Basis für die Transformation nach

---

[2] Elektroniksystem- und Logistik-GmbH

XML.

Listing 7.1: ASCII-File eines MSCs

```
/* ESG MSC Editor Version 3.3.1 */ mscdocument Untitled;
    reference Beispiel_MSC utilities msc Beispiel_MSC;
p1: instance /*FBlockName: '', InstanceId: '', InstanceDevice:
    '', MostCatalogAlias: 'USERDEF', ExtensionCatalogAlias:
    ''*/;
p2: instance /*FBlockName: '', InstanceId: '', InstanceDevice:
    '', MostCatalogAlias: 'USERDEF', ExtensionCatalogAlias:
    ''*/;
p2: label L0;
out USERDEF.x1,1 to p1;
p1: in USERDEF.x1,1 from p2;
all: opt begin;
all: alt begin;
p1: label L1;
out USERDEF.y1,2 to p2;
p2: in USERDEF.y1,2 from p1;
p2: label L2;
out USERDEF.x2,3 to p1;
p1: in USERDEF.x2,3 from p2;
alt;
p1: label L3;
out USERDEF.y2,4 to p2;
p2: in USERDEF.y2,4 from p1;
p2: label L4;
out USERDEF.x3,5 to p1;
p1: in USERDEF.x3,5 from p2;
alt end;
opt end;
p1: label L5;
out USERDEF.y3,6 to p2;
p2: in USERDEF.y3,6 from p1;
p1: endinstance;
p2: endinstance;
endmsc;
```

7.4.2 Transformation in XML-Darstellung

MSC-Editoren speichern erstellte MSCs i.d.R. in unterschiedlichen Formaten ab. Daher ist es für eine (generische) Umsetzung einer Transformation von MSCs in Automaten sinnvoll, eine gemeinsame Beschreibungs-Basis einzuführen. Im Rahmen von CompA wird als gemeinsame Basis ein MSC-Metamodell verwendet, das auf Basis von EMF (Eclipse Modeling Framework) [41][42][20] entworfen wurde. Dieses

MSC-Metamodell ist der Ausgangspunkt für alle weiteren Berechnungen bzw. Transformationen im Rahmen des CompA-Ansatzes. Mittels Parser können beliebige MSC-Formate in MSC-(EMF/XML)-Instanzen transformiert werden und auf dieser Basis verglichen werden. Dies hat den Vorteil, dass für verschiedene MSC-Editoren lediglich ein spezifischer Parser erstellt werden muss (vgl. Abbildung 7.7). Das im Rahmen von CompA verwendete MSC-Metamodell und der MSC-Parser wurde bei der BMW Group entwickelt. Der MSC-Parser transformiert ASCII-Files in eine korrespondierende XML-Instanz. In Abbildung 7.8 ist beispielsweise die XML-/EMF-Instanz des ASCII-Files aus Listing 7.1 dargestellt.

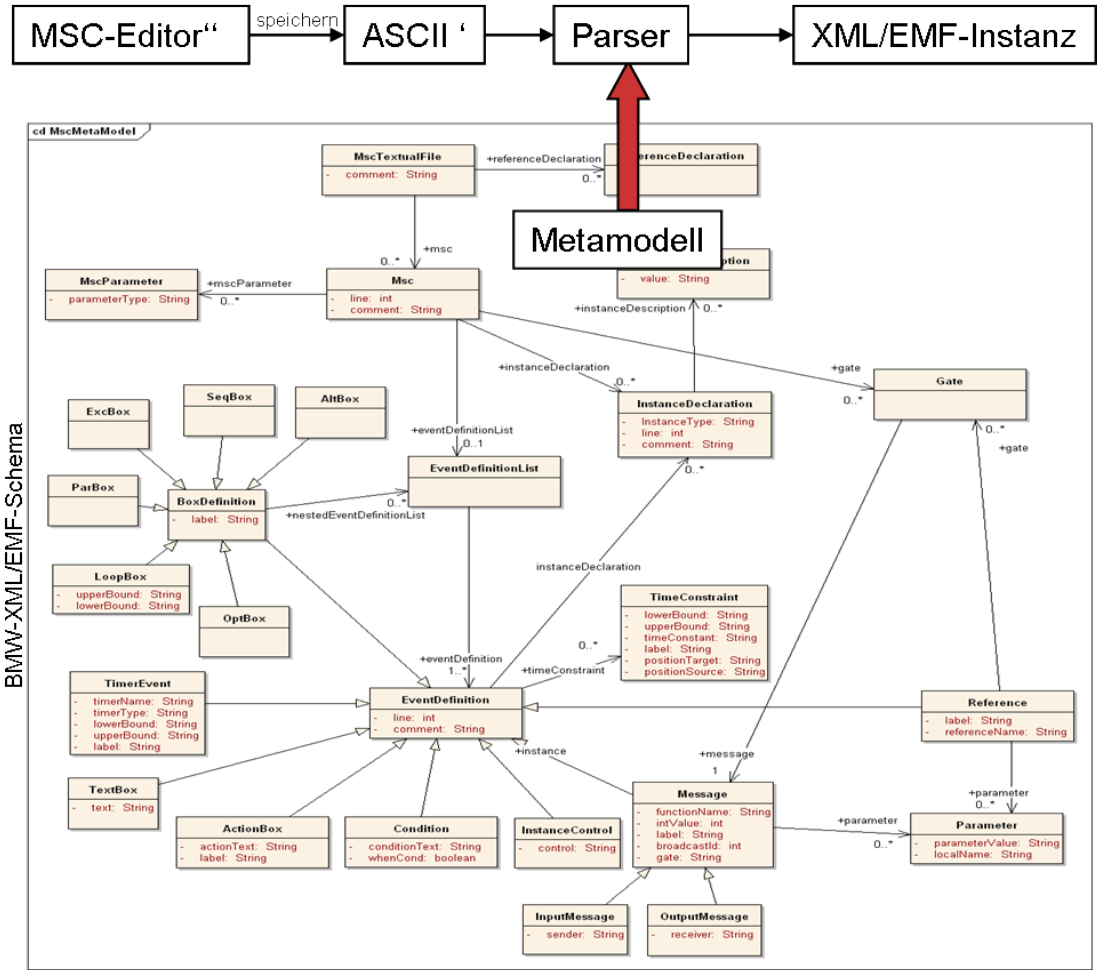
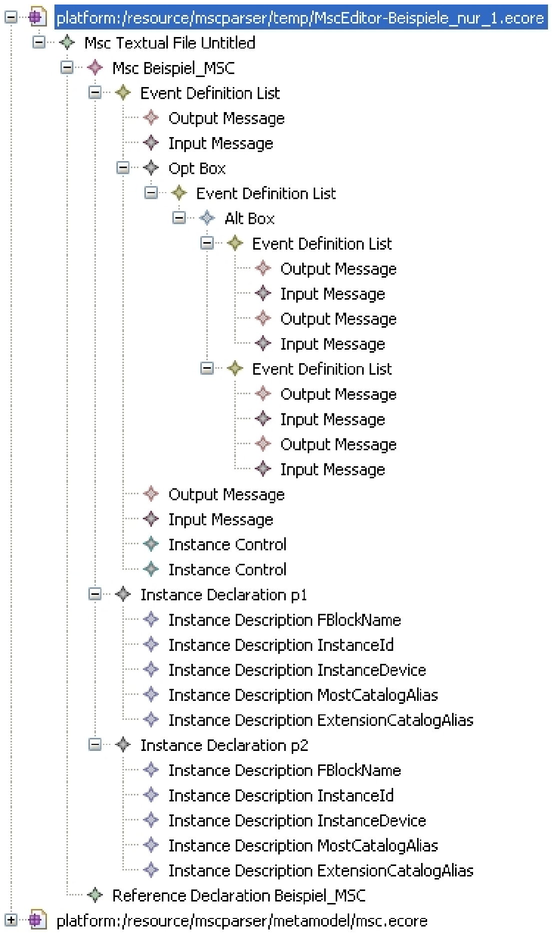


Abbildung 7.7: Transformation in XML-/EMF-Instanz



- platform:/resource/mscparser/temp/MscEditor-Beispiele_nur_1.ecore
  - Msc Textual File Untitled
    - Msc Beispiel_MSC
      - Event Definition List
        - Output Message
        - Input Message
        - Opt Box
          - Event Definition List
            - Alt Box
              - Event Definition List
                - Output Message
                - Input Message
                - Output Message
                - Input Message
              - Event Definition List
                - Output Message
                - Input Message
                - Output Message
                - Input Message
        - Output Message
        - Input Message
        - Instance Control
        - Instance Control
      - Instance Declaration p1
        - Instance Description FBlockName
        - Instance Description InstanceId
        - Instance Description InstanceDevice
        - Instance Description MostCatalogAlias
        - Instance Description ExtensionCatalogAlias
      - Instance Declaration p2
        - Instance Description FBlockName
        - Instance Description InstanceId
        - Instance Description InstanceDevice
        - Instance Description MostCatalogAlias
        - Instance Description ExtensionCatalogAlias
    - Reference Declaration Beispiel_MSC
- platform:/resource/mscparser/metamodel/msc.ecore

Abbildung 7.8: Beispiel für Abbildung eines MSC-XML-/EMF-Baumes

7.4.3 Transformation in Automaten-Darstellung

Die auf Basis des MSC-Metamodells geparsten XML-/EMF-Instanzen bilden die Ausgangsbasis für alle weiteren Berechnungen. Die Transformation von MSCs in Automaten gliedert sich in folgende Schritte:

1. Dekomposition von MSCs in die Instanzen p_i
2. Transformation der dekomponierten Instanzen p_i in endliche Automaten

Bei der Transformation können drei unterschiedliche Automaten-Typen auftreten:

- Nicht-deterministische endliche Automaten (NEA)
- Nicht-deterministische endliche Automaten mit ϵ-Transitionen (ϵ-NEA)
- Deterministische endliche Automaten (DEA)

Dekomposition der MSCs

Jede Instanz eines MSCs beschreibt, analog zu einem endlichen Automaten, ein Ein-/Ausgangsverhalten und kann daher separat in einen Automaten transformiert werden. In Abbildung 7.9 ist ein MSC, bestehend aus zwei Instanzen, dargestellt. Jede Instanz wird separat in einen Automaten transformiert. Hierfür werden die MSCs an den Instanzen „aufgetrennt". Zur Sicherstellung, dass die Information, welche Ereignisse zwischen welchen Instanzen ausgetauscht werden, nicht verloren geht, werden die Ereignisse mit einem Präfix ([!,?][Sender-/Empfänger-Instanz]) versehen. Diese Präfixe werden durch einen Doppelpunkt getrennt und dem Ereignisnamen vorangestellt:

- Die Symbole [!,?] kennzeichnen analog zu [87] das Senden(!)/Empfangen(?).
- Die Zuordnung, welche Instanz ein Ereignis sendet und welche Instanz ein Ereignis empfängt, wird durch die Notation [Sender-/Empfängerinstanz] gekennzeichnet.

Beispiel: In Abbildung 7.9 wird ein Ereignis y_1 von Instanz p_1 zu Instanz p_2 gesendet. Durch die Dekomposition erhält das Ereignis das Präfix $!p_1/p_2 : y_1$. Einen Sonderfall bei der Dekomposition stellen die *Gates* bzw. das *Environment* dar.

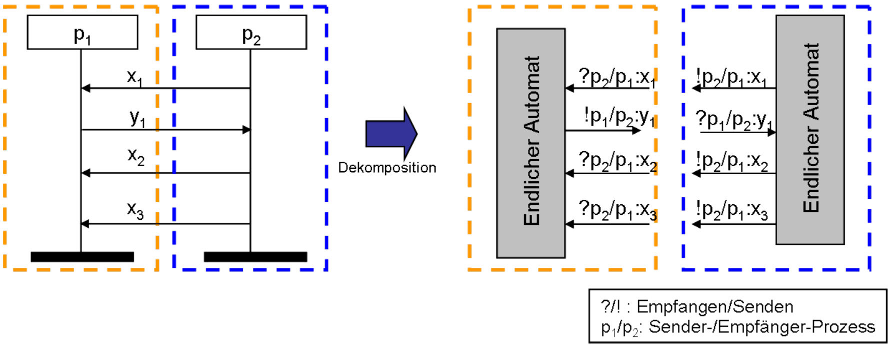


Abbildung 7.9: Dekomposition und Einführung einer Nomenklatur

Gates - Environment Messages werden nicht nur über Instanzen ausgetauscht, sondern können auch an Gates oder an die Systemumgebung (Environment) gesendet, aber auch von dort empfangen werden (vgl. Kapitel 2.3.2). Gates und/oder Environment besitzen nicht, wie Instanzen, eine sequentielle Ordnung der Sende-/Empfangsereignisse und werden daher nicht in separate Automaten transformiert. Diese Elemente finden bereits Beachtung bei der Dekomposition bzw. bei der Festlegung der Nachbarschaftsbeziehungen und werden den Messages bzw. deren Ereignissen zugewiesen. In Abbildung 7.10 ist die Dekomposition von Gates und Environment dargestellt.

- *Environment:* Ereignisse, die von einer Systemumgebung gesendet/empfangen werden, werden über das Präfix *env* gekennzeichnet.

- *Gate:* Ereignisse, die von einem Gate gesendet/empfangen werden, werden über das Präfix *gate_(Name des Gates)* gekennzeichnet.

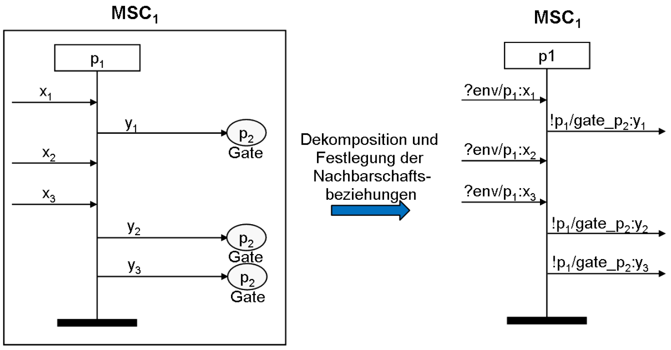


Abbildung 7.10: Dekomposition von MSC-Element *Gate* und *Environment*

Transformation von dekomponierten Instanzen p_i in endliche Automaten

Nach der Dekomposition wird jede Instanz p_i separat in einen endlichen Automaten überführt. Hierzu werden die Instanzen von oben nach unten traversiert und die MSC-Elemente mit Hilfe von Transformationsregeln in einen korrespondieren endlichen Automaten umgewandelt.

Endliche Automaten $A = (Q, \Sigma, \delta, q_0, F)$ bestehen aus Zuständen $Q = \{q_i, ...\}$, Übergangsfunktionen δ und einem Eingabealphabet Σ. Zusätzlich werden der *initiale*-Zustand q_0 und die *finalen* Zustände $F = \{q_n, ...\}$ explizit gekennzeichnet. Um komplexe MSC-Elemente wie z.B. Loop-, Alternative-Blöcke, etc. in Automaten zu transformieren, wird das Eingabealphabet zusätzlich um ϵ-Übergangsfunktionen erweitert (vgl. Grundlagen Automaten in Kapitel 2.4).

Für die unterschiedlichen MSC-Elemente werden spezifische Transformationsregeln benötigt. Der CompA-Ansatz bietet für die, in ITU-Z.120 [80], definierten MSC-Elemente spezifische Transformationsregel an. Die Transformationsregeln sind so modular, rekursiv und hierarchisch aufgebaut, dass auch verschachtelte MSC-Inline-Expressions (z.B. Optional-Operator innerhalb eines Alternative-Operator) in Automaten überführt werden können. Für jedes MSC-Element existiert also eine spezifische Transformationsregel, die jeweils ein MSC-Element in einen korrespondieren Automaten überführt. Die Automaten werden anschließend über Konkatenation zusammengeführt. In Abbildung 7.11 ist ein MSC und der zugehörige „transformierte“ Automat dargestellt. Die MSC-Elemente $p_0/p_1 : x_1$ und $p_0/p_1 : x_2$ werden jeweils in einen Automaten überführt und über Konkatenation zusammengefügt. Das MSC-Element *Alternative-Operator* wird ebenfalls in einen spezifischen Automaten transformiert und über Konkatenation mit vorhergehenden Automaten verbunden.

In Algorithmus 4 ist die Traversierung der Instanzen, der Aufruf der Transformationsregeln und das Verbinden der transformierten Automaten beschrieben. In den folgenden

Algorithm 4 (A,z)=transformiere_MSC_Elemente(z)

```
Input: $z$ // Index des aktuellen Zustands;
Output: $A$ // Automat; $z$ // Index des aktuellen Zustands;;
$A = \emptyset$
// Wiederhole solange bis Prozess- oder Section-Ende erreicht
repeat
    m=getnext_MSC-Element() // Hole nächstes MSC-Element
    if (m==Ereignis) then
        (B,z):=transformiere_Ereignis(m,z);
    else if (m==AltBox) then
        (B,z):=transformiere_Alternative(z);
    else if (m==OptBox) then
        (B,z):=transformiere_Optional(z);
    else if (m==LoopBox$< n >$) then
        (B,z):=transformiere_Loop<n>(z);
    else if ... then
        ....
    else if (m==LoopBox$< n, inf >$) then
        (B,z):=transformiere_Loop<n,inf>(z);
    else if (m==ExcBox) then
        (B,z):=transformiere_Exception(z);
    else if ... then
        ....
    end if
    // Füge Automaten $A, B$ über Konkatenation zusammen
    $(A) := Konkatenation_{v}on_{A}utomaten(A, B)$
until (m==Instanz-Ende) $\vee$ (m==Section-Ende)
return (A,z);
```

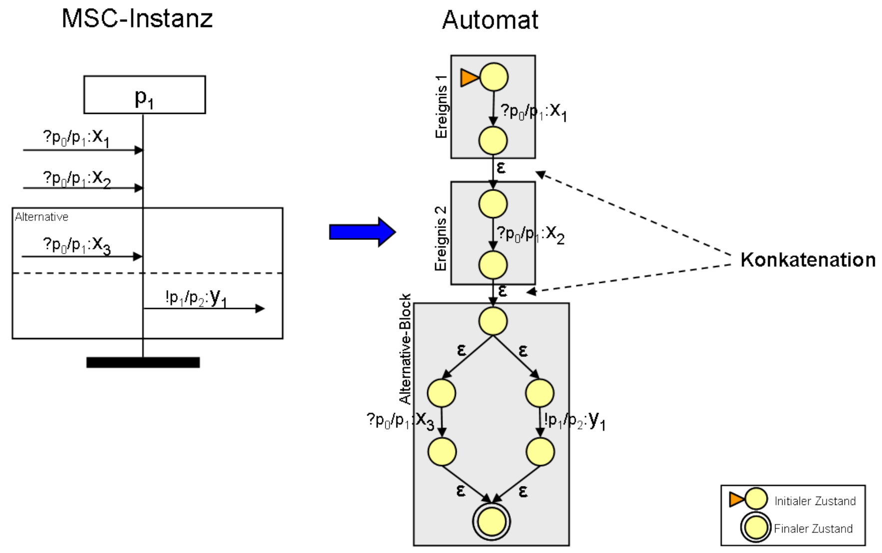


Abbildung 7.11: MSC-Transformation in Automaten

Abschnitten werden die einzelnen Transformationsregeln beschrieben.

Ereignis Ein MSC-Ereignis *e* wird bei einer Transformation als Eingabesymbol interpretiert und somit als Erweiterung vom Eingabealphabet Σ. Dadurch entsteht ein endlicher Automat, bestehend aus einem Start-Zustand q_z, einem akzeptierenden Zustand q_{z+1} und einer Übergangsfunktion $\delta(q_z, e) = q_{z+1}$.
In Abbildung 7.12 ist die Transformation für das Ereignis $e = (p_0/p_1 : x_1)$ dargestellt. Der erzeugte endliche Automat besteht aus einem Start-Zustand $q_z = q_0$, einem akzeptierenden Zustand $q_{z+1} = q_1$ und der Übergangsfunktion $\delta(q_0, (p_0/p_1 : x_1)) = q_1$.
Die Transformationsvorschrift ist in Algorithmus 5 beschrieben.

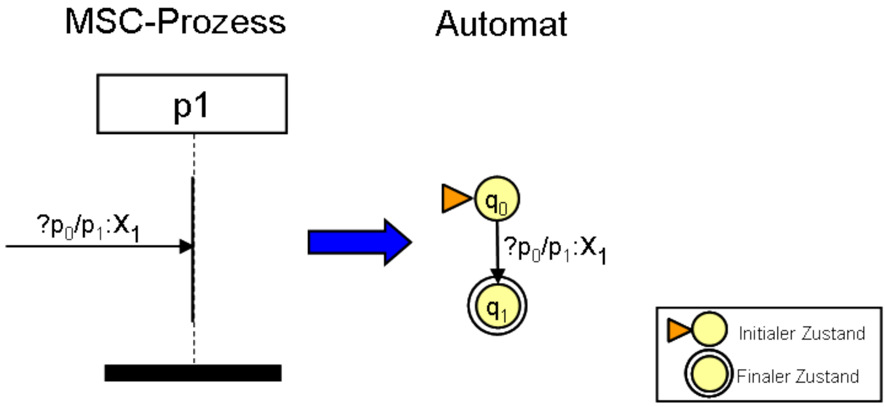


Abbildung 7.12: Transformation eines Ereignisses in einen Automaten

```
Algorithm 5 (A,z)=transformiere_Ereignis($e, z$)
Input: $e$ // Ereignis; $z$ // Index des aktuellen Zustands;
Output: $A$ // Automat;
// erzeuge zwei Zustände
z++; add $q_z$;
z++; add $q_z$;
// erzeuge $\delta$ von Zustand $q_{z-1}$ auf $q_z$ über Ereignis $e$
add $\delta(q_{z-1}, e) = q_z$
// markiere $q_{z-1}$ als Anfangs-Zustand
$q_0 \leftarrow q_{z-1}$
// markiere $q_z$ als akzeptierenden Zustand
$F \leftarrow q_z$
// erzeugter Automat
$A = ((Q = \{q_{z-1}, q_z\}), (\Sigma = \{e\}), \delta, q_{z-1}, (F = \{q_z\}))$
return $(A, z)$
```

Condition, Action Die Basic-MSC Elemente *Condition* und *Action* können, abstrakt gesehen, als Ereignisse interpretiert werden, d.h. nur wenn das Ereignis *Condition* anliegt, kann auch die Ereignissequenz „weiterlaufen".
Mit dieser Annahme können *Condition* und *Action* analog zu den Ereignissen behandelt werden. Bei der Transformation von *Condition* und *Action* werden Automaten bestehend aus einem *initialen* und einem *finalen* Zustand erzeugt. Der „Inhalt" von *Condition* und *Action* wird dabei als Übergangsfunktion interpretiert. Um die Elemente *Condition* und *Action* von den Ereignissen unterscheiden zu können, erhalten diese die Präfixe „*Condition:*" und „*Action:*".
In Abbildung 7.13 ist eine *Condition* $< x == 1 >$ und ein *Action*-Operator mit Inhalt $start : Fenster\ auf$ dargestellt. Bei der Transformation wird der Inhalt der *Condition* und des *Actions-Operators* in die korrespondierende Übergangsfunktionen *Condition:x==1* und *Action:start:Fenster auf* transformiert.

Timer Das Basic-Element *Timer* kann, wie die Elemente *Condition* und *Action*, bei der Transformation ebenfalls analog zu den Ereignissen behandelt werden.
Mittels Timer kann in MSCs ein zeitliches Verhalten abgebildet werden. Die Timer starten und enden innerhalb einer Ereignissequenz eines MSCs. Daher können die Elemente, die einen Timer abbilden auch als Übergangsfunktionen abgebildet werden (vgl. Abbildung 7.14). Zur Unterscheidung der Timervarianten, werden den Timer-Übergangsfunktionen bei der Transformation die Präfixe (Start_, Stopp_, Reset_) vorangestellt.

Alternative-Operator Bei der Transformation eines *Alternative*-Operators wird jede *Section* separat betrachtet. So wird der Inhalt jeder *Section* z.B. Ereignisse, etc. entsprechend den Transformationsregeln in einen Automaten transformiert. Im Anschluss werden die erzeugten Automaten der Sections durch *Vereinigung* mittels ϵ-Übergängen

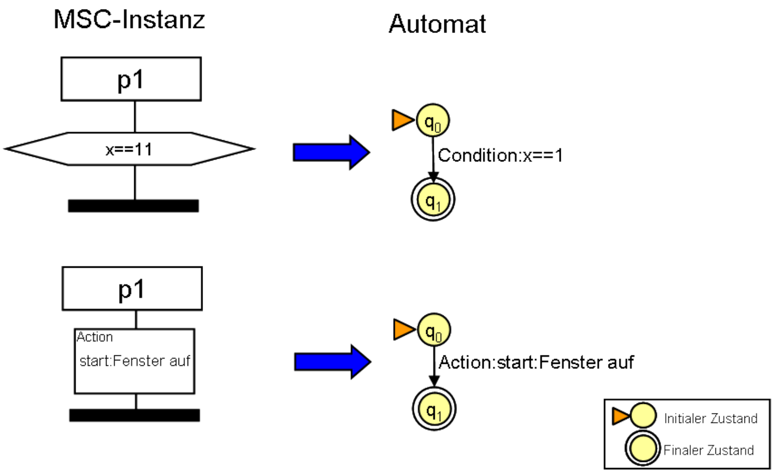


Abbildung 7.13: Transformation von Condition und Action

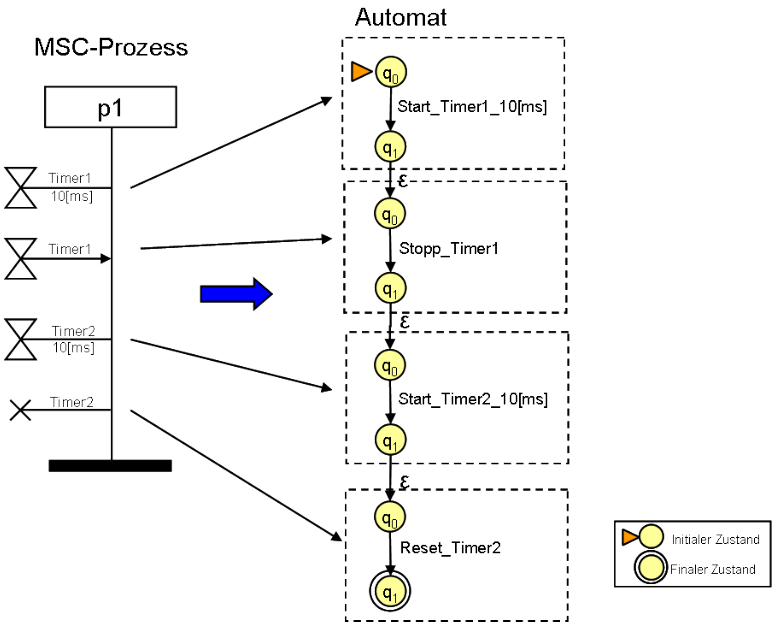


Abbildung 7.14: Transformation von Timer

zusammengeführt (vgl. Abbildung 7.15).

Das Ergebnis der Transformation ist ein ϵ-NEA. Die „vereinigten“ Automaten verzweigen sich über ϵ-Übergänge und bilden so die „alternativen“ Ereignissequenzen. In Abbildung 7.16 ist ein Beispiel dargestellt. Nach dem Empfang des Signals x_1 gibt es zwei Alternativen: 1) Entweder das Signal y_1 wird gesendet oder 2) das Signal x_2 wird empfangen. Bei der Transformation wird zuerst das Ereignis x_1 in einen Automaten A_1 transformiert. Anschließend ist der *Alternative*-Operator an der Reihe. Hierzu werden

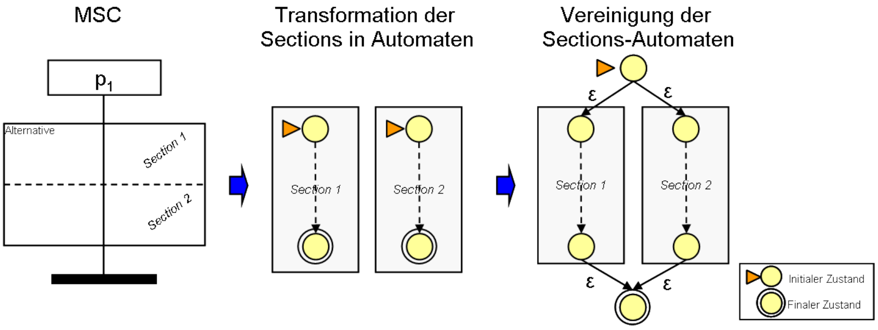


Abbildung 7.15: Transformation eines Alternative-Operators

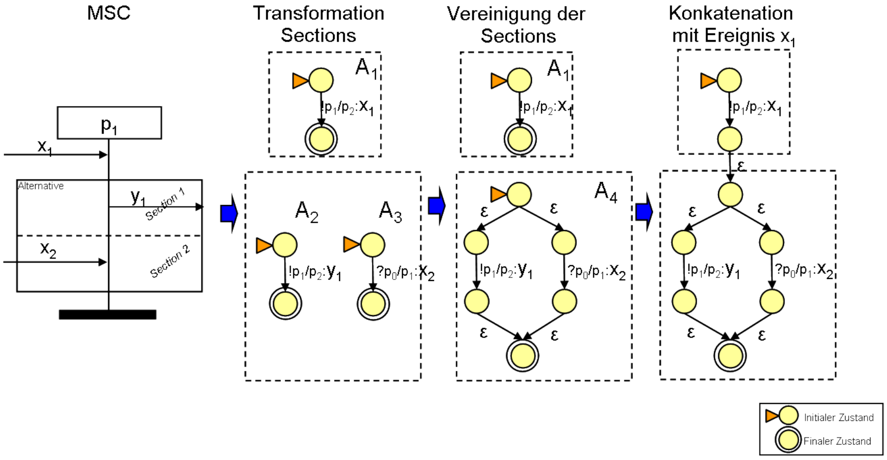


Abbildung 7.16: Beispiel für Transformation eines Alternative-Operators

bei der Transformation die Inhalte von *Section 1* und *Section 2*, also deren Ereignisse, in separate Automaten (A_2, A_3) transformiert. Diese beiden Automaten (A_2, A_3) werden zu einem Automaten A_4 vereinigt. A_4 ist der zum *Alternative*-Operator korrespondierende Automat. Zuletzt wird der Automat A_1 (für das Ereignis x_1) und der Automat A_4 (für den *Alternative*-Operator) über Konkatenation aneinander gereiht.
Der Algorithmus ist in Algorithmus 6 beschrieben.

Optional-Operator Das besondere Kennzeichen eines *Optional*-Operators ist, dass die *Section* eines *Optional*-Operators ausgeführt werden kann, aber nicht ausgeführt werden muss. Um dies im Automaten zu realisieren, muss ein zusätzlicher ϵ-Übergang eingefügt werden.
Bei der Transformation wird der Inhalt der *Section* in einen Automaten transformiert. Um zu modellieren, dass die *Section* ausgeführt werden kann, aber nicht muss, wird der *initiale Zustand* und der *finale Zustand* des erzeugten Automaten über eine ϵ-Übergangsfunktion miteinander verbunden (vgl. Abbildung 7.17). Die Transformationsregel

```
Algorithm 6 (A,z)=transformiere_Alternative($z$)
Input: $z$ // Index des aktuellen Zustands
Output: $A$ // Automat $A$
$A = \emptyset$;
// Erzeuge pro Section einen Automaten
$m = get\_Anzahl\_sections$;
for (n=1 to m) do
    $(B_n, z) := transformiere\_MSC\_Elemente(z)$;
end for
// vereinige die erzeugten Automaten
$(A, z) := Vereinigung\_von\_Automaten(z, m, \{B_n\})$;
return $(A, z)$
```

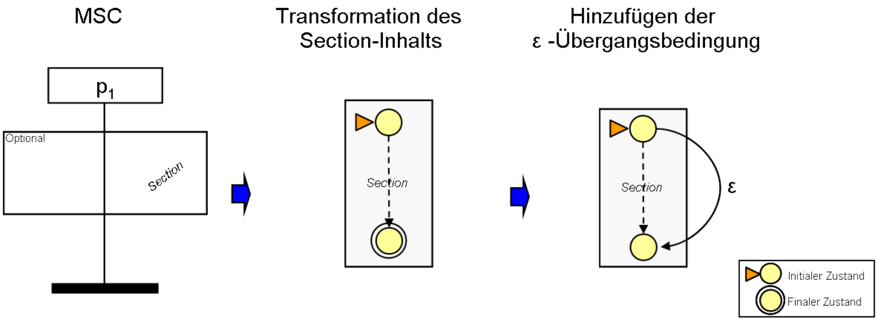


Abbildung 7.17: Transformation eines Optional-Operators

ist in Algorithmus 7 beschrieben.

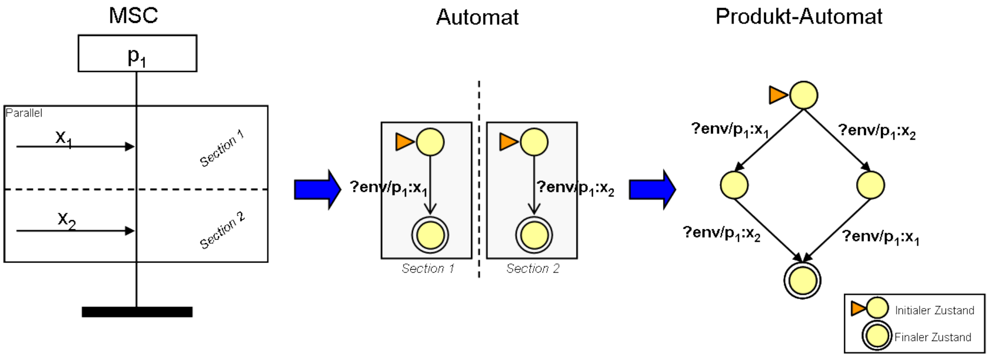


Abbildung 7.18: Transformation eines Parallel-Operators

Parallel-Operator Basis-Automaten können paralleles Verhalten nicht direkt abbilden. In der Automatentheorie existieren Ansätze mit denen parallele Automatenkon-

Algorithm 7 (A,z)=transformiere_Optional(z)

```
Input: $z$ // Index des aktuellen Zustands
Output: $A$ // Automat
// Transformiere alle Elemente innerhalb von Optional
$(A, z) := transformiere\_MSC\_Elemente(z)$;
// Füge zusätzlichen $\epsilon$-Übergang hinzu
$A = (Q, \Sigma, \delta, q_{0_A}, \{q_z\})$;
$\Sigma \leftarrow \epsilon$;
add $\delta(q_{0_A}, \epsilon) = q_z$;
return $(A, z)$
```

strukte in Produktautomaten [74] überführt werden können. Diese Methode kann auch hier angewendet werden. In Abbildung 7.18 ist dargestellt, wie ein *Parallel*-Operator in einen Produktautomaten überführt wird. Jede Section des *Parallel*-Operators wird separat in einen Automaten transformiert und anschließend der Produktautomat gebildet. Formal setzt sich ein Produktautomat A aus zwei Automaten A_L und A_M wie folgt zusammen [74].
$A = (Q_L \times Q_M, \Sigma, \delta, (q_L, q_M), F_L \times F_M)$
Parallel-Operatoren können auch verschachtelt sein. In Abbildung 7.19 ist der komplexe Fall dargestellt, dass eine Section eines Parallel-Operators einen weiteren Parallel-Operator enthält. Die Sections von *Parallel* werden jeweils separat in Automaten überführt und so lange als parallel geführt, bis alle Elemente der *Sections* traversiert wurden. Wird ein weiterer *Parallel*-Operator entdeckt, wird dieser auch als paralleler Automat dargestellt (1. Schritt). Anschließend werden die parallelen Automaten rekursiv in einen Produktautomaten überführt. So wird in unserem Beispiel erst der parallele Automat in Section 1 (2. Schritt) und dann die beiden übergeordneten parallelen Automaten zu einem Produktautomaten zusammengeführt (4. Schritt). Um die Berechnung des Produktautomaten effizient zu gestalten, wird vor der Generierung eines Produktautomaten eine Überprüfung auf ϵ-Übergänge durchgeführt und diese eliminiert (vgl. 3. Schritt). Dadurch können die Automaten um „überflüssige“ Zustände und Übergänge reduziert werden, wodurch die Komplexität der Berechnung von Produktautomaten reduziert wird.

Loop-Operator Der Inhalt eines Loop-Operators wird nach den Transformationsregeln in einen endlichen Automaten A transformiert. Die Besonderheit des Loop-Operators ist, dass der Operator unterschiedlich parametrisiert sein kann. In Kapitel 2.3.3 wurden hierfür die möglichen Parametrisierungen bereits ausführlich dargestellt. Es lassen sich folgende Parametrisierungen unterscheiden:

- Loop$< n >$
- Loop$< n, m >$
- Loop$< inf >$

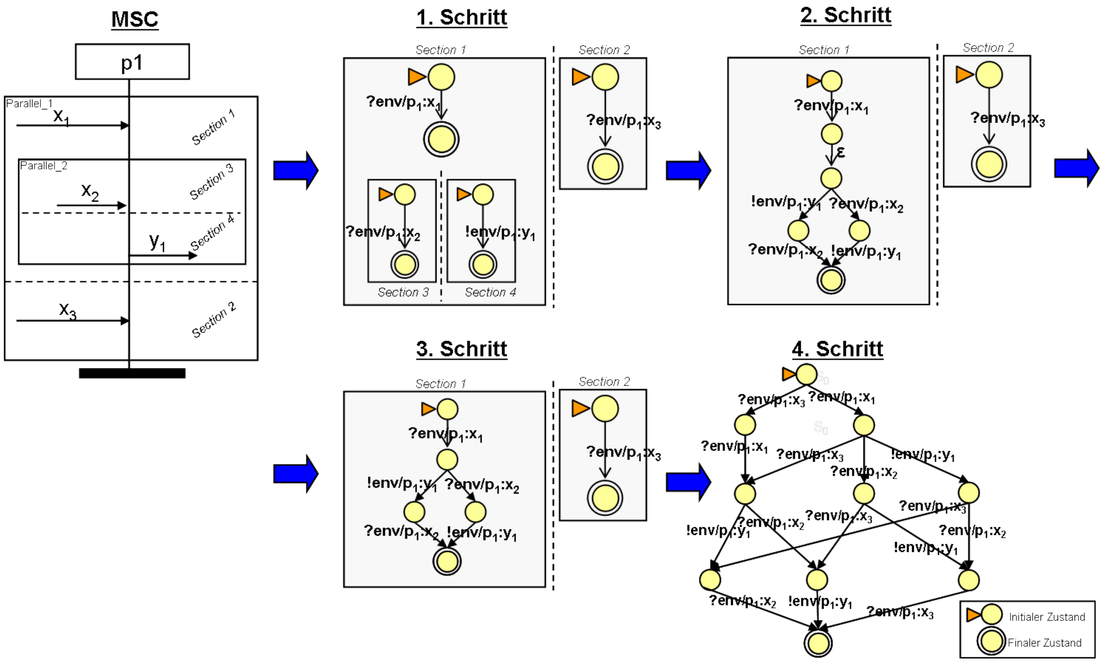


Abbildung 7.19: Transformation eines Parallel-Operators, der mit einem weiteren Parallel-Operator verschachtelt ist

- Loop$< n, inf >$

Eine Sonderrolle bei der Transformation spielt der Parameter „inf". Dieser besagt, dass eine Loop-Schleife unendlich ausgeführt werden kann. Die einzelnen Transformationsregeln sind in den Algorithmen 8, 9, 10 und 11 beschrieben.

Loop$< n >$ Der Parameter $< n >$ bedeutet, dass ein Loop-Operator exakt n-mal (mit $n \in \mathbb{N}$) ausgeführt wird. Bei der Transformation wird zuerst der Inhalt des Loop-Operators in einen einzelnen Automaten A transformiert. Anschließend wird der Automat A n-mal sequentiell aneinander gereiht (vgl. Algorithmus 8). In Abbildung 7.20 ist die Transformation bildlich dargestellt.

Loop$< n, m >$ Der Parameter $< n, m >$ bedeutet, dass ein MSC-Loop-Operator mindestens n-mal und maximal m-mal ausgeführt werden kann, wobei $n, m \in \mathbb{N}$. Um dieses Verhalten auf Automaten zu übertragen, müssen ϵ-Übergänge zusätzlich hinzugefügt werden.
In Abbildung 7.21 ist die schrittweise Transformation dargestellt. Zuerst wird der Inhalt des Loop-Operators in einen einzelnen Automaten A transformiert. Dieser Automat kann also mindestens n-mal und maximal m-mal durchlaufen werden. Um dies abzubilden, wird der Automat A zuerst n-mal konkateniert und so die Mindestanzahl der Durchläufe abgebildet. Die nächsten (m-n) Durchläufe bzw. Automaten können,

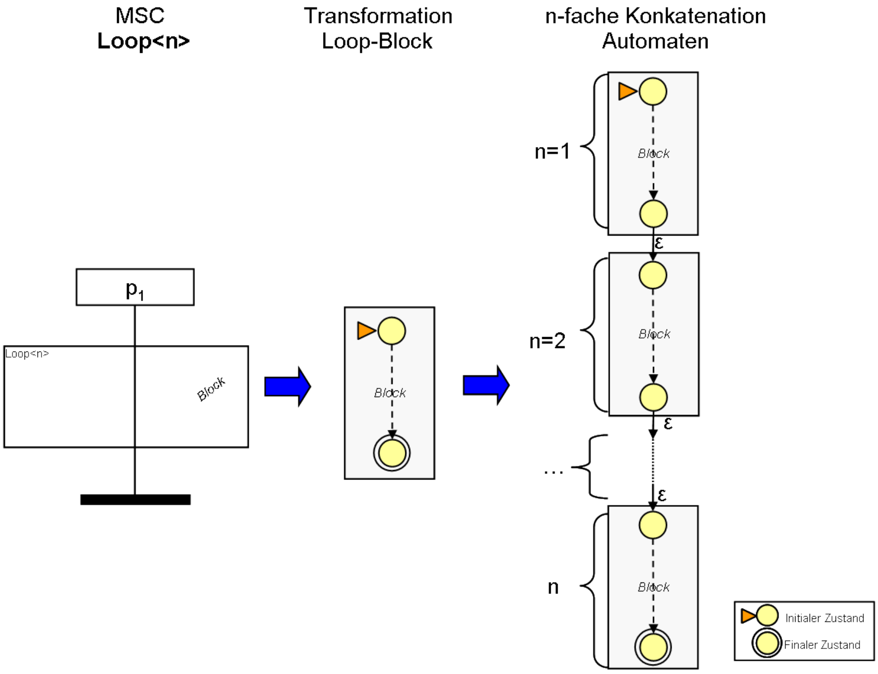


Abbildung 7.20: Transformation von Loop-Operator mit Parametersatz *Loop<n>*

müssen aber nicht mehr stattfinden bzw. durchlaufen werden. Dies wird dargestellt, indem bei den nächsten (m-n) Automaten (ab Automat n+1) der *initiale* Zustand über einen ϵ-Übergang mit dem *finalen* Zustand verbunden wird. Dadurch entsteht der Automat A'. Dieser Automat wird nun (m-n)-mal konkateniert und auch mit dem Automat A verkettet (vgl. Algorithmus 9).

Sonderfall Loop$< inf >$**:** Diese Parametrisierung stellt einen Sonderfall von *Loop*$< n >$ mit *n=inf* dar. inf steht für $infinite$ und bedeutet, dass die Loop-Schleife unendlich durchlaufen wird. Die unendliche Schleife wird bei der Transformation wie folgt realisiert (vgl. Abbildung 7.22).

Algorithm 8 (A,z)=transformiere_Loop$< n >$(z)

```
Input: $z$ // Index des aktuellen Zustands
Output: $A$ COMMENTKonstruierter Automat; $z$ // Index des aktuellen Zustands
// Transformiere inneren Block
for x=1 to n do
    goto Anfang_der_Section // Springe zum Beginn der Section
    $(B, z) = transformiere\_MSC\_Element(z)$
    $(A) = Konkatenation\_MSC\_Prozess(A, B)$
end for
return $(A, z)$
```

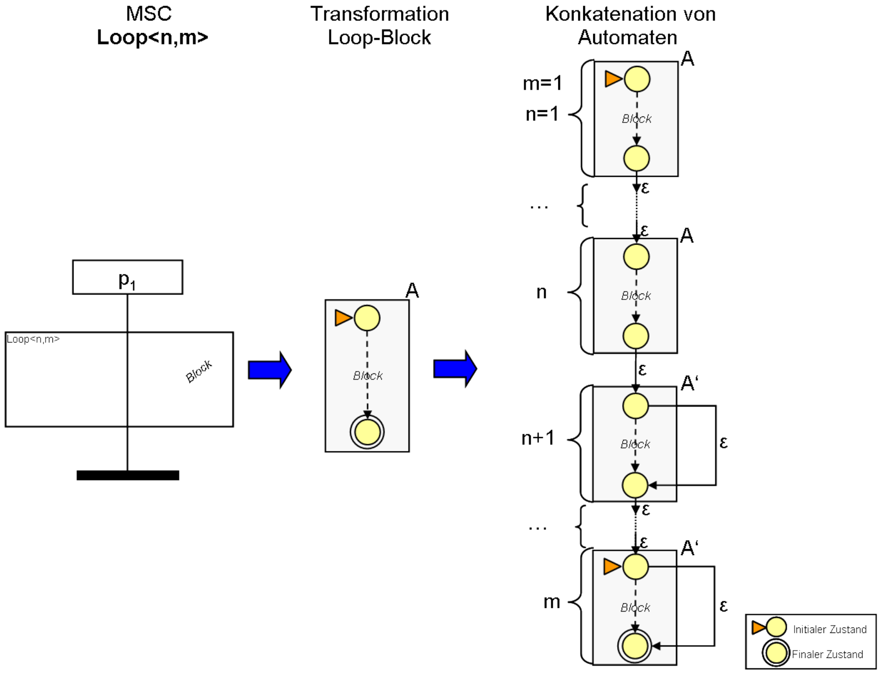


Abbildung 7.21: Transformation von Loop-Operator mit Parametersatz *Loop<n,m>*

Zuerst wird der Inhalt des Loop-Operators in einen Automaten A transformiert. Anschließend wird vom *finalen* Zustand (q_z) des Automaten A ein ϵ-Übergang auf den *initialen* Zustand (q_0) des Automaten hinzugefügt ($q_0 = \delta(q_z, \epsilon)$). Hierdurch wird ein Schleifendurchlauf aufgebaut. Da diese Schleife nicht „verlassen“ werden darf, wird eine fiktive Übergangsfunktion mit dem Parameter inf hinzugefügt, die niemals „TRUE“ wird. Zusätzlich wird ein neuer akzeptierender Zustand hinzugefügt, mit $q_{z+1} = \delta(q_z, inf)$. Anschließend wird q_z als akzeptierender Zustand entfernt. Durch dieses Konstrukt wird eine unendliche Schleife im Automaten realisiert (vgl. Algorithmus 10).

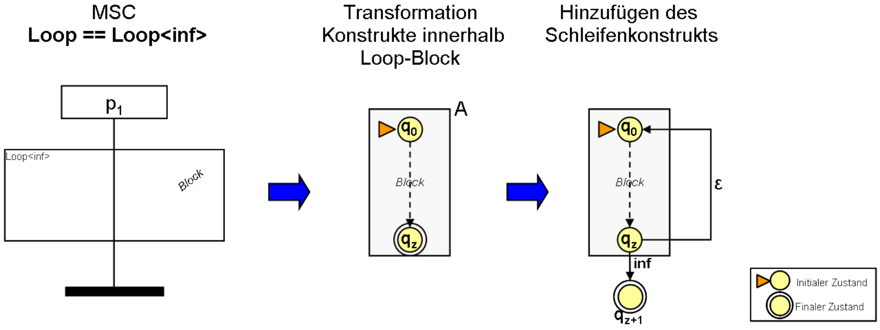


Abbildung 7.22: Transformation eines Loop-Operators mit Parametersatz *Loop<inf>*

Algorithm 9 (A,z)=transformiere_Loop$< n, m >$(z)

```
Input: $z$ // Index des aktuellen Zustands
Output: $A$ // Konstruierter Automat; $z$ // Index des aktuellen Zustands
for x=1 to n do
    goto Anfang_der_Section // Springe zum Beginn der Section
    $(B, z) = transformiere\_MSC\_Element(z)$
    $(C) = Konkatenation\_MSC\_Prozess(C, B)$
end for
for x=n to m do
    goto Anfang_der_Section // Springe zum Beginn der Section
    // Transformiere alle Elemente innerhalb Section
    $(D, z) := transformiere\_MSC\_Elemente(z)$;
    // Füge zusätzlichen $\epsilon$-Übergang hinzu
    $D = (Q_D, \Sigma_D, \delta_D, q_{0_D}, F_D = \{q_z\})$;
    $\Sigma_D \leftarrow \epsilon$
    add $\delta_D(q_{0_D}, \epsilon) = q_z$;
    $(E) = Konkatenation\_MSC\_Prozess(E, D)$
end for
$(A) = Konkatenation\_MSC\_Prozess(C, E)$
return $(A, z)$
```

Algorithm 10 (A,z)=transformiere_Loop$< inf >$(z)

```
Input: $z$ // Index des aktuellen Zustands
Output: $A$ // Konstruierter Automat; $z$ // Index des aktuellen Zustands
// Transformiere alle Elemente innerhalb Section
$(A, z) = transformiere\_MSC\_Element(z)$
// Erzeugter Automat ist vom Typ $A = (Q, \Sigma, \delta, q_{0_A}, F = \{q_z\})$
add $\delta(q_z, \epsilon) = q_{0_A}$; // erzeuge $\epsilon$-Übergang von finalem Zustand $q_z$ auf initialen
    Zustand $q_{0_A}$ von Automat $A$
z++; add $q_z$; // erzeuge neuen Zustand $q_z$
add $\delta(q_{z-1}, (inf)) = q_z$; // erzeuge Übergangsfunktion „inf“ von Zustand $q_{z-1}$ auf
    neuen Zustand $q_z$
// Markiere den neuen Zustand als den neuen finalen Zustand;
$F \leftarrow q_z$;
return $(A, z)$
```

Sonderfall Loop$< n, inf >$: Dieser Fall stellt einen Sonderfall von Loop$< n, m >$ mit *m=inf* dar. Eine Anwendung der Transformationsregel von Loop$< n, m >$ würde bedeuten, dass *m* Automaten, also in diesem Fall unendlich viele Automaten, konkateniert werden müssten. Eine Kombination mit der Transformationsregel von Loop$< inf >$ ist nicht möglich, da die benötigte Schleife nach Durchlauf der Mindestanzahl *n* verlassen darf. Daher wird der Fall Loop$< n, inf >$ gesondert betrachtet. Bei der Transformation wird zuerst der Inhalt des Loop-Operators in einen Automaten A transformiert. Anschließend werden 3 Fälle unterschieden, die in Abbildung 7.23 dargestellt sind (vgl. Algorithmus 11).

1. Fall: Loop$< 1, inf >$
Der *finale* Zustand von Automat A wird über einen ϵ-Übergang mit dessen *initialen* Zustand verbunden. Dadurch erhält man eine Schleife. Zum Verlassen der Schleife bzw. um den *finalen* Zustand des Automaten zu erreichen, muss der Automat mindestens 1-mal durchlaufen werden. Aufgrund des „rückführenden“ ϵ-Übergangs kann der Automat aber auch beliebig oft durchlaufen werden.

2. Fall: Loop$< 0, inf >$
Der erzeugte Automat aus dem 1. Fall (Loop$< 1, inf >$) wird mindestens 1-mal durchlaufen. Um die Möglichkeit abzubilden, dass die Schleife nicht durchlaufen werden muss, wird dem Automaten aus Loop$< 1, inf >$ ein weiterer ϵ-Übergang hinzugefügt. Und zwar wird der initiale Zustand über den ϵ-Übergang mit dem finalen Zustand verbunden. Auf diese Weise kann das Durchlaufen des Automaten übersprungen werden.

3. Fall: Loop$< n, inf >$ **mit** $n > 1$
Der 3. Fall baut ebenfalls auf den 1. Fall auf. Um abzubilden, dass der Automat mindestens n-mal durchlaufen wird, wird der Automat A (n-1)-mal konkateniert. Anschließend wird ein Automat, der analog zu dem Automaten aus dem 1. Fall (Loop$< 1, inf >$) aufgebaut ist, damit konkateniert.

Exception-Operator Der Exception-Operator dient zur Behandlung von Ausnahmefällen. Dies bedeutet, wenn der Exception-Operator durchlaufen wird, wird das MSC beendet.

Bei der Transformation wird im 1. Schritt der Inhalt des Exception-Operators in einen Automaten A transformiert und im 2. Schritt werden alle Elemente nach dem Exception-Operator in einen Automaten B überführt. Im 3. Schritt werden die beiden Automaten *A* und *B* über ϵ-Übergänge mit einem zusätzlichen Zustand verbunden (vgl. Algorithmus 12). Der neue Zustand wird der *initiale* Zustand. Der finale Zustand des Automaten A, also des Exceptions-Blocks, wird zusätzlich explizit gekennzeichnet. Dies hat den Hintergrund, dass bei weiteren Konkatenationen dieser *finale* Zustand nicht mehr berücksichtigt werden soll. Denn wenn dieser Zustand erreicht wird, ist das MSC und damit die korrespondierende Ereignissequenz beendet.

Algorithm 11 (A,z)=transformiere_Loop$< n, inf >$(z)

```
Input: $y$ // Parameter der Loop-Box Loop$< n, m >$; $z$ // Index des aktuellen Zustands
Output: $A$ (Automat)
if $(n = 0)$ then
    // Transformiere inneren Block
    $(A, z) = transformiere\_MSC\_Element(z)$ // $A = (Q, \Sigma, \delta, q_{0_A}, F = q_z)$;
    // Erzeuge $\epsilon$-Übergang von finalem Zustand $q_z$ auf initialen Zustand $q_{0_A}$ und
    umgekehrt
    add $\delta(q_z, \epsilon) = q_{0_A}$;
    add $\delta(q_{0_A}, \epsilon)) = q_z$;
else if $(n = 1)$ then
    // Transformiere inneren Block
    $(A, z) = transformiere\_MSC\_Element(z)$ // $A = (Q, \Sigma, \delta, q_{0_A}, F = q_z)$;
    // Erzeuge $\epsilon$-Übergang von finalem Zustand $q_z$ auf initialen Zustand $q_{0_A}$
    add $\delta(q_z, \epsilon) = q_{0_A}$;
else if $(n > 1)$ then
    for (x=1 to (n-1)) do
        goto Anfang_der_Section // Springe zum Beginn der Section
        $(B, z) = transformiere\_MSC\_Element(z)$
        $(C) = Konkatenation\_MSC\_Prozess(C, B)$
    end for
    // Transformiere inneren Block
    goto Anfang_der_Section // Springe zum Beginn der Section
    $(A, z) = transformiere\_MSC\_Element(z)$ // $A = (Q, \Sigma, \delta, q_{0_A}, F = q_z)$;
    // Erzeuge $\epsilon$-Übergang von finalem Zustand $q_z$ auf initialen Zustand $q_{0_A}$
    add $\delta(q_z, \epsilon) = q_{0_A}$;
    $(A) = Konkatenation\_MSC\_Prozess(C, A)$
end if
return $(A, z)$
```

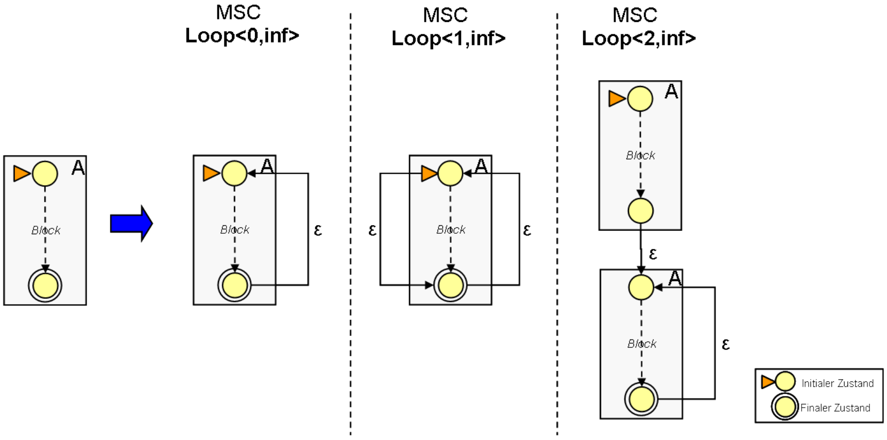


Abbildung 7.23: Transformation eines Loop-Operators mit Parametersatz *Loop<n,inf>*

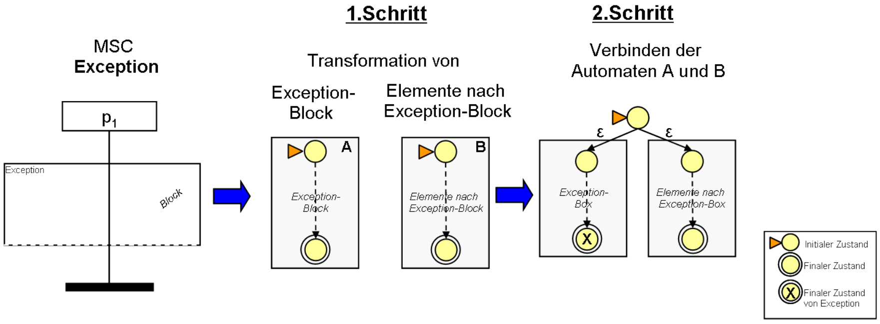


Abbildung 7.24: Transformation von Exception-Operator

Der konstruierte Automat bietet also durch die ϵ-Verzweigung die Alternative, dass entweder der Exception-Operator durchlaufen wird oder dass die Ereignisse nach dem Exception-Operator ausgeführt werden.

Coregion Die sequentielle Reihenfolge von Sende-/Empfangsereignissen wird innerhalb einer *Coregion* aufgehoben. Für eine Transformation in einen Automaten hat dies zur Folge, dass bei n-Ereignissen n!-verschiedene Ereignissequenzen abgebildet werden.
Bei der Transformation werden die Ereignisse in separate endliche Automaten transformiert. Anschließend werden die möglichen Permutationen berechnet und die Automaten einer möglichen Permutation über Konkatenation zusammengefügt. Zuletzt werden diese Automaten über Vereinigung zusammengefügt.
In Abbildung 7.25 ist eine MSC-Instanz mit zwei Ereignissen dargestellt. Da sich beide Ereignisse $?p_0/p_1{:}x_1$, $?p_0/p_1{:}x_2$ innerhalb einer Coregion befinden, ist die Reihenfolge irrelevant. Die beiden Ereignisse werden in die endlichen Automaten A und B trans-

Algorithm 12 (A,z)=transformiere_Exception(z)

Input: z // Index des aktuellen Zustands;

Output: A // Automat;

```
// Transformation aller Elemente innerhalb der Exception Operator
$(B,z) := transformiere\_MSC\_Element(z)$ // $B = (Q_B, \Sigma_B, \delta_B, q_{0_B}, F_B)$;
// Transformation aller Elemente nach dem Exception Operator
$(C,z) := transformiere\_MSC\_Element(z)$ // $B = (Q_C, \Sigma_C, \delta_C, q_{0_C}, F_C)$;
// $A = (Q_A, \Sigma_A, \delta_A, q_{0_A}, F_A)$;$B = (Q_B, \Sigma_B, \delta_B, q_{0_B}, F_B)$
// Erzeuge neuen Automat A aus B und C
$A = (Q_A, \Sigma_A, \delta_A, q_{0_A}, F_A)$;
$Q_A \leftarrow Q_B \cup Q_C$;
$\Sigma_A \leftarrow \Sigma_B \cup \Sigma_C$;
$\delta_A \leftarrow \delta_B \cup \cup \delta_C$;
// Verbinde neuen Zustand mit den initialen Zuständen $q_0$ von A und B
z++; add $q_z$;
add $\delta(q_z, \epsilon) = q_{0_B}$
add $\delta(q_z, \epsilon) = q_{0_C}$
// Erzeuge neuen Automat A aus B und C
$A = (Q_A, \Sigma_A, \delta_A, q_{0_A}, F_A)$;
$Q_A \leftarrow Q_B \cup Q_C \cup q_z$;
$\Sigma_A \leftarrow \Sigma_B \cup \Sigma_C \cup \epsilon$;
$\delta_A \leftarrow \delta_B \cup \delta_C \cup \delta$;
// markiere $q_z$ als neuen initialen Zustand
$q_{0_A} \leftarrow q_z$ // markiere $q_z$ als initialen Zustand
$F_A \leftarrow F_B \cup F_C$ // definiere die finalen Zustände
$M \leftarrow F_B$ // Markiere akzeptierende Zustände von B explizit, so dass diese bei weiteren Konkatenationen nicht berücksichtigt werden
return $(A, z, M)$
```

formiert und anschließend über Konkatenation aneinander gereiht. Da es 2 Ereignisse sind, gibt es hierfür $2! = 2$ verschiedene Permutations-Möglichkeiten der Konkatenation ($L(A)L(B)$ und $L(B)L(A)$). Anschließend werden die möglichen Konkatenationen über Vereinigung miteinander verbunden, so das der neue Automat die Sprache $L = L(A)L(B) \cup L(B)L(A))$ akzeptiert.

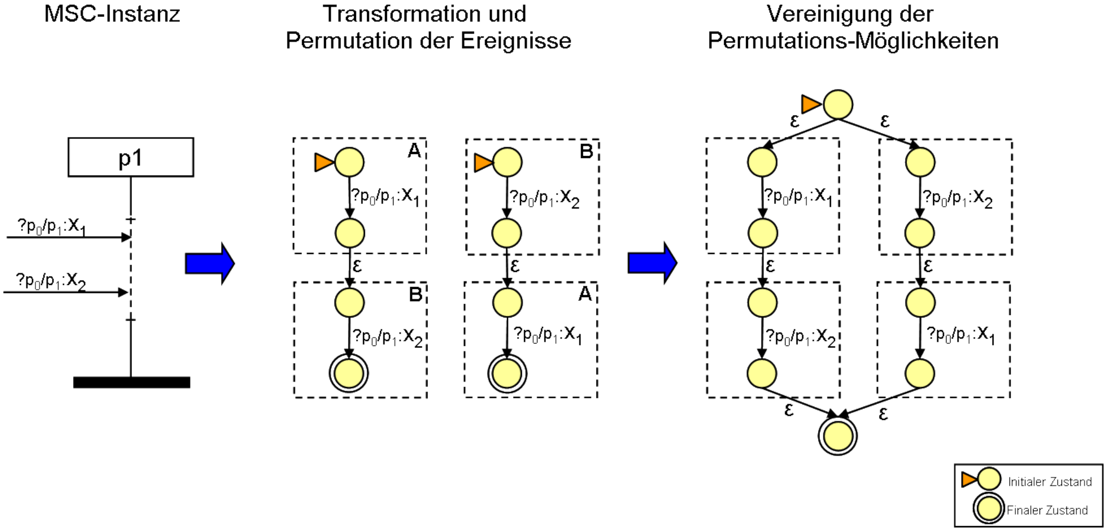


Abbildung 7.25: Transformation von Coregion

7.4.4 Determinierung der Automaten

Voraussetzung für die Anwendung der heuristischen Vergleichsmethode, die in Kapitel 7.4.5 beschrieben wird, sind deterministische Automaten. In der Automatentheorie sind z.B. in [74] Methoden zur Determinierung von endlichen Automaten beschrieben. Dies erfolgt in zwei Schritten:

1. **ϵ-Eliminierung**
   Entfernung aller ϵ-Transitionen. Somit wird der ϵ-NEA in einen NEA überführt.

2. **Determinierung**
   Umwandlung von NEA in DEA.

Ein positiver Nebeneffekt bei der Erzeugung deterministischer Automaten ist, dass durch die ϵ-Eliminierung die Komplexität bzw. die Größe der Automaten verringert bzw. verkleinert werden kann. Die beiden Methoden sind in der Automatentheorie beschrieben und daher wird an dieser Stelle auf eine weitere Ausführung verzichtet.
In der Automatentheorie wurde auch ein Verfahren zur vollständigen Reduktion von endlichen deterministischen Automaten definiert [74]. Vollständige reduzierte Automaten besitzen die minimal mögliche Anzahl an Zuständen. Dies wird erreicht, indem auf

DEAs die Minimierung [74] angewendet wird. In [2][9] wird die Laufzeit von verschiedenen Minimierungs-Verfahren bewertet. Die Komplexität der Hopcroft-Minimierung [74] beträgt z.B. $O(n\ log\ n)$. Für den CompA-Ansatz ist die Minimierung jedoch nicht zwingend notwendig.

7.4.5 Kompatibilitätsanalyse

In den vorhergehenden Abschnitten wurden Transformationsregeln und die Determinierung beschrieben, mit Hilfe derer aus MSCs deterministische Automaten erzeugt werden. Ob zwei MSCs ein zueinander rückwärtskompatibles Verhalten beschreiben, wird auf Basis der deterministischen Automaten analysiert.

In diesem Abschnitt wird nun die Methode beschrieben, mit der zwei deterministische Automaten auf Rückwärtskompatibilität verglichen werden können. In Abbildung 7.26 sind zwei MSCs (MSC_1 und MSC_2) dargestellt, die in korrespondierende deterministische Automaten (A_1, A_2) umgewandelt worden sind. Da jedes MSC nur eine Instanz besitzt, wird jeweils auch nur ein Automat erzeugt: $MSC_1 \rightarrow A_1$ und $MSC_2 \rightarrow A_2$.

Bevor nun auf die Details der Vergleichsmethode an sich eingegangen wird, wird im nächsten Abschnitt die Definition für Rückwärtskompatibilität von MSCs auf die transformierten Automaten übertragen.

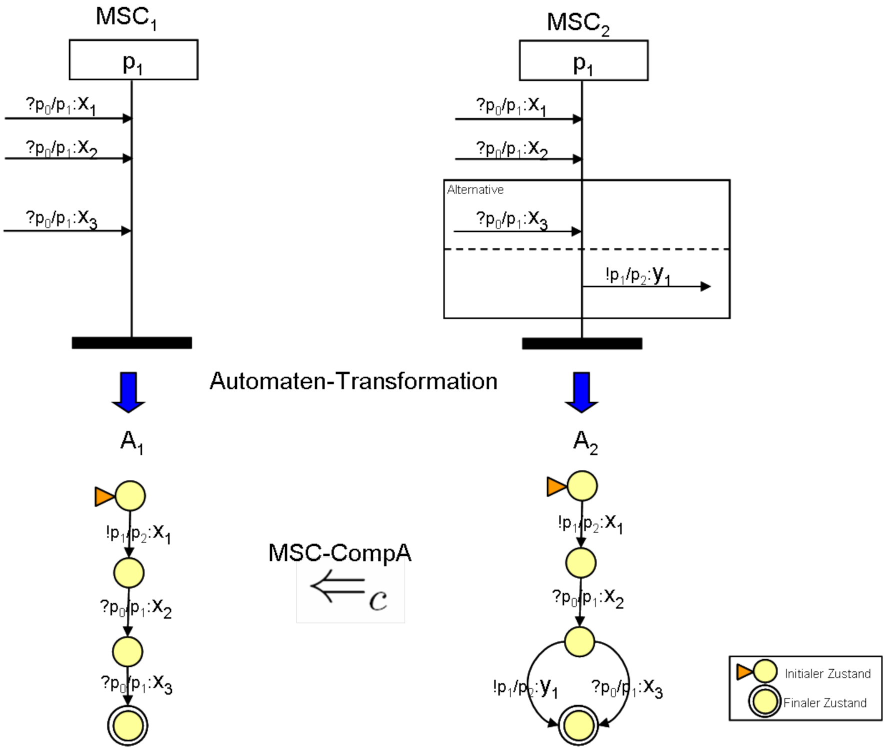


Abbildung 7.26: Rückwärtskompatibilitäts-Analyse auf Basis von Automaten

Erweiterung Kompatibilitätsdefinition auf Automaten

In Definition 6 (Kapitel 7.2) wurde Rückwärtskompatibilität von MSCs definiert durch:
$(MSC_1 \Leftarrow_c MSC_2) \Leftrightarrow (S_1 \subseteq S_2 \wedge E_1 \Leftarrow_c E_2 \wedge P_1 \Leftarrow_c P_2)$
Da im Rahmen von CompA die Rückwärtskompatibilität von MSCs auf Basis von Automaten untersucht wird, wird die Definition 6 auf Automaten erweitert. Hierzu werden die Transformationen genauer betrachtet.
Bei der Transformation wird für jede Instanz eines MSCs ein Automat erzeugt. Die Menge der Ereignissequenzen S_1 eines MSC_1 kann sich also aus mehreren Automaten zusammensetzen. Um $S_1 \subseteq S_2$ zu prüfen, muss die Menge aller Automaten M_1 von MSC_1 mit der Menge M_2 von MSC_2 verglichen werden. Damit kann Definition 6 für Automaten wie folgt angepasst werden:

Definition 8 *MSC_2 ist genau dann zu MSC_1 rückwärtskompatibel, wenn die Menge aller Automaten M_1 von MSC_1 eine echte Teilmenge der Menge der Automaten M_2 von MSC_2 darstellen, wobei die Übergangsfunktionen der Automaten rückwärtskompatibel sein dürfen.*
$(MSC_1 \Leftarrow_c MSC_2) \Leftrightarrow ((M_1 \subseteq M_2)\ mit\ \delta_{M_1} \Leftarrow_c \delta_{M_2})$

Die Rückwärtskompatibilität bzgl. der Menge der Instanzen $P_1 \Leftarrow_c P_2$ muss hier nicht weiter explizit betrachtet werden, da die Nachbarschaftsbeziehungen $o : E \rightarrow P$ bei der Dekomposition auf die Ereignisse und damit bei der Transformation auf die Übergangsfunktionen übertragen wurden.
Um die Definition 8 zu prüfen, müssen die Automaten der einzelnen Mengen miteinander verglichen werden. Die Rückwärtskompatibilität zweier Automaten wird wie folgt definiert:

Definition 9 *Zwei Automaten A_1 und A_2 sind genau dann zueinander rückwärtskompatibel, wenn der Automat A_1 eine echte Teilmenge von A_2 darstellt ($A_1 \subseteq A_2$) und wenn die zugehörigen Übergangsfunktionen zueinander rückwärtskompatibel sind.*
$(A_1 \Leftarrow_c A_2) \Leftrightarrow (A_1 \subseteq A_2\ mit\ \delta_{A_1} \Leftarrow_c \delta_{A_2})$

Auf Basis dieser Definitionen wird im folgenden Abschnitt die Vergleichsmethode definiert.

Vergleichsmethode

Zum Vergleich von Automaten wurde im Rahmen von CompA ein heuristischer Vergleichsansatz definiert. Es existieren zwar bereits mächtige kommerzielle Model-Checking-Tools, diese sind jedoch zum einen sehr teuer und zum anderen können diese Tools keine Kompatibilitäts-Ergebnisse, die im Rahmen von *XML-CompA* generiert werden, berücksichtigen.
Aufbauend auf den Definitionen 8 und 9 muss die, zum Nachweis von Rückwärtskompatibilität, definierte Heuristik folgende Punkte analysieren können:

- Die Menge M_1 der endlichen Automaten von MSC_1 muss eine echte Teilmenge von M_2, der Menge der endlichen Automaten von MSC_2 sein.

- Die Übergangsfunktionen $\delta(M_1)$ der Menge der Automaten M_1 muss rückwärtskompatibel zu den Übergangsfunktionen $\delta(M_2)$ der Menge der Automaten M_2 sein.

Bevor auf die Analyse der Teilmengeneigenschaft eingegangen wird, wird die Analyse der Rückwärtskompatibilität der Übergangsfunktionen betrachtet.

Analyse der Ereignisse auf Kompatibilität Zwei MSC-Messages können kompatibel sein, wenn diese sich, im einfachsten Fall, nur im Namen unterscheiden. Ob diese Signale zueinander rückwärtskompatibel sind oder nicht, kann nur durch Wissen von Experten entschieden werden. Daher wurde in die Vergleichsmethode die Möglichkeit integriert, über eine Kompatibilitäts-Tabelle (vgl. Tabelle 7.1) die Messages zu definieren, die zueinander rückwärtskompatibel sind. In Tabelle 7.1 ist so beispielsweise festgelegt, dass Message x_2 zu Message x_1 und Message x_3 zu Message x_1 rückwärtskompatibel ist.

Ereignisse/Messages von SG_1	**Ereignisse/Messages von SG_2**
x_1	x_2
x_1	x_3

Tabelle 7.1: Kompatibilitätstabelle für MSC-Ereignisse

Ausblick: Im Projekt „*XML-CompA*" wurde Rückwärtskompatibilität von Signalen geprüft. Das Ergebnis dieser Prüfung wird ebenfalls in dieser Tabelle abgelegt werden und kann somit für *MSC-CompA* verwendet werden.

Analyse des deterministischen Automaten auf Teilmengen-Eigenschaft der Ereignissequenzen In diesem Schritt wird geprüft, ob die Menge der Automaten M_1 eine vollständige Teilmenge der Automaten M_2 ist. Hierzu werden alle Automaten von M_1 einzeln mit allen Automaten von M_2 verglichen. Der Vergleich ist in Algorithmus[3] 13 und 14 beschrieben und wird an den Automaten aus Abbildung 7.26 erklärt.

Gegeben sind zwei Automaten A_1 und A_2, die bzgl. Rückwärtskompatibilität verglichen werden sollen. Die Automaten werden ausgehend vom *initialen Zustand* parallel traversiert. Hierbei werden sowohl die Zustände als auch die Übergangsfunktionen miteinander verglichen. Bei dem Vergleich der Zustände ist wichtig, dass, wenn im Automat A_1 ein *finaler Zustand* erreicht wird, dieser auch im Automaten A_2 erreicht werden muss.

Beim Vergleich von zwei Übergangsfunktionen werden zuerst die Präfixe (hier: $?p_1/p_2$:) miteinander verglichen. Diese müssen äquivalent sein. Anschließend werden die Ereignisse verglichen. Diese müssen entweder äquivalent oder in der Kompatibilitäts-Tabelle (vgl. Tabelle 7.1) als rückwärtskompatibel deklariert sein.

---

[3] Die Kompatibilitäts-Tabelle wurde der Einfachheit halber im Algorithmus nicht berücksichtigt.

Die traversierten Zustände von A_1 und A_2 werden in Tupel als bereits traversiert markiert. Dadurch kann die Traversierung abgebrochen werden, wenn ein markiertes Tupel, nochmal traversiert werden sollte. Auf diese Weise lassen sich Schleifen erkennen. Die parallele Traversierung der Automaten läuft so lange, bis der Automat A_1 komplett traversiert wurde oder eine Inkompatibilität auftritt.

Algorithm 13 CompA_Vergleich_von_Prozessen(A, B)

Input: $A = (Q_A, \Sigma_A, \delta_A, q_{A_0}, F_A)$; $B = (Q_B, \Sigma_B, \delta_B, q_{B_0}, F_B)$ // Automaten A und B

```
if ($q_{A_0} \in F_A \wedge q_{B_0} \notin F_B$) then
    print „Automat $B$ ist nicht rückwärtskompatibel zu Automat $A$, d.h. $A \nLeftarrow_c B$“
    exit;
else
    $search\_\delta(q_{A_0}, q_{B_0}, A, B)$;
    // Falls Programm bis hierher durchlaufen, dann $A \Leftarrow_c B$
    print „Automat $B$ ist rückwärtskompatibel zu Automat $A$, d.h. $A \Leftarrow_c B$“
end if
```

7.5 Beispiel für Transformation eines MSC in einen Automaten

Beispiel 1: In Abbildung 7.27 ist ein MSC dargestellt, das aus 6 Ereignissen und einem Optional-Operator, in dem ein Alternative-Operator mit 2 Sections verschachtelt ist, besteht. Das MSC wird mit Hilfe der in Abschnitt 7.4.3 definierten Transformationsregeln in einen endlichen Automaten transformiert. Hierzu werden die einzelnen Ereignisse entsprechend transformiert, konkateniert und vereinigt. Das Ergebnis ist ein ϵ-NEA. Anschließend wird dieser Automat determiniert. Hierzu werden im 1.Schritt durch die ϵ-Eliminierung die ϵ-Übergänge entfernt und im 2. Schritt wird der Automat determiniert.

Beispiel 2: Das gleiche Vorgehen wird mit dem Automaten aus Abbildung 7.29 wiederholt. Das MSC besteht zwar ebenfalls aus 6 Ereignissen, aber aus einem Alternative-Operator mit 3 Sections, wobei eine dieser Sections leer ist. Wird das MSC mit Hilfe der Transformationsregeln in einen Automaten transformiert, so entsteht ein ϵ-NEA. Die rote gestrichelte Linie kennzeichnet dabei den korrespondierenden Alternative-Operator. Der ϵ-NEA wird nun mit Hilfe der Methoden ϵ-Eliminierung und Determinierung in einen DEA überführt (vgl. Abbildung 7.30).

Algorithm 14 $search\_\delta(q_A, q_B, A, B)$

```
Input: $(A, q_A)$ // $Automat A$ und aktueller Zustand; $(B, q_B)$ // $Automat B$ und aktueller Zustand
Output: Aussage, ob Ereignissequenzen von Automat A und Automat B kompatibel;
for $i = 1$ to $Anzahl\_\delta(q_A)$ do
    $x = get\_\Sigma(q_A, i)$;// liefert sequentiell das Eingabealphabet für $q_A$
    $y = search\_for\_\Sigma(q_B, x)$;// Prüfe, ob für Alphabet $x \in \Sigma(q_A)$ auch gilt $x \in \Sigma(q_B)$; Ergebnis TRUE/FALSE
    if (y=FALSE) then
        print „Automat $B$ ist nicht rückwärtskompatibel zu Automat $A$, d.h. $A \nLeftarrow_c B$“
        exit;
    end if
    $q_{A+1} = \delta(q_A, x)$;
    $q_{B+1} = \delta(q_B, x)$;
    if $(q_{A+1} \in F_A) \wedge (q_{B+1} \notin F_B)$ then
        print „Automat $B$ ist nicht rückwärtskompatibel zu Automat $A$, d.h. $A \nLeftarrow_c B$“
        exit;
    else if $(q_{A+1}, q_{B+1}) = markiert$ then
        return
    else
        $Markiere(q_{A+1}, q_{B+1})$; // Markiere Tupel, als bereits besucht
        $search\_\delta(q_A, q_B, A, B)$;
    end if
end for
return
```

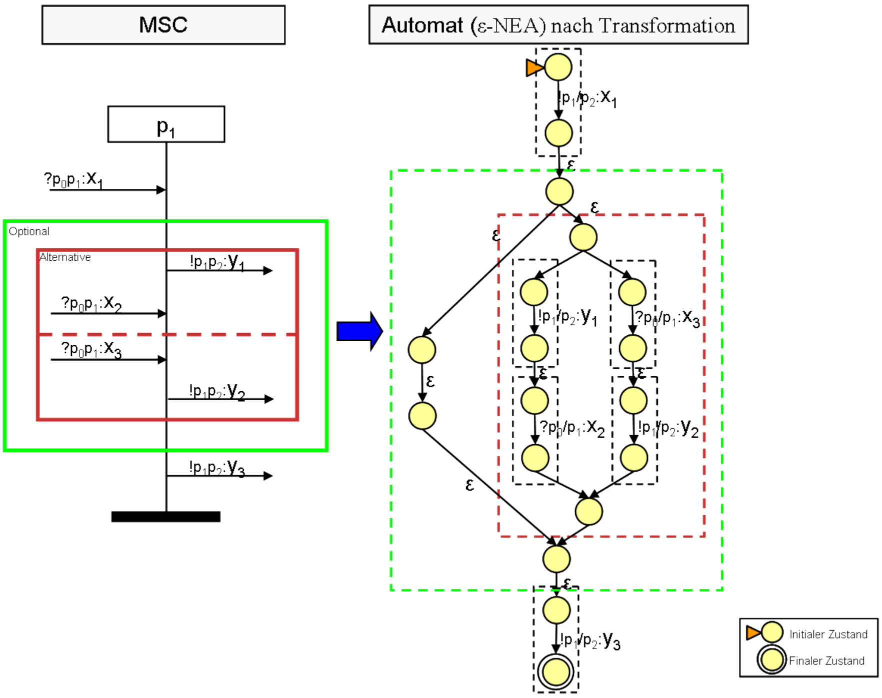


Abbildung 7.27: **Beispiel 1:** Automaten-Transformation

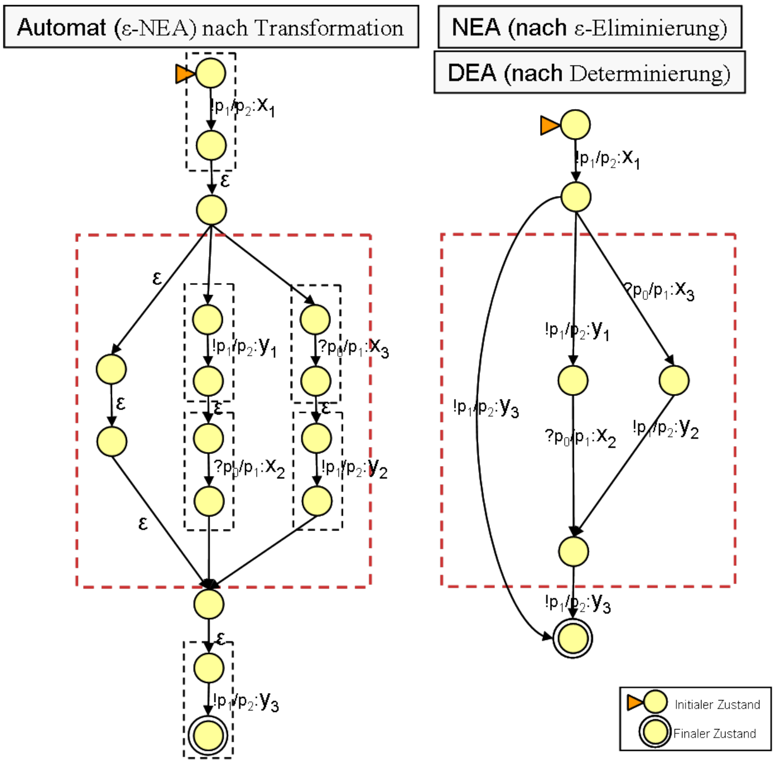


Abbildung 7.30: **Beispiel 2:** Determinierung des Automaten

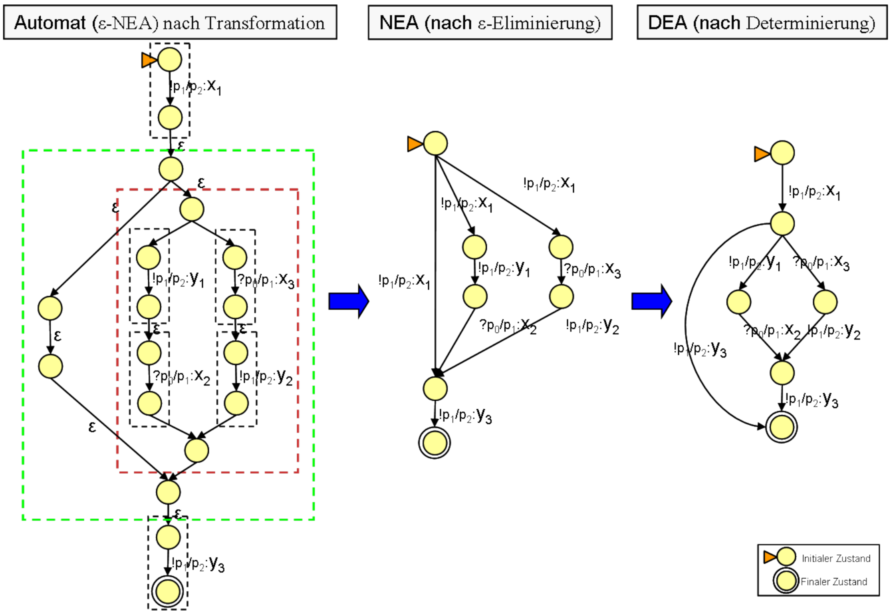


Abbildung 7.28: **Beispiel 1:** Determinierung

Vergleicht man die beiden deterministischen Automaten aus **Beispiel 1** und **Beispiel 2**, so zeigt sich, dass diese äquivalent sind. In Abbildung 7.31 sind die beiden MSCs und die deterministischen Automaten gegenübergestellt. Obwohl die MSCs eine unterschiedliche Darstellung besitzen, beschreiben diese das gleiche dynamische Verhalten, wie die Äquivalenz der beiden Automaten beweist.

7.6 Zusammenfassung

In diesem Kapitel wurde das Konzept der MSC-Vergleichsmethode (*MSC-CompA*) beschrieben. Das Konzept basiert auf einer Transformation von MSC-Sprachkonstrukten in endliche Automaten, die anschließend auf Rückwärtskompatibilität geprüft werden. Hierfür wurde die Definition von Rückwärtskompatibilität von Kapitel 3 auf MSCs übertragen und anschließend auf endliche Automaten erweitert. Schwerpunkt dieses Kapitels waren die Transformationsregeln, mit Hilfe derer beliebige MSC-Konstrukte in endliche Automaten überführt werden können. Die Regeln wurden dabei so konstruiert, dass auch verschachtelte Inline-Expressions transformiert werden können. Alle Transformationsregeln wurden detailliert erläutert und z.T. durch Algorithmen ergänzt. Ferner wurde in diesem Kapitel ein heuristischer Vergleichsansatz beschrieben. Mit Hilfe dieses Ansatzes können deterministische Automaten verglichen und Aussagen bzgl. deren Rückwärtskompatibilität getroffen werden.

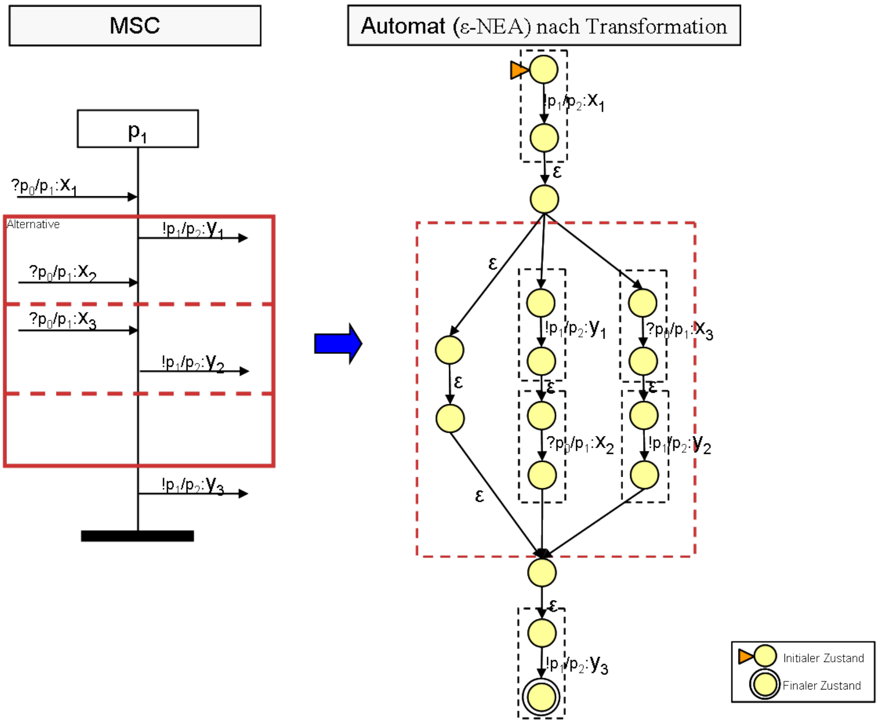


Abbildung 7.29: **Beispiel 2:** Automaten-Transformation

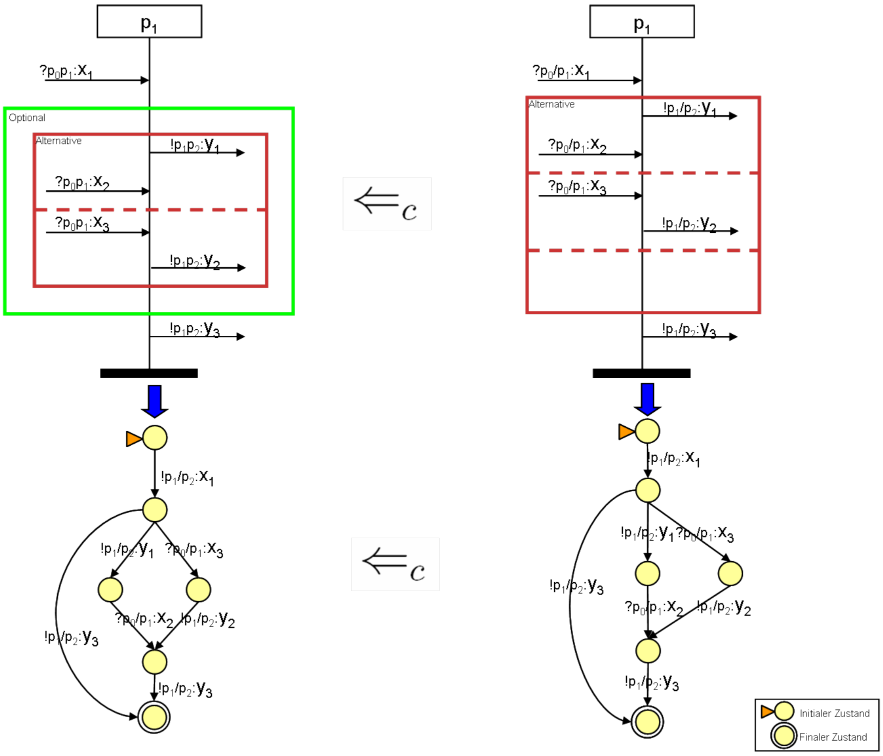


Abbildung 7.31: Gegenüberstellung des deterministischen Automaten aus **Beispiel 1** und **Beispiel 2**

8 Implementierung der Methodik

Die in dieser Arbeit entwickelten Konzepte wurden als Software-Module (SW-Module) auf der Software-Plattform *CAMP (Common Application Module Platform)* implementiert. In den folgenden Abschnitten werden die Plattform und die SW-Module genauer erläutert:

- *Software-Plattform CAMP: CAMP* stellt eine Integrations-Plattform zur Verfügung, auf der SW-Module entwickelt und ausgeführt werden können (s. Abschnitt 8.1).
- *SG-Editor:* Das Modul SG-Editor (Steuergeräte-Editor) ist eine graphische Bedienoberfläche (GUI) zur Spezifikation von Steuergeräten. In Abschnitt 8.2 wird sowohl das GUI-Konzept für den SG-Editor behandelt, als auch kurz auf dessen Implementierung eingegangen.
- *XML-CompA:* Das Modul *XML-CompA* setzt die XML-Vergleichsmethode aus Kapitel 6 um (s. Abschnitt 8.3).
- *MSC-CompA:* Das Modul *MSC-CompA* setzt die MSC-Vergleichsmethode aus Kapitel 7 um (s. Abschnitt 8.4).

8.1 Zielplattform CAMP

CAMP (Common Application Module Platform) ist eine auf Java[1] entwickelte Software-Plattform, auf deren Basis Software-Module entwickelt und als Module bzw. Plug-Ins ausgeführt werden können. Die Plattform wurde nach dem Model-View-Kontroller Konzept [50] an der TU-Paderborn und der TU-Chemnitz entwickelt [126]. CAMP wird u.a. in dem Forschungsprojekt IPQ [136][137] an der TU-Chemnitz eingesetzt. Ziel von CAMP ist es, eine Integrationsplattform für SW-Module zur Verfügung zu stellen, so dass diese miteinander agieren können. Hierfür stellt CAMP verschiedene Basis-Funktionen bereit:

- grundlegende GUI-Struktur
- Implementierung von GUI-Events (bspw. Klicks)
- Basisfunktionalität zum Anlegen, Öffnen, Speichern und Schließen von Dateien

---

[1] CAMP-Implementierung wurde im Rahmen von CompA auf Java 1.6 portiert

- weitgehende Konfigurationsmöglichkeiten bezüglich Abhängigkeiten zwischen Modulen
- Schnittstelle zur Bereitstellung von Diensten zwischen verschiedenen Software-Modulen

Im Rahmen des IPQ bzw. CAMP-Projektes wurden zwei Module (XML-API, XML-Editor) entwickelt, auf die auch im Rahmen von CompA zurückgegriffen wird.

XML-API Die XML-API [126] integriert mehrere APIs, indem eine weitere Software-Schicht über vorhandene APIs gelegt wird. Mit Hilfe dieser Software-Schicht werden vereinfachte Funktionsaufrufe zur Verfügung gestellt, mit denen XML-Instanzen eingelesen, traversiert und bearbeitet werden können. Die XML-API integriert Xerces [5] als XML Parser, um XML-Instanzen einzulesen. Xalan [4] wird eingesetzt, um die XPath Funktionalitäten zu integrieren und um die DOM Struktur der XML-Instanzen im Hauptspeicher zu verarbeiten. Zudem wird die XSD-API mit eingebunden, um der XML-API auch Funktionalitäten zur XML Schemaverarbeitung zu ermöglichen.

XML-Editor Ziel des XML-Editors ist, dass durch Hinterlegen eines XML-Schemas eine XML-Instanz durch „Drag & Drop“ entsprechend des Schemas erstellt werden kann [45]. Hierfür werden grundlegende Operationen zur Manipulation von XML-Instanzen zur Verfügung gestellt. Diese werden auch von den *CompA*-Modulen (z.B. SG-Editor) genutzt.

8.2 SG-Editor

Im Rahmen von CompA werden Steuergeräte auf Basis eines XML-SG-Schemas spezifiziert. Hierzu wird für jedes Steuergerät eine eigene XML-SG-Instanz erstellt. Zur Erstellung von XML-SG-Instanzen gibt es eine Vielzahl an unterschiedlichen XML-Editoren wie den *XML-Editor* von CAMP oder den Editoren von kommerziellen Tools wie z.B. von Altova [146]. Diese XML-Editoren besitzen zwar einen sehr mächtigen Funktionsumfang, eignen sich jedoch für eine intuitive Erstellung von Steuergeräte-Spezifikationen nur bedingt. Daher wurde zur Spezifikation von Steuergeräten ein spezifische graphische Bedienoberfläche (GUI) definiert und als Modul „*SG-Editor*“ in CAMP integriert. Hierbei wurden folgende Punkte fokussiert:

- *einfache und intuitive Bedienbarkeit*
- *Erstellung der Instanzen auf Basis des XML-SG-Schemas:* Der Nutzer wird durch das XML-SG-Schema bei der Erstellung der XML-SG-Instanzen „geführt“, d.h. das XML-SG-Schema gibt den Rahmen vor und die XML-SG-Instanzen werden direkt daraus abgeleitet.

- *Befüllung der XML-SG-Instanzen über Tabellen:* Bei der Erstellung von XML-SG-Instanzen kommt es häufig vor, dass Elemente mehrfach instanziiert werden z.B. Pins. Hier ist es sinnvoll, dass die mehrfach instanziierten Pfade, die auf dem gleichen Schema-Pfad beruhen, in einer Tabelle dargestellt werden.

- *Generischer Aufbau:* Der Aufbau eines XML-SG-Schemas kann je nach Anwendungsgebiet unterschiedlich sein. Die GUI muss also generisch aufgebaut sein, so dass beliebige XML-SG-Schemata verwendet werden können.

Das Modul *SG-Editor* stellt für die GUI drei verschiedene Eingabe-Oberflächen zur Verfügung (vgl. Abbildung 8.1):

- ***Graphische Notation:*** Diese Ebene dient einer abstrakten graphischen Strukturierung (vgl. Abschnitt 8.2) und spiegelt zum Teil das Sichtenkonzept wieder.

- ***Tabellarische Notation:*** Diese Ebene stellt eine Detaillierung der graphischen Notation dar und dient vorrangig dem Befüllen einer XML-SG-Instanz mit Werten.

- ***Relationen-Notation:*** Diese Ebene dient der Spezifikation von internen Verweisen bzw. Abhängigkeiten (ID und IDref).

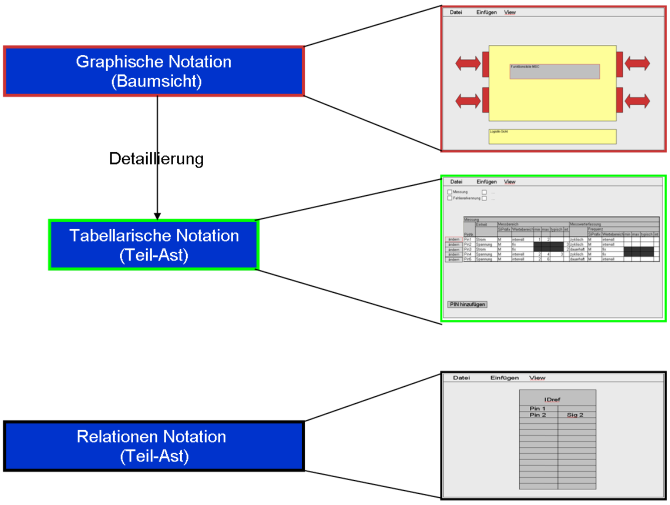


Abbildung 8.1: Ebenen-Modell der graphischen Benutzeroberfläche (GUI)

In den nächsten Abschnitten wird eine kurze Übersicht zu den Leistungsmerkmalen des GUI-Konzeptes gegeben. Für weitere Informationen zur Implementierung sei auf [83] verwiesen.

Graphische Notation

Die graphische Notation dient als Einstiegs-Ebene bzw. zur anschaulichen Strukturierung der obersten Schema-Ebene. Hier können verschiedene XML-Pfade graphisch repräsentiert werden. In Abbildung 8.2 ist links das XML-SG-Schema mit den obersten Ebenen abgebildet. Rechts daneben ist die graphische Notation dargestellt. Die Graphische Notation stellt ein Steuergerät mit Interfaces, Signalen, etc. dar. Die Pfeile deuten den Bezug zwischen XML-Schema-Elementen und der visuellen Darstellung an. So ist beispielsweise der XML-Schema-Pfad $Interface$ durch einen roten Block repräsentiert.

Auf der graphischen Notations-Ebene können weitere Elemente hinzugefügt werden. Durch das Hinzufügen weiterer Blöcke werden die entsprechenden Pfade auch im Instanzdokument instanziiert. Durch Doppelklick auf das Interface in der graphischen Darstellung (roter Block) öffnet sich die zugehörige tabellarische Darstellung.

Alle im SG-Editor zu visualisierenden Elemente müssen im XML-SG-Schema markiert werden. Die Markieren-Semantik wird dabei über das Attribut GUI realisiert, welche einen Wert aus dem folgenden Tupel annehmen kann: ’interface’, ’signalblock’, ’fct_msc’, ’fct_ascet’, ’fct_simulink’, ’logistic’, ’dependencies’, ’hardware’, ’geometrie’. Anhand des gesetzten Attributwertes kann der Editor entscheiden, welche Schema-Bäume in Verbindung zu welchem visuellen Element gesetzt werden müssen. Für künftige Anwendungen könnte der Editor erweitert werden, so dass der Nutzer selbst die Zuordnungen und die Definition der visuellen Eigenschaften der graphischen Elemente definieren kann.

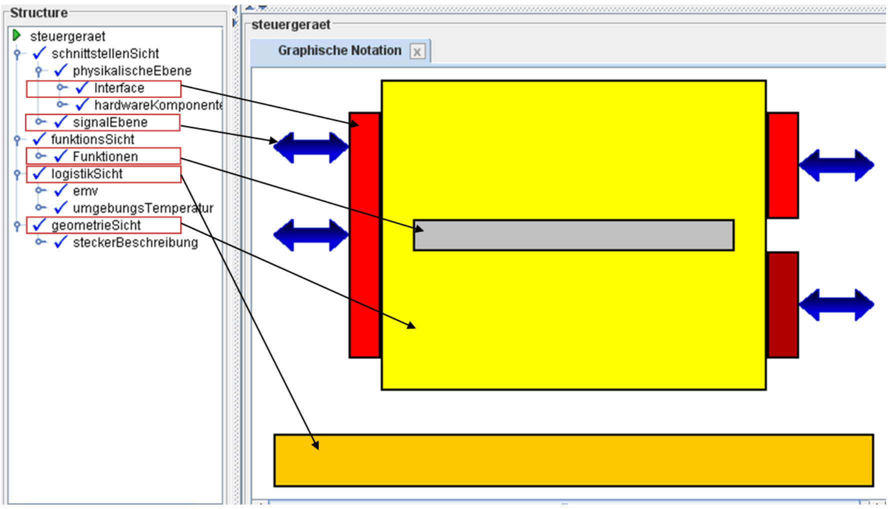


Abbildung 8.2: Aufbau der graphischen Notation

Tabellarische Notation

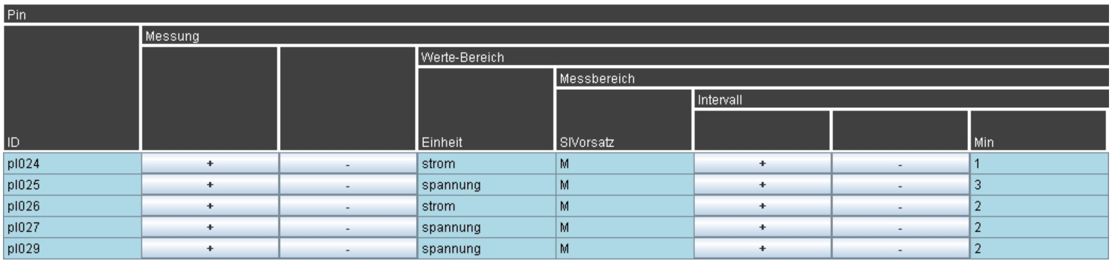
Im XML-SG-Schema sind viele, der zu spezifizierende Elemente, vom Typ $min - /maxOccurs > 1$. Diese Elemente können (inkl. der zugehörigen Kind-Elemente) mehrfach in der XML-SG-Instanz auftreten. Zur Unterstützung dieses Falls eignet sich sehr gut eine tabellarische Notation.
Bei der tabellarischen Notation wird in x-Richtung der XML-Pfad entsprechend des XML-SG-Schemas dargestellt. Das Besondere bei diesem Ansatz ist, dass die hierarchische Struktur mit dargestellt wird. In y-Richtung werden die mehrfach angelegten XML-Pfade angelegt. In Listing 8.1 ist ein kleiner Ausschnitt aus einer XML-SG-Instanz zu sehen, der die Eigenschaft $Messung$ von Pins charakterisiert.

Listing 8.1: Ausschnitt aus Spezifikation für Messwerterfassung eines Pins

```
...
<Pin ID=”pI024”>
    ...
    <Messung>
        <Werte−Bereich CompAregel=”ISG1innerhalbSG2” Einheit=”
               strom”>
            <Messbereich SIVorsatz=”M”>
                <Intervall>
                    <Min>1</Min>
                     ...
                </Intervall>
            </Messbereich>
             .....
        </Werte−Bereich>
    </Messung>
</Pin>
...
```

In Abbildung 8.3 ist die zugehörige tabellarische Notation der Eigenschaft *Messung* von mehreren *Pins* dargestellt. Alle zu einem *Pin* gehörigen Kind-Elemente und deren hierarchische Strukturierung sind (analog zum Schema bzw. der Instanz) in der Kopfzeile dargestellt. Die oberste Ebene ist *Pin* mit dem Attribut *ID*. *Messung* ist ein Kind-Element von *Pin*. *Messung* gliedert sich wiederum in *Wertebereich* und dieser in *Messbereich*. Der *Messbereich* selbst wird durch ein *Intervall* spezifiziert. *Messung* ist als optional im Schema deklariert. Für diesen Fall sind in der tabellarischen Notation die Kästchen mit + und - vorgesehen. Mit Hilfe dieser kann definiert werden, ob ein optionales Element hinzugefügt oder aber wieder entfernt werden soll.
Der direkte Vergleich der beiden Darstellungen aus Listing 8.1 und Abbildung 8.3 zeigt anschaulich, dass die tabellarische Notation für eine Spezifikation erheblich übersichtlicher ist als der XML-Code.
Die tabellarische Notation bietet noch weitere Besonderheiten:

- $choice$-Elemente werden als Pull-Down-Auswahl dargestellt. Zusätzlich werden,



Pin							
	Messung						
			Werte-Bereich				
				Messbereich			
					Intervall		
ID			Einheit	SIVorsatz			Min
pl024	+	-	strom	M	+	-	1
pl025	+	-	spannung	M	+	-	3
pl026	+	-	strom	M	+	-	2
pl027	+	-	spannung	M	+	-	2
pl029	+	-	spannung	M	+	-	2

Abbildung 8.3: Aufbau der tabellarischen Notation

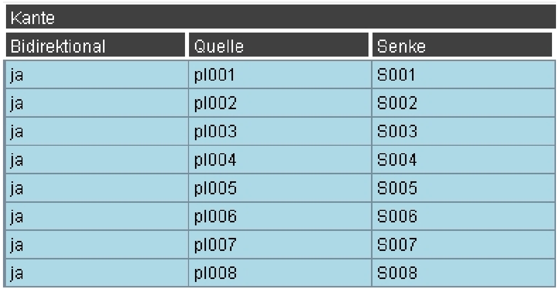
entsprechend des ausgewählten Elements, die nicht zu befüllenden Elemente ausgegraut.

- $restriction$-Elemente werden ebenfalls als Pull-Down-Auswahl dargestellt.
- $optionale$-Elemente können in der Tabelle als zu befüllende Elemente aktiviert werden.
- *kontinuierliche Validierung*. Alle eingegeben Werte werden auf Validität geprüft und es wird geprüft an welchen Stellen Elemente noch spezifiziert werden müssen. Die Felder werden mit „rot“ gekennzeichnet.

Algorithmus für tabellarische Darstellung Die graphische Benutzeroberfläche muss für alle nicht-rekursiven Schemata funktionieren. Daher wurde für den Aufbau der tabellarischen Notation folgender generischer Algorithmus definiert:
Zunächst wird das zugrundeliegende Schema für das zu betrachtende Element ausgelesen und als Tabellenkopf referenziert. Dieser Tabellenkopf besteht aus einer Datenstruktur, die jeder Spalte genau einen Knoten aus dem Schema zuordnet. Dies ist wichtig, um die Instanz später in der Tabelle positionieren zu können. Für jedes Unterelement mit der Beschränkung $maxOccurs > 1$ wird eine neue Zeile in die Tabelle eingefügt, wobei in jeder Zeile z ein XML-Element e referenziert wird mit $Tabelle(z) \rightarrow e$. Um nun die Transformation abzuschließen, müssen die Elemente im Subbaum des jeweiligen Zeilenelementes in ihre richtige Spaltenposition gebracht werden. Dies geschieht mit Hilfe des Tabellenkopfes, der das Schema-Element der aktuellen Spalte definiert. Zusammen mit dem XML-Element existiert damit eine eindeutige Abbildung: $Tabelle(zeile, spalte) \rightarrow Instanzelement$.

Relationen-Notation

XML-SG-Schema erlaubt auch interne Verweise mit *ID* und *ID-Ref* (vgl. Kapitel 2.2.3). Ein Beispiel hierfür ist der Verweis zwischen Signal und Pin, um zu kennzeichnen, welches Signal an welchem Pin anliegt. Für die Zuordnung der ID zu ID-Ref und umgekehrt, wurde die Relationen-Notation eingeführt. Um gerichtete Abhängigkeiten darzustellen, besteht die Möglichkeit, die ID oder die IDref als Quelle bzw. Senke zu spezifizieren. In Abbildung 8.4 ist die Tabelle angegeben, die ebenfalls generisch auf Basis des Schemas angelegt wird. In der ersten Zeilen ist eine Abhängigkeit zwischen einem Pin mit der ID *pI001* und einem Signal mit der ID *S001* definiert. Die Abhängigkeit ist ungerichtet bzw. bidirektional.



Kante

Bidirektional	Quelle	Senke
ja	pI001	S001
ja	pI002	S002
ja	pI003	S003
ja	pI004	S004
ja	pI005	S005
ja	pI006	S006
ja	pI007	S007
ja	pI008	S008

Abbildung 8.4: GUI zur Eingabe von Abhängigkeiten (Relationen-Notation)

8.3 XML-CompA

In diesem Abschnitt wird das Modul *XML-CompA* vorgestellt, welches die XML-Vergleichsmethode umsetzt. Das Modul wurde in Java programmiert und die zugehörige GUI setzt auf Java Swing[2] [92] auf. Die GUI unterteilt sich in vier Hauptbereiche (vgl. Abbildung 8.5).

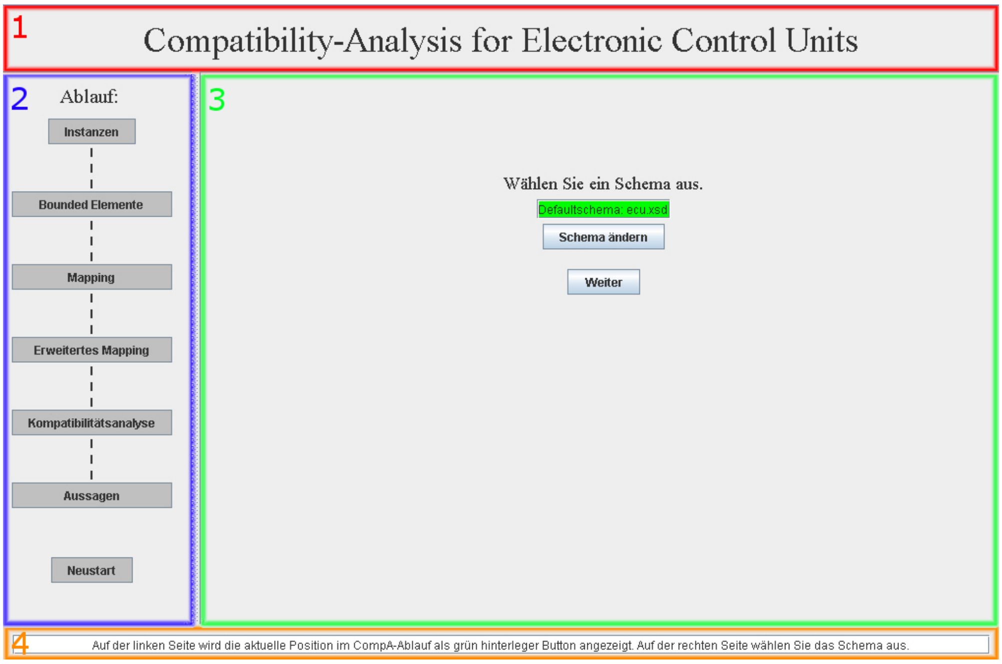
1. *Hauptüberschriften:* Anzeige des aktuellen Prozess-Schritts.
2. *Ablauf/Menü:* Ablauf bzw. Prozess der Kompatibilitätsanalyse. Die einzelnen Prozess-Schritte fungieren gleichzeitig als Menü. Durch Klicken auf einen Prozess-Schritt werden die Ergebnisse im Informationsbereich dargestellt (vgl. 3. Hauptbereich). Der aktuelle Prozess-Schritt wird „grün“ hinterlegt.
3. *Informationsbereich:* Eingabe von spezifischen Daten bzw. Ausgabe von Ergebnissen.
4. *Hilfe:* Anzeige von User-Informationen zu den Buttons.

Der Ablauf spiegelt die in Kapitel 6 beschriebene 4-Schritt-Methode wieder.

- Auswahl
- Mapping
- Kompatibilitätsanalyse

---

[2] Swing ist eine Programmierschnittstelle von Java, die zur Programmierung von graphischen Benutzeroberflächen eingesetzt wird.



Wählen Sie ein Schema aus.

Auf der linken Seite wird die aktuelle Position im CompA-Ablauf als grün hinterleger Button angezeigt. Auf der rechten Seite wählen Sie das Schema aus.

Abbildung 8.5: GUI für XML-Vergleichsmethode (*XML-CompA*)

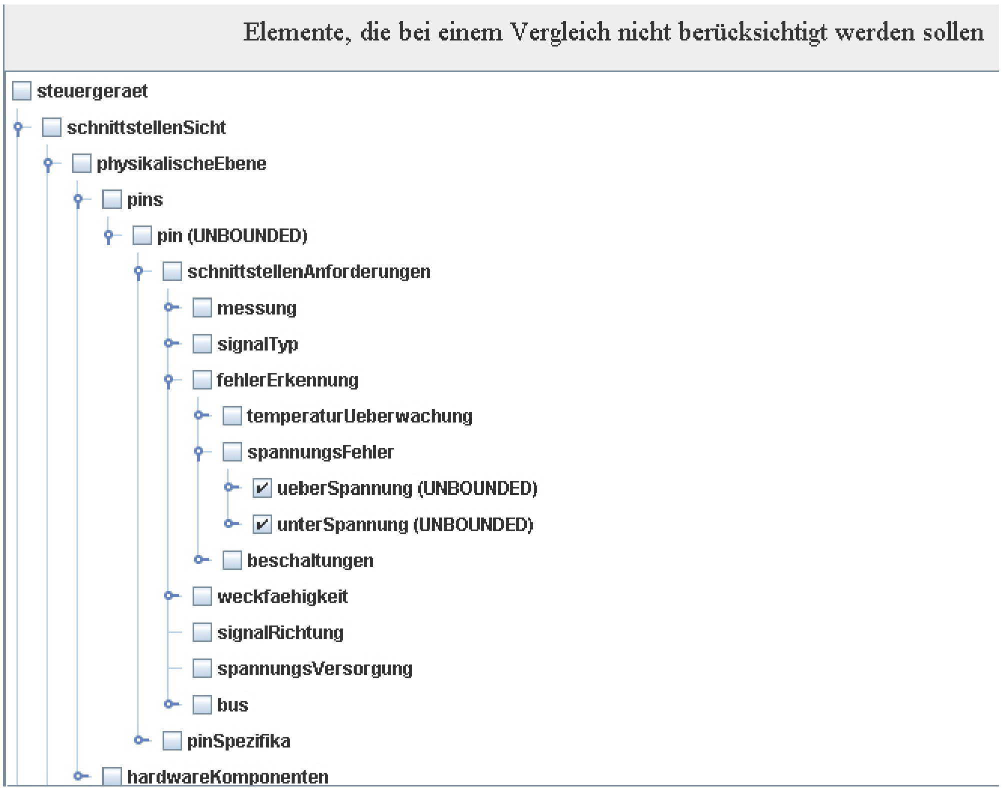
- Kompatibilitätsaussagen

In der Implementierung wurde jedoch die Kompatibilitätsanalyse von *bounded*- bzw. *ordered*-Elementen nach vorne gezogen, da bei diesen Elementen kein Mapping notwendig ist. Das Mapping gliedert sich in die Schritte *Mapping* und *erweitertes Mapping*. Dem Benutzer können die Zwischenergebnisse der einzelnen Mappingschritte ausgegeben werden.
In den nächsten Abschnitten wird kurz auf die 4-Hauptschritte eingegangen. Für detaillierte Informationen bzgl. der konkreten Implementierung sei auf [132] verwiesen.

Auswahl Nachdem der Benutzer das XML-SG-Schema, das den zu vergleichenden XML-SG-Instanzen zugrunde liegt, ausgewählt hat, wird das Schema geparst und anschließend dem Benutzer als *JTree* angezeigt (vgl. Abbildung 8.6). Hier kann die Entscheidung getroffen werden, welche Elemente bei der Kompatibilitätsanalyse nicht berücksichtigt werden sollen.

Mapping Nachdem das Mapping durchgeführt wurde, werden dem Benutzer die Ergebnisse angezeigt. Hierbei stehen dem Benutzer folgende Funktionalitäten zur Verfügung.

- Anzeige der Elemente, auf denen ein Mapping durchgeführt wurde (vgl. Abbildung 8.7 (links)).



Elemente, die bei einem Vergleich nicht berücksichtigt werden sollen

- ☐ steuergeraet
  - ☐ schnittstellenSicht
    - ☐ physikalischeEbene
      - ☐ pins
        - ☐ pin (UNBOUNDED)
          - ☐ schnittstellenAnforderungen
            - ☐ messung
            - ☐ signalTyp
            - ☐ fehlerErkennung
              - ☐ temperaturUeberwachung
              - ☐ spannungsFehler
                - ☑ ueberSpannung (UNBOUNDED)
                - ☑ unterSpannung (UNBOUNDED)
              - ☐ beschaltungen
            - ☐ weckfaehigkeit
            - ☐ signalRichtung
            - ☐ spannungsVersorgung
            - ☐ bus
          - ☐ pinSpezifika
      - ☐ hardwareKomponenten

Abbildung 8.6: Auswahl der Elemente, die bei der Kompatibilitätsanalyse nicht berücksichtigt werden sollen

- Anzeige Mappinggrade der gemappten Elemente (vgl. Kapitel 6.4.2).
- Manuelle Übersteuerung der automatisch generierten Mappinggrade. Hierfür kann der Mappinggrad mit Werten zwischen 0 und 100 verändert werden (vgl. Abbildung 8.7 (rechts)).

Kompatibilitätsanalyse Die Auswertung der Kompatibilitätsanalyse wird analog zu den Mappingergebnissen angezeigt (Abbildung 8.8). Zusätzlich werden neben dem Kompatibilitätsgrad (CompAgrad), der mit der Rückwärtskompatibilität einzelner Elemente korreliert, die Regeln angegeben, die zur Kompatibilitätsberechnung herangezogen wurden.

Kompatibilitätsaussagen Der letzte Hauptschritt ist die Generierung von Kompatibilitätsaussagen (vgl. Abbildung 8.9). Die Anzeigen werden hierbei durch farbliche Hintergründe unterstützt. Die Werte werden bei 100% *grün*, bei 99% bis 75% *gelb* und ansonsten *rot* hinterlegt. Folgende Aussagen werden getroffen, die in Kapitel 6.6 bereits detailliert erläutert wurden:
Mapping

Zuordnung: Element aus Steuergerät 1 zu Element aus Steuergerät 2

Steuergerät 1	Steuergerät 2	Mappinggrad
ID: pl00000001	ID: pl00000001	100 (Mit Enter bestätigen)
ID: pl00000001	ID: pl00000002	89.29 (Mit Enter bestätigen)
ID: pl00000001	ID: pl00000005	86.81 (Mit Enter bestätigen)
ID: pl00000001	ID: pl00000007	85.99 (Mit Enter bestätigen)
ID: pl00000001	ID: pl00000009	85.44 (Mit Enter bestätigen)
ID: pl00000001	ID: pl00000008	85.16 (Mit Enter bestätigen)
ID: pl00000001	ID: pl00000003	82.69 (Mit Enter bestätigen)
ID: pl00000001	ID: pl0000000A	47.45 (Mit Enter bestätigen)
ID: pl00000002	ID: pl00000002	100 (Mit Enter bestätigen)

Abbildung 8.7: Darstellung der Mapping-Ergebnisse

- *Mapping-Aussage[%]*
- *Mapping-Aussage*
- *Rechenzeit*

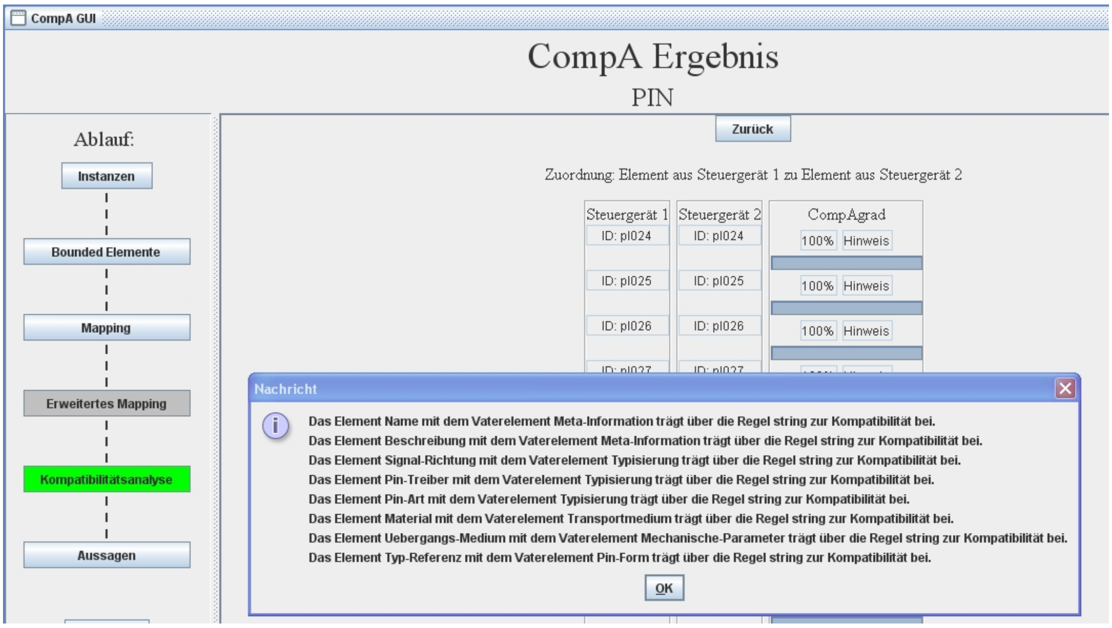
Rückwärtskompatibilität

- *Rückwärtskompatibilitäts-Aussage[%]*
- *Rückwärtskompatibilitäts-Aussage*
- *Rechenzeit*
- *Gesamt-Aussage (rückwärtskompatibel / nicht rückwärtskompatibel)*

Als zusätzliches Feature wurde ein **Äquivalenzcheck** integriert. Dieser besagt, wie viele der verglichenen Elemente äquivalent sind.

8.4 MSC-CompA

In dem Modul *MSC-CompA* ist die Vergleichsmethode für Message Sequence Charts (MSC) umgesetzt. Das Modul baut auf der gleichen SW-Struktur wie *XML-CompA* auf. In Abbildung 8.10 ist die Startseite von *MSC-CompA* dargestellt. Links befindet sich das Ablaufdiagramm der MSC-Vergleichsmethode, analog zu dem im Kapitel 7 beschriebenen Konzept. Die Details zur Implementierung sind in [93] beschrieben. An dieser Stelle seien die Hauptschritte der MSC-Vergleichsmethode herausgegriffen.



Steuergerät 1	Steuergerät 2	CompAgrad
ID: pl024	ID: pl024	100% Hinweis
ID: pl025	ID: pl025	100% Hinweis
ID: pl026	ID: pl026	100% Hinweis

Das Element Name mit dem Vaterelement Meta-Information trägt über die Regel string zur Kompatibilität bei.
Das Element Beschreibung mit dem Vaterelement Meta-Information trägt über die Regel string zur Kompatibilität bei.
Das Element Signal-Richtung mit dem Vaterelement Typisierung trägt über die Regel string zur Kompatibilität bei.
Das Element Pin-Treiber mit dem Vaterelement Typisierung trägt über die Regel string zur Kompatibilität bei.
Das Element Pin-Art mit dem Vaterelement Typisierung trägt über die Regel string zur Kompatibilität bei.
Das Element Material mit dem Vaterelement Transportmedium trägt über die Regel string zur Kompatibilität bei.
Das Element Uebergangs-Medium mit dem Vaterelement Mechanische-Parameter trägt über die Regel string zur Kompatibilität bei.
Das Element Typ-Referenz mit dem Vaterelement Pin-Form trägt über die Regel string zur Kompatibilität bei.

Abbildung 8.8: Anzeige der Kompatibilitätsanalyse

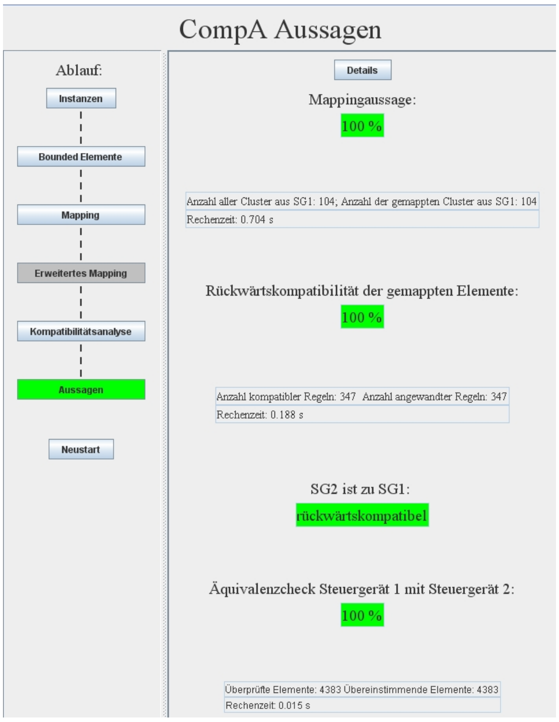
Transformation-XML Bevor die MSC-Vergleichsmethode startet, kann ausgewählt werden, ob MSCs als Set oder nur einzelne Szenarien von MSCs miteinander verglichen werden sollen. In Abbildung 8.11 ist die Auswahl für MSC-Szenarien dargestellt. In diesem Fall wird das Szenario „10_Events“ mit dem Szenario „10_Events_und_Timer_ohne_Constraints“ verglichen. Die ausgewählten MSCs bzw. MSC-Szenarien werden auf Basis des MSC-Metamodells (vgl. Kapitel 7.4.2) in eine XML-Darstellung überführt.

Transformation-EA Im Schritt *Transformation-EA* werden die in XML vorliegenden *Message Sequence Charts* in endliche Automaten (EA) transformiert. Das Ergebnis der Automaten-Transformation wird, aufgespalten nach MSC-Szenario, als Adjazenzliste dargestellt. In Abbildung 8.12 ist die Liste eines Szenarios(MSC_1) dargestellt. Der Automat ist unter TransferList zu finden. Zu lesen ist beispielsweise die erste Zeile der TransferList wie folgt: *1->!Instanz_1 Instanz_2:USERDEF.x_1->2*. Von Zustand *1* wird ein Signal *USERDEF.x_1* zu Zustand 2 gesendet. Dabei hat das Signal das Präfix[3] „*!Instanz_1 Instanz_2:*“. Nach der Transformation erfolgt die Determinierung der endlichen Automaten.

Aussagen Auf Basis von deterministischen Automaten setzt die MSC-Vergleichsmethode an. Nach dem Durchlauf der MSC-Vergleichsmethode werden die Kompatibilitätsaussagen generiert. In Abbildung 8.13 ist die Oberfläche der Ausgabe

---

[3] In der Implementierung ist der Trennungsstrich / zwischen den Instanzen durch ein Leerzeichen ersetzt.



Mappingaussage:
100 %

Anzahl aller Cluster aus SG1: 104; Anzahl der gemappten Cluster aus SG1: 104
Rechenzeit: 0.704 s

Rückwärtskompatibilität der gemappten Elemente:
100 %

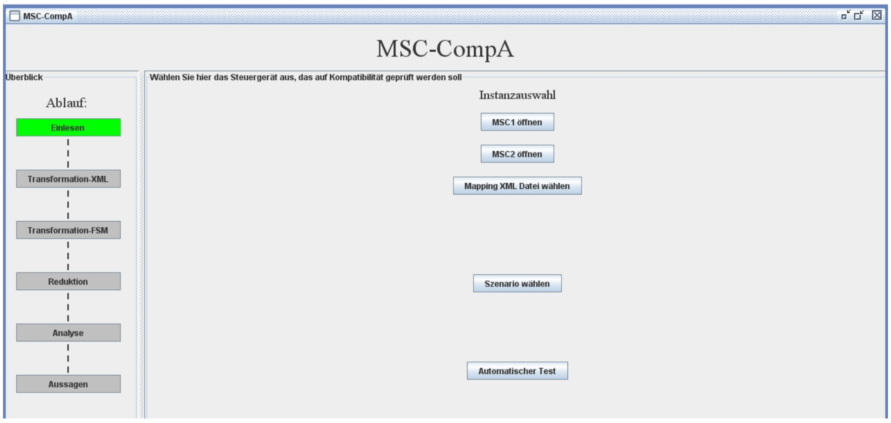
Anzahl kompatibler Regeln: 347 Anzahl angewandter Regeln: 347
Rechenzeit: 0.188 s

SG2 ist zu SG1:
rückwärtskompatibel

Äquivalenzcheck Steuergerät 1 mit Steuergerät 2:
100 %

Überprüfte Elemente: 4383 Übereinstimmende Elemente: 4383
Rechenzeit: 0.015 s

Abbildung 8.9: Anzeige der Kompatibilitätsaussagen



Wählen Sie hier das Steuergerät aus, das auf Kompatibilität geprüft werden soll

Abbildung 8.10: GUI für MSC-Vergleichsmethode (*MSC-CompA*)

dargestellt.
Als erstes wird für jeden Automat die Anzahl der Zustände nach

- der Transformation,
- der ϵ-Eliminierung und
- der Determinierung

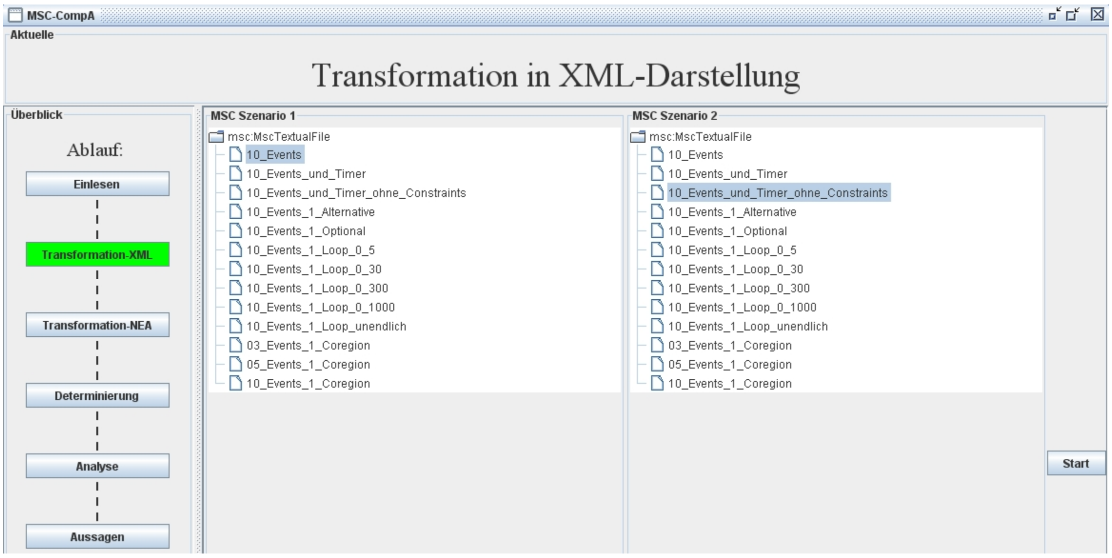
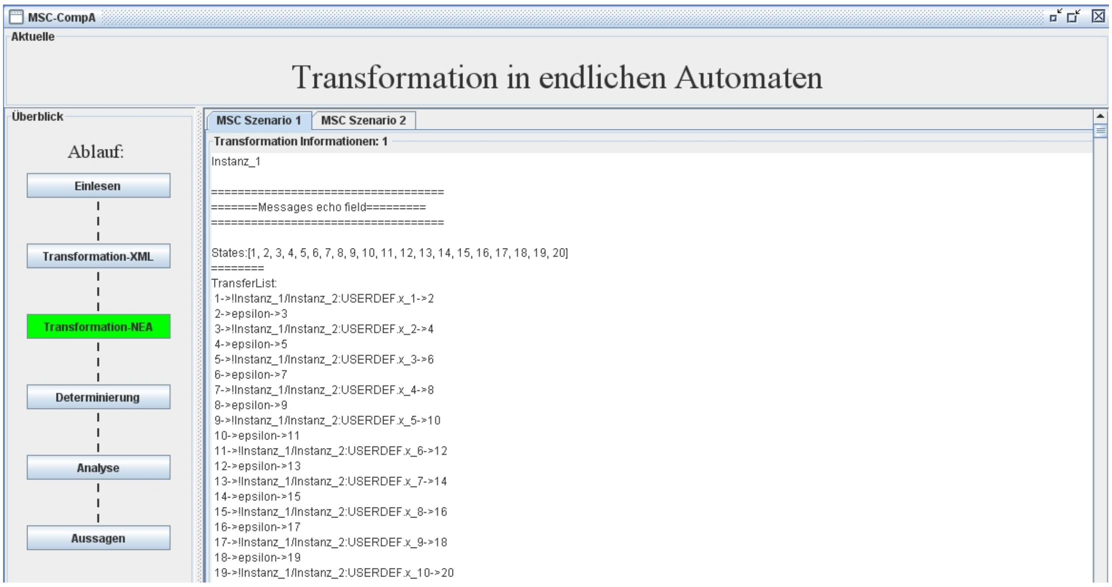


Abbildung 8.11: Auswahl der zu analysierenden MSC-Szenarien



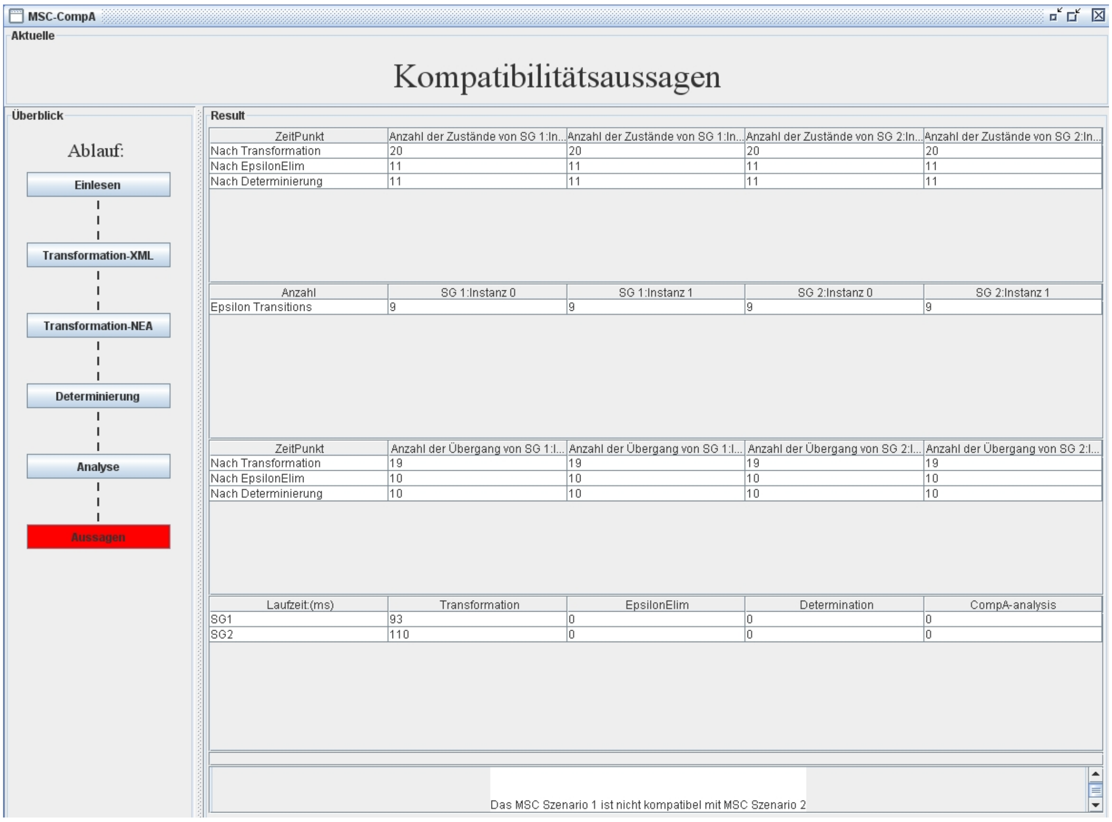
```
Instanz_1

====================================
=======Messages echo field=========
====================================

States:[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
========
TransferList:
 1->!Instanz_1/Instanz_2:USERDEF.x_1->2
 2->epsilon->3
 3->!Instanz_1/Instanz_2:USERDEF.x_2->4
 4->epsilon->5
 5->!Instanz_1/Instanz_2:USERDEF.x_3->6
 6->epsilon->7
 7->!Instanz_1/Instanz_2:USERDEF.x_4->8
 8->epsilon->9
 9->!Instanz_1/Instanz_2:USERDEF.x_5->10
 10->epsilon->11
 11->!Instanz_1/Instanz_2:USERDEF.x_6->12
 12->epsilon->13
 13->!Instanz_1/Instanz_2:USERDEF.x_7->14
 14->epsilon->15
 15->!Instanz_1/Instanz_2:USERDEF.x_8->16
 16->epsilon->17
 17->!Instanz_1/Instanz_2:USERDEF.x_9->18
 18->epsilon->19
 19->!Instanz_1/Instanz_2:USERDEF.x_10->20
```

Abbildung 8.12: Transformation-EA: Transformation in endliche Automaten (EA)

angegeben. Als zweites wird die Anzahl der ϵ-Übergänge und als drittes wird für jeden Automaten die Anzahl der Übergangsfunktionen, analog zu den Zuständen, angegeben. Bei den Übergangsfunktionen sind die ϵ-Übergänge mitgezählt. Jeder einzelne Automat wird dabei in einer Spalte dargestellt. Zusätzlich wird die Laufzeit bzw. Rechenzeit für die ϵ-Eliminierung und Determinierung ausgegeben. Abschließend wird die Aussage bzgl. der Rückwärtskompatibilität generiert. In diesem Fall sind die Ereignissequenzen von MSC_2 zu MSC_1 nicht rückwärtskompatibel.



ZeitPunkt	Anzahl der Zustände von SG 1:In...	Anzahl der Zustände von SG 1:In...	Anzahl der Zustände von SG 2:In...	Anzahl der Zustände von SG 2:In...
Nach Transformation	20	20	20	20
Nach EpsilonElim	11	11	11	11
Nach Determinierung	11	11	11	11

Anzahl	SG 1:Instanz 0	SG 1:Instanz 1	SG 2:Instanz 0	SG 2:Instanz 1
Epsilon Transitions	9	9	9	9

ZeitPunkt	Anzahl der Übergang von SG 1:I...	Anzahl der Übergang von SG 1:I...	Anzahl der Übergang von SG 2:I...	Anzahl der Übergang von SG 2:I...
Nach Transformation	19	19	19	19
Nach EpsilonElim	10	10	10	10
Nach Determinierung	10	10	10	10

Laufzeit:(ms)	Transformation	EpsilonElim	Determination	CompA-analysis
SG1	93	0	0	0
SG2	110	0	0	0

Das MSC Szenario 1 ist nicht kompatibel mit MSC Szenario 2

Abbildung 8.13: Auswahl der zu analysierenden MSC-Szenarien

8.5 Zusammenfassung

In diesem Kapitel wurden die im Rahmen von CompA entwickelten Tools bzw. Module (SG-Editor, *XML-CompA* und *MSC-CompA*) vorgestellt. Alle Module wurden für die Softwareumgebung CAMP entwickelt.

Das *Modul SG-Editor* stellt ein spezifisches Werkzeug zur Verfügung, mit dem XML-SG-Instanzen auf Basis eines beliebigen, nicht-rekursiven XML-SG-Metamodells beschrieben werden können. Hierfür wurden drei Oberflächen zur Verfügung gestellt: die graphische Notation, die tabellarische Notation und die Relationen-Notation.

Das *Modul XML-CompA* setzt die XML-Vergleichsmethode *XML-CompA* um. Das Tool kann XML-SG-Instanzen einlesen, einzelne Elemente ausblenden und die Instanzen unter Anwendung von Kompatibilitätsregeln vergleichen. Die Ergebnisse werden dem Benutzer detailliert angezeigt.

Das *Modul MSC-CompA* setzt das entwickelte Konzept zum MSC-Vergleich (*MSC-CompA*) um. Das Tool bietet einen durchgängigen Weg vom Einlesen der MSCs bis zur

Analyse. Die Zwischenergebnisse werden dem Benutzer, wie bei *XML-CompA*, immer zur Verfügung gestellt.
Mit den entwickelten Tools steht eine durchgängige Tool-Kette zur Verfügung, die den Prozess von der Spezifikation der Steuergeräte (mittels XML und/oder MSCs) bis hin zur Analyse der Steuergeräte-Spezifikationen bzgl. Rückwärtskompatibilität abdeckt.

9 Validierung der XML-Vergleichsmethode (*XML-CompA*)

Zur Validierung der XML-Vergleichsmethode (*XML-CompA*) werden jeweils zwei spezifische XML-SG-Instanzen miteinander verglichen. Im Rahmen von CompA wurden für drei Steuergeräte XML-SG-Instanzen erstellt:

- SMG-Steuergerät: Getriebesteuergerät für das *Sequentielle Manuelle Getriebe*[1] *(SMG)* [54].
  Das Steuergerät dient zur Ansteuerung des automatisierten Handschaltgetriebes, das in mehreren BMW Modellen zum Einsatz kam (BMW 3er, 5er, 6er, Z4). Im Rahmen von *CompA* wurde das Lastenheft des Steuergerätes hinsichtlich aller Pins, Signale, etc. in XML überführt. Die Funktionsmodelle wurden nicht überführt, da diese *confidental* sind.

- JBBFE3: Das Steuergerät *Junction Box Beifahrer (JBBFE3)* ist ein aktuelles Steuergerät und momentan auch in mehreren BMW Modellen im Einsatz.
  Das Steuergerät umfasst ca. 60 Funktionen u.a. auch die Funktion des Fensterhebers [85]. Die Funktion des Fensterhebers wurde mittels MSC-Modellen nachgebildet. Der Fokus bei der Portierung in eine XML-Spezifikation lag bei diesem Steuergerät auf den Interfaces und den Pins.

- ZKE: Das Steuergerät *Zentrale Karosserie-Elektronik (*ZKE*)* kann als Vorgänger-Steuergerät von *JBBFE3* betrachtet werden [133].
  Bei diesem Steuergerät handelt es sich bereits um ein „älteres“ Steuergerät. Ebenso, wie bei *JBBFE3*, lag der Fokus bei der Portierung in eine XML-Spezifikation auf den Interfaces und Pins.

Die Methode *XML-CompA* wurde auf Basis von verschiedenen Schemata (*XML-SG-*$Schema_1$*, XML-SG-*$Schema_2$) validiert. Hierzu wurde z.B. das *XML-SG-*$Schema_1$ von *JBBFE3* verändert. Zusätzlich wurden auf Basis von *JBBFE3* valide XML-SG-Instanzen für das *XML-SG-*$Schema_2$ erzeugt. Diese XML-SG-Instanzen werden als *JBBFE2* oder *JBBFE1* bezeichnet. In der Tabelle ist eine Übersicht der erzeugten XML-SG-Instanzen zu sehen. In Abbildung 9.1 ist das Konzept der Validierung dargestellt.

---

[1] Eine Entwicklungsprojekt der BMW Group und Magneti Marelli

	Schema$_1$	Schema$_2$	Anzahl Interfaces	Anzahl Pins	Anzahl Signale	Anzahl Zeilen XML-Code
SMG	x		3	45	45	3283
JBBFE3	x		4	140	4	7546
ZKE	x		3	94	3	6058
JBBFE1/2		x	3	7	11	1363

Tabelle 9.1: Übersicht der erzeugten XML-SG-Instanzen

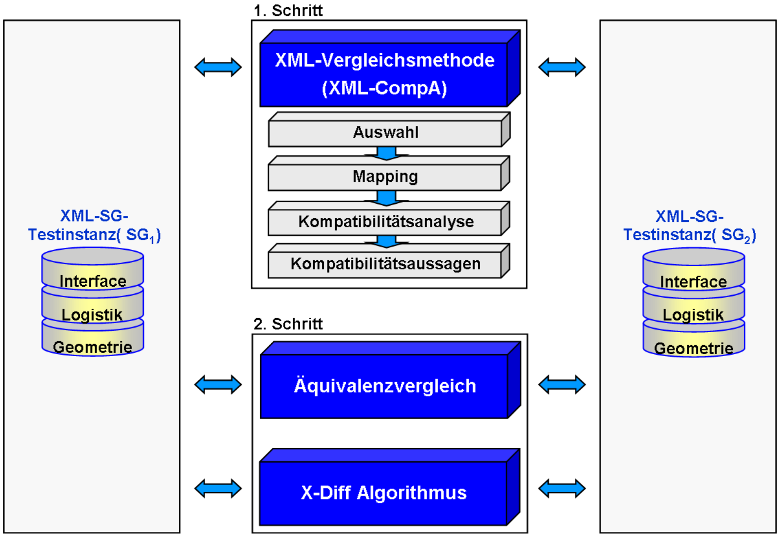


Abbildung 9.1: Konzept zur Validierung der XML-Vergleichsmethode

Im 1. Schritt wird die XML-Vergleichsmethode *XML-CompA* mit Hilfe von XML-SG-Testinstanzen validiert. Die XML-SG-Testinstanzen wurden dabei so gewählt bzw. definiert, dass damit die einzelnen Schritte von *XML-CompA* (Auswahl, Mapping, Kompatibilitätsanalyse, Kompatibilitätsaussage) auf Korrektheit überprüft werden können. Im 2. Schritt werden die gleichen XML-SG-Testinstanzen mit zwei weiteren Vergleichsansätzen analysiert:

1. *Äquivalenzvergleich*

2. *X-Diff Algorithmus* (vgl. Kapitel 4.3.1)

Im Anschluss daran werden die Ergebnisse der drei Vergleichsmethoden bewertet und so Aussagen bzgl. der Leistungsfähigkeit der Vergleichsmethoden getroffen.

9.1 Validierung von *XML-CompA*

In diesem Abschnitt wird die korrekte Funktion von *XML-CompA* validiert. Hierzu werden zwei XML-SG-Testinstanzen bzgl. Rückwärtskompatibilität verglichen und die Vergleichsergebnisse in spezifischen Tabellen dokumentiert.

9.1.1 Validierungskonzept

Jede Zeile der Tabelle entspricht einer XML-SG-Testinstanz. Für die XML-SG-Testinstanzen wurden Test-Szenarien definiert, anhand derer die in Kapitel 6.1 beschriebenen Herausforderungen untersucht werden können:

- **Äquivalenz (ÄQ):** Validierung, ob die Äquivalenz zweier XML-SG-Testinstanzen korrekt erkannt wird.
- **Ordnungsstruktur-Zuordnung (OZ):** Validierung, ob ungeordnete Strukturen korrekt erkannt werden.
- **Kompatibilitätsregeln (KR):** Validierung, ob die in dieser Arbeit definierten Kompatibilitätsregeln korrekt funktionieren (vgl. Kapitel 6.5.3).
- **Ausblendung von Strukturinformationen (ASI):** Validierung, ob aufgrund spezifischer Ausblendung von Elementen entsprechende Vergleichsergebnisse generiert werden.
- **Optionale oder nicht-relevante Informationen (OI):** Validierung, ob spezifische Elemente z.B. optionale Elemente nicht berücksichtigt werden.
- **Abhängigkeiten (AH):** Validierung, ob Abhängigkeiten in den XML-SG-Testinstanzen erkannt und im Mapping korrekt berücksichtigt werden.

In den Spalten sind die Informationen wie folgt hinterlegt:
Test-Szenario

- (1. Spalte) Eindeutige Nummer des Testfalls.
- (2. Spalte) Bezeichnung bzw. Beschreibung des 1. Testszenarios. Dies entspricht der XML-SG-Instanz von SG_1.
- (3. Spalte) Bezeichnung bzw. Beschreibung des 2. Testszenarios. Dies entspricht der XML-SG-Instanz von SG_2.
- (4. Spalte) Bewertung des von *XML-CompA* generierten Ergebnisses.

Erwartete Ergebnisse

- (5./6. Spalte) Beschreibung der erwarteten Analyse-Ergebnisse mit (*XML-CompA*).

  - (5. Spalte) Erwartungswert *Mapping*: Aussage, wie viel Prozent der Cluster von SG_1 dem Steuergerät SG_2 zugeordnet werden können.
  - (6. Spalte) Erwartungswert *Rückwärtskompatibilität*: Aussage, ob Steuergerät SG_2 rückwärtskompatibel zu SG_1 ist ($SG_1 \Leftarrow_c SG_2$).

Ergebnisse des Mappings

- (7.-10. Spalte) Ergebnisse des Mappings, d.h. wie viele Cluster von SG_1 den Clustern von SG_2 zugeordnet werden konnten.
  - (7. Spalte) Rechenzeit, die für die Berechnung des Mappings benötigt wird.
  - (8. Spalte) **Mapping-Aussage (MA):** Anzahl der Cluster, die von SG_1 auf SG_2 gemappt werden konnten im Verhältnis zu allen Clustern von SG_1. $\textbf{\textit{MA}}= \left(\frac{Anzahl\ gemappter\ Cluster}{Anzahl\ aller\ Cluster\ von\ SG_1}\right)$
  - (9. Spalte) Mapping-Aussage (MA) umgerechnet in [%].
- (10. Spalte) Erweitertes Mapping: Anzahl der Elemente, die durch das Erweiterte Mapping neu zugeordnet werden.

Ergebnisse der Kompatibilitätsanalyse/-aussage

- (11. Spalte) Rechenzeit, die für die Berechnung der Kompatibilitätsanalyse und -aussage benötigt wird.
- (12. Spalte) **Rückwärtskompatibilitäts-Aussage (RKA):** Verhältnis der angewandten Kompatibilitätsregeln mit Aussage „rückwärtskompatibel“ zu allen angewandten Kompatibilitätsregeln. $\textbf{\textit{(RKA)}}= \left(\frac{Anzahl\ kompatibler\ Kompatibilitaetsregeln}{Anzahl\ angewandter\ Kompatibilitaetsregeln}\right)$
- (13. Spalte) Rückwärtskompatibilitäts-Aussage (RKA) umgerechnet in [%].
- (14. Spalte) Aussage bzgl. Rückwärtskompatibilität von Steuergerät SG_2 zu SG_1 (rückwärtskompatibel / nicht rückwärtskompatibel). Dies ist dann der Fall, wenn alle Cluster von SG_1 auf SG_2 gemappt werden konnten und alle angewendeten Rückwärtskompatibilitäts-Regeln eine Kompatibilität anzeigen, d.h. RKA=100%.

Zur Validierung wurden ca. 31 Testfälle generiert. In Abbildung 9.2 ist die Zuordnung der Testfälle zu den Herausforderungen dargestellt. Alle Aspekte konnten damit erfolgreich evaluiert werden.

9.1.2 Ergebnisse von *XML-CompA*

In Tabelle 9.2, 9.3, 9.4 und 9.5 sind die Ergebnisse hinterlegt, die bei dem Vergleich von XML-SG-Testinstanzen mittels *XML-CompA* generiert wurden.

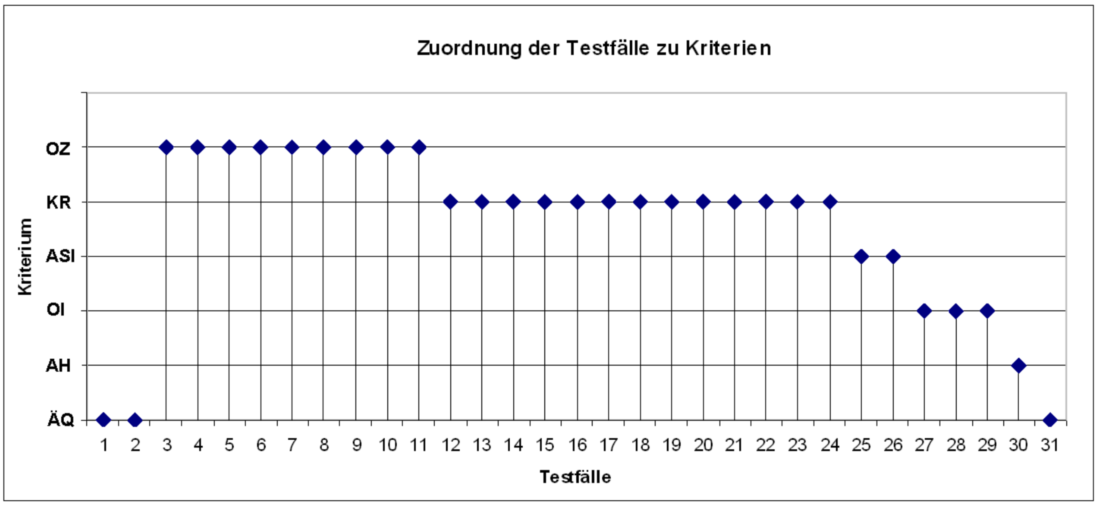


Abbildung 9.2: Zuordnung der Testfälle zu den einzelnen Herausforderungen

				Erwartetes		Mapping				Analyse mit Kompatibilitätsregeln			
							Ähnlichkeits-Mapping		Erweit.-Mapping				
Testfall-Nummer	Testinstanz $_1$ (SG$_1$)	Testinstanz $_2$ (SG$_2$)	Status	Mapping-Aussage (MA) [%]	Rückwärtskompatibilität (ja/nein)	Rechenzeit für Mapping	Mapping-Aussage	Mapping-Aussage (MA) [%]	Anzahl der neu gemappten Cluster	Rechenzeit für Kompatibilitätsanalyse	Rückwärtskompatibilitäts-Aussage (RKA)	Rückwärtskompatibilitäts-Aussage (RKA) [%]	Rückwärtskompatibilität
Äquivalente Testinstanzen													
1	**SMG**	**SMG**	ok	100%	ja	2,21s	135/135	100%	--	309ms	484/484	100%	rückwärtskompatibel
2	**JBBF1**	**JBBF1**	ok	100%	ja	0,5s	98/98	100%	--	94ms	232/232	100%	rückwärtskompatibel

Tabelle 9.2: Erkennen von äquivalenten Instanzen durch *XML-CompA*

Validierung äquivalenter/identischer Instanzen In Testfall-Nr. 1 und 2 (Tabelle 9.2) werden zwei gleiche XML-SG-Instanzen miteinander verglichen. Ziel dieser Tests ist, zu überprüfen, ob diese Instanzen von *XML-CompA* als äquivalent bzw. rückwärtskompatibel erkannt werden. Da die Instanzen identisch sind, müssen alle Cluster gemappt werden können (*Mapping-Aussage(MA)=100%)* und alle angewandten Kompatibilitätsregeln müssen auch Rückwärtskompatibilität bestätigen (*Rückwärtskompatibilitäts-Aussage (RKA)=100%.*

Fazit: Für alle äquivalenten bzw. identischen Instanzen funktioniert das Verfahren korrekt.

Validierung Ordnungsstruktur-Zuordnung In den Testfällen-Nr. 3,4,5,6,11 (Tabelle 9.3) wurde in den Testinstanzen die Reihenfolge von unordered-Elementen geändert. Die Verdrehung erfolgte dabei sowohl auf einer Ebene (Testfall-Nr. 3,11) als auch auf mehreren Ebenen (Testfall-Nr. 4,5,6). In diesen Testfällen muss *XML-CompA*

				Erwartetes		Mapping				Analyse mit Kompatibilitätsregeln			
							Ähnlichkeits-Mapping		Erweit.-Mapping				
Testfall-Nummer	**Testinstanz $_1$(SG_1)**	**Testinstanz $_2$(SG_2)**	**Status**	**Mapping-Aussage (MA) [%]**	**Rückwärtskompatibilität (ja/nein)**	**Rechenzeit für Mapping**	**Mapping-Aussage**	**Mapping-Aussage (MA) [%]**	**Anzahl der neu gemappten Cluster**	**Rechenzeit für Kompatibilitätsanalyse**	**Rückwärtskompatibilitäts-Aussage (RKA)**	**Rückwärtskompatibilitäts-Aussage (RKA) [%]**	**Rückwärtskompatibilität**
Ordnungsstruktur-Zuordung													
3	**SMG**	**SMG_V1** - Interfaces vertauscht	ok	100%	ja	1,08s	135/135	100%	--	204ms	484/484	100%	rückwärtskompatibel
4	**SMG**	**SMG_V2** - Interfaces vertauscht - Pins innerhalb Interface iI001 vertauscht	ok	100%	ja	1,17s	135/135	100%	--	156ms	484/484	100%	rückwärtskompatibel
5	**SMG**	**SMG_V3** - Interfaces vertauscht - Pins innerhalb Interface iI001 vertauscht - Signale-Semantik innerhalb der Signale vertauscht	ok	100%	ja	1,28s	135/135	100%	--	156ms	484/484	100%	rückwärtskompatibel
6	**SMG**	**SMG_V10** - interfaces vertauscht - Pins innerhalb Interface 1 vertauscht - Pins innerhalb Interace 3 vertauscht	ok	100%	ja	1,19s	135/135	100%	--	172ms	484/484	100%	rückwärtskompatibel
7	**SMG_V6** - zusätzliche unordered-Elemente / Cluster	**SMG**	ok	<100%	nein	1,27s	135/139	97,12%	--	140ms	484/484	100%	nicht rückwärtskompatibel
8	**SMG**	**SMG_V6** - zusätzliche unordered-Elemente/Cluster	ok	100%	ja	1,20s	135/135	100%	--	140ms	484/484	100%	rückwärtskompatibel
9	**SMG**	**SMG_V9** - Signalsemantik von verschiedenen Signalen vertauscht d.h. nicht innerhalb eines Signals (S006<-->S005;S007<-->S004)	ok	100%	nein	2,031	135/135	100%	--	375ms	479/481	99,58%	nicht rückwärtskompatibel
10	**JBBF1**	**Anzahl** - unordered-Elemente/Cluster entfernt	ok	100%	nein	0,203s	56/96	58,33%	--	15ms	68/118	57,63%	nicht rückwärtskompatibel
11	**JBBF1**	**Reihenfolge** - Reihenfolge vertauscht	ok	100%	ja	0,452s	98/98	100%	--	31ms	232/232	100%	rückwärtskompatibel

Tabelle 9.3: Validierung der Erkennung von ungeordneten Strukturen

also die Änderung der Reihenfolge in den Instanzen erkennen und die unordered-Elemente bzw. 135 Cluster beim Mapping korrekt zuordnen. Die Mapping-Aussage (MA=100%) zeigt, dass alle Elemente zugeordnet werden konnten. Da auch keine Werte verändert wurden, muss das Ergebnis der Rückwärtskompatibilitäts-Aussage (RKA) gleich 100% sein. Dies ist in den genannten Testfällen der Fall.

In Testfall-Nr. 9 (Tabelle 9.3) wurde nicht nur die Reihenfolge verändert, sondern es wurden unordered Elemente in der $Testinstanz_2$ gegeneinander vertauscht, d.h. ihre hierarchische Zuordnung verändert. Beim Mapping wird nun aufgrund der Vertauschung z.B. *S006* von (SG_1) auf *S005* von (SG_2) zugeordnet. Wie das Ergebnis zeigt kann *XML-CompA* alle 135 Cluster von SG_1 auf SG_2 zuordnen. Dass die zugeordneten

Elemente jedoch nicht identisch bzw. nicht rückwärtskompatibel sind, zeigt die Kompatibilitätsanalyse. Von den 481 angewandten Kompatibilitätsregeln verletzten zwei die Kompatibilitätskriterien (RKA=99,58%).

In den Testfällen mit Nr. 7,8,10 (Tabelle 9.3) wurde die Anzahl der XML-Elemente in den Testinstanzen verändert. In Testfall-Nr. 7 wurden der $Testinstanz_1$ vier neue unordered-Elemente bzw. Cluster hinzugefügt, also die Anzahl der Cluster auf 139 erhöht ($SG_1 > SG_2$). Ansonsten sind die $Testinstanz_1$ und $Testinstanz_2$ identisch. Beim Mapping wird deutlich, dass genau diese vier neuen Cluster nicht zugeordnet werden können, sondern nur 135 Cluster (MA=135/139). Alle Elemente, die zugeordnet werden können, müssen rückwärtskompatibel sein, da die Instanzen bis auf die vier zusätzlichen Cluster identisch sind. Das Ergebnis von RKA ist 100%. Da jedoch nicht alle Cluster zu 100% zugeordnet werden konnten, kann SG_1 keine echte Teilmenge von SG_2 sein ($SG_1 \nsubseteq SG_2$) und somit gilt $SG_1 \not\Leftarrow_c SG_2$. In Testfall-Nr. 8 ist der Fall genau umgekehrt zu Testfall-Nr. 7: $Testinstanz_2$ verfügt über vier zusätzliche unordered-Elemente bzw. Cluster und ist ansonsten identisch zu $Testinstanz_1$. In diesem Fall stellt SG_1 also eine echte Teilmenge von SG_2 dar ($SG_1 \subseteq SG_2$). *XML-CompA* kann also alle 135 Cluster von (SG_1) zuordnen (MA=100%) und auch RKA ergibt 100%. Dieser Testfall simuliert den realen Fall, dass in einem Steuergerät SG_2 neue Zusatzfunktionen integriert wurden und das Steuergerät ansonsten identisch zum Vorgänger-Steuergerät SG_1 ist.

In Testfall-Nr. 10 (Tabelle 9.3) wurden bei der $Testinstanz_2$ mehrere *unordered* und *ordered* Elemente entfernt. Daher können nicht alle Cluster von $Testinstanz_1$ der $Testinstanz_2$ zugeordnet werden (MA=56/96). Aufgrund der fehlenden *ordered* Elemente können bei den zugeordneten Elementen die Kompatibilitätsregeln nicht vollständig angewendet werden (RKA=68/118). Die Ergebnisse zeigen, dass die fehlenden *unordered* und *ordered* korrekt erkannt werden.

Fazit: Die Testergebnisse belegen, dass die Methode *XML-CompA* ungeordnete Strukturen erkennen und die Elemente von Steuergeräte SG_1 dem Steuergerät SG_2 zuordnen kann.

Validierung Kompatibilitätsregeln Die Testfälle mit Nr. 12-24 (Tabelle 9.4) stellen spezifische Fälle dar, mit denen die Funktionsfähigkeit der Kompatibilitätsregeln getestet werden. In *XML-CompA* wurden spezifische Kompatibilitätsregeln implementiert (vgl. Kapitel 6.5.3).

Die Testfälle mit Nr. 12-22 wurden jeweils zur Validierung einer spezifischen Kompatibilitätsregel entworfen. Hierzu wurden die Testinstanzen pro Testfall so manipuliert[2], dass die Testinstanzen zwar nicht mehr äquivalent sind, aber aufgrund der spezifischen Kompatibilitätsregeln noch als zueinander rückwärtskompatibel betrachtet werden können. Dass die Kompatibilitätsregeln korrekt umgesetzt sind, ist daran zu erkennen, dass sowohl die Mapping-Aussage als auch die Rückwärtskompatibilitäts-Aussage 100% ergeben und somit die manipulierten Instanzen als zueinander rückwärtskompa-

---

[2] Zur Validierung der Kompatibilitätsregeln wurde auch die Testinstanz SMG um zusätzliche Elemente erweitert und als SMG_V1 abgespeichert

tibel gelten.

In den Testfällen mit Nr. 24,25 wurden die Werte in den Testinstanzen so verändert, dass die Kompatibilitätsregeln „verletzt“ werden und daher die Instanzen als nicht rückwärtskompatibel erkannt werden. Beim Mapping wurden die Cluster der Testinstanzen korrekt zugeordnet (MA=100$\%$). Bei der Rückwärtskompatibilitäts-Aussage zeigt sich dann, dass Kompatibilitätsregeln entsprechend den Manipulationen verletzt werden. In beiden Testfällen wird die Inkompatibilität korrekt erkannt.

Fazit: Die Testergebnisse zeigen, dass die implementierten Kompatibilitätsregeln korrekt funktionieren und dass diese das Anwendungsfeld zur Erkennung von Rückwärtskompatibilität stark erweitern.

Ausblendung von Strukturinformationen Die Testfälle mit Nr. 25,26 (Tabelle 9.5) wurden zum Nachweis der korrekten Kompatibilitätsberechnung bei Ausblendung von Strukturinformationen definiert.

Hierzu wurden analog zu Kapitel 6.3 zwei Testinstanzen definiert, die die gleichen Pins, Signale etc. besitzen, jedoch mit dem Unterschied, dass $Testinstanz_1$ ein Interface und $Testinstanz_2$ zwei Interfaces besitzt und die Pins auf die Interfaces verteilt sind. In Testfall Nr. 25 werden die beiden Testinstanzen miteinander verglichen und es zeigt sich, dass nur ein Teil der Cluster von $Testinstanz_1$ auf $Testinstanz_2$ zugeordnet werden kann (MA=68/91). Die zugeordneten Cluster sind hierbei jeweils zu 100$\%$ rückwärtskompatibel. Veranschaulicht bedeutet dies, dass das eine Interface von $Testinstanz_1$ auf ein Interface von $Testinstanz_2$ zugeordnet wurde. Da das Interface($Testinstanz_1$) mehr Pins enthält als Interface($Testinstanz_2$) konnten nicht alle Cluster zugeordnet werden.

In Testfall-Nr. 26 werden die Interfaces ausgeblendet. Dadurch ist die hierarchische Zuordnung der Pins zu den Interfaces aufgehoben und somit können alle Pins von $Testinstanz_1$ der $Testinstanz_2$ zugeordnet werden. Dies zeigt sich zum einen an *MA=100$\%$*. Da die beiden Testinstanzen bis auf die Interfaces identisch waren, ist das Ergebnis der *Rückwärtskompatibilitäts-Aussage=100$\%$*.

Auf Basis dieses Ergebnisses kann abgeleitet werden, dass ein Steckeradapter, der die beiden Interfaces von $Testinstanz_2$ auf ein Interface zusammenführt (analog zu $Testinstanz_1$), Rückwärtskompatibilität herstellen kann.

Fazit: *XML-CompA* bietet durch das Feature, einzelne Elemente für die Kompatibilitätsanalyse dediziert auszublenden, ein sehr mächtiges Werkzeug, um spezifische Analysen zur Erreichung von Rückwärtskompatibilität durchzuführen.

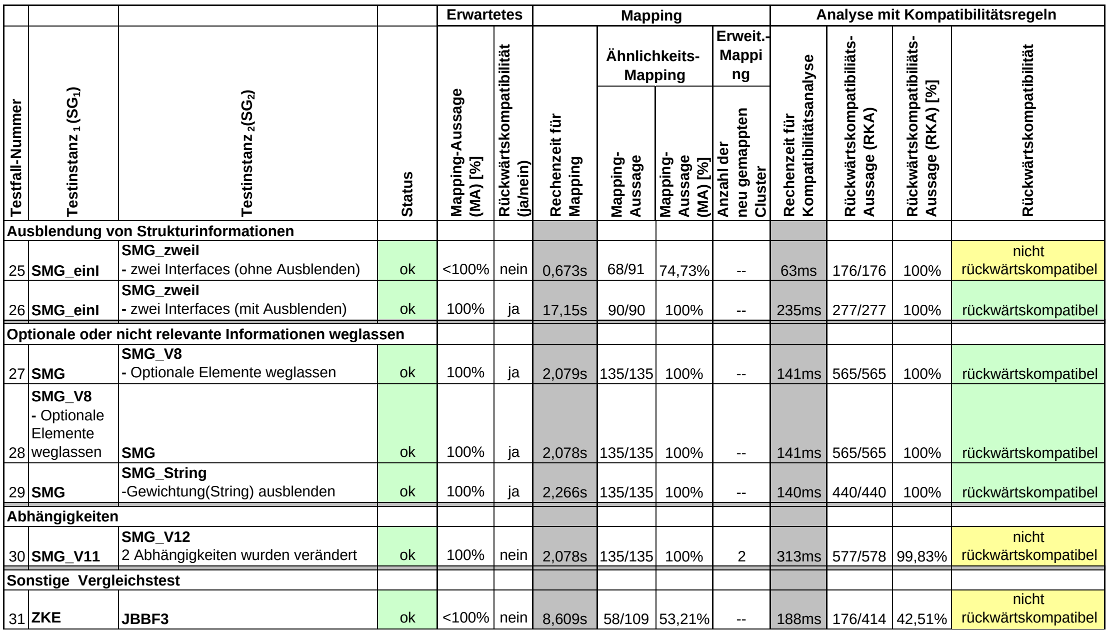
Optionale oder nicht relevante Informationen weglassen Mit Testfall-Nr. 27,28 (Tabelle 9.5) wird nachgewiesen, dass optionale Elemente, die nicht zur Analyse von Rückwärtskompatibilität beitragen, automatisch „ausgeblendet“ werden. Hierzu wurden in Testfall-Nr. 27 optionale Elemente in $Testinstanz_2$ und in Testfall-Nr. 28 optionale Elemente in $Testinstanz_1$ entfernt. In beiden Testfällen werden die *unordered*-Elemente bzw. Cluster einander korrekt zugeordnet und die Rückwärtskompatibilität ergibt korrekterweise 100$\%$. Somit könnten Zulieferer eigene optionale Elemente

				Erwartetes		Mapping				Analyse mit Kompatibilitätsregeln			
							Ähnlichkeits-Mapping		Erweit.-Mapping				
Testfall-Nummer	Testinstanz $_1$ (SG$_1$)	Testinstanz $_2$(SG$_2$)	Status	Mapping-Aussage (MA) [%]	Rückwärtskompatibilität (ja/nein)	Rechenzeit für Mapping	Mapping-Aussage	Mapping-Aussage (MA) [%]	Anzahl der neu gemappten Cluster	Rechenzeit für Kompatibilitätsanalyse	Rückwärtskompatibiliäts-Aussage (RKA)	Rückwärtskompatibiliäts-Aussage (RKA) [%]	Rückwärtskompatibilität
Kompatibilitätsregeln													
12	**SMG**	**SMG_V101** **(ignorieren)** - Meta-Information/Name von „Pin 4“ auf „Pin 123“ von Pin mit ID=pI003 geändert	ok	100%	ja	1,28s	135/135	100%	--	344ms	484/484	100%	rückwärtskompatibel
14	**SMG**	**SMG_V103** **(ISG1innerhalbSG2)** - Leistung/.../Max von Pin mit ID=p001 von 5 auf 10 geändert	ok	100%	ja	1,24s	135/135	100%	--	235ms	484/484	100%	rückwärtskompatibel
15	**SMG**	**SMG_V104** **(ISG2innerhalbSG1)** - Aufwach-Zeit/.../Max von Pin mit ID=p001 von 10 auf 5 geändert	ok	100%	ja	1,281s	135/135	100%	--	219ms	484/484	100%	rückwärtskompatibel
16	**SMG_V1**	**SMG_V1_V105** **(ImaxSG2kleinergleichmaxSG1)** - Flanke-Steigend/.../Max von Highside mit ID=C001 von 10 auf 6 geändert	ok	100%	ja	1,25s	135/135	100%	--	203ms	492/492	100%	rückwärtskompatibel
18	**SMG_V1**	**SMG_V1_V107** **(IuntereobereGrenzeplusminusX)** - Frequenz/.../Min von 3 auf -2 und Frequenz/.../Max von 78 auf 75 bei Signal S001 geändert	ok	100%	ja	2,16s	138/138	100%	--	422ms	492/492	100%	rückwärtskompatibel
19	**SMG_V1**	**SMG_V1_V108** **(FSG2greaterSG1)** - Lebensdauer von Highside mit ID=C001 von 10 auf 20 geändert	ok	100%	ja	2,17s	138/138	100%	--	438ms	492/492	100%	rückwärtskompatibel
20	**SMG**	**SMG_V1_V109** **(FSG2groessergleichSG1plusX)** - Signalsemantik/.../High-Pegel-Offset von 2 auf 2.1 bei Signal S008 geändert	ok	100%	ja	2,1s	138/138	100%	--	234ms	492/492	100%	rückwärtskompatibel
21	**SMG_V1**	**SMG_V1_V110** **(FSG2kleinergleichSG1minusX)** - Flanke-Steigend/.../Max von 10 auf 9.9 und Flanke-Steigend/Min von 5 auf 4.9 geändert bei Highside mit ID=C001	ok	100%	ja	2,21s	138/138	100%	--	375ms	492/492	100%	rückwärtskompatibel
22	**SMG**	**SMG_V111** **(FSG2innerhalbSG1plusminusX)** - Aufwach-Zeit/.../Max von Pin mit ID=p001 von 10 auf 8 und Aufwach-Zeit/.../Typisch von 5 auf 4 geändert	ok	100%	ja	1,45s	135/135	100%	--	250ms	484/484	100%	rückwärtskompatibel
23	**SMG**	**SMG_V5** **(verschiedene CompA-Regeln verletzt)** - Belastungssparameter tiefgesetzt bei Interface II1	ok	100%	nein	1,27s	135/135	100%	--	94ms	471/484	97,31%	nicht rückwärtskompatibel
24	**SMG**	**SMG_V7** - Spezifische Werte sind inkompatibel	ok	100%	nein	1,25s	135/135	100%	--	94ms	479/484	98,97%	nicht rückwärtskompatibel

Tabelle 9.4: Validierung der Kompatibilitätsregeln von *XML-CompA*

verwenden, die bei der Kompatibilitätsanalyse nicht berücksichtigt werden sollen.

In Testfall-Nr. 29 (Tabelle 9.5) wird die korrekte Funktion der *Gewichtung* überprüft. Die Gewichtung dient zum Ausblenden von spezifischen Informationen, die keinen Beitrag zur Kompatibilitätsanalyse haben bzw. diese verfälschen würden. Ein typisches Beispiel hierfür sind die internen Verweise ID und IDREF. In $Testinstanz_2$ wurde hierfür ein String verändert und aktiv über eine *Gewichtung=0* ausgeblendet. Dadurch



				Erwartetes		Mapping				Analyse mit Kompatibilitätsregeln			
							Ähnlichkeits-Mapping		Erweit.-Mapping				
Testfall-Nummer	Testinstanz $_1$ (SG$_1$)	Testinstanz $_2$(SG$_2$)	Status	Mapping-Aussage (MA) [%]	Rückwärtskompatibilität (ja/nein)	Rechenzeit für Mapping	Mapping-Aussage	Mapping-Aussage (MA) [%]	Anzahl der neu gemappten Cluster	Rechenzeit für Kompatibilitätsanalyse	Rückwärtskompatibiläts-Aussage (RKA)	Rückwärtskompatibiläts-Aussage (RKA) [%]	Rückwärtskompatibilität
Ausblendung von Strukturinformationen													
25	**SMG_einl**	**SMG_zweil** **-** zwei Interfaces (ohne Ausblenden)	ok	<100%	nein	0,673s	68/91	74,73%	--	63ms	176/176	100%	nicht rückwärtskompatibel
26	**SMG_einl**	**SMG_zweil** **-** zwei Interfaces (mit Ausblenden)	ok	100%	ja	17,15s	90/90	100%	--	235ms	277/277	100%	rückwärtskompatibel
Optionale oder nicht relevante Informationen weglassen													
27	**SMG**	**SMG_V8** **-** Optionale Elemente weglassen	ok	100%	ja	2,079s	135/135	100%	--	141ms	565/565	100%	rückwärtskompatibel
28	**SMG_V8** **-** Optionale Elemente weglassen	**SMG**	ok	100%	ja	2,078s	135/135	100%	--	141ms	565/565	100%	rückwärtskompatibel
29	**SMG**	**SMG_String** -Gewichtung(String) ausblenden	ok	100%	ja	2,266s	135/135	100%	--	140ms	440/440	100%	rückwärtskompatibel
Abhängigkeiten													
30	**SMG_V11**	**SMG_V12** 2 Abhängigkeiten wurden verändert	ok	100%	nein	2,078s	135/135	100%	2	313ms	577/578	99,83%	nicht rückwärtskompatibel
Sonstige Vergleichstest													
31	**ZKE**	**JBBF3**	ok	<100%	nein	8,609s	58/109	53,21%	--	188ms	176/414	42,51%	nicht rückwärtskompatibel

Tabelle 9.5: Validierung der Ausblendung von Strukturinformationen, optionalen Informationen und der Berücksichtigung von Abhängigkeiten

ergibt sich, obwohl die Instanzen nicht äquivalent sind, dass die beiden Testinstanzen zueinander rückwärtskompatibel sind.

Fazit: Elemente, die die Kompatibilitätsanalyse verfälschen würden, können mittels *XML-CompA* von vornherein ausgeblendet werden. Die Testfälle belegen sowohl die korrekte Implementierung als auch die Notwendigkeit für dieses Feature.

Abhängigkeiten Durch Testfall-Nr. 30 (Tabelle 9.5) wird nachgewiesen, dass beim Mapping von Elementen die Abhängigkeiten korrekt berücksichtigt werden. Hierzu wurden zwei identische XML-SG-Testinstanzen angelegt, mit dem Unterschied, dass sich die Abhängigkeiten an zwei Stellen unterscheiden. In $Testinstanz_1$ wurde spezifiziert, dass das Signal S_1 an Pin P_1 und das Signal S_2 an Pin P_2 anliegt. $Testinstanz_2$ wurde mittels der Abhängigkeiten jedoch so manipuliert, dass das Signal S_1 an Pin P_2 und das Signal S_2 an Pin P_1 anliegt. Beim *Ähnlichkeits-Mapping* werden die Abhängigkeiten noch nicht berücksichtigt und daher werden 100% der Elemente zugeordnet, wobei

$$P_1(Testinstanz_1) \rightarrow P_1(Testinstanz_2)$$

$$P_2(Testinstanz_1) \rightarrow P_2(Testinstanz_2)$$

$$S_1(Testinstanz_1) \rightarrow S_1(Testinstanz_2)$$

$$S_2(Testinstanz_1) \rightarrow S_2(Testinstanz_2)$$

Durch das *Erweiterte Mapping* werden die Abhängigkeiten bei der Zuordnung berücksichtigt und daher werden zwei Cluster „neu“ gemappt. Da in diesem Testfall dem *Pin* die höhere Priorität für das Mapping zugeordnet worden ist, wurden die Cluster nun wie folgt zugeordnet:

$$P_1(Testinstanz_1) \rightarrow P_1(Testinstanz_2)$$

$$P_2(Testinstanz_1) \rightarrow P_2(Testinstanz_2)$$

$$S_1(Testinstanz_1) \rightarrow S_2(Testinstanz_2)$$

$$S_2(Testinstanz_1) \rightarrow S_1(Testinstanz_2)$$

Fazit: *XML-CompA* kann mit Hilfe des *Erweiterten Mappings* Abhängigkeiten erkennen und diese in das Mapping valide mit einbeziehen. Nur dadurch ist eine korrekte Zuordnung der Elemente sichergestellt.

Vergleichstest zweier realer Steuergeräte In Testfall-Nr. 31 (Tabelle 9.5) wird die XML-SG-Instanz von Steuergerät *JBBFE3* gegen das „Vorgänger-Steuergerät“ *ZKE* verglichen. Das Ergebnis zeigt, dass über 50% der Cluster von *ZKE* auf *JBBFE3* zugeordnet werden können (MA=53,21%) und dass ca. die Hälfte der zugeordneten Elemente rückwärtskompatibel wären (RKA=42,51%). Entwickler können auf Basis dieser Ergebnisse spezifische Maßnahmen ableiten, um evtl. Rückwärtskompatibilität der spezifizierten Steuergeräte zu ermöglichen.

9.2 Vergleich mit weiteren Tools

Die Leistungsfähigkeit von *XML-CompA* wird durch einen Vergleich mit

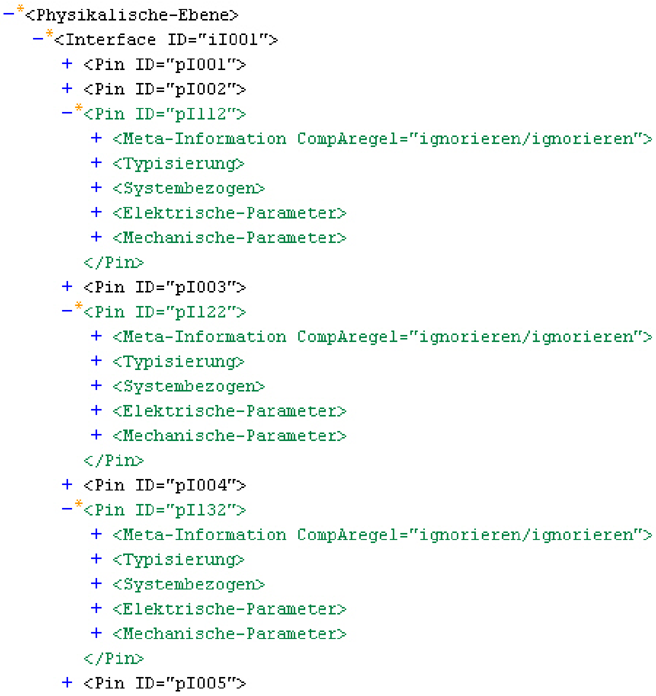
- einem *Äquivalenzvergleich* und
- einem *X-Diff Algorithmus* (vgl. Kapitel 4.3.1)

unter Beweis gestellt. Hierzu werden die in den Tabellen 9.2, 9.3, 9.4 und 9.5 beschriebenen Testfälle auch mit den beiden Vergleichsmethoden durchgeführt. Die Ergebnisse der beiden Vergleichsmethoden werden mit denen von *XML-CompA* verglichen.

Äquivalenzvergleich Der *Äquivalenzvergleich* überprüft, ob zwei XML-SG-Instanzen äquivalent sind. Hierzu werden in *XML-CompA* die XML-SG-Instanzen auf Matrizen portiert und der Vergleich auf Basis dieser Matrizen durchgeführt (vgl. [132]). Die *Äquivalenz-Aussage* sei wie folgt definiert:
Äquivalenz-Aussage$= \left(\frac{Anzahl\ aller\ uebereinstimmender\ XML-Elemente}{Anzahl\ aller\ vorhandener\ XML-Elemente}\right)$

X-Diff-Vergleich *X-Diff Algorithmen* dienen der *Change Detection*, d.h. der Detektion von Unterschieden zweier Dokumente.

Zum Vergleich der Testfälle über einen *X-Diff Algorithmus* wurde das Tool *Exchanger XML Professional 3.2* eingesetzt. Das Ergebnis des Vergleichs ist ein „Editskript“, in dem die Unterschiede der beiden Dokumente farblich hervorgehoben werden (vgl. Abbildung 9.3). Zur Generierung einer *X-Diff-Aussage* werden alle detektierten Unterschiede in den XML-SG-Instanzen bzw. den Testinstanzen gezählt. Eine weitere Gegenüberstellung der detektierten Unterschiede z.B. gegenüber den verglichenen Zeilen ist nicht notwendig, da an dieser Stelle nur die Aussage relevant ist, ob der *X-Diff Algorithmus* die Manipulationen an den Testinstanzen korrekt erkennt. Die *X-Diff-Aussage* sei wie folgt definiert:



```
-*<Physikalische-Ebene>
  -*<Interface ID="iI001">
    + <Pin ID="pI001">
    + <Pin ID="pI002">
    -*<Pin ID="pI112">
      + <Meta-Information CompAregel="ignorieren/ignorieren">
      + <Typisierung>
      + <Systembezogen>
      + <Elektrische-Parameter>
      + <Mechanische-Parameter>
      </Pin>
    + <Pin ID="pI003">
    -*<Pin ID="pI122">
      + <Meta-Information CompAregel="ignorieren/ignorieren">
      + <Typisierung>
      + <Systembezogen>
      + <Elektrische-Parameter>
      + <Mechanische-Parameter>
      </Pin>
    + <Pin ID="pI004">
    -*<Pin ID="pI132">
      + <Meta-Information CompAregel="ignorieren/ignorieren">
      + <Typisierung>
      + <Systembezogen>
      + <Elektrische-Parameter>
      + <Mechanische-Parameter>
      </Pin>
    + <Pin ID="pI005">
```

Abbildung 9.3: Ergebnis des Vergleichs mit Exchanger XML Professional 3.2

X-Diff-Aussage$= (Anzahl\ detektierter\ Unterschiede)$

9.2.1 Validierungskonzept

Zum Vergleich der Ergebnisse der Methoden *Äquivalenzvergleich* und *X-Diff Algorithmus* mit *XML-CompA* wurden diese in den Tabellen 9.6, 9.7 und 9.8 abgelegt, die wie folgt aufgebaut sind:

Test-Szenario

- (1. Spalte) Eindeutige Nummer des Testfalls

- (2. Spalte) Bezeichung bzw. Beschreibung des 1. Testszenarios. Dies entspricht der XML-SG-Instanz von SG_1
- (3. Spalte) Bezeichung bzw. Beschreibung des 2. Testszenarios. Dies entspricht der XML-SG-Instanz von SG_2
- (4. Spalte) Status, ob die generierten Vergleichsergebnisse „ok“ bzw. plausibel sind

Ergebnis von *XML-CompA*

- (5. Spalte) Mapping-Aussage in [%]
- (6. Spalte) Rückwärtskompatibilitäts-Aussage in [%]
- (7. Spalte) Ergebnis, ob Testinstanzen zueinander *rückwärtskompatibel* bzw. *nicht rückwärtskompatibel* sind

Ergebnis von *Äquivalenzvergleich*

- (8. Spalte) Äquivalenz-Aussage
- (9. Spalte) Äquivalenz-Aussage in [%]
- (10. Spalte) Aussage, ob die Testinstanzen auf Basis des *Äquivalenzvergleichs äquivalent* bzw. *nicht äquivalent* sind

Ergebnis von *X-Diff Vergleichs*

- (11. Spalte) *X-Diff-Aussage*: Anzahl der detektierten Unterschiede
- (12. Spalte) Aussage, ob die Testinstanzen auf Basis des *X-Diff Vergleichs äquivalent* bzw. *nicht äquivalent* sind

9.2.2 Ergebnisse des Vergleichs

Validierung gleiche Instanzen In Testfall-Nr. 1 und 2 (Tabelle 9.6) werden zwei gleiche Instanzen miteinander verglichen. Alle drei Vergleichsmethoden erkennen, dass es sich um äquivalente bzw. rückwärtskompatible XML-SG-Instanzen handelt.

Validierung Ordnungsstruktur-Zuordnung Die Testfälle mit Nr. 3-11 (Tabelle 9.6) dienen der Evaluierung, ob bzw. wie gut die XML-Vergleichsmethoden ungeordnete Strukturen erkennen und diese korrekt einander zuordnen.
In den Testfällen Nr. 3,4,6,11 wurde bei $Testinstanz_2$ eine Links-Rechts-Vertauschung von ungeordneten Elementen vorgenommen, ansonsten sind die Testinstanzen identisch. *XML-CompA* und der *X-Diff Vergleich* erkennen diese Permutation korrekt und geben als Ergebnis *rückwärtskompatibel* bzw. *äquivalent* aus. Der Äquivalenzvergleich kann nur geordnete Strukturen erkennen und gibt daher als Ergebnis *nicht äquivalent*

Testszenarien				XML-CompA			Äquivalenzvergleich			X-Diff Vergleich	
Testfall-Nummer	**Testinstanz$_1$(SG$_1$)**	**Testinstanz$_2$(SG$_2$)**	**Status**	**Mapping-Aussage [%]**	**Rückwärts-kompatibiliäts-Aussage [%]**	**Rückwärts-kompatibilität**	**Äquivalenz-Aussage**	**Äquivalenz-Aussage [%]**	**äquivalent / nicht äquivalent**	**X-Diff-Aussage**	**äquivalent / nicht äquivalent**
Äquivalente Testinstanzen											
1	**SMG**	**SMG**	ok	100%	100%	rückwärtskompatibel	6033/6033	100%	äquivalent	0	äquivalent
2	**JBBF1**	**JBBF1**	ok	100%	100%	rückwärtskompatibel	1230/1230	100%	äquivalent	0	äquivalent
Ordnungsstruktur-Zuordung											
3	**SMG**	**SMG_V1** - Interfaces vertauscht	ok	100%	100%	rückwärtskompatibel	3422/8019	42,67%	nicht äquivalent	0	äquivalent
4	**SMG**	**SMG_V2** - Interfaces vertauscht - Pins innerhalb Interface iI001 vertauscht	ok	100%	100%	rückwärtskompatibel	3420/8019	42,65%	nicht äquivalent	0	äquivalent
5	**SMG**	**SMG_V3** - Interfaces vertauscht - Pins innerhalb Interface iI001 vertauscht - Signale-Semantik innerhalb der Signale vertauscht	ok	100%	100%	rückwärtskompatibel	3421/8019	42,68%	nicht äquivalent	2	nicht äquivalent
6	**SMG**	**SMG_V10** - interfaces vertauscht - Pins innerhalb Interface 1 vertauscht - Pins innerhalb Interace 3 vertauscht	ok	100%	100%	rückwärtskompatibel	3408/8035	42,41%	nicht äquivalent	0	äquivalent
7	**SMG_V6** - zusätzliche unordered-Elemente/Cluster	**SMG**	ok	97,12%	100%	nicht rückwärtskompatibel	4215/24537	17,18%	nicht äquivalent	4	nicht äquivalent
8	**SMG**	**SMG_V6** - zusätzliche unordered-Elemente/Cluster	ok	100%	100%	rückwärtskompatibel	4215/24537	17,18%	nicht äquivalent	4	nicht äquivalent
9	**SMG**	**SMG_V9** - Signalsemantik von verschiedenen Signalen vertauscht d.h. nicht innerhalb eines Signals (S006<-->S005;S007<-->S004)	ok	100%	99,58%	nicht rückwärtskompatibel	5838/6106	95,61%	nicht äquivalent	38	nicht äquivalent
10	**JBBF1**	**Anzahl** - unordered-Elemente/Cluster entfernt	ok	58,33%	57,63%	nicht rückwärtskompatibel	432/6138	7,04%	nicht äquivalent	110	nicht äquivalent
11	**JBBF1**	**Reihenfolge** - Reihenfolge vertauscht	ok	100%	100%	rückwärtskompatibel	1129/1305	86,51%	nicht äquivalent	0	äquivalent

Tabelle 9.6: Ergebnisse der XML-Vergleichsmethoden (1. Teil)

aus.
Ein interessanter Testfall ist Nr. 5: Hier wurde eine Signal-Semantik von zwei Signalen vertauscht. Die vertauschte Signal-Semantik ist inhaltlich identisch, bis auf die ID (Verwendung für Referenzierung von inneren Abhängigkeiten). Der *X-Diff Vergleich* erkennt, dass es aufgrund der ID zwei Unterschiede bei den Instanzen gibt und daher sind für den *X-Diff Vergleich* die beiden Instanzen *nicht äquivalent*. Auch für den *Äquivalenzvergleich* sind die Testinstanzen *nicht äquivalent*. Für die Rückwärtskompatibilität spielt die ID keine direkte Rolle und wird in XML-CompA nicht berücksichtigt. Daher erkennt XML-CompA diesen Testfall als rückwärtskompatibel.

Validierung Kompatibilitätsregeln Die Testfälle mit Nr. 12-22 (Tabelle 9.7) stellen spezifische Fälle dar, mit denen die Funktionsfähigkeit der Kompatibilitätsregeln getestet wird. Hierzu wurden die Testinstanzen derart manipuliert, dass diese bei einem

Testszenarien				XML-CompA			Äquivalenzvergleich			X-Diff Vergleich	
Testfall-Nummer	**Testinstanz$_1$ (SG$_1$)**	**Testinstanz$_2$(SG$_2$)**	**Status**	**Mapping-Aussage [%]**	**Rückwärts-kompatibiliäts-Aussage [%]**	**Rückwärts-kompatibilität**	**Äquivalenz-Aussage**	**Äquivalenz-Aussage [%]**	**äquivalent / nicht äquivalent**	**X-Diff-Aussage**	**äquivalent / nicht äquivalent**
Kompatibilitätsregeln											
12	**SMG**	**SMG_V101** **(ignorieren)** - Meta-Information/Name von „Pin 4“ auf „Pin 123“ von Pin mit ID=pI003 geändert	ok	100%	100%	rückwärtskompatibel	6031/6033	99,97%	nicht äquivalent	1	nicht äquivalent
14	**SMG**	**SMG_V103** **(ISG1innerhalbSG2)** - Leistung/.../Max von Pin mit ID=p001 von 5 auf 10 geändert	ok	100%	100%	rückwärtskompatibel	6031/6033	99,97%	nicht äquivalent	1	nicht äquivalent
15	**SMG**	**SMG_V104** **(ISG2innerhalbSG1)** - Aufwach-Zeit/.../Max von Pin mit ID=p001 von 10 auf 5 geändert	ok	100%	100%	rückwärtskompatibel	6031/6033	99,97%	nicht äquivalent	1	nicht äquivalent
16	**SMG_V1**	**SMG_V1_V105** **(ImaxSG2kleinergleichmaxSG1)** - Flanke-Steigend/.../Max von Highside mit ID=C001 von 10 auf 6 geändert	ok	100%	100%	rückwärtskompatibel	6077/6078	99,98%	nicht äquivalent	1	nicht äquivalent
18	**SMG_V1**	**SMG_V1_V107** **(IuntereobereGrenzeplusminusX)** - Frequenz/.../Min von 3 auf -2 und Frequenz/.../Max von 78 auf 75 bei Signal S001 geändert	ok	100%	100%	rückwärtskompatibel	6074/6078	99,93%	nicht äquivalent	1	nicht äquivalent
19	**SMG_V1**	**SMG_V1_V108** **(FSG2greaterSG1)** - Lebensdauer von Highside mit ID=C001 von 10 auf 20 geändert	ok	100%	100%	rückwärtskompatibel	6077/6078	99,98%	nicht äquivalent	1	nicht äquivalent
20	**SMG**	**SMG_V1_V109** **(FSG2groessergleichSG1plusX)** - Signalsemantik/.../High-Pegel-Offset von 2 auf 2.1 bei Signal S008 geändert	ok	100%	100%	rückwärtskompatibel	6076/6078	99,97%	nicht äquivalent	1	nicht äquivalent
21	**SMG_V1**	**SMG_V1_V110** **(FSG2kleinergleichSG1minusX)** - Flanke-Steigend/.../Max von 10 auf 9.9 und Flanke-Steigend/Min von 5 auf 4.9 geändert bei Highside mit ID=C001	ok	100%	100%	rückwärtskompatibel	6076/6078	99,97%	nicht äquivalent	1	nicht äquivalent
22	**SMG**	**SMG_V111** **(FSG2innerhalbSG1plusminusX)** - Aufwach-Zeit/.../Max von Pin mit ID=p001 von 10 auf 8 und Aufwach-Zeit/.../Typisch von 5 auf 4 geändert	ok	100%	100%	rückwärtskompatibel	6035/6027	99,87%	nicht äquivalent	2	nicht äquivalent
23	**SMG**	**SMG_V7** - Spezifische Werte sind inkompatibel	ok	100%	97,31%	nicht rückwärtskompatibel	6007/6033	99,57%	nicht äquivalent	13	nicht äquivalent
24	**SMG**	**SMG_V7** - Spezifische Werte sind inkompatibel	ok	100%	98,97%	nicht rückwärtskompatibel	6011/6033	99,64%	nicht äquivalent	9	nicht äquivalent

Tabelle 9.7: Ergebnisse der XML-Vergleichsmethoden (2. Teil)

Vergleich mit *XML-CompA* als *rückwärtskompatibel* erkannt werden. Für den *Äquivalenzvergleich* und den *X-Diff Vergleich* stellt die Manipulation der Testinstanzen eine Verletzung der Äquivalenz dar und daher werden diese Fälle von beiden Vergleichsmethoden als *nicht äquivalent* erkannt.

Validierung Ausblendung Strukturinformationen In den Testfällen-Nr. 25,26 (Tabelle 9.8) wird validiert, ob eine Ausblendung von Elementen, die zu einer Inkompatibilität führen, korrekt erkannt werden. Die Testinstanzen unterscheiden sich in der Anzahl an Interfaces und der Zuordnung der Pins zu den Interfaces. Daher ist das Ergebnis der Vergleichsmethoden für Testfall-Nr. 25 *inkompatibel* bzw. *nicht äquivalent*. Würde man die Interfaces ausblenden, wären beide Testinstanzen äquivalent. In Testfall-Nr.

Testszenarien				XML-CompA			Äquivalenzvergleich			X-Diff Vergleich	
Testfall-Nummer	Testinstanz$_1$(SG$_1$)	Testinstanz$_2$(SG$_2$)	Status	Mapping-Aussage [%]	Rückwärts-kompatibiliäts-Aussage [%]	Rückwärts-kompatibilität	Äquivalenz-Aussage	Äquivalenz-Aussage [%]	äquivalent / nicht äquivalent	X-Diff-Aussage	äquivalent / nicht äquivalent
Ausblendung von Strukturinformationen											
25	**SMG_einl**	**SMG_zweil** zwei Interfaces (ohne Ausblenden)	ok	74,73%	100%	nicht rückwärtskompatibel	2483/4705	52,77%	nicht äquivalent	1	nicht äquivalent
26	**SMG_einl**	**SMG_zweil** zwei Interfaces (mit Ausblenden)	ok	100%	100%	rückwärtskompatibel	1969/1969	100,00%	äquivalent	1	nicht äquivalent
Optionale oder nicht relevante Informationen weglassen											
27	**SMG**	**SMG_V8** **-** Optionale Elemente weglassen	ok	100%	100%	rückwärtskompatibel	5811/6033	96,32%	nicht äquivalent	12	nicht äquivalent
28	**-** Optionale Elemente	**SMG**	ok	100%	100%	rückwärtskompatibel	5811/6033	96,32%	nicht äquivalent	12	nicht äquivalent
29	**SMG**	**SMG_String** -Gewichtung(String) ausblenden	ok	100%	100%	rückwärtskompatibel	6027/6033	99,90%	nicht äquivalent	3	nicht äquivalent
Abhängigkeiten											
30	**SMG_V11**	**SMG_V12** 2 Abhängigkeiten wurden verändert	ok	100%	99,83%	nicht rückwärtskompatibel	6031/6033	99,97%	nicht äquivalent	2	nicht äquivalent
Sonstige Vergleichstest											
31	**ZKE**	**JBBF3_v2**	ok	53,21%	42,51%	nicht rückwärtskompatibel	5963/610008	0,98%	nicht äquivalent	948	nicht äquivalent

Tabelle 9.8: Ergebnisse der XML-Vergleichsmethoden (3. Teil)

26 wird dies überprüft. Durch das Ausblenden erkennt *XML-CompA* die Testinstanzen als *rückwärtskompatibel*. Der *Äquivalenzvergleich* wurde ebenfalls im Rahmen des *CompA*-Projekts implementiert und besitzt damit auch das Feature, dass ein Ausblenden von Strukturinformationen möglich ist. Daher kann dieser *Äquivalenzvergleich* die Testinstanzen als *äquivalent* erkennen. Der *X-Diff Vergleich* bietet keine Möglichkeit für einen Vergleich spezifische Informationen auszublenden und daher ist die *X-Diff-Aussage*: *nicht äquivalent*.

Validierung sonstiger Features Die Testfälle mit Nr. 27-30 (Tabelle 9.8) stellen weitere spezifische Testfälle dar, die nur mit von *XML-CompA* erkannt werden können.

- optionale oder nicht relevante Informationen wegzulassen,

- Abhängigkeiten zu berücksichtigen.

Ein *X-Diff Vergleich* und ein *Äquivalenzvergleich* kann nur erkennen, dass die Instanzen nicht äquivalent sind. In der Tabelle wird deutlich, dass immer, wenn *XML-CompA* Testinstanzen als *nicht rückwärtskompatibel* detektiert, diese auch nicht äquivalent sind. Sind die Testinstanzen jedoch *nicht äquivalent*, können diese dennoch zueinander *rückwärtskompatibel* sein.

9.3 Auswertung

9.3.1 Gegenüberstellung unterschiedlicher Vergleichsmethoden

Im folgenden Abschnitt werden die Merkmale der Vergleichsmethoden *XML-CompA*, *Äquivalenzvergleich* und Vergleich mit *X-Diff Algorithmen* einander gegenübergestellt. Hierzu wurden zwei Netzdiagramme erstellt. Für die Netzdiagramme wurden spezifische Testfälle herausgegriffen und bewertet. Das Kriterium für die Bewertung war, ob die Vergleichsmethoden erkennen, dass zwei XML-SG-Instanzen zu einander

- rückwärtskompatibel,
- äquivalent,
- nicht äquivalent / nicht kompatibel

sind. Die Basis für die Bewertung sind die Ergebnisse aus den Tabellen 9.6, 9.7 und 9.8. In Abbildung 9.4 sind die ersten sechs spezifischen Testfälle dargestellt. Der erste Testfall (vgl. Testfall Nr. 1,2) sind äquivalente XML-SG-Instanzen. Die Äquivalenz wird vom *Äquivalenzvergleich* und den X-Diff Algorithmen korrekt erkannt. Die Methode *XML-CompA* bewertet die XML-SG-Instanzen als rückwärtskompatibel. Auch diese Aussage ist korrekt.

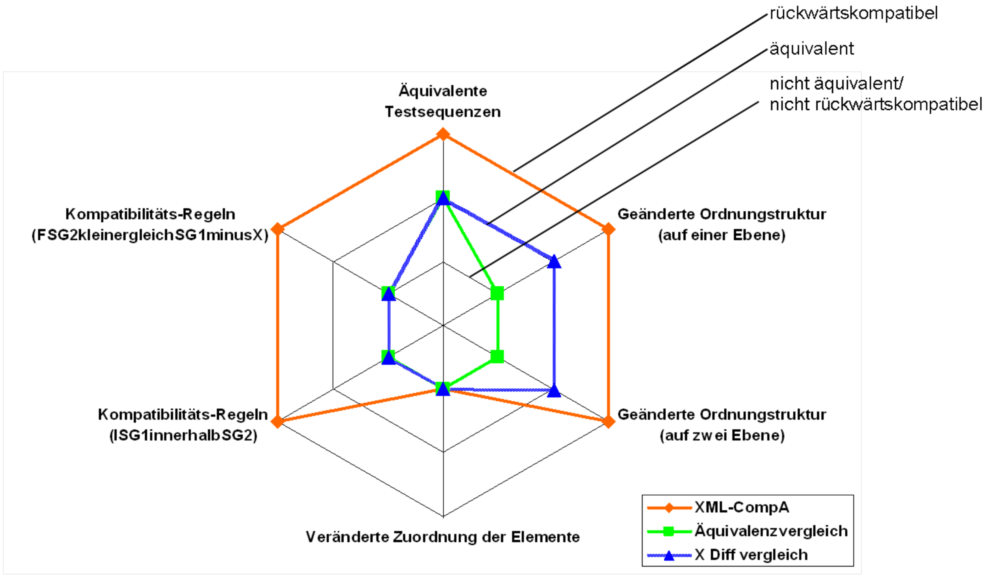


Abbildung 9.4: Auswertung (Teil 1) der Vergleichsmethoden

In den Testfällen *Geänderte Ordnungsstruktur* wurde eine links-rechts Vertauschung

von ungeordneten Elementen (vgl. unordered tree Kapitel 2.2.3) auf einer (bspw. Testfall Nr. 3) und auf zwei Hierarchie-Ebenen (Testfall Nr. 4,5,6) vorgenommen. Der *Äquivalenzvergleich* erkennt die Elemente als nicht äquivalent. Der *X-Diff Algorithmus* kann die ungeordneten Strukturen korrekt erkennen und betrachtet beide Instanzen als äquivalent. *XML-CompA* erkennt die geänderte Ordnungsstruktur ebenfalls korrekt und bewertet daher die Instanzen als rückwärtskompatibel.
Wird die hierarchische Zuordnung von einzelnen Elementen geändert (Testfall Nr. 9), so erkennen alle Vergleichsmethoden, dass die Instanzen nicht äquivalent/nicht kompatibel sind.
Werden Werte von XML-SG-Instanzen innerhalb eines spezifischen Wertebereichs verändert (vgl. Testfall Nr. 17,25), können diese trotz der Unterschiedlichkeit rückwärtskompatibel sein. Dies kann mittels Kompatibilitätsregeln überprüft werden. Diese Möglichkeit bietet nur *XML-CompA*. Wie in der Abbildung 9.4 deutlich zu sehen, werden aufgrund der Kompatibilitätsregeln *FSG2kleinergleichSG1minusX* und *ISG1innerhalbSG2* die XML-SG-Instanzen als kompatibel erkannt. In Abbildung 9.5

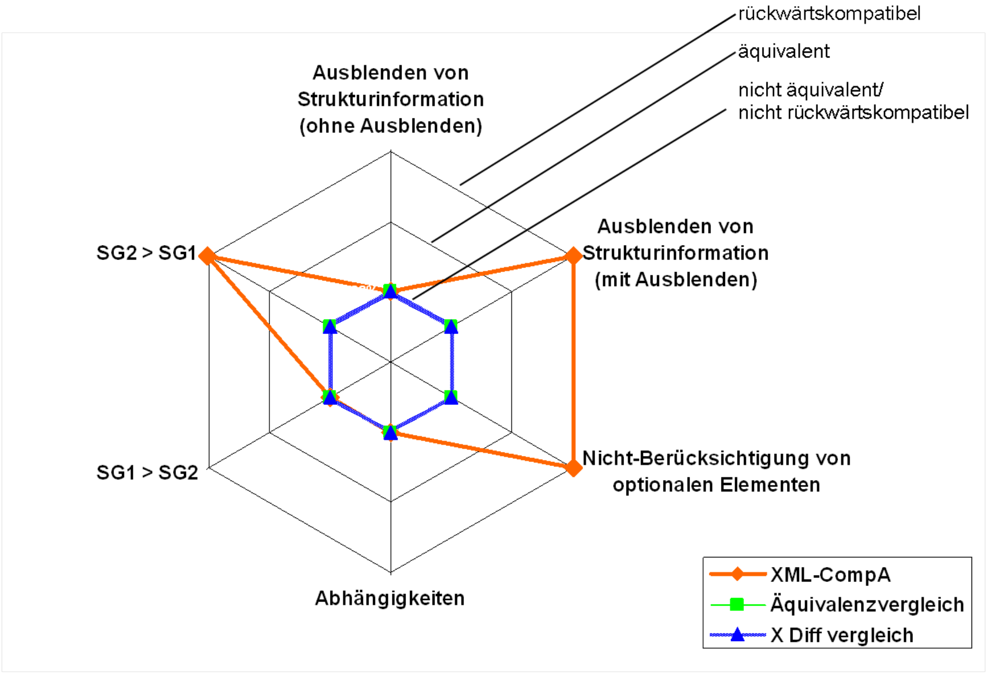


Abbildung 9.5: Auswertung (Teil 2) der Vergleichsmethoden

sind die zweiten sechs spezifischen Testfälle dargestellt. Zuerst werden zwei XML-SG-Instanzen verglichen, die nicht kompatibel sind (ohne Ausblenden von Strukturinformation s. Testfall Nr. 26) aber durch Ausblenden von spezifischen Strukturinformationen als rückwärtskompatibel betrachtet werden können. *XML-CompA* erkennt erkennt beide Fälle korrekt. *X-Diff Algorithmen* und *Äquivalenzvergleiche* bieten nicht die Möglichkeit, spezifische Elemente für einen Vergleich auszublenden.

Des Weiteren bietet *XML-CompA* die Möglichkeit, dass Elemente, die als optional gekennzeichnet sind, nicht bei der Analyse berücksichtigt werden. Optionale Elemente sind Elemente, die in einer XML-SG-Instanz spezifiziert sein können, aber nicht müssen. Daher müssen diese bei der Rückwärtskompatibilitätsanalyse auch nicht berücksichtigt werden. Da sich die beiden XML-SG-Instanzen (vgl. Testfall Nr. 28) nur in den optionalen Elementen unterscheiden, erkennt *XML-CompA* diese als rückwärtskompatibel und die anderen beiden Methoden als nicht äquivalent.

Mittels Abhängigkeiten kann gekennzeichnet werden, dass beispielsweise ein spezifisches Signal an einem bestimmten Pin anliegt. Unterscheiden sich zwei ansonsten identische XML-SG-Instanzen in ihren Abhängigkeiten (vgl. Testfall Nr. 31), so sind diese weder äquivalent noch rückwärtskompatibel.

Werden in SG_1 mehr Elemente spezifiziert als in SG_2 (vgl. Testfall Nr. 7), so stellt SG_1 keine Teilmenge mehr von SG_2 dar und kann daher nicht mehr rückwärtskompatibel und/oder äquivalent sein.

Werden in SG_2 mehr Elemente als in SG_1 spezifiziert, dann kann SG_1 eine echte Teilmenge von SG_2 sein (vgl. Testfall Nr. 8).

Die Abbildungen 9.4 und 9.5 zeigen anschaulich, dass *XML-CompA* die in Kapitel 6.1 aufgeführten Herausforderungen erfüllt. Hierfür wurden in *XML-CompA* spezifische Methoden integriert, die heutige XML-Vergleichsmethoden (bspw. *X-Diff Algorithmen*) nicht bieten.

9.3.2 Evaluierung Rechenzeit

In diesem Abschnitt wird die Rechenzeit von *XML-CompA* analysiert. Für die Schritte *Mapping* und *Kompatibilitätsanalyse* wurden Zeitmessungen implementiert. Die einzelnen Rechenzeiten wurden für jeden Testfall separat erfasst und in den Tabellen 9.2 bis 9.5 mit eingetragen. Für das Mapping ergibt sich über die 31 Testfälle eine durchschnittliche Rechenzeit von ca. 2,19s pro Testfall. Für die Kompatibilitätsanalyse ergibt sich eine durchschnittliche Rechenzeit von ca. 204ms. Die Kompatibilitätsanalyse ist damit im Schnitt um den Faktor 10 schneller als das Mapping. Hier zeigt sich der Vorteil der 4-Schritt-Methode von XML-CompA. Durch das Mapping werden die Elemente vorab einander zugeordnet. Durch die korrekte hierarchische Zuordnung der Elemente kann die Analyse der Rückwärtskompatibilität auf einen Attribut-/Strukturvergleich von spezifischen XML-Elementen beschränkt werden. Hierdurch erklärt sich auch der große Unterschied in der Rechenzeit zwischen *Mapping* und *Kompatibilitätsanalyse*.

Die längste Rechenzeit wird für das Mapping in Testfall-Nr. 26 mit 17,15s benötigt. In diesem Beispiel müssen, aufgrund der Ausblendung des Interfaces, auf der obersten Ebene

- 24 Pins von SG_1 zu 24 Pins von SG_2
- 24 Signale von SG_1 zu 24 Signale von SG_2

zugeordnet werden. Dadurch werden alleine auf der obersten Hierarchie-Ebene $(24 \cdot 24) \cdot 2 = 1152$ Vergleiche an Clustern durchgeführt.

9.4 Zusammenfassung

In diesem Kapitel wurde die Validierung der XML-Vergleichsmethode *XML-CompA* beschrieben.
Zur Validierung von *XML-CompA* wurden spezifische XML-SG-Instanzen angelegt, anhand derer alle Herausforderungen aus Kapitel 6.1 wie z.B. Berücksichtigung von Kompatibilitätsregeln oder Abhängigkeiten validiert werden konnten. Im Ganzen wurden hierzu 31 Testfälle vorgestellt. Die Validierung der Testfälle zeigte, dass alle eingebauten Unterschiede in den Testinstanzen korrekt erkannt wurden. Somit konnte auch die korrekte Funktion von *XML-CompA* nachgewiesen werden.
Zur Validierung der Leistungsfähigkeit von *XML-CompA* wurden die Testfälle mit zwei weiteren Vergleichsmethoden (*X-Diff Algorithmus, Äquivalenzvergleich*) analysiert. Hier wurde deutlich, dass *XML-CompA* speziell zur Analyse von Rückwärtskompatibilität entwickelt wurde und entscheidende Vorteile gegenüber den beiden „herkömmlichen“ Vergleichsmethoden aufweist.

10 Validierung der MSC-Vergleichsmethode (*MSC-CompA*)

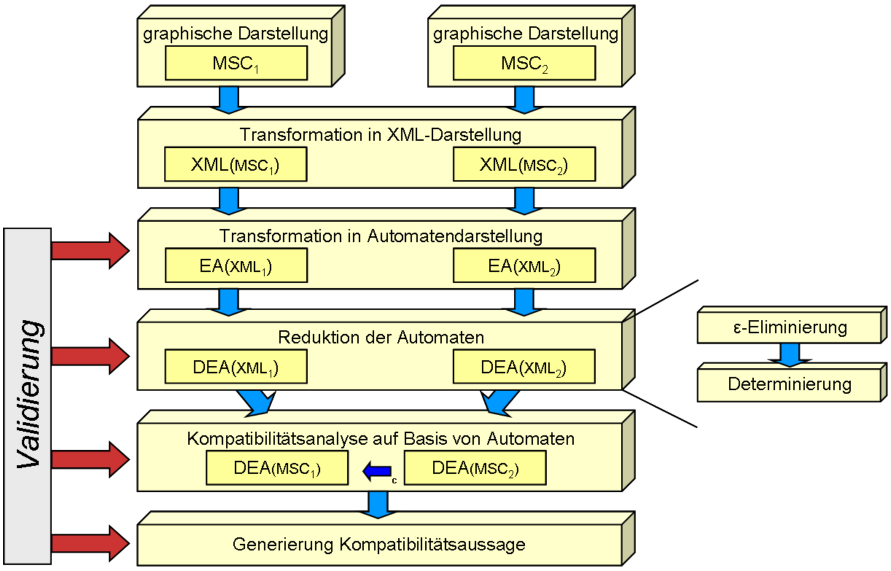


Abbildung 10.1: Konzept zur Validierung der MSC-Vergleichsmethode (*MSC-CompA*)

In diesem Abschnitt wird die Validierung der MSC-Vergleichsmethode *MSC-CompA* vorgestellt. Hierzu werden die Ergebnisse der vier wesentlichen Schritte des Konzeptes betrachtet und auf Korrektheit überprüft (vgl. Abbildung 10.1):

- Transformation in Automatendarstellung.
- Determinierung der Automaten: Die Determinierung der Automaten unterteilt sich wiederum in zwei Schritte, deren Ergebnisse ebenfalls mit validiert werden:
    1. ϵ-Eliminierung
    2. Determinierung
- Kompatibilitätsanalyse auf Basis von Automaten.
- Generierung von Kompatibilitätsaussagen.

Die Validierung des Schritts *Transformation in XML-Darstellung* ist nicht explizit notwendig, da dieser durch die Validierung der nachfolgenden Schritte bereits mit abgedeckt ist. Würde die *XML-Transformation* falsche Ergebnisse liefern, wären die darauf aufbauenden Automaten ebenfalls nicht korrekt.
Die Beschreibung der Validierung von MSC-CompA gliedert sich in folgende Schwerpunkte:

1. Validierung der Transformationsregeln zur Generierung endlicher Automaten. Hierbei wird auch die Determinierung der Automaten implizit validiert.
2. Validierung der Rückwärtskompatibilitätsanalyse auf Basis der generierten Automaten.
3. Bewertung der Leistungsfähigkeit des Tools.

10.1 Validierung der Transformationsregeln von Basic-MSCs und Inline-Expressions

Die Transformationsregeln zur Erzeugung endlicher Automaten auf Basis von MSCs wurden entsprechend dem in Kapitel 7 beschriebenen Konzept implementiert. In diesem Abschnitt wird nun die Validierung dieser Transformationsregeln vorgestellt. Die Validierung der Transformationsregeln wurde dabei stufenweise aufgebaut, d.h. von einfachen MSC-Konstrukten zu komplexen MSC-Konstrukten (vgl. Abbildung 10.2):

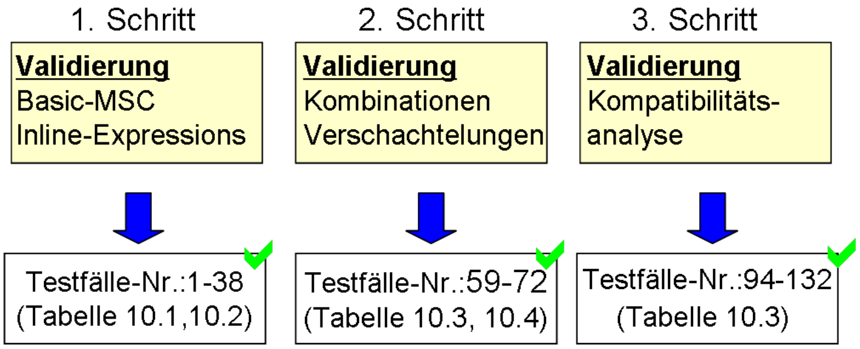


Abbildung 10.2: Vorgehen zur Validierung

1. Validierung der einzelnen Basic-MSC Sprachkonstrukte (Instanzen, Ereignisse, Action, Timer, Condition).
2. Validierung der einzelnen Inline-Expressions (Optional, Alternative, Loop, Coregion, Parallel, Exception).

3. Validierung von MSC-Sprachkonstrukten in Kombination mit weiteren Sprachkonstrukten (z.B. Alternative + Optional) oder verschachtelt (z.B. Alternative innerhalb Optional).

Insgesamt wurden ca. 132 MSC-Testfälle für die Validierung erstellt. Bei allen Testfällen wurden die generierten Automaten auf Vollständigkeit und Korrektheit geprüft. Zur Vereinfachung der Validierungs-Beschreibung werden die generierten Automaten nicht konkret vorgestellt, sondern die Anzahl der erzeugten Zustände und Übergangsfunktionen auf Richtigkeit plausibilisiert.

Vergleichskonzept Die Ergebnisse der Validierung wurden in mehreren Tabellen 10.1, 10.2 und 10.3 zusammengefasst. Die Tabellen folgen einem spezifischen Aufbau. Jede Zeile entspricht einem MSC-Testfall und in den Spalten sind die Informationen wie folgt hinterlegt:

- (1. Spalte) Eindeutige Nr. des MSC-Testfalls.
- (2. Spalte) Beschreibung des MSC-Sprachkonstrukts.
- (3./4. Spalte) Beschreibung der MSC-Konstrukte, d.h. aus wie vielen Instanzen und Ereignissen das MSC aufgebaut ist.
- (5. Spalte) Status, ob der Testfall das richtige Ergebnis geliefert hat.
- (6.-12. Spalte) Beschreibung der Ergebnisse der Automaten-Transformation. Diese gliedern sich in folgende Punkte:
  - (6. Spalte) Rechenzeit für Automaten-Transformation.
  - (7.-12. Spalte) Transformations-Ergebnis: Jede Instanz eines MSCs wird in einen separaten Automaten transformiert. Für die Validierung werden MSCs mit 1 bis 2 Instanzen verwendet. Daher können pro MSC-Testfall 1-2 Automaten auftreten. Für jeden Automat werden dabei folgende Werte abgespeichert:
    - Zustände: Anzahl der Zustände, die bei der Transformation erzeugt werden.
    - Übergangsfunktionen: Anzahl der Übergangsfunktionen, die bei der Transformation erzeugt werden. Aber ohne ϵ-Übergangsfunktionen.
    - ϵ-Übergangsfunktionen: Anzahl der ϵ-Übergangsfunktionen, die bei den Transformationen zusätzlich entstehen.
- (13.-19. Spalte) Beschreibung der Ergebnisse nach der Determinierung. Der Schritt ϵ-Eliminierung ist eine Voraussetzung für die Determinierung. Im Folgenden wird daher die ϵ-Eliminierung der Determinierung immer implizit untergeordnet.
  - (13. Spalte) Rechenzeit für Determinierung.

- (14.-17. Spalte) Aufbau der erzeugten deterministischen Automaten (DEA). Für jeden DEA werden dabei folgende Werte abgespeichert:
    - Zustände: Anzahl der Zustände, nach der ϵ-Eliminierung
    - Übergangsfunktionen: Anzahl der Übergangsfunktionen, die bei der Transformation erzeugt werden. ϵ-Übergänge werden aufgrund der ϵ-Eliminierung nicht weiter berücksichtigt.

10.1.1 Transformationsregeln von Basic-MSC-Konstrukten

In Tabelle 10.1 sind spezifische Testfälle bzw. Test-Szenarien dargestellt, mit Hilfe derer die Transformationsregeln der einzelnen Basic-MSC-Konstrukte (*Instanzen*, *Ereignisse*, *Action*, *Timer*, *Condition*, *Comment*) validiert werden. In Testfall-Nr. 1 existiert eine Instanz ohne Ereignis. Instanzen dienen als Ausgangspunkt für die Dekomposition,

		MSC-Elemente			Transformation endliche Automaten							nach Determinierung				
						Automat 1 (^= 1. Instanz)			Automat 2 (^= 2. Instanz)				Automat 1 (^= 1. Instanz)		Automat 2 (^= 2. Instanz)	
Tesfall Nr.	**MSC-Sprachkonstrukte**	Anzahl Instanzen	Anzahl Ereignisse	Status	Rechenzeit Transformation [ms]	Zustände	\epsilon-Übergangsfunktionen	Übergangsfunktionen	Zustände	\epsilon-Übergangsfunktionen	Übergangsfunktionen	Rechenzeit Determinierung [ms]	Zustände	Übergangsfunktionen	Zustände	Übergangsfunktionen
	Basic-MSC-Konstrukte															
	Instanzen															
1	1 Instanz (+ kein Ereignis)	1	0	ok												
2	1 Instanz (+ Ereignis von Environment)	1	1	ok	47	2	0	1				2	2	1		
3	2 Instanzen (+ Ereignis zwischen den Instanzen)	2	1	ok	141	2	0	1	2	0	1	2	2	1	2	1
	Ereignisse															
5	von Gate	1	1	ok	62	2	0	1				2	2	1	0	0
6	von Environment	1	1	ok	47	2	0	1				2	2	1	0	0
7	zwischen Instanzen	2	1	ok	125	2	0	1	2	0	1	2	2	1	2	1
	Action															
8	Action-Block (über 1 Instanz)	1	0	ok	47	2	0	1				2	2	1		
	Timer															
9	Timer-Start	1	0	ok	47	2	0	1				2	2	1		
10	Time-out	1	0	ok	39	2	0	1				2	2	1		
11	Timer-Stopp	1	0	ok	47	2	0	1				2	2	1		
	Condition															
14	Condition-Block (über 1 Instanz)	1	0	ok	39	2	0	1				2	2	1		
15	Condition-Block (über 2 Instanzen)	2	0	ok	71	2	0	1	2	0	1	2	2	1	2	1
	Comment															
16	Comment-Block (über 2 Instanzen)	2	0	ok												

Tabelle 10.1: Ergebnis der Validierung von Basic-MSC Konstrukten

d.h. pro Instanz ist ein separater Automat anzulegen. Eine Instanz ist zwar eine notwendige Voraussetzung für ein MSC, allerdings leistet die Instanz selbst keinen Beitrag zum Automaten.
Jedes einzelne Basic-MSC Konstrukt (Testfall-Nr. 2-15) wird bei der Transformation in einen endlichen Automaten überführt, der aus einem *initialen Zustand*, einer *Übergangsfunktion* und einem *finalen Zustand* besteht (vgl. Kapitel 7.4.3). Dies bestätigen die Testfälle in Tabelle 10.1. In allen Testfällen ist das Ergebnis der Transformation eines Basic-MSC-Konstrukts jeweils 2 Zustände und eine Übergangsfunktion. Bei der Transformation eines Basic-MSC-Konstrukts wird kein ϵ-Übergang erzeugt.
In der Tabelle wird deutlich, dass jede dekomponierte Instanz genau einem Automaten entspricht. Die Testfälle-Nr. 2,5,6,8,9,10,14 bestehen aus einer Instanz und haben als Ergebnis daher einen Automaten. Die Testfälle-Nr. 3,7,15 besitzen zwei Instanzen und damit auch zwei korrespondierende Automaten.
Das *Comment*-Element (Testfall Nr. 16) hat keinen Einfluss auf die Ereignissequenzen und wird daher auch nicht transformiert.

10.1.2 Transformationsregeln von Inline-Expressions

In Tabelle 10.2 ist die Validierung der Inline-Expressions dargestellt. Die Validierungen folgender Inline-Expressions bzw. Operatoren werden hier vorgestellt: *Optional*, *Alternative*, *Loop*, *Coregion*, *Parallel*, *Exception*.
Inline-Expressions sind Operatoren, mit Hilfe derer spezifische Ereignissequenzen erzeugt werden können. Voraussetzung hierfür ist, dass ein Ereignis oder ein anderes Basic-MSC Konstrukt innerhalb einer Inline-Expression vorhanden ist. „Leere“ Inline-Expressions werden daher bei der Validierung nicht weiter betrachtet.
Bei der Transformation von Inline-Expressions kommen ϵ-Übergänge zum Einsatz. Diese werden durch die ϵ-Eliminierung entfernt. Im Folgenden wird kurz auf die einzelnen Testfälle eingegangen.

Optional In den Testfällen-Nr. 18,19 wird die Transformation des Optional-Operators validiert. Bei der Transformation werden zuerst alle Elemente innerhalb des Operators transformiert.
In beiden Testfällen ist ein Ereignis vorhanden und daher wird jeweils ein Basis-Automat bestehend aus einem *initialen*, einem *finalen Zustand* und einer Übergangsfunktion erzeugt. Zusätzlich wird der *initiale* und der *finale Zustand* über eine ϵ-*Übergangsfunktion* verbunden (vgl. Kapitel 7.4.3). Hierdurch wird modelliert, dass ein Automat ausgeführt werden kann, aber nicht muss. Zusammengefasst werden bei der Transformation 2 Zustände, 1 ϵ-Übergang und 1 Übergangsfunktion erzeugt (vgl. Testfall-Nr. 18).
In Testfall-Nr. 19 erstreckt sich der der Optional-Operator über 2 Instanzen. Da pro Instanz ein Automat generiert wird, werden bei diesem Testfall also zwei Automaten erzeugt.

Tesfall Nr.	MSC-Sprachkonstrukte	MSC-Elemente		Status	Transformation endliche Automaten							nach Determinierung				
						Automat 1 (^= 1. Instanz)			Automat 2 (^= 2. Instanz)				Automat 1 (^= 1. Instanz)		Automat 2 (^= 2. Instanz)	
		Anzahl Instanzen	Anzahl Ereignisse		Rechenzeit Transformation [ms]	Zustände	\epsilon-Übergangsfunktionen	Übergangsfunktionen	Zustände	\epsilon-Übergangsfunktionen	Übergangsfunktionen	Rechenzeit Determinierung [ms]	Zustände	Übergangsfunktionen	Zustände	Übergangsfunktionen
	Inline-Expressions															
	Optional															
18	Optional (über 1 Instanzen + 1 Ereignis von Env.)	1	1	ok	62	2	1	1				2	2	1		
19	Optional (über 2 Instanzen + 1 Ereignis)	2	1	ok	107	2	1	1	2	1	1	9	2	1	2	1
	Alternative															
21	Alternative: 1 Section (über 1 Instanzen + 1 Ereignis v. Env)	1	1	ok	62	4	2	1				2	2	1		
22	Alternative: 1 Section (über 2 Instanzen + 1 Ereignis)	2	1	ok	63	4	2	1	4	2	1	2	2	1	2	1
23	Alternative: 2 Section (über 2 Instanzen + 1 Ereignis/Section)	2	2	ok	70	6	4	2	6	4	2	17	3	2	3	2
	Loop															
25	Loop <1> (über 2 Instanzen + 1 Ereignis)	2	1	ok	55	2	0	1	2	0	1	2	2	1	2	1
26	Loop <10> (über 2 Instanzen + 1 Ereignis)	2	1	ok	54	20	9	10	20	9	10	2	11	10	12	10
27	Loop <0,10> (über 2 Instanzen + 1 Ereignis)	2	1	ok	66	20	19	10	20	19	10	17	11	10	12	10
28	Loop <inf> (über 2 Instanzen + 1 Ereignis)	2	1	ok	56	3	1	2	3	1	2	2	3	1	3	1
29	Loop <0,inf> (über 2 Instanzen + 1 Ereignis)	2	1	ok	62	2	2	1	2	2	1	2	1	1	1	1
	Co-Region															
31	Co-Region (über 1 Instanzen + 1 Ereignis von Env)	1	1	ok	94	4	2	1				2	3	1		
32	Co-Region (über 2 Instanzen + 1 Ereignis)	2	1	ok	94	4	2	1	4	2	1	2	3	1	3	1
33	Co-Region (über 2 Instanzen + 2 Ereignis)	2	2	ok	62	10	6	4	10	6	4	2	6	4	6	4
34	Co-Region (über 2 Instanzen + 3 Ereignis)	2	3	ok	109	38	24	18	38	24	18	2	17	15	17	15
	Parallel															
36	Parallel: 2 Section (über 2 Instanzen + 1 Ereignis)	2	1	ok	115	4	0	4	4	0	4	17	4	4	4	4
37	Parallel: 2 Section (über 2 Instanzen + 2 Ereignis/Section)	2	2	ok	164	9	0	14	9	0	12	32	9	12	9	12
	Exception															
38	Exception (über 2 Instanzen: 1 Ereignis innerhalb + 1 Ereignis nach Exception)	2	2	ok	32	5	2	2	5	2	2	2	3	2	3	2

Tabelle 10.2: Ergebnisse der Validierung von Inline-Expressions

Alternative In den Testfällen-Nr. 21-23 wird der *Alternative*-Operator validiert. Wie beim *Optional*-Operator werden zuerst die Elemente innerhalb des Operators in Automaten transformiert. Eine Besonderheit des *Alternative*-Operators ist, dass dieser aus mehreren *Sections* bestehen kann. Die *Sections* werden über Vereinigung miteinander verbunden. Bei einer Vereinigung werden 2 zusätzliche Zustände und pro zu vereinigenden Automat 2 ϵ-Übergänge erzeugt. Der Mechanismus der Vereinigung findet auch

dann statt, wenn der *Alternative*-Operator nur aus einer *Section* besteht.
Beispiel (Testfall-Nr. 21): Das MSC besteht aus einer Section mit einem Ereignis. Der generierte Automat hat demnach 4 Zustände. 2 Zustände durch die Transformation des Ereignisses und 2 Zustände aus der Vereinigung. Des Weiteren werden bei der Transformation 1 Übergangsfunktion und bei der Vereinigung 2 ϵ-Übergänge erzeugt.
Beispiel (Testfall-Nr. 23): Dieses MSC hat 2 Sections mit jeweils 1 Ereignis. Bei der Transformation der *Sections* entstehen 2 Automaten, mit jeweils 2 Zuständen und 1 Übergangsfunktion. Durch die Vereinigung der Automaten werden 2 weitere Zustände und 4 ϵ-Übergänge hinzugefügt. Damit muss der transformierte Automat aus 6 Zuständen, 4 ϵ-Übergängen und 2 Übergangsfunktionen bestehen.

Loop In den Testfällen-Nr. 25-29 werden die Transformationsregeln für den Loop-Operator validiert. Hierbei werden die spezifischen Fälle des Loop-Operators unterschieden:

- **Loop**$<\ n\ >$**:** In Testfall-Nr. 25 (Loop$<\ 1\ >$) ist 1 Ereignis innerhalb einer Loop dargestellt. Bei der Transformation wird das Ereignis in einen Automaten (2 Zustände, 1 Übergangsfunktion) transformiert. In Testfall-Nr. 26 (Loop$<\ 10\ >$) wird die Loop 10-mal wiederholt. Hierzu wird der Automat, der bei der Transformation des Loop-Blocks entsteht, 10 mal hintereinander konkateniert. In diesem Testfall erhält man also einen Automaten bestehend aus *10·2 Zustände*, *10·1 Übergangsfunktionen* und *9 ϵ-Übergänge* aus der Konkatenation.
- **Loop**$<\ n,m\ >$**:** Der Aufbau des Testfall-Nr. 27 (Loop$<\ 0,10\ >$) unterscheidet sich von Testfall-Nr. 26 nur im Parameter des Loop-Operators. Für die Transformation der Anzahl an minimalen Wiederholungen werden zusätzliche ϵ-Übergänge verwendet. Da die minimale Wiederholungsanzahl gleich 0 ist, darf jeder zu konkatenierende Automat „übersprungen“ werden. Daher wird der initiale und der finale Zustand jedes zu konkatenierenden Automaten über einen ϵ-Übergang verbunden. Der generierte Automat von (Loop$<\ 0,10\ >$) hat somit 10 ϵ-Übergänge mehr als der Automat von (Loop$<\ 10\ >$).
- **Loop**$<\ inf\ >$**:** Der Loop-Operator von Testfall-Nr. 28 hat 1 Ereignis. Dieses wird in einen Automaten transformiert und der *finale Zustand* mit dem *initialen Zustand* über einen *ϵ-Übergang* verbunden. Anschließend wird 1 weiterer Zustand hinzugefügt (dies wird auch der neue *finale Zustand*) und über eine neue Übergangsfunktion „*inf*“ verbunden. Das Ergebnis der Transformation ist damit ein Automat bestehend aus 3 Zuständen, 1 ϵ-Übergang und 2 Übergangsfunktionen.

Coregion In den Testfällen-Nr. 31-34 wird die Transformation der *Coregion* validiert. Für Ereignisse innerhalb einer Coregion ist die zeitliche Reihenfolge aufgehoben. Die somit erlaubte Permutation der Ereignisse wird durch die spezifische Transformationsregel berücksichtigt. Hierfür werden alle möglichen Ereignissequenzen des MSC

auf einzelne Automaten abgebildet und anschließend werden diese vereinigt.
Beispiel (Testfall-Nr. 33): Die Coregion enthält 2 Ereignisse. Damit sind $2! = 2$ Ereignissequenzen möglich. Jede Ereignissequenz bzw. der zugehörige Automat wird aus den 2 Ereignissen durch Permutation der Ereignisse aufgebaut. Hierbei wird jedes Ereignis in einen Automaten mit 2 Zuständen und 1 Übergangsfunktion transformiert. Anschließend werden die Automaten der beiden Ereignisse über einen ϵ-Übergang konkateniert (entsprechend der Permutation). Somit entsteht eine Ereignissequenz aus jeweils 4 Zuständen, 1 ϵ-Übergang und 2 Übergangsfunktionen. Diese Ereignissequenzen bzw. die korrespondieren Automaten werden abschließend über Vereinigung zusammengefügt. Somit entsteht ein Automat, der aus [2·(4 Zuständen pro Ereignissequenz) + 2 Zustände aus Vereinigung] = 10 Zuständen, [2·(1 ϵ-Übergang pro Ereignissequenz) + 4 ϵ-Übergänge aus Vereinigung]=6 ϵ-Übergänge und [2·2 Übergangsfunktionen] = 4 Übergangsfunktionen besteht. Die Plausibilitätsprüfung der weiteren Testfälle erfolgt analog.

Parallel In den Testfällen-Nr. 36-37 werden die Transformationsregeln für den *Parallel*-Operator validiert. Bei der Transformation wird jede Section separat in ein Automaten transformiert und anschließend werden die Automaten zu einem Produktautomaten zusammengefügt.
Beispiel: In Testfall-Nr. 36 wird jede Section in einen Automaten bestehend aus 2 Zuständen und 1 Übergangsfunktion transformiert. Der erzeugte Produktautomat besteht also aus $(2 \cdot 2) = 4$ Zuständen und aus 4 Übergangsfunktionen (vgl. Abbildung 10.3).

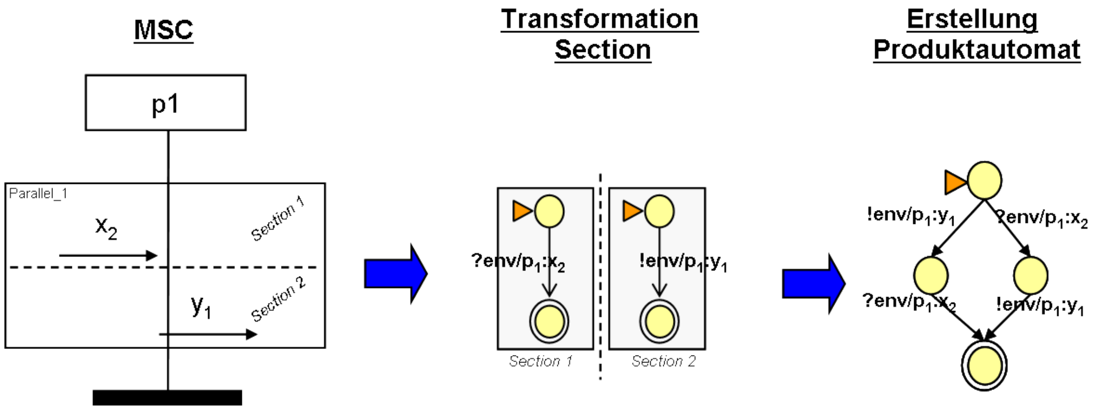


Abbildung 10.3: Transformation von Testfall-Nr. 36

Exception Bei der Transformation des Exception-Operators (vgl. Testfall-Nr. 38) werden zuerst alle MSC-Elemente innerhalb des Operators und anschließend alle MSC-Elemente nach dem Operator in jeweils einen separaten Automaten transformiert. In dem Testfall liegt 1 Ereignis innerhalb und 1 Ereignis nach der Exception und somit werden je 2 Automaten mit je 2 Zuständen und 2 Übergangsfunktionen erzeugt. Zusätzlich wird 1 neuer Zustand eingefügt, der die initialen Zustände der separaten Automaten über ϵ-Übergänge verbindet. Anschließend wird der neue Zustand als initialer Zustand gekennzeichnet. Insgesamt werden damit 5 Zustände, 2 Übergangsfunktionen und 2 ϵ-Übergänge erzeugt.

10.1.3 Validierung Transformationsregeln von kombinierten bzw. verschachtelten MSC-Konstrukten

In Tabelle 10.1 und 10.2 wurde gezeigt, dass die Transformationsregeln sowohl für die Basic-MSC und als auch die Inline-Expressions „valide“ sind. Um zu zeigen, dass auch komplexe Sprachkonstrukte transformiert werden können, wurden verschiedene MSC-Sprachkonstrukte miteinander kombiniert bzw. verschachtelt (vgl. Tabelle 10.3).

					Transformation endliche Automaten							nach Determinierung				
		MSC-Elemente				Automat 1 (^= 1. Instanz)			Automat 2 (^= 2. Instanz)				Automat 1 (^= 1. Instanz)		Automat 2 (^= 2. Instanz)	
Tesfall Nr.	**MSC-Sprachkonstrukte**	Anzahl Instanzen	Anzahl Ereignisse	Status	Rechenzeit Transformation [ms]	Zustände	\epsilon-Übergangsfunktionen	Übergangsfunktionen	Zustände	\epsilon-Übergangsfunktionen	Übergangsfunktionen	Rechenzeit Determinierung [ms]	Zustände	Übergangsfunktionen	Zustände	Übergangsfunktionen
	Kombination von MSC-Elemente															
59	1 Action 1 Condition Ereignisse (jeweils 1 Ereignis zischen Operatoren) 1 Ereignis von Environment 1 Ereignis von Gate 5 Ereignisse zwischen Instanzen (für Timer) Timer -Start-Timer -Stop-Timer -Time-Out	2	9	ok	4406	28	13	14	18	8	9	2	15	14	10	9
60	2 Ereignisse 1 Optinonal (mit 2 Ereignissen) 1 Alternative-Block (mit 2 Sections & 2 Ereignissen/Section)	2	8	ok	1641	18	11	8	18	11	8	2	9	10	9	10

Tabelle 10.3: Validierung von kombinierten MSC-Konstrukten

Kombination von Sprachkonstrukten

In Testfall Nr. 59/60 wurden verschiedene Sprachkonstrukte sequentiell aneinander gereiht. In Testfall-Nr. 59 werden an einer Instanz ($Instanz_1$) 14 Basic-MSC aneinander gereiht. Gemäß der Transformationsregeln besteht ein Automat eines Basic-MSC Konstrukts aus 2 Zuständen und 1 Übergangsfunktion. Die erzeugten Automaten werden anschließend über Konkatenation, also über ϵ-Übergänge miteinander verknüpft. Das Ergebnis für $Instanz_1$ ist somit ein Automat bestehend aus $14 \cdot 2 = 28$ Zuständen, $14 \cdot 1 = 14$ Übergangsfunktionen und 13 ϵ-Übergängen. An $Instanz_1$ liegen Signale von/zum Environment an, die nicht an der $Instanz_2$ anliegen. Dies erklärt den Unterschied an Zuständen zwischen den Automaten.

Verschachtelung von Inline-Expression

In diesem Abschnitt werden die Transformationsregeln anhand von verschachtelten Inline-Expressions validiert. Die Testfälle sind in der Tabelle 10.4 dargestellt. Im Folgenden werden ausgewählte Fälle exemplarisch vorgestellt.

Optional-Alternative In den Testfällen-Nr. 61-64 wird die Verschachtelung von *Optional* und *Alternative* validiert. Beispiel Testfall-Nr. 62. Das Szenario besteht aus einem *Alternative*-Operator mit zwei *Sections*. Innerhalb einer Section ist ein *Optional*-Operator mit 2 Ereignissen integriert. Jede *Section* von *Alternative* enthält 2 Ereignisse. Da sich die Operatoren über 2 Instanzen erstrecken, werden zwei Automaten generiert. Bei der Transformation der ersten *Section* (ohne *Optional*-Operator) werden die 2 Ereignisse in einen Automaten bestehend aus 4 Zuständen, 1 ϵ-Übergang und 2 Übergangsfunktionen transformiert. Bei der Transformation der zweiten Section werden die 2 Ereignisse ebenfalls in einen Automaten transformiert und anschließend mit dem Automaten des *Optional*-Operators konkateniert. Der Automat für den *Optional*-Operator besteht, aufgrund von 2 Ereignissen, aus 4 Zuständen, 2 ϵ-Übergängen und 2 Übergangsfunktionen. Anschließend werden die Automaten vereinigt (plus 2 Zustände und 4 ϵ-Übergänge). Als Ergebnis wird ein Automat generiert, bestehend aus 14 Zuständen, 9 ϵ-Übergängen und 6 Übergangsfunktionen.

Loop-Optional-Alternative In den Testfällen-Nr. 65-68 wurde die Verschachtelung mit Loop-Operatoren validiert. Dies sei beispielhaft an Testfall-Nr. 66 *(Loop* $< 0, 10 >$*)* erläutert. Der Testfall besteht aus zwei Instanzen p_1 und p_2. Das Signal x_1 wird innerhalb einer Loop mit der Parametrisierung $< 0, 10 >$ ausgetauscht. Innerhalb der Loop befindet sich ein *Optional*-Operator mit 2 Ereignissen x_3 und x_4. Der Optional-Operator wiederum enthält einen *Alternative*-Operator mit zwei *Sections*. Jede *Section* enthält wiederum zwei Ereignisse x_5 und x_6. Durch die Transformation entstehen zwei Automaten. Jeder erzeugte Automat enthält 132 Zustände, 91 ϵ-Übergänge und 161 Übergangsfunktionen. Durch die anschließende ϵ-Eliminierung und Determinierung werden alle ϵ-Übergänge eliminiert und die Anzahl der Zustände auf 71 und der Übergangsfunktionen auf 88 reduziert.

Alternative - (Optional) - Coregion In den Testfällen-Nr. 69-72 werden die Operatoren *Alternative*, *Optional* und *Coregion* verschachtelt. In Testfall-Nr. 72 (vgl. Abbildung 10.4) ist beispielsweise ein MSC mit einem Alternative-Operator und 2 Sections mit je 2 Ereignissen dargestellt. Innerhalb einer Section ist ein Optional-Operator mit 2 Ereignissen integriert, der zusätzlich eine Coregion mit 2 Ereignissen aufweist. Zur Transformation: Die Ereignisse der ersten Section werden in separate Automaten transformiert und anschließend konkateniert. In der zweiten Section wird das erste Ereignis

Tesfall Nr.	MSC-Sprachkonstrukte	MSC-Elemente		Status	Transformation endliche Automaten							nach Determinierung				
						Automat 1 (^= 1. Instanz)			Automat 2 (^= 2. Instanz)				Automat 1 (^= 1. Instanz)		Automat 2 (^= 2. Instanz)	
		Anzahl Instanzen	Anzahl Ereignisse		Rechenzeit Transformation [ms]	Zustände	\epsilon-Übergangsfunktionen	Übergangsfunktionen	Zustände	\epsilon-Übergangsfunktionen	Übergangsfunktionen	Rechenzeit Determinierung [ms]	Zustände	Übergangsfunktionen	Zustände	Übergangsfunktionen
	Verschachtelung Inline-Expressions															
	Optional- Alternative															
61	Optional - (über 2 Instanzen & 1 Ereignis) Alternative - (über 2 Instanzen & 1 Sections & 1 Ereignis/Section)	2	2	ok	125	6	4	2	6	4	2	2	3	2	3	2
62	Optional - (über 2 Instanzen & 2 Ereignisse) Alternative - (über 2 Instanzen & 2 Sections & 2 Ereignisse/Section)	2	6	ok	105	14	9	6	14	9	6	2	7	6	7	6
	Alternative - Optional															
63	Alternative - (über 2 Instanzen & 1 Sections & 1 Ereignis/Section) Optional - (über 2 Instanzen & 1 Ereignisse)	2	2	ok	281	6	4	2	6	4	2	2	3	2	3	2
64	Alternative - (über 2 Instanzen & 2 Sections & 2 Ereignisse/Section) Optional - (über 2 Instanzen & 2 Ereignisse)	2	6	ok	547	14	9	6	14	9	6	2	7	7	7	7
	Loop-Optional- Alternative															
65	Loop <1> - (über 2 Instanzen + 1 Ereignis) Optional - (über 2 Instanzen & 2 Ereignisse) Alternative - (über 2 Instanzen & 2 Sections & 2 Ereignisse/Section)	2	7	ok	304	16	10	7	16	10	7	2	8	7	8	7
66	Loop <0,10> - (über 2 Instanzen + 1 Ereignis) Optional - (über 2 Instanzen & 2 Ereignisse) Alternative - (über 2 Instanzen & 2 Sections & 2 Ereignisse/Section)	2	7	ok	1157	160	119	70	160	119	70	24	71	88	71	88
67	Loop <0,inf> - (über 2 Instanzen + 1 Ereignis) Optional - (über 2 Instanzen & 2 Ereignisse) Alternative - (über 2 Instanzen & 2 Sections & 2 Ereignisse/Section)	2	7	ok	473	16	12	7	16	12	7	17	8	10	8	10
68	Loop <inf> - (über 2 Instanzen + 1 Ereignis) Optional - (über 2 Instanzen & 2 Ereignisse) Alternative - (über 2 Instanzen & 2 Sections & 2 Ereignisse/Section)	2	7	ok	437	17	11	19	17	11	19	2	10	13	10	13
	Alternative - Co-Region															
69	Alternative - (über 2 Instanzen & 1 Section & 1 Ereignis/Section) Coregion - (über 2 Instanzen & 1 Ereignis)	2	2	ok	203	6	3	2	6	3	2	22	3	2	3	2
70	Alternative - (über 2 Instanzen & 2 Section & 2 Ereignis/Section) Coregion - (über 2 Instanzen & 2 Ereignis)	2	6	ok	250	20	13	8	14	8	6	17	9	8	7	6
	Alternative - Optional - Co-Region															
71	Alternative - (über 2 Instanzen & 1 Section & 1 Ereignis/Section) Optional - (über 2 Instanzen & 1 Ereignisse) Coregion - (über 2 Instanzen & 1 Ereignis)	2	3	ok	203	8	5	3	8	5	3	48	4	3	4	3
72	Alternative - (über 2 Instanzen & 2 Section & 2 Ereignis/Section) Optional - (über 2 Instanzen & 2 Ereignisse) Coregion - (über 2 Instanzen & 2 Ereignis)	2	8	ok	661	24	16	10	24	16	10	29	11	12	11	12

Tabelle 10.4: Validierung von verschachtelten MSC-Konstrukten

in einen Automaten transformiert und mit dem Automaten des Optional-Operators konkateniert. Der Optional-Operator enthält eine Coregion, die in einen Automaten bestehend aus 10 Zuständen, 6 ϵ-Übergängen und 4 Übergangsfunktionen transformiert und mit den Ereignissen x_1, x_2 konkateniert wird. Abschließend werden die Automaten der beiden Sections vereinigt. Das Ergebnis ist ebenfalls in Abbildung 10.4 dargestellt. Der Automat besteht aus 24 Zuständen, 16 ϵ-Übergängen und 10 Übergangsfunktionen.

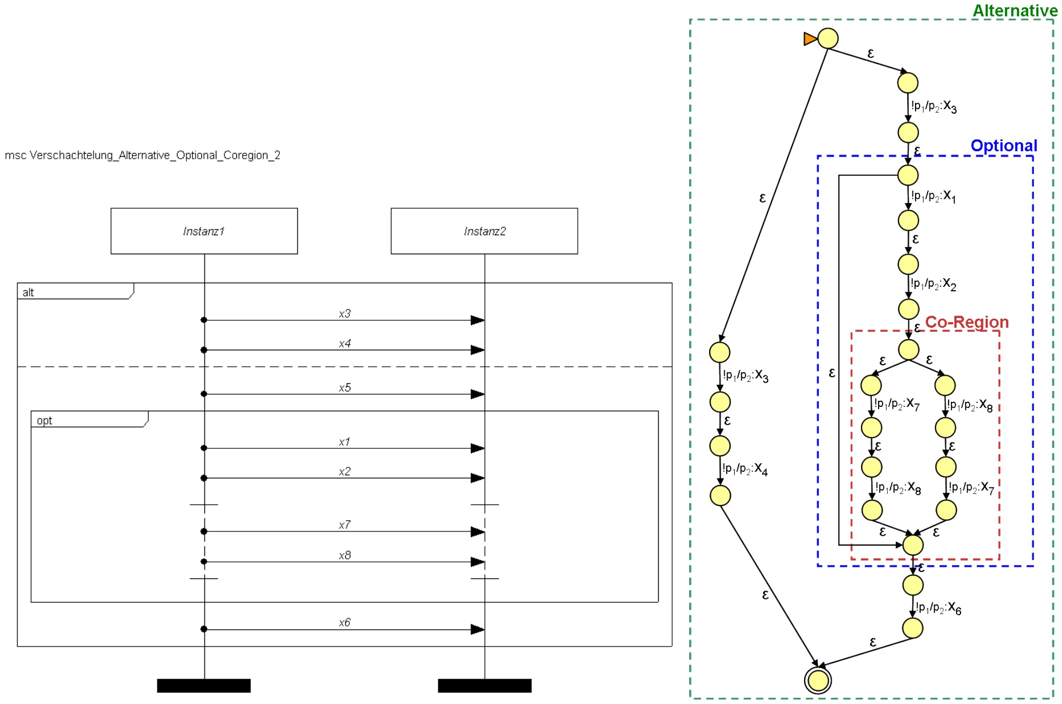


Abbildung 10.4: Beispiel für Verschachtelung Alternative - Optional - Coregion Testfall-Nr. 72

10.2 Validierung der Vergleichsanalyse für Rückwärtskompatibilität

In den vorherigen Abschnitten wurde sowohl die Gültigkeit der Transformationsregeln als auch die Performance der Transformationen und Reduktionen untersucht. Um eine Kompatibilitätsaussage bzgl. der Rückwärtskompatibilität von MSCs zu erhalten, werden die Automaten miteinander verglichen. In diesem Abschnitt wird die heuristische Vergleichsmethode aus Kapitel 7.4.5 validiert.

Zur Validierung der Vergleichsmethode wurden 42 MSC-Testfälle erstellt (vgl. Abbildung 10.5). Die Testfälle wurden sowohl für Basic-MSCs (Ereignisse, Timer), für Inline-Expressions (Alternative, Optional, Loop, Coregion, Parallel) als auch für ein

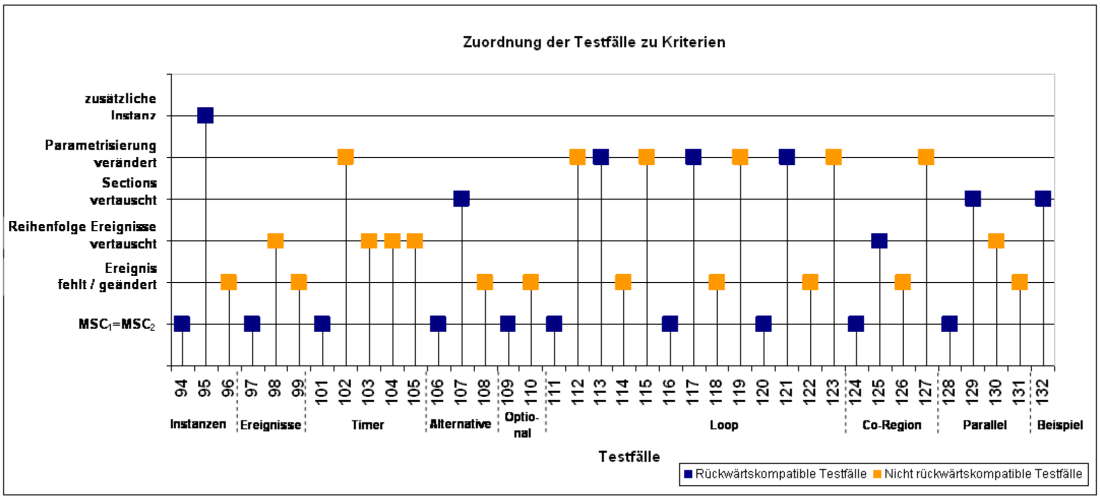


Abbildung 10.5: Zuordnung der Testfälle (*MSC-CompA*)

komplexes Beispiel und einen Belastungstest generiert. Hierbei wurden die Testfälle so konstruiert, dass das erwartete Kompatibilitäts-Ergebnis vorausgesagt werden kann.

Vergleichskonzept Die Ergebnisse des Vergleichs sind in den Tabellen 10.5 und 10.6 dargestellt. Die Tabellen ist hierbei wie folgt aufgebaut:

- (1. Spalte) Nummer des Testfalls.
- (2. Spalte) Bezeichnung des MSC-Konstrukts bzw. Testfall-Kategorie.
- (3. Spalte) Beschreibung des 1. Testszenarios MSC_1.
- (4. Spalte) Beschreibung des 2. Testszenarios MSC_2.
- (5. Spalte) Status, ob der Testfall „ok“ ist, d.h. das richtige Ergebnis geliefert hat oder evtl. noch nicht implementiert ist.
- (6. Spalte) Ergebnis des MSC-Vergleichsalgorithmus (kompatibel/nicht kompatibel).

Die Testfälle sind je nach MSC-Konstrukt (Instanzen, Ereignisse, Operatoren, etc.) kategorisiert. Der erste Testfall jeder Kategorie (z.B. Testfall Nr. 94,97,101,106,109,111,116,120,124) überprüft den Fall, dass MSC_1=MSC_2. In diesem Fall muss das Ergebnis des Vergleichs *„rückwärtskompatibel“* lauten.

Testfall Nr.		MSC-Szenario 1	MSC-Szenario 2	Ergebnis	Ergebnis Kompatibilitätsanalyse rückwärtskompatibel/ nicht rückwärtskompatibel
94	**Instanzen**	1 Instanz 1 Ereignis zum Env 1 Ereignis vom Env	**MSC_1 == MSC_2**	**ok**	**rückwärtskompatibel**
95			**MSC1 echte Teilmenge von MSC2** 2 Instanzen: 1.Instanz: - 1 Ereignis zum Env - 1 Ereignis vom Env 2. Instanz - 1 Ereignis zum Env	**ok**	**rückwärtskompatibel**
96			**MSC1 keine echte Teilmenge von MSC2** 2 Instanzen: - 1 Ereignis zum Env - 1 Ereignis vom Env - 1 Ereignis zwischen Instanz 1 und 2	**ok**	**nicht rückwärtskompatibel**
97	**Ereignisse**	1 Ereignis: Environment 1 Ereignis: Gate 1 Ereignis: zwischen Instanzen	**MSC_1 == MSC_2**	**ok**	**rückwärtskompatibel**
98			**Reihenfolge von Ereignissen vertauscht** 1 Ereignis: Environment 1 Ereignis: zwischen Instanzen 1 Ereignis: Gate	**ok**	**rückwärtskompatibel**
99			**Ereignis-Name geändert**	**ok**	**nicht rückwärtskompatibel**
101	**Timer**	10 Ereignisse mit - Start-Timer [10ms] - Time-Out - Stop-Timer	**MSC_1 == MSC_2**	**ok**	**rückwärtskompatibel**
102			**Start Timer [20ms]**	**ok**	**nicht rückwärtskompatibel**
103			**Start Timer an anderer Position [10ms]**	**ok**	**nicht rückwärtskompatibel**
104			**Time-Out an anderer Position**	**ok**	**nicht rückwärtskompatibel**
105			**Stopp Timer an anderer Position**	**ok**	**nicht rückwärtskompatibel**
106	**Alternative**	Alternative: 2 Section (über 2 Instanzen + 2 Ereignis/Section)	**MSC_1 == MSC_2**	**ok**	**rückwärtskompatibel**
107			**Section 1 und Section 2 vertauscht**	**ok**	**rückwärtskompatibel**
108			**Section 1 und Section 2 vertauscht & 1 Ereignis fehlt**	**ok**	**nicht rückwärtskompatibel**
109	**Optional**	Optional (über 2 Instanzen + 2 Ereignis)	**MSC_1 == MSC_2**	**ok**	**rückwärtskompatibel**
110			**1 Ereignis fehlt**	**ok**	**nicht rückwärtskompatibel**

Tabelle 10.5: Validierung der MSC-Kompatibilitätsanalyse (Teil 1)

Testfälle In den Testfällen-Nr. 94-99 werden zwei MSC-Szenarien mit Manipulationen an den Ereignissen überprüft. So wurde in Testfall-Nr. 98 die Reihenfolge der Signale und in Nr. 99 der Name eines Signals verändert. In beiden Testfällen muss das Ergebnis „nicht rückwärtskompatibel“ sein.

In den Testfällen-Nr. 101-105 wird der Vergleich von *Timer* validiert. Hierzu wurden sowohl die Timer-Parameter (Testfall-Nr. 102) als auch die Reihenfolge der Timer verändert (Testfälle-Nr. 103-105). Jede Veränderung der Timer-Spezifikation führt zu einer Inkompatibilität. Dies belegen auch die Ergebnisse.

Die Testfälle Nr. 106-108 analysieren, ob eine Vertauschung bei *Alternative* korrekt erkannt wird. Hierzu wurden zum einen die *Sections* vom *Alternative*-Operator vertauscht (Testfall-Nr. 107) und zum anderen wurde zusätzlich ein Ereignis entfernt (Testfall-Nr. 108). Die Vertauschung der Sections führt zu keinem Kompatibilitätsverlust. Wird jedoch zusätzlich ein Ereignis entfernt, so sind MSC_1 und MSC_2 inkompatibel.

In den Testfällen Nr. 111-123 wurden Vergleichstests mit dem Loop-Operator durchgeführt. Dies ist insofern interessant, da im Loop-Operator zusätzlich Parameter spezifiziert sein können, die angeben, wie oft die Loop durchlaufen werden soll.

Testfall Nr.		MSC-Szenario 1	MSC-Szenario 2	Ergebnis	Ergebnis Kompatibilitätsanalyse
					rückwärtskompatibel/ nicht rückwärtskompatibel
111	**Loop**	Loop <10> - (über 2 Instanzen + 3 Ereignis)	**MSC_1 == MSC_2**	**ok**	**rückwärtskompatibel**
112			**MSC_1 keine Teilmenge von MSC_2** Loop <15> - (über 2 Instanzen + 3 Ereignis)	**ok**	**nicht rückwärtskompatibel**
113			**MSC_1 echte Teilmenge von MSC_2** Loop <0,15> - (über 2 Instanzen + 3 Ereignis)	**ok**	**rückwärtskompatibel**
114			**MSC_1 keine Teilmenge von MSC_2 & Ereignis fehlt** Loop <15> - (über 2 Instanzen + 3 Ereignis)	**ok**	**nicht rückwärtskompatibel**
115			**MSC_1 keine Teilmenge von MSC_2** Loop <5> - (über 2 Instanzen + 3 Ereignis)	**ok**	**nicht rückwärtskompatibel**
116		Loop <2,10> - (über 2 Instanzen + 3 Ereignisse)	**MSC_1 == MSC_2**	**ok**	**rückwärtskompatibel**
117			**MSC_1 echte Teilmenge von MSC_2** Loop <1,15>	**ok**	**rückwärtskompatibel**
118			**MSC_1 keine Teilmenge von MSC_2 & Ereignis fehlt** Loop <1,15>	**ok**	**nicht rückwärtskompatibel**
119			**MSC_1 keine Teilmenge von MSC_2** Loop <15> - (über 2 Instanzen + 3 Ereignis)	**ok**	**nicht rückwärtskompatibel**
120		Loop <2,inf> - (über 2 Instanzen + 3 Ereignisse)	**MSC_1 == MSC_2**	**ok**	**rückwärtskompatibel**
121			**MSC_1 echte Teilmenge von MSC_2** Loop <1,inf>	**ok**	**rückwärtskompatibel**
122			**MSC_1 keine Teilmenge von MSC_2, Ereignis fehlt** Loop <1,inf>	**ok**	**nicht rückwärtskompatibel**
123			**MSC_1 keine Teilmenge von MSC_2** Loop <15>	**ok**	**nicht rückwärtskompatibel**
124	**Co-Region**	Coregion - (über 2 Instanzen & 2 Ereignisse)	**MSC_1 == MSC_2**	**ok**	**rückwärtskompatibel**
125			**Reihenfolge von Ereignissen vertauscht**	**ok**	**rückwärtskompatibel**
126			**1 Ereignis fehlt**	**ok**	**nicht rückwärtskompatibel**
127			**Coregion verändert** - MSC hat zwei Instanzen - Coregion nur über 1 Instanz	**ok**	**nicht rückwärtskompatibel**
132	**Beispiel**	- 2 Ereignisse - Optional innerhalb Alternative mit (mit 2 Sections & je 2 Signalen/Section)	- 2 Ereignisse - Alaternative mit (mit 3 Sections & je 2 Signalen/Section eine Section leer)	**ok**	**rückwärtskompatibel**

Tabelle 10.6: Validierung der MSC-Kompatibilitätsanalyse (Teil 2)

So wird in den Testfällen Nr. 113,117,121 getestet, ob die Ereignissequenzen von MSC_1 eine echte Teilmenge von MSC_2 sind ($MSC_1 \subseteq MSC_2$). In diesem Fall gilt MSC_2 rückwärtskompatibel zu MSC_1. Der komplementäre Fall wurde in Nr. 112,114,115,118,119,123 überprüft.

Die Testfälle Nr. 124-127 überprüften die Vergleichsmethode anhand der Coregion. Es zeigt sich, dass die Ereignisse innerhalb einer Coregion vertauscht werden können (Testfall-Nr. 125), ohne Einfluss auf das Kompatibilitätsergebnis. Wird jedoch ein Ergebnis entfernt, sind die MSCs nicht mehr rückwärtskompatibel zueinander (Testfall-Nr. 126,127).

Der Testfall Nr. 132 überprüft, ob zwei MSCs, die eine unterschiedliche graphische

Darstellung haben, aber das gleiche dynamische Verhalten beschreiben, als rückwärtskompatibel erkannt werden. Hierzu wurde das Beispiel aus Kapitel 7.5 abgebildet (vgl. Abbildung 10.6). Das Ergebnis bestätigt die Theorie.

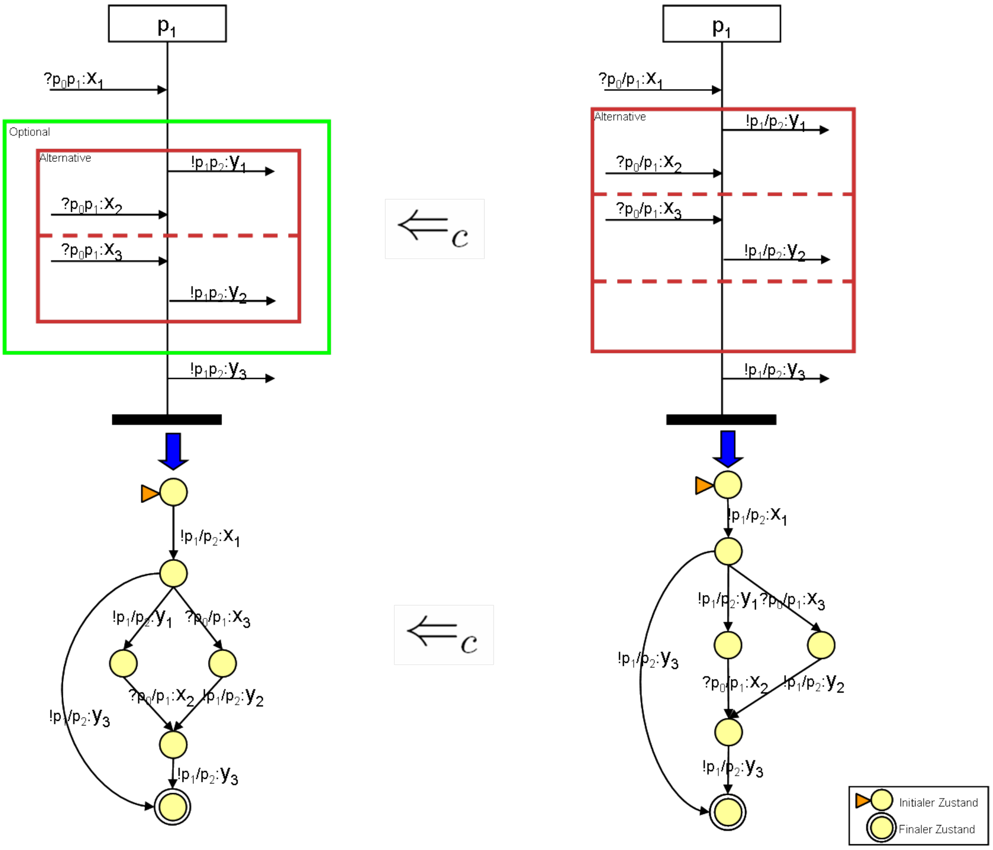


Abbildung 10.6: Beispiel aus Kapitel 7.5, Abbildung 7.31

10.3 Auswertung

In den vorangegangenen Abschnitten wurde die Validierung der einzelnen Transformationsregeln und des heuristischen Vergleichsansatzes vorgestellt. In diesem Abschnitt wird nun auf die Effizienz und die Performance der Methoden eingegangen.

10.3.1 Vergleich verschiedener Transformationsregeln

In Abbildung 10.7 ist gegenübergestellt, wie viele Zustände, Übergangsfunktionen und ϵ-Übergänge bei der Transformation verschiedener MSC-Konstrukte erzeugt werden. Hierzu wurden Testfälle mit den Konstrukten *Ereignisse*, *Condition*, *Optional*, *Alternative*, *Parallel*, *Loop* und *Coregion* erzeugt. Jeder Testfall besteht aus 6 Basic-MSC-

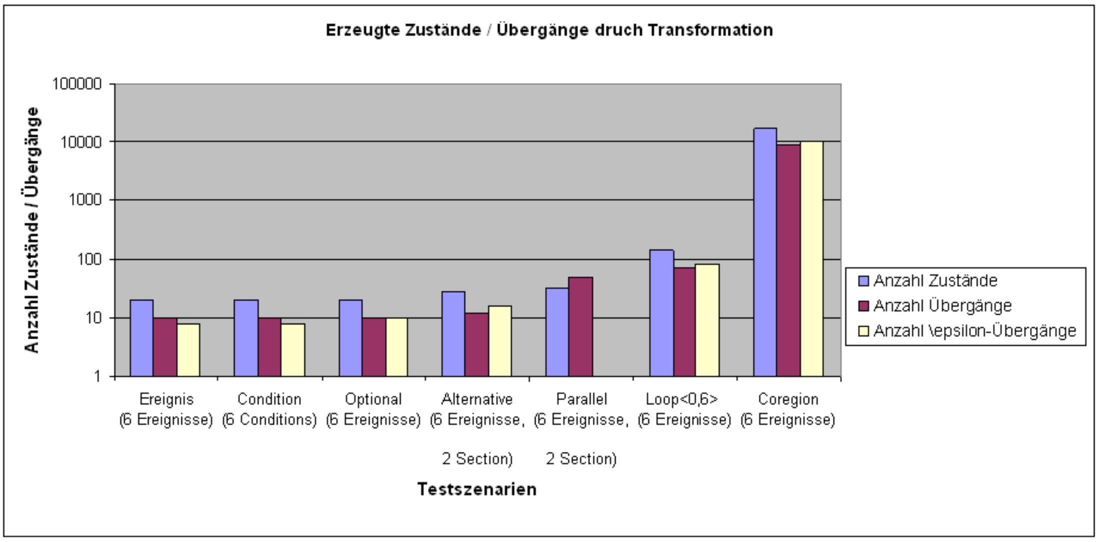


Abbildung 10.7: Erzeugte Zustände und Übergänge durch Transformation

Elemente (*Ereignisse*, *Conditions*) und der jeweiligen Inline-Expressions, sofern diese angegeben ist. In der Abbildung wird deutlich, dass beim *Optional*-Operator gegenüber den Basic-MSCs ein zusätzlicher ϵ-Übergang und beim *Alternative*-Operator zusätzliche Zustände und ϵ-Übergänge hinzugefügt werden. Bei der Transformation des Parallel-Operators werden die Sections des Operators separat in Automaten überführt und anschließend wird der Produktautomat daraus gebildet. Um die Komplexität zu verringern wird vor der Erzeugung des Produktautomaten eine ϵ-Eliminierung durchgeführt. Daher enthält der Produktautomat des Parallel-Operators keine ϵ-Übergänge. Beim *Loop*-Operator wird der erzeugte Automat von 6 Ereignissen, aufgrund des Parameters $< 0, 6 >$, 6 mal konkateniert. Daher besitzt dieser Automat ca. 6-mal mehr Zustände, Übergangsfunktionen und ϵ-Übergänge als *Ereignisse*. *Coregion* ist das komplexeste Konstrukt. Hier werden mit 6 *Ereignissen* fast 1000-mal mehr Elemente erzeugt als bei *Ereignis*.

10.3.2 Determinierung der generierten Automaten

Nach der Transformation werden die Automaten determiniert, d.h. es wird eine ϵ-Eliminierung und eine Determinierung durchgeführt. Durch die Eliminierung von ϵ-Übergängen verringert sich die Anzahl der Elemente von Automaten sehr stark. Um dies darzustellen, wird die Anzahl der Zustände und Übergangsfunktionen (inkl. der ϵ-Übergänge) der Anzahl an Zuständen und Übergangsfunktionen nach der Determinierung gegenübergestellt (vgl. Abbildung 10.8). Es wird deutlich, dass die Übergangsfunktionen und die Zustände aufgrund der ϵ-Eliminierung um ca. 50% reduziert werden. Dadurch wird der Vergleichsalgorithmus zur Rückwärtskompatibilitätsanalyse, der auf deterministische Automaten aufsetzt, erheblich effizienter.

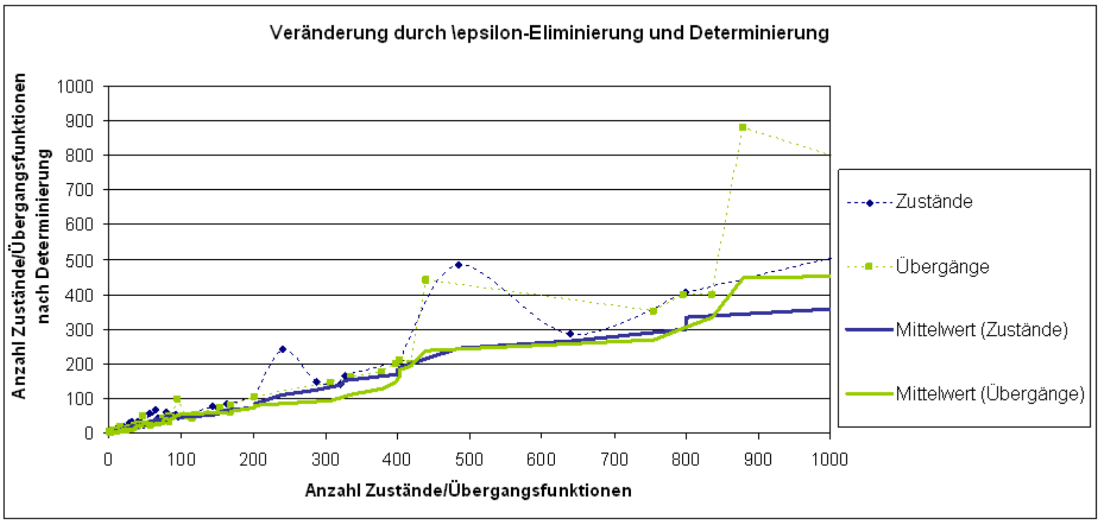


Abbildung 10.8: Veränderung der Anzahl an Zuständen und Übergangsfunktionen durch Determinierung

10.3.3 Vergleich Rechenzeiten

In Abbildung 10.9 sind die Rechenzeiten der Transformation, der Determinierung und der Kompatibilitätsanalyse abgebildet. Bei der Validierung wurden die Rechenzeiten in [ms] für die einzelnen Schritte gemessen und der Anzahl an Zuständen und Übergangsfunktionen (inkl. ϵ-Übergängen) gegenübergestellt. In der Graphik ist deutlich zu sehen, dass die Berechnung der Automaten-Transformation bei bis zu 3500 Elementen am längsten dauert. Ab 3500 Elementen steigt die Komplexität zur Berechnung der Determinierung (inkl. ϵ-Eliminierung) stark an. Die Rechenzeit für die Kompatibilitätsanalyse ist gegenüber der Transformation und der Determinierung sehr kurz und veranschaulicht die Effizienz des Vergleichsalgorithmus zur Kompatibilitätsanalyse. Die Rechenzeiten für die einzelnen Schritte sind für die Komplexität der Berechnungen sehr kurz. So dauert beispielsweise die Transformation von zwei äquivalenten MSCs mit jeweils einer Loop$< 0, 6 >$ und 6 Ereignissen (vgl. Abbildung 10.7) 218ms. Dabei werden 288 Zustände und 308 Übergangsfunktionen (inkl. ϵ-Übergänge) erzeugt. Die Determinierung (inkl. ϵ-Eliminierung) benötigt für diese Automaten 314ms. Die Rechenzeit ist hierbei aufgrund der Schleifen länger als im Durchschnitt. Die Automaten können durch die Determinierung (inkl. ϵ-Eliminierung) auf 148 Zustände und 144 Übergangsfunktionen reduziert werden. Die Analyse bzgl. der Rückwärtskompatibilität der beiden MSCs benötigt nur ca. 1 ms.

10.4 Zusammenfassung

In diesem Kapitel wurde die Validierung der MSC-Vergleichsmethode (*MSC-CompA*) schrittweise beschrieben.

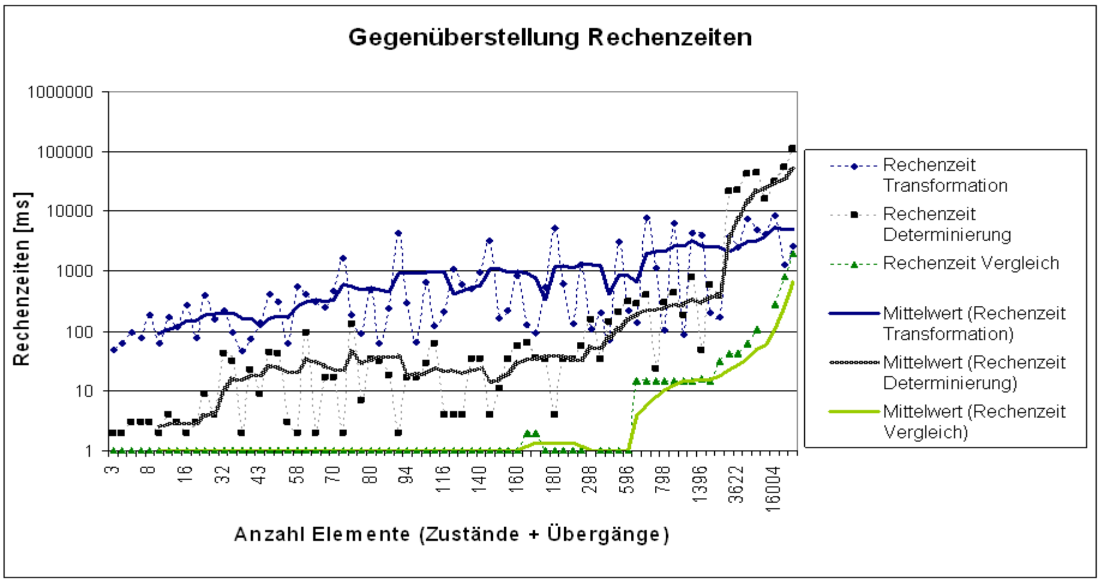


Abbildung 10.9: Gegenüberstellung von Rechenzeit und Anzahl an Zuständen

Im ersten Schritt wurden hierzu die einzelnen Transformationsregeln aus Kapitel 7 separat untersucht. Das Ergebnis zeigte, dass alle implementierten Transformationsregeln korrekt arbeiten.

Im zweiten Schritt wurden die MSC-Sprachkonstrukte kombiniert und/oder verschachtelt, so dass komplexe MSC-Szenarien generiert wurden. Auch hier lieferten die Ergebnisse den Nachweis, dass die Transformationsregeln auch komplexe, wie z.B. verschachtelte, MSC-Sprachkonstrukte korrekt transformieren.

Aufbauend auf den Ergebnissen aus Schritt 1 und Schritt 2 wurde eine Auswertung der Performance der Transformationsregeln erstellt. Diese zeigte deutlich, dass die für die Transformationsregeln benötigte Rechenzeit sehr kurz ist.

Nach dem Nachweis, dass die Transformationsregeln korrekt funktionieren, wurde im dritten Schritt der heuristische Ansatz zum Vergleich der deterministischen Automaten validiert. Die Ergebnisse des Vergleichs belegten die korrekte Arbeitsweise der Vergleichsmethode.

Durch den schrittweisen Aufbau konnte sehr gut nachgewiesen werden, dass sowohl alle Transformationen, die Determinierung als auch der heuristische Vergleichsansatz allen Anforderungen gerecht werden.

11 Zusammenfassung und Ausblick

11.1 Zusammenfassung

Das Forschungsthema *Compatibility Analysis for Electronic Control Units (CompA)* wurde im Rahmen dieser Dissertation vollständig bearbeitet und in diesem Dokument zusammengefasst.
Die Zielsetzung von *CompA* war, eine Methode zu entwickeln, mit der Steuergeräte auf Basis ihrer Spezifikationen bzgl. Rückwärtskompatibilität analysiert werden können. Hierfür wurde ein durchgängiges Gesamtkonzept entworfen, das drei Schwerpunkte fokussierte:

- Formalisierung und Instanziierung: Definition einer Methode zum Entwurf eines Steuergeräte-Metamodells (SG-Metamodell) auf Basis von XML-Schema (XML-SG-Schema). Aus dem XML-SG-Schema können XML-Steuergeräte-Spezifikationen (XML-SG-Instanzen) abgeleitet werden. Dynamisches Verhalten wird mittels Message Sequence Charts (MSC) spezifiziert.
- XML-Vergleichsmethode: Definition einer Methode zur Analyse, ob zwei Steuergeräte aufgrund der zugehörigen XML-SG-Instanzen zueinander rückwärtskompatibel sind.
- MSC-Vergleichsmethode: Definition einer Methode zur Analyse, ob das von zwei MSCs spezifizierte Verhalten zueinander rückwärtskompatibel ist.

11.1.1 Entwurf eines abstrakten Steuergeräte-Metamodells (SG-Metamodell)

Um auf Basis von Spezifikationen Aussagen bzgl. der Rückwärtskompatibilität von Steuergeräten treffen zu können, müssen die Spezifikationen miteinander verglichen werden. Dies ist nur sinnvoll möglich, wenn diese formalen Spezifikationsvorgaben genügen. Aus diesem Grund wurde im Rahmen von *CompA* ein Konzept zur Erstellung von SG-Metamodellen entwickelt. SG-Metamodelle legen fest, welche Elemente von Steuergeräten spezifiziert werden müssen, um Analysen bzgl. der Rückwärtskompatibilität von Steuergeräten durchführen zu können. Hierbei wurde zwischen statischen und dynamischen Eigenschaften von Steuergeräten unterschieden. Das entwickelte SG-Metamodell basiert auf abstrakten Strukturierungsregeln, die sich in vier Ebenen untergliedern: Die *Sichten*-Ebene, die *abstrakte Klassen*-Ebene, die *abstrakte Attribut*-Ebene

und die *abstrakte Attribut Typ*-Ebene. Um IT-technisch mit dem SG-Metamodell zu arbeiten, wurde eine Portierung der SG-Metamodelle auf XML-Schema (XML-SG-Schema) vorgenommen. Aus dem XML-SG-Schema können dann SG-Spezifikationen (XML-SG-Instanzen) abgeleitet werden.
Auf Basis dieser XML-SG-Instanzen bauen die weiteren Vergleichsmethoden auf.

11.1.2 XML-Vergleichsmethode (*XML-CompA*)

Die XML-SG-Instanzen bilden die Basis für die Kompatibilitätsanalyse von Steuergeräten. Es wurde gezeigt, dass heutige XML-Vergleichsmethoden nicht ausreichen, um umfangreiche Aussagen bzgl. der Rückwärtskompatibilität von Steuergeräten zu generieren. Aus diesem Grund wurde die spezifische XML-Vergleichsmethode „*XML-CompA*“ zur Kompatibilitätsanalyse entwickelt. Der Ansatz baut auf einer 4-Schritt-Methode auf:

- ***Auswahl:*** Festlegung, welche XML-Elemente bei der Kompatibilitätsanalyse eine Rolle spielen.
- ***Mapping:*** Zuordnung der zu vergleichenden Elemente von *XML-SG-Instanz(*SG_1*)* zur *XML-SG-Instanz(*SG_2*)*.
- ***Kompatibilitätsanalyse:*** Analyse der zugeordneten Elemente bzgl. Rückwärtskompatibilität, unter Einbeziehung von Kompatibilitätsregeln.
- ***Kompatibilitätsaussagen:*** Generierung von Aussagen bzgl. Rückwärtskompatibilität von Steuergerät SG_2 zu SG_1.

Die entwickelte Vergleichsmethode XML-CompA bietet gegenüber aktuellen XML-Vergleichsmethoden folgende Features:

- Spezifische Elemente können für die Kompatibilitätsanalyse ausgeblendet werden. Damit lassen sich unterschiedliche Kompatibilitätsszenarien evaluieren, nach dem Motto: Was wäre, wenn man z.B. die Anordnung der Interfaces nicht berücksichtigen würde?
- Innere Abhängigkeiten bzw. Verweise (ID/IDREF) von XML-SG-Instanzen werden bei der Analyse korrekt berücksichtigt.
- XML-Elemente können bzgl. Ihrer Relevanz zur Kompatibilitätsanalyse unterschiedlich gewichtet werden.
- Zur Analyse können spezifische Kompatibilitätsregeln herangezogen werden.

Das Konzept gibt dem Entwickler die Möglichkeit, durch einen Vergleich von XML-SG-Instanzen die inkompatiblen Elemente bzw. Anforderungen zu identifizieren. Daraus kann der Entwickler anschließend Lösungen zur Sicherstellung von Rückwärtskompatibilität ableiten.

11.1.3 MSC-Vergleichsmethode (*MSC-CompA*)

Message Sequence Charts (MSC) werden verwendet, um dynamisches Verhalten zu spezifizieren. Um zu analysieren, ob das spezifizierte Verhalten von zwei MSCs zueinander rückwärtskompatibel ist, werden diese verglichen. Das Problem hierbei ist, dass ein äquivalentes bzw. kompatibles Verhalten mittels MSCs unterschiedlich spezifiziert sein kann. Um dies zu erkennen, wurde im Rahmen dieser Arbeit eine MSC-Vergleichsmethode definiert. Das Konzept der entwickelten Methode ist, die MSCs in endliche Automaten zu transformieren und anschließend mittels eines heuristischen Vergleichsansatzes bzgl. Rückwärtskompatibilität zu analysieren. Die entwickelte Methode, die in dieser Arbeit ausführlich beschrieben wurde, baut auf folgenden Schritten auf:

- *Graphische Darstellung*: MSCs werden mittels MSC-Editoren graphisch erstellt und in ASCII exportiert.

- *Transformation in XML-Darstellung*: ASCII-Code wird auf Basis eines MSC-Metamodells in MSC-XML-Instanzen portiert.

- *Transformation in Automatendarstellung*: MSC-XML-Instanzen werden in endliche Automaten (EA) transformiert.

- *Determinierung der Automaten*: Endliche Automaten (EA) werden durch ϵ-Eliminierung reduziert und durch Determinierung in deterministische endliche Automaten überführt (DEA).

- *Kompatibilitätsanalyse auf Basis von Automaten*: Vergleich der deterministischen endlichen Automaten ($DEA(MSC_1)$, $DEA(MSC_2)$) und Erzeugung von Aussagen bzgl. deren Rückwärtskompatibilität.

Jeder Schritt wurde im Rahmen dieser Arbeit detailliert erläutert.

11.2 Ergebnisse

Die Ergebnisse dieser Arbeit lassen sich in zwei Kategorien unterteilen:

1. Konzepte und Methoden

2. Implementierungen

Alle Konzepte, Methoden und Implementierungen wurde mit Hilfe mehrerer Testfälle validiert.

11.2.1 Konzepte und Methoden

Im Rahmen dieser Arbeit entstand eine Reihe von neuen Konzepten und Methoden. Angefangen bei dem Gesamtkonzept, das einen durchgängigen Ansatz zur Analyse von Steuergeräten bzgl. Rückwärtskompatibilität präsentiert. Um das Gesamtkonzept zu realisieren, wurden verschiedene Methoden entwickelt, wie z.B. der Spezifikationsansatz für Steuergeräte, der auf XML-SG-Metamodellen basiert. Hierfür wurden Strukturierungsregeln wie beispielsweise das Sichten-Konzept definiert, aber auch die Überführung in ein XML-SG-Schema vorgestellt. Des Weiteren wurden im Rahmen dieser Dissertation die benötigten Vergleichsmethoden zur Analyse bzgl. Rückwärtskompatibilität entwickelt. Die XML-Vergleichsmethode *XML-CompA* ist ein neuartiger Ansatz, um XML-Instanzen bzgl. Rückwärtskompatibilität effizient zu analysieren. Die MSC-Vergleichsmethode erweitert einen Ansatz [87] um Transformationsvorgaben für verschachtelte MSCs und definiert einen effizienten heuristischen Vergleichsansatz zur Analyse bzgl. der Rückwärtskompatibilität des definierten Verhaltens.

11.2.2 Implementierungen

Alle entwickelten Konzepte wurden im Rahmen dieser Arbeit realisiert und validiert:

- **XML-SG-Schema:** Erstellung eines XML-SG-Schemas zur hinreichenden Spezifikation von Steuergeräten.
- **Software-Module:** Implementierung der Konzepte
  - **SG-Editor:** Graphische Bedienoberfläche (GUI), die den Benutzer unterstützt, auf Basis der XML-SG-Metamodell Steuergeräte-Spezifikationen (XML-SG-Instanzen) zu erstellen.
  - **XML-CompA:** Umsetzung der Konzepte der Vergleichsmethode XML-CompA.
  - **MSC-CompA:** Umsetzung der Konzepte der Vergleichsmethode MSC-CompA.

Vorhandene Spezifikationen von verschiedenen Steuergeräten wurden mit Hilfe des CompA-Ansatzes nachgebildet. Auf Basis dieser XML-SG-Instanzen konnte die Korrektheit, Funktionalität und Effizienz der entwickelten Konzepte, Methoden und Implementierungen nachgewiesen werden.

11.3 Ausblick

CompA bietet einen durchgängigen Ansatz zur Analyse von Rückwärtskompatibilität, dennoch sind verschiedene Erweiterungen denkbar.
Ein interessanter Punkt ist z.B. die Ausweitung der Kompatibilitäts-Definitionen und der Konzepte auf die Analyse von Vorwärtskompatibilität von Steuergeräten.

Konzeptionell können die entwickelten Methoden diese Analysen bereits behandeln, allerdings sind Änderungen an den integrierten Kompatibilitätsregeln notwendig.
Ein weiterer Ansatzpunkt ist die Ausweitung der entwickelten Methoden auf unterschiedliche Steuergeräte. Dies wird zu Erweiterungen des SG-Metamodells führen. Für die Erweiterung des SG-Metamodells ist auch eine Zusammenführung der in MSR [106] definierten Elemente denkbar und sinnvoll.
Um sich wiederholende Spezifikationsvorgaben im XML-SG-Schema (z.B. Intervall) nur einmal zu spezifizieren und an anderen Stellen zu referenzieren, empfiehlt sich die Portierung des XML-SG-Schema auf ein rekursives XML-SG-Schema. Hierfür müssen die Konzepte bzw. Strukturierungsregeln nur geringfügig angepasst werden. Die Implementierung des *SG-Editors* und von *XML-CompA* ist hierfür jedoch grundlegend zu überarbeiten.
Eine weitere vielversprechende Erweiterung des CompA-Ansatzes ist die Einbeziehung des Themas *Test und Absicherung* von Steuergeräten. Bei einer Erweiterung des XML-SG-Schemas auf Spezifikationsvorgaben für Testfälle, können die entwickelten Methoden eins zu eins angewendet werden und so auch *Testspezifikationen* bei der Kompatibilitätsbetrachtung berücksichtigt werden. Das Gleiche gilt auch für die *Technischen Produktbeschreibungen* der Zulieferer.

Diese Arbeit beschäftigte sich mit einem zentralen Thema aus der Industrie: der Rückwärtskompatibilitäts-Analyse von Steuergeräten. Alle entwickelten Konzepte und Implementierungen sind an Beispielen aus der Industrie erfolgreich erprobt worden und stellen eine Basis für weitere Entwicklungen dar. Gerade die Erweiterung des Ansatzes auf den Bereich *Test-und Absicherung* ist sehr vielversprechend und von großem Industrie-Interesse.

Literaturverzeichnis

[1] 8879:1986, ISO: *Information Processing - Text and Office Systems - Standard Generalized Markup Language (SGML)*. 1996

[2] ALMEIDA, Marco ; MOREIRA, Nelma ; REIS, Rogerio: On the performance of automata minimization algorithms / Departamento de Ciencia de Computadores. 2007. – Forschungsbericht

[3] ALUR, Rajeev ; HOLZMANN, Gerard ; PELED, Doron: An Analyzer for Message Sequence Charts. In: *Workshop on Tools and Algorithms for the construction and Analysis of Systems, TACAS96*. Lecture Notes in Computer Science, März 1996, S. 35–48

[4] THE APACHE XML PROJECT (Hrsg.): *Xalan: XSL stylesheet processors in Java*. The Apache XML Project, August 2003

[5] THE APACHE XML PROJECT (Hrsg.): *Xerces2 Java Parser Readme*. The Apache XML Project, August 2003

[6] BAUMERT, Andreas: *SGML-Grundlagen-Goldfarb, Mosher und Lorie (GML)*. Online-Publikation. `http://www.recherche-und-text.de/wwwpubls/sgml03.htm`. Version: November 2003

[7] BEHME, Henning ; MINTERT, Stefan: *XML in der Praxis; Professionelles Web-Publishing mit der Extensible Markup Language*. 2. Auflage. Addison-Wesley, 2000. – ISBN 3827316367

[8] BERARD, B. ; BIDOIT, M. ; FINKEL, A. ; LAROUSSINIE, F. ; PETIT, A. ; PETRUCCI, L. ; SCHNOEBELEN, P. ; MCKENZIE, P: *Systems and Software Verification: Model-Checking Techniques and Tools*. 1 edition (August 9, 2001). Springer Verlag, 2001. – ISBN 3540415238

[9] BERSTEL, Jean ; CARTON, Olivier: On the Complexity of Hopcroft's State Minimization Algorithm. In: DOMARATZKI, Michael (Hrsg.) ; OKHOTIN, Alexander (Hrsg.) ; SALOMAA, Kai (Hrsg.) ; YU, Sheng (Hrsg.): *CIAA* Bd. 3317. Springer, 2004. – ISBN 3–540–24318–6, S. 35–44

[10] BMW-GROUP: *Vorlage Musterlastenheft*. confidental - unpblished, 2006

[11] BOBDA, Christophe: *Synthesis of dataflow graphs for reconfigurable systems using temporal partitioning and temporal placement*, Universität Paderborn, Dissertation, 2003

[12] BOBDA, Christophe ; AHMADINIA, Ali: Dynamic Interconnection of Reconfigurable Modules on Reconfigurable Devices. In: *IEEE Design & Test of Computers, Special Issue* (2005)

[13] BOBDA, Christophe ; STEENBOCK, Nils: Singular Value Decomposition on Distributed Reconfigurable Systems. In: *12th IEEE International Workshop on Rapid System Prototyping*. Monterey, California, USA, 2001

[14] BOBDA, Christophe ; STEENBOCK, Nils: A Rapid Prototyping Environment for Distributed Reconfigurable Systems. In: *13th IEEE International Workshop On Rapid System Prototyping(RSP'02), Darmstadt Germany*, 2002

[15] BOOCH, G. ; RUMBAUGH, J. ; JACOBSEN, I.: *The Unified Modeling Language User Guide*. Addison-Wesley, 1999. – ISBN 0201571684

[16] BORN, Marc ; HOLZ, Eckhardt ; KATH, Olaf: *Softwareentwicklung mit UML 2 , Die neuen Entwurfstechniken UML 2, MOF 2 und MDA*. Addison-Wesley-Verlag, 2004. – ISBN 3827322561

[17] BOSCH: *Spezifikation Motorsteuergerät Motronic MS 5*. Spezifikation, 2007

[18] BRAATZ, Benjamin: *Vorstellung 3. Meilenstein, Deterministische Automaten, Konstruktionen*. Vorlesung, Dezember 2006. – TU-Berlin

[19] BRAATZ, Benjamin: *Konstruktion und Vergleiche auf Automaten*. Vorlesung, Juni 2007. – TU-Berlin

[20] BUDINSKY, Frank ; STEINBERG, David ; MERKS, Ed ; ELLERSICK, Raymond ; GROSE, Timothy J.: *Eclipse Modeling Framework*. Addison-Wesley, 2004. – ISBN 0–13–142542–0

[21] BUSS, Samuel: Alogtime Algorithms for Tree Isomorphism, Comparison, and Canonization. In: *KGC '97: Proceedings of the 5th Kurt Gödel Colloquium on Computational Logic and Proof Theory*. London, UK : Springer-Verlag, 1997. – ISBN 3–540–63385–5, S. 18–33

[22] CAI, J. ; PAIGE, R. ; TARJAN, R.: More efficient bottom-up multi-pattern matching in trees. In: *Theor. Comput. Sci.* 106 (1992), Nr. 1, S. 21–60

[23] CHAWATHE, Sudarshan S.: Comparing Hierarchical Data in External Memory. In: *VLDB '99: Proceedings of the 25th International Conference on Very Large Data Bases*. San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 1999. – ISBN 1–55860–615–7, S. 90–101

[24] CHAWATHE, Sudarshan S. ; GARCIA-MOLINA, Hector: Meaningful change detection in structural data. In: *International Conference on Management of Data*. ACM Press, 1997. – ISBN 0–89791–911–4, S. 26–37

[25] Chawathe, Sudarshan S. ; Rajaraman, Anand ; Garcia-Molina, Hector ; Widom, Jennifer: Change Detection in hierarchically structured information. In: *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, Montreal, Quebec, Canada, June 4-6, 1996.* ACM Press, 1996, S. 493–504

[26] Clarke ; Grumberg ; Peled: *Model Checking.* MIT Press, 2000. – ISBN 0–262–03270–8

[27] Cobena, G. ; Abiteboul, S. ; Marian, A.: A comparative study for XML change detection / Institut National de Recherche en Informatique et en Automatique, Rocquencourt, France. 2002. – Forschungsbericht

[28] Cobena, G. ; Abiteboul, S. ; Marian, A.: Detecting Changes in XML Documents. In: *ICDE, Proceedings of the 18th International Conference on Data Engineering.* IEEE Computer Society, 2002. – ISSN 1063–6382

[29] Consortium, W3: *Extensible Markup Language (XML).* `http://www.w3.org/TR/1998/REC-xml-19980210`. Version: Februar 1998

[30] Consortium, W3: *XML Path Language (XPath).* `http://www.w3.org/TR/xpath`. Version: November 1999

[31] Consortium, W3: *XSL Transformations (XSLT).* `http://www.w3.org/TR/xslt`. Version: November 1999

[32] Consortium, W3: *Document Object Model (DOM).* `http://www.w3.org/DOM/`. Version: 2004

[33] Consortium, W3: *Extensible Markup Language (XML) 1.1.* `http://www.w3.org/TR/2004/REC-xml11-20040204/`. Version: April 2004

[34] Consortium, W3: *SGML Resources.* `http://www.w3.org/MarkUp/SGML/`. Version: März 2004

[35] Consortium, W3: *Extensible Markup Language (XML) 1.0 (Fourth Edition).* `http://www.w3.org/TR/2006/PER-xml-20060614/`. Version: June 2006

[36] Damm, Werner ; Harel, David: LSCs: Breathing Life into Message Sequence Charts. In: *FMOODS'99 IFIP TC6/WG6.1 Third International Conference on Formal Methods for Open Object-Based Distributed Systems*, 1999

[37] DIN ISO 8402 1995-08, Ziffer 2.: *DIN Norm -Kompatibilität,.* `http://www2.din.de/`. Version: 1995

[38] Droste, Matthias: *Effiziente Verhaltensmodellierung zur Systemintegration*, TU-München, Diplomarbeit, 2006

[39] E. LEONARDI, S. M. S. Bhowmick B. S. Bhowmick: XANDY: Detecting Changes in Large Unordered XML Documents Using Relational Databases / University Singapore, University Missouri-Rolla. 2005. – Forschungsbericht

[40] EBBINGHAUS, H.-D.: *Einführung in die Mengenlehre*. BI Wissenschaftsverlag, 1994 (3). – ISBN 3–411–17113–8

[41] ECLIPSE.ORG: *Eclipse Modelig Framework*. `http://www.eclipse.org/modeling/emf`. Version: 2007

[42] ECLIPSE.ORG: *Generating an EMF Model using XML Schema (XSD)*. `www.eclipse.org`. Version: 2007

[43] EPSTEIN, D. ; CURBERA, F.: XML TreeDiff by IBM alphaWorks / IBM. 1998. – Forschungsbericht

[44] ETAS: *ASCET-Produktfamilie*. `http://de.etasgroup.com/products/ascet/`. Version: 2007

[45] FAUST, Philipp Matthias und H. Matthias und Hoven: *Modellierung und automatische Auswertung von Regeln zur Beschreibung von Abhängigkeiten in XML-Dokumenten*, Universität Paderborn, Diplomarbeit, Januar 2004

[46] FEIJS, Loe M. G.: Generating FSMs from interworkings. In: *Distributed Computing* 12 (1999), S. 31–40

[47] FERRARO, Pascal ; QUANGRAOUA, Aida: Local mapping between unordered trees / laBRI. 2005. – Forschungsbericht

[48] FERRARO, Pascal ; TICHIT, Laurent ; DULUCQ, Serge: Local similarity between trees. In: *5èmes Journées Ouvertes Biologie Informatique Mathématiques*, 2004. – Montréal, Canada

[49] FORM, Thomas: *Datenbussysteme in Straßenfahrzeugen*. Vorlesung; Lehrstuhl für elektronische Fahrzeugsysteme. `www.ifr.ing.tu-bs.de/lehre/downloads/skripte/skript_dbf_1.pdf`. Version: November 2006

[50] GAMMA, Erich ; HELM, Richard ; JOHNSON, Ralph ; VLISSIDES, John: *Entwurfsmuster-Elemente wiederverwendbarer objektorientierter Software*. Addison-Wesley, 2004. – ISBN 978–3827321992

[51] GLOCKNER, Matthias: Kompatibilitätsanalyse für integrierte Automobil-Steuergeräte / TU-Chemnitz. 2006. – Forschungsbericht

[52] GLOCKNER, Matthias ; BROSS, Marco ; HOCKGEIGER, Elmar: *Automatisiertes Testen elektronischer Getriebesteuerungen*. IAV-Symposium 2003, Oktober 2003

[53] GLOCKNER, Matthias ; HARDT, Wolfram ; FUCHS, Maximilian ; MACHT, Michael: Kompatibilitätsanalyse dynamischen Verhaltens von integrierten Automobil-Steuergeräten. In: *Wissenschaftliche Schriftenreihe Eingebettete, selbstorganisierende Systeme* Bd. 5. Dresden, Germany : TUDpress, Mai 2007. – ISBN 978–3–940046–28–4, S. 11 – 16

[54] GLOCKNER, Matthias ; MAYER, Thomas: *Lastenheft: Sequentielles Schaltgetriebe, BMW Group*. unpublished, confidental, 2004

[55] GOLDFARB, Charles F. ; PRESCOD, Paul ; VERLAG, Addison W. (Hrsg.): *XML Handbuch*. Prentice Hall, 1999. – ISBN 3827317126

[56] GRABOWSKI, J. ; GRAUBMANN, P. ; RUDOLPH, E.: The Standardization of Message Sequence Charts. In: *Tagungsband zum IEEE Software Engineering Standards Symposium 1993*. IEEE, September 1993

[57] GRABOWSKI, Jens: *Test Case Generation and Test Case Specification with Message Sequence Charts*, Universität Bern, Diss., Februar 1994. – MSC

[58] GRABOWSKI, Jens ; RUDOLPH, Ekkart ; SCHMITT, Michael: Die Spezifikationssprachen MSC und SDL; Teil 1: Message Sequence Chart (MSC). In: *Automatisierungstechnik* 49 (2001), Dezember, S. 19–22

[59] GRAUBMANN, P. ; RUDOLPH: Hyper MSCs and Sequence Diagrams for Use Case Modeling and Testing. In: *Tagungsband zur UML2000*. 3rd International Conference on the Unified Modeling Language. Springer, Oktober 2000

[60] HAENELT, Karin: *Endliche Automaten - Äquivalenz regulärer Ausdrücke und endlicher Automaten*. Vorlesung, Mai 2006. – Universität Heidelberg

[61] HARDT, Wolfram: *HW/SW-Codesign auf Basis von C-Programmen unter Performanz-Gesichtspunkten*. Paderborn, Germany, University of Paderborn, Diss., 1996

[62] HARDT, Wolfram ; KERN, Wolfgang (Hrsg.) ; RAMMIG, Franz-Josef (Hrsg.): *Integration von Verzögerungszeit-Invarianz in den Entwurf eingebetteter Systeme*. Shaker Verlag, 2002. – ISBN 3–8265–8251–9

[63] HARDT, Wolfram ; KLEINJOHANN, Bernd: Flysig: Dataflow Oriented Delay-Insensitive Processor for Rapid Prototyping of Signal Processing. In: *Nineth IEEE International Workshop on Rapid System Prototyping*. IEEE Computer Society Press, 1999

[64] HARDT, Wolfram ; RAMMIG, Franz ; BÖKE, Carsten ; STROOP, Joachim ; RETTBERG, Achim ; DEL CASTILLO, G. ; KLEINJOHANN, Bernd ; TEICH, Jürgen: IP-based System Design within the PARADISE Design Environment. In: *Journal of Systems Architecture – the Euromicro Journal* (2001)

[65] HARDT, Wolfram ; ROSENSTIEL, Wolfgang: Prototyping of Tightly Coupled Hardware/Software-Systems. In: *Kluwer Journal: Design Automation for Embedded Systems* 2 (1997), Nr. 1

[66] HAROLD, Eliotte R.: *Die XML Bibel.* 2. Auflage. MIT Press, 2002. – ISBN 3826608216

[67] HAROLD, Elliote R. ; MEANS, W. S.: *XML IN A NUTSHHELL.* 2. Auflage. O'REILLY, 2003. – ISBN 3–89721–337–0

[68] HARREL, David ; KUGLER, Hillel: Synthesizing State-Based Object Systems from LSC Specifications. In: *Lecture Notes in Computer Science* 2088 (2001), Oktober, Nr. MCS99-20

[69] HAUGEN, Oystein: *Comparing UML2.0 Interactions and MSC-2000.* 4th International SDL and MSC Workshop, 2004. – SAM 2004

[70] HEINKEL, Ulrich: *Interface Specification and Validation using Extended Timing Diagrams.* PhD Forum DAC, San Francisco, 1998

[71] HEINKEL, Ulrich: *Formale Spezifikation und Validierung digitaler Schaltungsbeschreibungen mit Zeitdiagrammen*, Friedrich-Alexander-Universität Erlangen-Nürnberg, Diss., 1999

[72] HELLER, Ulrich ; SEIDL, Michael ; PFEIFROTH, Johannes ; BAUER, Gertrud: *MSC Editor - A Visual Editor for Message Sequence Charts with Function Catalog Support*, 2006. (Version 3.3.1)

[73] HENRIKSEN, Jesper G. ; MUKUND, Madhavan ; KUMARM, K. N. ; SOHONI, Milind ; THIAGARAJAN, P.S.: A theory of regular MSC languages. In: *Information and Computation* 202 (2005), Nr. 1, S. 1–38. – ISSN 0890–5401

[74] HOPCROFT, John E. ; MOTWANI, Rajeev ; ULLMAN, Jeffry D.: *Einführung in die Automatentheorie, Formale Sprachen und Komplexitätstheorie.* 2. überarbeitete Auflage. Addison-Wesley, 2002. – ISBN 3–8273–7020–5

[75] HOTTINGER, Daniel ; MEYER, Franziska: *XML-Diff-Algorithmen.* Semesterarbeit, Juli 2005

[76] IBM, Corporation: *XML Schema Infoset Model.* `http://www.eclipse.org/xsd/`. Version: August 2003

[77] ISO/IEC: *Information Technology - OSI Service Conventions.* ISO/IEC JTC1/SC 21 N 6341, Geneva, 1991

[78] *Kapitel* Z.120. In: ITU: *Message Sequence Chart (MSC).* ITU-T, 1993

[79] *Kapitel* Z.120. In: ITU: *Message Sequence Chart (MSC).* ITU-T, 1996

[80] *Kapitel* Z.120. In: ITU: *Message Sequence Chart (MSC).* ITU-T, 2000

[81] JERINIC, V. ; LANGER, J. ; HEINKEL, U. ; MUELLER, D: New Methods and Coverage Metrics for Functional Verification. In: *Design Automation and Test in Europe (DATE)*, 2006. – ISBN 3–9810801–0–6. – München

[82] KAIZHONG, Zhang ; SHASHA, Dennis: Simple fast algorithms for the editing distance between trees and related problems. In: *SIAM J. Comput.* 18 (1989), Nr. 6, S. 1245–1262. – ISSN 0097–5397

[83] KARRAS, Marcel: *Graphische Bedienöberfläche (GUI) zur Erstellung von XML-Dokumenten auf Basis nicht rekursiver XML-Schemata.* Studienarbeit, 2007

[84] KLEINOD, Ekkart: Modellbasierte Systementwicklung in der Automobilindustrie / Fraunhofer ISST. 2006 (77). – Forschungsbericht

[85] KLINGSEISEN, Jens: *Lastenheft: Junction Box Beifahrer (JBBFE3), BMW Group.* unpublished, confidental, 2006

[86] KLUS, Holger: *Erfassung, Beschreibung und Darstellung von Clusterstrukturen mittels XML-Technologien*, Fachbereich Informatik, Universität Kaiserslautern, Projektarbeit, August 2004

[87] KRÜGER, Ingolf H.: *Distributed System Design with Message Sequence Charts*, Technische Universität München, Diss., 2000

[88] LA FONTAINE, R.: *A Delta Format for XML: Identifying changes in XML and representing the changes in XML.* 2001

[89] LEE, Nam H. ; CHA, Sung D.: Generating A Reduced Finite State Machine from Concurrent Scenarios using Static Partial Order Method. In: *Journal of research and practice in information technology* 36 (2004), Nr. 3, S. 145–156

[90] LEE, Nam H. ; KIM, Tai H. ; CHA, Sung D.: Construction of Global Finite State Machine for Testing Task Interactions written in Message Sequene Charts. In: *SEKE'02* (2002)

[91] LÖTZBEYER, Heiko ; SANDNER, Robert ; NÖLLE, Oliver ; HEPPERGER, Peter ; BRAUN, Peter ; SLOTOSCH, Oscar: *Dynamische Schnittstellen-Modellierung.* BMW Group (intern) - Draft, Dezember 2005

[92] LOY, Marc ; ECKSTEIN, Robert ; WOOD, Dave: *Java Swing.* 2. Auflage. O'Reilly, 2002. – ISBN 0596004087

[93] MA, Zheng: *Analyse von Schnittstellenkompatibilität von Steuergeräten auf Basis von MSC-Beschreibungen*, TU-Chemnitz, Diplomarbeit, 2007

[94] MAIER, Karin: *Die langfristige Ersatzteilversorgung mit elektronischen Komponenten*, Fachhochschule Landshut, Diplomarbeit, 2003

[95] MARCHAL, Benoit: *XML - BY EXAMPLE.* Pierce, John, 2000. – ISBN 0–7897–2242–9

[96] MARIAN, Amelie ; YAHIA, Sihem A.: Adaptive Processing of Top-k Queries in XML. In: *icde* (2005), S. 162–173. – ISSN 1084–4627

[97] MARUYAMA, H. ; TAMURA, K. ; URAMOTO, R.: Digest values for DOM (DOM-Hash) proposal / IBM Tokyo Research Laboratory. 1999. – Forschungsbericht

[98] MATHWORKS: *Simulink.* `http://www.mathworks.de/products/simulink/`. Version: 2007

[99] MICROSOFT, Corporation: XML Diff and Patch / Microsoft Corporation. 2005. – Forschungsbericht

[100] MIDDENDORF, Lars ; MÜHLBAUER, Felix ; UMLAUF, Georg ; BOBDA, Christophe: Embedded Vertex Shader in FPGA. In: *IFIP International Federation for Information Processing.* Springer, 2007

[101] MIDDENDORF, Stefan ; SINGER, Reiner ; HEID, Jörn: *Java Programmierhandbuch und Referenz für die Java-2-Plattform.* 2002

[102] MINIAOUI, S. ; WENTLAND FORTE, M.: XML Mining: From Trees to Strings? In: *Second International Conference on Intelligent Computing and Information Systems (ICICIS 2005)*, 2005. – Kairo

[103] MOKOMA: *Vorhabensbeschreibung MOKOMA - Verbundprojekt in der Forschungsoffensive Software Engineering 2006.* confidental - unpublished, 2006

[104] In: MOST, Corporation: *MOST MSC Cookbook.* Rev 1.2. 2005, S. 62

[105] MOUAT, A.: Diffxml. Version: 2002. `http://diffxml.sourceforge.net`. 2002. – Forschungsbericht

[106] MSR-CONSORTIUM: *Entwicklung von Methoden, Definition von Standards, Nachfolgende Implementierung.* `http://www.msr-wg.de/msr.html`. Version: 2007

[107] MSR-MEDOC: *Element-und Attributdokumentation MSRSYS V1.2.0a.* `http://www.msr-wg.de/medoc/download/msrsys/v120a/msrsys-eadoc-de/msrsys-eadoc-de.pdf`. Version: Dezember 2001

[108] MSR-MEDOC: *Benutzerhandbuch MSRSYS V1.2.0a.* `http://www.msr-wg.de/medoc/download/msrsys/v120a/msrsys-ug-de/msrsys-ug-de.pdf`. Version: Februar 2002

[109] MSR-MEDOC: *MSR Working Group MEDOC, Presentation of Results.* `http://www.msr-wg.de/medoc/download/literature/msr-tr-medoc-en/msr-tr-medoc-en.pdf`. Version: Juli 2002

[110] MSR-MEDOC: *Proceesing-Guide für MSRSYS DTD.* `http://www.msr-wg.de/medoc/download/msrsys/v120a/msrsys-pg-de/msrsys-pg-de.pdf`. Version: Februar 2002

[111] MSR-MEDOC: *Structure Priciples of the MSRSYS DTD.* `http://www.msr-wg.de/medoc/download/msrsys/v120a/msrsys-sp-en/msrsys-sp-en.pdf`. Version: Februar 2002

[112] MSR-MEDOC: *Element Attribute Documentation V2.3.0.* `http://www.msr-wg.de/medoc/download/msrsw/v230/msrsw_v230-eadoc-en/msrsw_v2_3_0.sl-eadoc.pdf`. Version: Mai 2005

[113] MSR-VHDL-AMS: *Requirements, goals, and tasks of MSR Working Group VHDL-AMS.* `http://www.msr-wg.de/vhdl-ams/download/wg-present/presentation_graphic_v1.pdf`. Version: Juli 2000

[114] NAYAK, Richi ; WITT, Rebecca ; TONEY, Anton: Data Mining and XML Documents. In: *Proceedings International Conference on Internet Computing, IC'2002* Bd. 3, 2002. – Las Vegas

[115] NOLTE, Daniela ; KOOP, Johannes ; UNGERMANN, Christoph: *Validierung von XML-Dokumenten.* Ausarbeitung (Prof. Gössner). `http://goessner.net/download/learn/mwt/ws2005/presentations/XmlValidierung.pdf`. Version: Dezember 2005

[116] PETERS, Gabriele: *Endlicher Automat, reguläre Grammatik und regulärer Ausdruck.* Vorlesung, 2007

[117] PROSS, U. ; RICHTER, A. ; HEINKEL, U.: Plattform zur formalisierten Spezifikationserfassung. In: *8. Chemnitzer Fachtagung Mikrosystemtechnik Chemnitz*, 2007

[118] REICHELT, Stephan: *Entwicklung einer Spezifikationsbeschreibung für statische Eigenschaften von Steuergeräte-Interfaces*, TU-Chemnitz, Diplomarbeit, September 2006

[119] RENNER, T. ; BLUHM, T. ; SCHNEIDER, A. ; KNAEBLEIN, J. ; ZAVALA, R. ; HEINKEL, U.: Formale Spezifikation und Verifikation abstrakter Beschreibungen von Telekommunikationsprotokollen. In: *9. GI/ITG/GMM Workshop Methoden und Beschreibungssprachen*, 2006. – ISBN 3–9810287–1–6. – Dresden

[120] RUDOLPH, E. ; GRABOWSKI, J. ; GRAUBMANN, P.: Towards an Harmonization of UML-Sequence Diagrams and MSC. In: *SDL'99-The next Millenium.* Elsevier Science Publishers, Juni 1999

[121] SCHÄUFFELE, Jörg ; ZURAWKA, Thomas ; VIEWEG (Hrsg.): *Automotive Software Engineering.* 2004

[122] SCHIEFERDECKER, Ina ; GRABOWSKI, Jens: The Graphical Format of TTCN-3 in the Context of MSC and UML. In: *SAM2002, LNCS 2599* (2003), S. 233–252

[123] SCHLIEDER, Torsten ; NAUMANN, Felix: Approximate Tree Embedding for Querying XML Data / Humboldt-Universitaet zu Berlin. 2000. – Forschungsbericht

[124] SCHMITT, Thomas: *Evaluierung von Bewertungsmethoden für das Baukastenprinzip*, Fachhochschule Landshut, Diplomarbeit, 2005

[125] SCHNEIDER, A. ; WALTER, S. ; LANGER, J. ; HEINKEL, U.: Concept for Automatic Visualization of Abstract System Specifications. In: *Sixth International Conference on Quality Software (QSIC'06)*, 2006. – ISBN 0–7695–2718–3. – Beijing (China)

[126] SCHOLAND, Andreas: *Konzeption und Implementierung einer Java-basierten Integrationsplattform für Software Module*, Universität Paderborn, Studienarbeit, August 2003

[127] SCHÖNING, Uwe: *Theoretische Informatik - kurzgefasst*. 4. Auflage. Spektrum, 2001. – ISBN 3–8274–1099–1

[128] SCHÜLER, Michael: *Netcharts*, RWTH Aachen, Diplomarbeit, 2005

[129] SKULSCHUS, Marcus ; WIEDERSTEIN, Marcus: *XML-Schema*. 1. Auflage. Galileo Press, 2004. – ISBN 3–89842–472–5

[130] SONG, Bhowmick: BioDIFF: An Effective Fast Change Detection Algorithm for Genomic and Proteomic Data. 2004. – Forschungsbericht

[131] TELELOGIC: *DOORS*. `http://www.telelogic.de/products/doors/index.cfm`. Version: 2007

[132] TIMMERMANN, Kay: *Entwurf von Vergleichsalgorithmen für XML Metamodelle*, TU Chemnitz, Diplomarbeit, Juli 2007

[133] UZUN: *Lastenheft: Zentrale Karroserie-Elektronik (ZKE V), BMW Group*. unpublished, confidental, 2004

[134] VALIENTE, G.: *Algorithms on Tress and Graphs*. Springer, Berlin, 2002. – ISBN 3540435506

[135] VDA, Verband der A.: *Automotive VDA-Standardvorlage Komponentenlastenheft*. `http://vda-qmc.de/de/index.php?catalog=1075`. Version: 2006

[136] VISARIUS, Markus ; HARDT, Wolfram: An IPQ Format based Toolbox to Support IP based Design. In: *it - Information Technology* (2005), S. 98–106

[137] VISARIUS, Markus ; LESSMANN, Johannes ; HARDT, Wolfram ; KELSO, Frank ; THRONICKE, Wolfgang: An XML Format based Integration Infrastructure for IP based Design. In: *Proceedings of the 16th Symposium on Integrated Circuits and Systems Design (SBCCI).* Sao Paulo, Brazil : IEEE Computer Society, September 2003, S. 119–124

[138] VISARIUS, Markus ; LESSMANN, Johannes ; KELSO, Frank ; HARDT, Wolfram: Generic Integration Infrastructure for IP based Design Processes and Tools with a Unified XML Format. In: *Integration - the VLSI journal* 37 (2004), September, Nr. 4, S. 289 – 321

[139] VLIST, Eric van d.: *XML-Schema.* 1. Auflage. O'Reilly, 2003. – ISBN 3–89721–345–1

[140] WALLENTOWITZ, Henning (Hrsg.) ; REIF, Konrad (Hrsg.): *Handbuch Kraftfahrzeugelektronik.* Wiesbaden : Vieweg Verlag, 2006. – ISBN 978–3528–03971–4

[141] WANG, Y. ; DEWITT, D. J. ; CAI, J. Y.: X-Diff: an effective change detection algorithm for XML documents. In: *Data Engineering, 2003. Proceedings. 19th International Conference on* (2003), 519–530. `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1260818`

[142] WATSON, Dennis G.: *Brief History Of Document Markup.* `http://edis.ifas.ufl.edu/BODY_AE038`. Version: 2002

[143] WHITTLE, Jon ; SCHUMANN, Johann: Generating statechart designs from scenarios. In: *International Conference on Software Engineering (ICSE)* (2000)

[144] WOHNHAAS, Achim ; MOSER, Rainer: Modellaustausch zwischen Steuergeräte-Entwicklungstools auf Basis einheitlicher grafischer Blockbibliotheken. In: *3. Stuttgarter Symposium.* Expert Verlag, 1999. – ISBN 3–8169–1751–8

[145] WOHNHAAS, Achim ; MOSER, Rainer ; BRANGS, Peter: Standardisierte Blockbibliothek zum modellbasierten Steuergeräte-Softwareentwurf als Basis für den Modellaustausch zwischen Entwicklungstools / MSR Megma. – Forschungsbericht

[146] XML-EDITOR: *Altova - xmlspy-2007.* `http://www.altova.com/products/xmlspy/xml_editor.html`. Version: 2007

[147] XU, Haiyuan ; WU, Quanyuan ; WANG, Huaimin ; YANG, Guogui ; JIA, Yan: KF-Diff+: Highly Efficient Change Detection Algorithm for XML Documents. In: *CoopIS/DOA/ODBASE*, 2002, S. 1273–1286

[148] ZEIGLER, Bernard P. ; PRAEHOFER, Herbert ; GON KIM, Tag: *Theory of Modeling and Simulation.* ACADEMIC PRESS, 2000 (Integration Discrete Event and Continuous Complex Dynamic Systems). – ISBN 0127784551

[149] ZEZULA, Pavel ; MANDREOLI, Federica ; MARTOGLIA, Riccardo: *Unordered XML Pattern Matching with Tree Signatures (Extended Abstract)*. SEBD 2004, Juni 2004

[150] ZHANG, K. ; SHASHA, D. ; STATMAN, R. ; SHASHA, D.: On the editing distance between unordered labeled trees. In: *Inf. Process. Lett.* 42 (1992), Nr. 3, S. 133–139. – ISSN 0020–0190

[151] ZHANG, Kaizhong: A New Editing based Distance between Unordered Labeled Trees. In: *CPM '93: Proceedings of the 4th Annual Symposium on Combinatorial Pattern Matching*. London, UK : Springer-Verlag, 1993. – ISBN 3–540–56764–X, S. 254–265

[152] ZHAO, Qiankun ; CHEN, Ling ; BHOWMICK, Sourav S. ; MADRIA, Sanjay: XML Structural Delta Mining: Issues and Challenges / University Singapore. 2005. – Forschungsbericht