Fortschritt-Berichte VDI

VDI

Reihe 8

Mess-,
Steuerungs- und
Regelungstechnik

Nr. 1261

Dipl.-Inform. Tina Mersch,
Verl

Regelbasierte Modelltransformation in prozessleittechnischen Laufzeitumgebungen

Lehrstuhl für
Prozessleittechnik
der RWTH Aachen

Regelbasierte Modelltransformation in prozessleittechnischen Laufzeitumgebungen

Von der Fakultät für Georessourcen und Materialtechnik
der Rheinisch-Westfälischen Technischen Hochschule Aachen

zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

genehmigte Dissertation

vorgelegt von Dipl.-Inform.

Tina Mersch

aus Sömmerda

Berichter: Univ.-Prof. Dr.-Ing. Ulrich Epple
Univ.-Prof. Dr. rer. nat. Andy Schürr

Tag der mündlichen Prüfung: 08.Dezember 2017

Diese Dissertation ist auf den Internetseiten der Hochschulbibliothek online verfügbar

Fortschritt-Berichte VDI

Reihe 8

Mess-, Steuerungs- und Regelungstechnik

Dipl.-Inform. Tina Mersch,
Verl

Nr. 1261

Regelbasierte Modelltransformation in prozessleittechnischen Laufzeitumgebungen

Lehrstuhl für
Prozessleittechnik
der RWTH Aachen

Mersch, Tina
Regelbasierte Modelltransformation in prozessleittechnischen Laufzeitumgebungen
Fortschr.-Ber. VDI Reihe 08 Nr. 1261. Düsseldorf: VDI Verlag 2018.
160 Seiten, 54 Bilder, 0 Tabellen.
ISBN 978-3-18-526108-1 ISSN 0178-9546,
€ 57,00/VDI-Mitgliederpreis € 51,30.

Für die Dokumentation: Modelltransformation – Automatisierungstechnik – Durchgängiges Engineering – Triple-Graph Grammatiken – TGG – Modellbasiertes Engineering – Regelbasiertes Engineering – Anlagenneutrale Automatisierungsfunktion – Automatisierung der Automatisierung

Der aus der Informatik stammende Ansatz der Modelltransformation mittels Triple-Graph-Grammatiken wird in die Welt der IEC61131-Sprachen überführt. Das dadurch entstandene Framework bietet die Grundlage für anlagenneutrale Automatisierungsfunktionen, die als Serienprodukt verkauft und per Modelltransformation anhand der Planungsdaten an die konkrete Anlage und die aktuellen Anforderungen angepasst werden können. Durch den Einsatz von Triple-Graph-Grammatiken ist es zudem möglich, Änderungen in der Automatisierungsfunktion in die Planungsdaten zurück zu spielen und somit zu dokumentiert. Das vorgestellte Konzept macht sich die starke Korrelation zwischen verschiedenen Modellen der Anlagenautomatisierung zu Nutze, indem es die Zusammenhänge und nicht das Modell selbst in den Fokus rückt. Das Wissen über diese Zusammenhänge wird dabei, abgelegt als Regeln, nutzbar für eine ganze Serie von Anlagen.

Bibliographische Information der Deutschen Bibliothek
Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliographie; detaillierte bibliographische Daten sind im Internet unter www.dnb.de abrufbar.

Bibliographic information published by the Deutsche Bibliothek
(German National Library)
The Deutsche Bibliothek lists this publication in the Deutsche Nationalbibliographie (German National Bibliography); detailed bibliographic data is available via Internet at www.dnb.de.

D82 (Diss. RWTH Aachen University, 2017)

ISSN 0178-9546
ISBN 978-3-18-526108-1

Vorwort

Die vorliegende Dissertation entstand während meiner Tätigkeit am Lehrstuhl für Prozessleittechnik der RWTH Aachen University. Ich möchte mich an dieser Stelle bei allen bedanken, die geholfen haben diese Arbeit erfolgreich abzuschließen. Mein besonderer Dank gilt dabei Herrn Professor Dr.-Ing. Ulrich Epple, der in seiner Rolle als Doktorvater und Chef durch spannende Diskussionen, neue Denkanstöße und durch die vielfältigen Möglichkeiten zum Austausch mit anderen Wissenschaftlern maßgeblich zum Gelingen dieser Arbeit beigetragen hat.

Aus der ursprünglich wagen Idee Modelltransformation in die Automatisierungstechnik zu bringen, hat sich insbesondere durch die langen und fachlich sehr lehrreichen Gespräche mit Herrn Professor Dr. rer. nat. Andy Schürr ein tragfähiges und wirklich spannendes Konzept entwickelt. Für die dabei aufgebrachte Geduld, die Nachsicht in vielen Dingen und nicht zuletzt für die Übernahme der Rolle des Zweitgutachters möchte ich mich bei ihm herzlich bedanken.

Auch den vielen Wegbegleitern sei ein Dank ausgesprochen. Besonders erwähnen möchte ich dabei Stefan Schmitz, dessen Ideen und Ansätze die Grundlage dieser Arbeit lieferten und Gustavo Quirós, der gerade in den ersten Phasen viele gute Ideen und Anwendungsmöglichkeiten für eine Modelltransformation in der Automatisierungstechnik beigetragen hat und mich dadurch motiviert hat, das Thema zu vertiefen. Auch möchte ich mich bei Marius Lauder bedanken, der mir in Gesprächen und gemeinsamen Arbeiten Einblicke in die Funktionsweise von TGGen gewährt und mir die Faszination dieses Ansatzes nähergebracht hat.

Ein besonderer Dank gilt meiner Familie. Angefangen bei meinen Eltern, die mir durch ihre Unterstützung auch bei ungewöhnlichen Ausbildungswünschen erst ermöglicht haben, diesen Werdegang einzuschlagen und bei meinen Kindern Liam und Tjard, die gerade in den heißen Phasen der Arbeit oft zurückstecken mussten, die mir aber auch immer wieder durch kleine Gesten über die unzähligen Tiefpunkte hinweggeholfen haben. Der größte Dank gilt jedoch meinem Mann Henning, der unendlich viel Geduld während der Entstehung der Arbeit aufgebracht hat und mich immer wieder motiviert hat, nicht aufzugeben.

Verl, im Oktober 2018 *Tina Mersch*

Most of the fundamental ideas of science are essentially simple, and may, as a rule, be expressed in a language comprehensible to everyone.

Albert Einstein

Inhaltsverzeichnis

Kurzfassung

Eine Umfrage unter 1800 Mitgliedern des Verbands Deutscher Maschinen- und Anlagenbauer (VDMA) [Sch12] zeigte, dass sich schon heute 61% der befragten Entwicklungs- und Konstruktionsingenieure den Herausforderungen bei der Entwicklung neuer Maschinen und Anlagen nicht mehr in vollem Umfang gewachsen fühlen. Mehr als die Hälfte der Befragten erwarten sogar, dass „die Technik, die für die Erstellung effizienter, leistungsfähiger und flexibler Maschinen benötigt wird, immer aufwendiger wird“. Zudem „nehmen Kompetenz und Qualifikation auf der Anwender- und Bedienerseite ab“. Dieses Zusammentreffen von steigender Komplexität und sinkendem Fachwissen verlangen nach neuen Methoden im Engineering von Anlagen. Anne Schneller, die diese Umfragen im Rahmen des VDI-Artikels vorstellte, schlägt vor, dass der Weg der Automatisierungstechnik in Richtung „Parametrieren statt Implementieren“ zu lenken ist, um diesen Herausforderungen auch in Zukunft gewachsen zu sein. Die vorliegende Arbeit leistet einen Beitrag dazu, dieses Paradigma auch für komplexe Automatisierungsfunktionen zugänglich zu machen.

Das in dieser Arbeit vorgestellte Konzept macht sich die starke Korrelation zwischen verschiedenen Modellen der Anlagenautomatisierung zu Nutze, indem es die Zusammenhänge und nicht das Modell selbst in den Fokus rückt. Die Verwendung von Modelltransformation als Basis einer anlagenneutralen Realisierung der Automatisierungsfunktion ermöglicht die Anpassung der Funktionalität an den konkreten Anlagenkontext durch Parametrieren mit den anlagenspezifischen Planungsdaten. Das Fachwissen wird dabei, abgelegt als Regeln, nutzbar für eine ganze Serie von Anlagen.

Die Methode der regelbasierten Modelltransformation hat ihren Ursprung in der Informatik, wo die entwickelten Ansätze bereits beachtliche Ergebnisse in den für sie geschaffenen Modellwelten erzielen. Trotz der langjährigen, erfolgreichen Entwicklung auf dem Gebiet der Modelltransformation stellt das Anwendungsgebiet der Automatisierungstechnik bisher eine besondere Herausforderung dar. Insbesondere semiformale Modellbeschreibungen, die Vielfalt der Modelle, erlaubte Varianzen in der Modellierung und multiple Quellmodelle erschweren den Einsatz von Standardverfahren oder machen ihn unmöglich. Nicht nur die hohen Anforderungen der Informatik an den Formalisierungsgrad und die Passgenauigkeit der beteiligten Modelle stellen eine Hürde bei der Zusammenführung der beiden Disziplinen dar, auch die konservative Einstellung der Automatisierungstechnik bringen besondere Herausforderung mit sich. Das in der Arbeit vorgestellte Konzept realisiert einen der erfolgversprechendsten Ansätze aus der Informatik und gliedert diesen nahtlos in für die Automatisierungstechnik übliche Programmiersprachen ein. Dem Applikateur bieten sich dadurch alle Freiheiten der kooperativen Nutzung von Modelltransformation und Standardprogrammierung.

Abstract

A survey of 1800 members of the German engineering association VDMA [Sch12] showed that 61% of development and design engineers surveyed don't feel up to the challenges in the development of new machines and equipment. More than one half of the respondents expect that the development of efficient, powerful and flexible machines will become even more complex in future. In addition, they predict that users and operators will become less qualified. This concurrence of increasing complexity and decreasing knowledge demands new methods in the engineering of plants. Anne Schneller, author of the VDI article about the survey, suggests that automation technology has to progress toward "parameterization instead implementation" to cope with these challenges. The work at hand aims to contribute to make this paradigm applicable for complex automation functions.

The approach presented in this work takes advantage of the strong correlation between different models of plants. It uses model transformation as the basis of a system-neutral development of automation functions. Those automation functions can be parametrized with the plant-specific planning data without further coding costs. The knowledge about the model correlations is stored once as rules, available for a large set of plants.

The method of rule-based model transformation has its roots in computer science, where the approaches already developed achieved significant results in the model worlds created for them. Despite of many years of progress in the field of model transformation, automation technology presents special challenges for adoption. In particular, semiformal model descriptions and the variances in modeling as well as multiple source models make the use of standard methods impossible. Not only are the high demands of computer science on formalization and the fit of the participating models a hurdle in merging the two disciplines but the conservative attitude of automation technology brings particular challenges with it as well. One requirement for the acceptance of these approaches in automation technology is the smooth integration of the concepts in the application domain without ignoring the domain experts. The concept presented realizes one of the most promising approaches from computer science and integrates it seamlessly into automation programming. With this approach the installation technician can combine model transformation and standard programming in accordance with his purposes.

1 Einleitung

Die steigende Komplexität bei der Automatisierung von Anlagen und der erwartete Fachkräftemangel werden insbesondere bei der Erstellung der Software ihre Auswirkung entfalten. Dies liegt daran, dass alleine 10% des Gesamtaufwandes beim Bau einer Anlage in der Erstellung der Software stecken. Darüber hinaus ist es schwierig, die Fehleranfälligkeit weiter zu minimieren und die Korrektheit der implementierten Funktionalität zu verifizieren [NA35]. Eine anlagenneutrale, modularisierte Softwarelösung, die durch Parametrierung an den konkreten Kontext angepasst wird, hat positive Auswirkungen auf die Projektierungszeit und Fehleranfälligkeit. Zudem ergeben sich Synergieeffekte für alle Phasen des Lebenszykluses einer Anlage. Das zu erwartende Potential eines solchen Ansatzes sowie die Zielsetzung der Arbeit werden in den folgenden Abschnitten konkretisiert.

1.1 Motivation

Applikationen zur Ansteuerung verfahrenstechnischer Anlagen sind in den meisten Fällen so individuell wie die anzusteuernde Anlage selbst. Doch auch wenn jede Applikation anlagenspezifisch erstellt werden muss, bilden branchenspezifische Basisfunktionen wie Einzelsteuerungen für die Aktorik/Sensorik, Verriegelungslogiken, Flusswegkontrolle etc. die Grundlage der Programmierung. Diese müssen bei gegebenem Anlagentyp immer in gleicher oder ähnlicher Form realisiert werden. Auch wenn die Umsetzung dieser Basisfunktionen mit Hilfe von Bibliotheken erfolgt, deren Bausteine „nur“ in die Applikation übernommen und parametriert werden müssen, sind die Basisfunktionen für den Applikateur meist reine Fließbandarbeit mit wenig kreativer Eigenleistung, „langweilig“ und „stupide“. Hinzu kommt, dass auf Grund des reinen Umfangs an Basisfunktionen viel Zeit in deren Umsetzung fließt. Für die Realisierung der anlagenspezifischen Applikationskomponenten, die spezielles Fachwissen und einen hohen Grad an Individualleistung vom Applikateur erfordern, steht entsprechend weniger Zeit zur Verfügung. Und das, obwohl gerade hier die Kernkompetenzen eines gut ausgebildeten Applikateurs liegen.

Auf der anderen Seite sind die Erhöhung der Flexibilität, der Skalierbarkeit und der Ausfallsicherheit nicht erst seit den Arbeiten im Rahmen der Initiative Industrie 4.0 [I4.0] Themen, mit denen sich Anlagenbauer und Applikateuer konfrontiert sehen. Dennoch erhöht die unter Leitung des Bundesministeriums für Bildung und Forschung (BMBF) und des Bundesministeriums für Wirtschaft und Energie (BMWi) laufende Initiative den Druck auf die Anlagenbauer, diese Anforderungen verstärkt zu berücksichtigen. Hinzu kommen neue Themen für die Realisierung zukunftsfähiger Anlagen wie die Ad-hoc-Anpassung der Produktion an veränderte

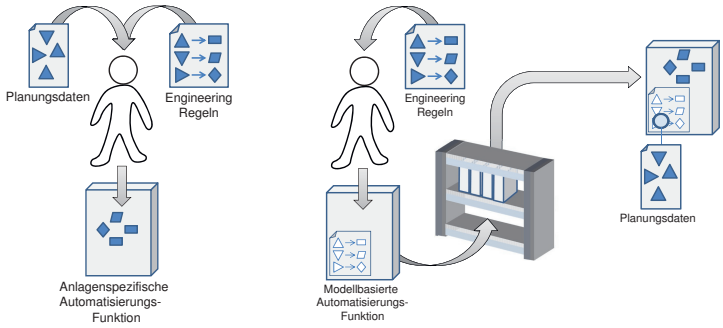
Marktanforderungen, die auftragsorientierte Vernetzung der Produktionsstätten, dynamische Anlagenstrukturen und damit einhergehend eine verstärkte Modularisierung der Anlagen.

Während bei der Modularisierung der Hardware auf die Erfahrungen der Fertigungstechnik zurückgegriffen werden kann, fehlt es bei der Entwicklung der passenden Softwaresysteme zur Automatisierung von modularen Anlagen noch an Erfahrungen [Mer+11]. Zwei grundlegende Ansätze sind hierbei denkbar. Der modulbasierte Ansatz, wie er in der Fertigungstechnik üblich ist, automatisiert jedes Hardwaremodul für sich. Dieser Ansatz hat den Vorteil, dass die Automatisierungssoftware optimal auf das Hardwaremodul abgestimmt und mit diesem als eine automatisierungstechnische Komponente ausgeliefert werden kann. Für den Bereich der Prozessautomatisierung ist diese Herangehensweise allerdings ungeeignet, da für gewöhnlich eine anlagenweite Automatisierung über die Modulgrenzen hinweg benötigt wird. Hier bietet sich der Einsatz modellbasierter Entwicklung an. Wenn es gelingt, Automatisierungslösungen anlagenneutral auf Basis der zugrundeliegenden Modelle zu implementieren, kann bei einer Hardwarerekonfiguration einer modularen Anlage die Automatisierungssoftware mit Hilfe des neuen Anlagenmodells automatisch angepasst werden. Dies erfordert jedoch einen grundlegenden Paradigmenwechsel in der Planung und Realisierung von Anlagen weg von der instanzgetriebenen hin zur modellgetriebenen Entwicklung. Auch Anlagen in herkömmlicher, nicht-modularer Bauweise können von einem solchen Paradigmenwechsel profitieren. Standardisierte Automatisierungsfunktionen könnten als Serienprodukt erworben, mit den Anlagenplanungsdaten konfiguriert und in Betrieb genommen werden. Neben der deutlichen Kostenreduktion, den verkürzten Entwicklungszeiten und dem Qualitätsgewinn durch Einsatz von Standardkomponenten, sind auch verkürzte Abnahme- und Prüfverfahren zu erwarten. Zwar bietet das Umfeld der Automatisierungstechnik gute Voraussetzungen für den Einsatz modellbasierter Entwicklung, das Potential bleibt zurzeit jedoch aus mehreren Gründen ungenutzt. So stehen für viele Abschnitte des Engineeringprozesses genormte bzw. standardisierte Modelle (z.B. [PandIX], [IEC61131]) zur Verfügung. Im Planungsprozess werden die Daten bereits modellübergreifend genutzt (z.B. [ST10]) oder zwischen den Modellen ausgetauscht (z.B. [IEC62424]). Statt die Implementierung von Leitsystemfunktionen aber unter Einbezug der vorhandenen Modellinformationen zu machen, werden durch explizites Ausprogrammieren der Funktion Insellösungen für konkrete Anlagen generiert.

1.2 Zielsetzung

Ziel dieser Arbeit ist es, den Applikateur entsprechend seiner Kernkompetenzen einzusetzen und ihn von der Umsetzung der Standardfunktionen zu entbinden. Dabei wird ausgenutzt, dass in der Automatisierungstechnik nicht nur die Anlagenplanung mit Hilfe von standardisierten Modellen erfolgt, sondern häufig auch die Implementierung. Neben explizit formulierten Modellen wie dem HMI [Sch10] oder der Prozessführung [SME05], liegen die Modelle häufig implizit über Engineeringregeln vor. Diese Engineeringregeln beschreiben die Zusammenhänge zwischen Planungsdaten und den zu erstellenden Objektstrukturen in der Implementationsphase.

In dieser Arbeit wird ein Konzept vorgestellt, das basierend auf regelbasierter Modelltransformation die bereits etablierte Arbeits- und Denkweise anhand von Engineeringregeln unterstützt.



(a) Instanzbasiert (b) Modellbasiert

Abbildung 1.1: Engineering einer Automatisierungsfunktion

Statt wie bisher die Engineeringregeln zu nutzen um von Hand die Planungsdaten in eine Implementierung zu überführen (vgl. Abbildung 1.1a), arbeitet der Applikateur auf Modellebene und beschreibt einmalig die Modellzusammenhänge in Form von Regeln (vgl. Abbildung 1.1b). Diese anlagenneutrale Herangehensweise über die Formulierung der zugrunde liegenden Engineeringregeln ermöglicht eine Verwendung der Automatisierungsfunktion als Serienprodukt, das anschließend mit den Planungsdaten für die Verwendung an der konkreten Anlage parametriert wird. Die Arbeit der wiederkehrenden, einfachen Regeln folgenden Instanziierung und Programmierung durch den Applikateur kann dadurch signifikant reduziert werden.

Hauptziel der Arbeit ist die Bereitstellung von kompletten Automatisierungsfunktionen als Serienprodukte. Abbildung 1.2 skizziert das angestrebte Vorgehen bei der Automatisierung einer verfahrenstechnischen Anlage. Die Umsetzung der Basisfunktionen soll mit Hilfe anlagenneutraler Automatisierungslösungen erfolgen. Diese sollen auf Grundlage von Planungsdaten und dem aktuellen Zustand der Anlage mittels regelbasierter Modelltransformation die eigentlichen Basisfunktionen anlagenspezifisch bereitstellen. Instanzen der anlagenneutralen Automatisierungslösungen, die realisierten Basisfunktionen und die anlagenspezifischen Applikationskomponenten müssen parallel zueinander in einem Laufzeitsystem ausgeführt werden können. Nur durch diese Integration ist es dem Applikateur möglich, die gesamte Anlagenautomatisierung auf einer Hardware zu realisieren und auf die Ergebnisse der modellbasiert erstellten Automatisierungsfunktionen zuzugreifen, um sie in seinen eigenen Lösungsstrategien nutzen zu können. Das in dieser Arbeit bereitgestellte Konzept muss daher eine vollständige Integration der Modelltransformation in die automatisierungstechnische Laufzeitumgebung ermöglichen. Unter dem Begriff Laufzeitumgebung/Laufzeitsystem wird dabei eine Speicherprogrammierbare Steuerung (SPS, engl. programmable logic controller, PLC) verstanden. Ist die Modelltransformation Teil der auf der SPS ausgeführten Applikation und wird somit während der Bearbeitung des Applikationscodes durchgeführt, so wird in dieser Arbeit von „Modelltransformation zur Laufzeit“ gesprochen.

Neben der Bereitstellung kompletter Automatisierungsfunktionen als anlagenneutrale Serienprodukte kommen aber auch andere Szenarien für den Einsatz regelbasierter Modelltransfor-

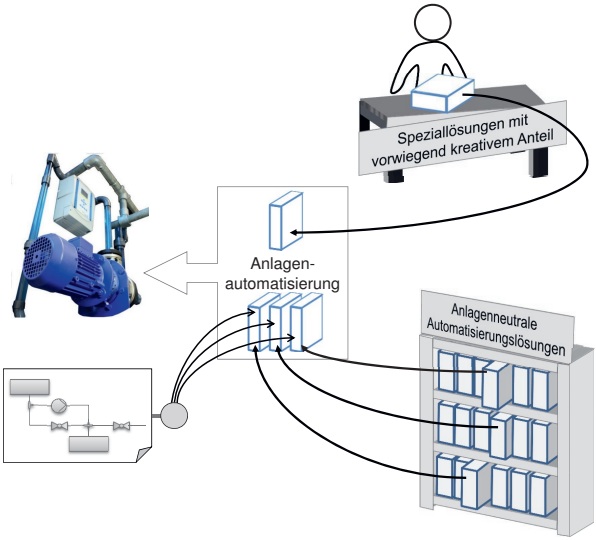


Abbildung 1.2: Konfigurieren statt Implementieren

mation in der Automatisierungstechnik in Frage. Einen kurzen Überblick über forcierte Einsatzmöglichkeiten sowie deren besondere Anforderungen an das zu entwickelnde Konzept geben die folgenden potentiellen Anwendungsszenarien.

S1. Einzelne Automatisierungsfunktion als Serienprodukt

Eine in sich abgeschlossene Funktionalität, die anlagenneutral formuliert werden kann, ist das Detektieren von Leckagestellen sowie unerwünschter Vermischungen verschiedener Medien in einer verfahrenstechnischen Anlage mit Hilfe der Flussweganalyse [Qui11; GWE14]. Basierend auf der Anlagenstruktur und dem aktuellen Zustand der Aktoren (z.B. Öffnungszustand von Ventilen) wird das Bedienbild mit entsprechenden Informationen angereichert. Drei Teilszenarien sind hier von prinzipiellem Interesse:

(a) Bei diesem Szenario soll der Anlagenfahrer eine Anfrage stellen können, welche Bereiche der Anlage durch das Öffnen eines konkreten Ventils oder durch das Anschalten einer Pumpe betroffen sind. Die Rohrleitungen, die ausgehend von diesem Aktor durch geöffnete Ventile und angeschaltete Pumpen miteinander verbunden sind, werden im Bedienbild farbig markiert.

(b) In verfahrenstechnischen Anlagen findet das Mischen von verschiedenen Medien im Allgemeinen in Reaktoren statt. Mischen in Rohren durch gleichzeitiges Einpumpen verschiedener Medien ist eher unüblich. Gleiches gilt für den Medienablass über die Systemgrenzen hinaus, sofern sich dort kein definierter Auffangpunkt (z.B. ein Tanklaster) befindet. Beide ungewollten Zustände sollen durch Warnungen im Bedienbild kenntlich gemacht werden.

(c) Das Mischen von verschiedenen Medien außerhalb der dafür vorgesehenen Reaktoren kann durch das Sperren der Anlage gegen unerwünschte Aktorbedienung verhindert werden. Dazu wird bei Aktoranfrage ein entsprechend Szenario S1.b) konfigurierter Suchlauf gestartet und ergebnisabhängig der Aktor freigegeben oder gesperrt.

Alle drei Teilszenarien lassen sich nach einfachen Regeln aus der Anlagenstruktur und dem aktuellen Anlagenzustand realisieren. Einmal implementiert kann die Flusswegeanalyse daher „von der Stange“ für jede Anlage eingesetzt werden. Für eine dynamisch umrüstbare Anlage, deren Struktur sich ständig ändert (z.B. M4P.AC), eignet sich der Einsatz des modellbasierten Ansatzes besonders gut. Statt einer Reprojektierung erfolgt lediglich eine Rekonfiguration anhand der neuen Strukturdaten der Anlage.

Ziel ist es, diese Funktionalitäten als eigenständige Serienprodukte bereitzustellen.

S2. Entwicklungsbegleitende Modelltransformation

Das Bedienbild einer konventionellen Anlage ist nur ein Beispiel für extrem spezialisierte Automatisierungsfunktionen. Die benötigten Anzeigen, Detailbilder und ähnliches sind zum Teil einzigartig und folgen keinen festen Regeln, nach denen sie aus den Planungsdaten erstellt werden können. Zwar gibt es bereits Ansätze, auch hier modellbasiert komplette Bedienbilder durch Anreicherung der Planungsdaten zu erstellen [UOS12], an dieser Stelle soll jedoch die Integration von modellbasierter Entwicklung in die konventionelle Bedienbilddarstellung im Vordergrund stehen.

Das Szenario sieht die modellbasierte Erstellung eines einfachen Bedienbildes vor, auf dem für jeden Sensor und jeden Aktor in den Planungsdaten ein repräsentierendes Symbol sowie eine Detailansicht für das Ablesen von Sensorwerten bzw. das Absetzen von Befehlen im Bedienbild erstellt wird. Eine anschließende Anpassung durch den Applikateur an die anlagenspezifischen Gegebenheiten muss dabei möglich bleiben. Inkrementelle Änderungen an den Planungsdaten müssen in das möglicherweise geänderte Bedienbild übertragen werden können.

S3. Konsistenzanalyse und Modellreparatur

Die Realisierung der Basisautomatisierung ist zumeist so anlagenspezifisch, dass eine regelbasierte Generierung nicht zielführend ist. Dennoch kann der Applikateur auch bei diesen Aufgaben durch eine regelbasierte Modelltransformation unterstützt werden, zum einen durch eine Konsistenzanalyse des fertigen Bedienbildes mit den zugehörigen Ansteuerbausteinen der Prozessführung und zum anderen durch eine regelbasierte Modellreparatur für den Fall, dass Inkonsistenzen zwischen Bedienbild und Prozessführung erkannt wurden.

Ausgangspunkt der drei Anwendungsszenarien ist das Anlagenstrukturmodell PandIX [PandIX] (vgl. Abbildung 1.3). PandIX ist ein Modell zur XML-basierten Repräsentation der für die Automatisierungstechnik relevanten Informationen des Rohrleitungs- und Instrumentenfließbildes (Abk. R&I-Fließbild) [DIN10628] einer verfahrenstechnischen Anlage. Neben der plattformunabhängigen Version nach [PandIX] kommt bei den Anwendungsszenarien auch die plattformabhängige Version ACPLT/PandIX für die ACPLT-Modellwelt zum Einsatz sowie dessen Instanz im Laufzeitsystem. Die ACPLT-Modellwelt (kurz ACPLT) ist die Modellierungsumgebung des

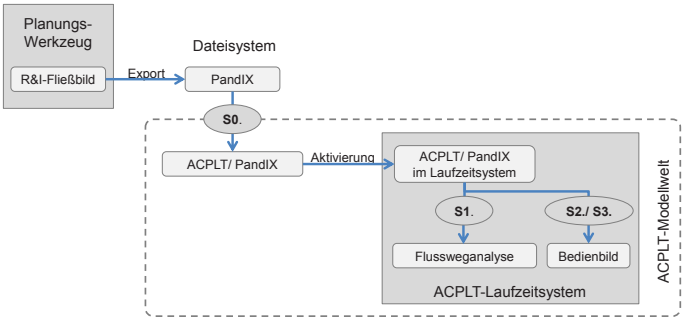


Abbildung 1.3: Anwendungsszenarien

Lehrstuhls für Prozessleittechnik der RWTH Aachen University. Sie besteht aus einem Satz vielfältiger prozessleittechnischer Modelle und ihrer Realisierung in einer Laufzeitumgebung.

Die aufgeführten Anwendungsszenarien basieren auf der Annahme, dass die zur Parametrierung benötigten Planungsdaten bereits im System vorliegen. In der Regel ist jedoch das Laufzeitsystem disjunkt vom Engineeringwerkzeug für die Planungsdaten. Ein viertes Anwendungsszenario des modellbasierten Ansatzes ist der Parametrierung von fertigen Softwaremodulen aus Anlagenplanungsdaten daher vorgeschaltet:

S0. Bereitstellung von Planungsdaten im Laufzeitsystem
Es wird ein bidirektionaler Datenaustausch zwischen den mit Hilfe von Planungswerkzeugen erstellen Dokumenten und einem erkundbaren Abbild der Planungsdaten im Laufzeitsystem bereitgestellt. Exemplarisch soll in diesem Anwendungsszenario das Anlagenstrukturmodell für die Weiterverarbeitung in S1, S2 und S3 aufbereitet werden.

1.3 Gliederung

Die Struktur der Arbeit (vgl. Abbildung 1.4) trägt der Interdisziplinarität des Themas Rechnung, indem zunächst der Stand der Technik beider Disziplinen erörtert und zueinander in Relation gesetzt wird. Aufbauend darauf erfolgt im Hauptteil der Arbeit die Entwicklung eines Modells, das die Methoden der Informatik für die Automatisierungstechnik zugänglich macht. In Kapitel 2 wird zunächst ein gemeinsames Verständnis für die Begriffe Modell, Modellhierarchie und formale Modellbeschreibung geschaffen. Es wird zudem eine Metrik entwickelt anhand derer sich Modelle bezüglich ihres Formalisierungsgrades einordnen lassen.

Durch seine zentrale Rolle in den Anwendungsszenarien bieten sich PandIX als Grundlage für die Verdeutlichung verschiedener Sachverhalte dieser Arbeit an. Es wird daher in den Beispielen regelmäßig aufgegriffen. Einen detaillierten Einblick in PandIX sowie in die übrigen, für die Anwendungsszenarien relevanten Modelle bietet Kapitel 3. Es werden bestehende Probleme bei der Modellierung aufgezeigt und Hinweise für die Entwicklung zukünfiger Modelle gege-

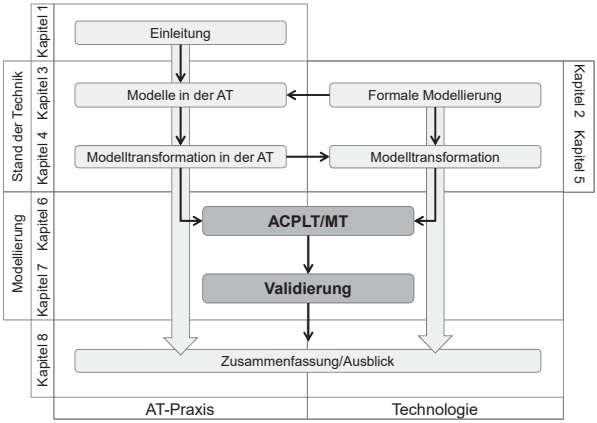


Abbildung 1.4: Struktur der Arbeit

ben. Modellzusammenhänge in der Automatisierungstechnik sowie der Stand der Technik bei der automatisierten Verarbeitung dieser Zusammenhänge werden in Kapitel 4 näher beleuchtet. Die dabei identifizierten Herausforderungen, denen sich eine Modelltransformation in der Automatisierungstechnik stellen muss dienen als Richtschnur für die Einführung des aus der Informatik stammenden Konzeptes der Tripel-Graph-Grammatiken in Kapitel 5. Kapitel 6 stellt ein Konzept für die Nutzung von Tripel-Graph-Grammatiken in der Anlagenautomatisierung vor, welches in Kapitel 7 anhand der zuvor identifizierten Szenarien und Anforderungen validiert wird. Kapitel 8 gibt eine Zusammenfassung der Arbeit sowie einen kurzen Ausblick auf potentielle neue Einsatzszenarien sowie sinnvolle Erweiterungen des vorgestellten Konzeptes.

2 Formale Modellierung

Der Fokus der vorliegenden Arbeit liegt auf dem Einsatz modellgetriebener Entwicklungsmethoden aus dem Bereich der Informatik in der Anwendungsdomäne der Ingenieurswissenschaften. Eine der größten Herausforderungen dabei stellt die unterschiedliche Herangehensweise dieser beiden Disziplinen an die Modellierung dar. Während in der Informatik die automatische Auswertbarkeit und somit die Modellierung nach klar definierten Regeln im Vordergrund steht, beschreiben Ingenieure die zu modellierenden Systeme meist mit Hilfe der beobachteten oder der erwünschten Systemeigenschaften. Typische Aussagen beim Aufeinandertreffen dieser beiden Herangehensweisen sind von Seiten der Informatiker: „Das ist nicht eindeutig modelliert“ und „es wird eine starke Typisierung benötigt“. Von den Ingenieuren kommen dabei Anforderungen wie: „Es sollen nicht alle Sonderfälle, sondern nur die grundlegende Idee modelliert werden“ oder „das Modell soll nach Bedarf später verfeinert werden“.

Disziplinübergreifende Arbeiten haben insbesondere drei Aspekte der Modellierung abzuwägen:

- deskriptiv vs. konstruktiv
- grafisch vs. textuell
- informal vs. formal

Dieses Kapitel sensibilisiert den Leser für die unterschiedlichen Modellierungsmethoden. Dazu werden zunächst die grundlegenden Begriffe eingeführt. Ausgehend vom zentralen Begriff des Modells stehen dabei insbesondere die formale Modellbeschreibung und die modellgetriebene Softwareentwicklung im Vordergrund.

Am Ende des Kapitels steht ein Bewertungsschema zur Verfügung, das dem Leser die Einordnung von Modellen hinsichtlich ihrer Eignung für die modellgetriebene Softwareentwicklung erlaubt. Zudem werden Hinweise für die Entwicklung neuer Modelle unter diesem Gesichtspunkt bereitgestellt.

2.1 Allgemeine Begriffsbestimmung

Auch wenn heute oftmals die von der Object Management Group (OMG) herausgegebenen Standards als Basis für modellgetriebenen Softwareentwicklung verwendet werden, so kommt man bei dem Thema Modelle nicht um den Modellbegriff von Stachowiak umher. In seinem Buch Allgemeine Modelltheorie [Sta73] definiert er Modelle wie folgt:

Definition 2.1 (Modell) *Modelle sind „Abbildungen [...] natürlicher oder künstlicher Originale“ (Abbildungsmerkmal), die unter Zuhilfenahme von Abstraktion (Verkürzungsmerkmal), die Attribute des Originals beleuchten, die „für bestimmte [...] Subjekte“ und einen bestimmten Zweck „innerhalb bestimmter Zeitintervalle“ von Interesse sind (Pragmatisches Merkmal).*

Viele von den 1973 von Stachowiak formulierten Ideen und Zusammenhänge finden sich heute unter anderem in den von der OMG veröffentlichten Standards so, oder in leicht veränderter Form wieder. Interessanter Grundgedanke beim Modellbegriff von Stachowiak ist die Betrachtung von Original und Modell als „Attributklassen“, also rein durch ihre Eigenschaften und Relationen beschriebene Objekte. Stachowiak unterteilt Modelle bzw. die für sie herangezogenen Attribute in unterschiedliche Stufen. Attribute, die ein Original beschreiben, ordnet er der nullten Stufe zu. Diesen nur in der realen Welt existierenden „uneigentlichen“ Attributen stellt er Eigenschaften von Individuen und Relationen zwischen Individuen als Attribute erster Stufe entgegen. Attribute zweiter Stufe beschreiben Eigenschaften von Attributen erster Stufe und Relationen zwischen Attributen erster Stufe.

Diese Idee wird ähnlich auch in den Standards der OMG aufgegriffen. Anders als bei Stachowiak beschränkt die OMG die Hierarchie jedoch auf eine Instanz- und drei Modellebenen. Modelle im Sinne von Definition 2.1 finden sich bei der OMG in der M1-Ebene. In dieser Schicht sind auch das Anlagenstrukturmodell PandIX sowie die meisten anderen Modelle aus dem Bereich der Automatisierungstechnik einzuordnen. Die Abbildung einer konkreten Anlagenstruktur mit Hilfe von PandIX wird als Laufzeitinstanz bezeichnet und ist in der Ebene M0 einzuordnen. Zur Beschreibung von M1 Modellen kommen die Modellierungssprachen aus der M2-Ebene zum Einsatz. Die OMG schlägt hierfür eine einheitliche domänenübergreifende Modellierungssprache (engl. Unified Modelling Language, UML) [UML] vor. Diese hat insbesondere in der Informatik und den Ingenieurswissenschaften mittlerweile einen hohen Durchdringungsgrad bei der Modellierung erlangt. Diese Meta-Modell-Ebene wiederum wird mit Hilfe der Modellbeschreibungssprachen aus der Meta-Meta-Modell-Ebene M3 definiert. Die OMG sieht die Meta-Object-Facility [MOF] als einziges M3-Modell vor, mit deren Hilfe UML definiert wird. Da von der OMG keine weiteren Meta-Ebenen vorgesehen sind, sind Modelle der M3-Ebene im Allgemeinen selbstbeschreibend.

Definition 2.2 (Modellierung - Modellbeschreibung - Modellbildung) *In dieser Arbeit wird der Begriff Modellierung für die Erstellung eines M1-Modells und der Begriff Modellbeschreibung für die eines M2-Modells verwendet. Bei der Verwendung des Begriffs Modellbildung wird von der konkreten Modellebene abstrahiert.*

Definition 2.3 (Selbstbeschreibung - Introspektion) *Ein Modell ist selbstbeschreibend/introspektiv, wenn für die Modellierung nur Elemente des Modells selbst zum Einsatz kommen.*

Neben den von der OMG vorgeschlagenen Kombination aus MOF und UML sind auch weitere Meta-(Meta-)Modelle denkbar. Dabei ist eine Zuordnung von Modellen zu einer der vier Ebenen nicht immer eindeutig möglich, wie die beispielhafte Abbildung der Modellhierarchien für die Modellierung von verfahrenstechnischen Anlagen mit Hilfe von PandIX in Abbildung 2.1 zeigt. Um die Vorteile einer domänenübergreifenden Modellierungssprache nutzen zu können ohne sich auf ihre Beschreibungsmittel einschränken zu müssen, können für ein Modell auch

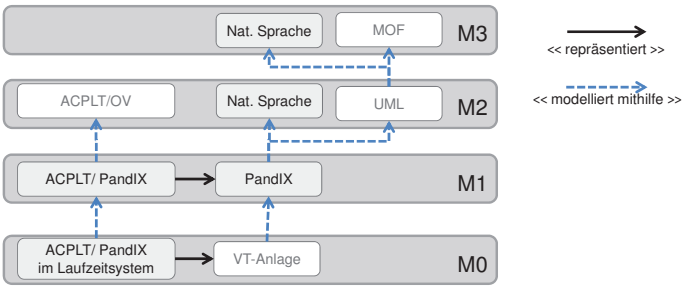


Abbildung 2.1: Einordnung von PandIX in die Modellhierarchie

mehrere Modellierungssprachen zum Einsatz kommen. So wird für die Beschreibung von Syntax und Semantik des PandIX-Modells aus Abbildung 2.1 sowohl UML als auch die natürliche Sprache eingesetzt. Auch wenn die natürliche Sprache im Allgemeinen in solchen Modellhierarchien außer Acht gelassen wird, so ist sie doch eine Meta-(Meta-)Sprache, die insbesondere in den Ingenieurswissenschaften oft zur Modellierung eingesetzt wird.

Definition 2.4 (Modellwelt) *Eine Menge aufeinander aufbauender bzw. harmonisierender Modelle bilden eine Modellwelt.*

Beispiel 2.1 *Ein Beispiel für eine recht umfangreiche Modellwelt für die Domäne der Prozessleittechnik bildet die ACPLT-Modellwelt. Sie besteht aus einem Satz vielfältiger prozessleittechnischer Modelle und ihrer Realisierung in der ACPLT-Laufzeitumgebung. Das M3-Modell und damit die Basis der ACPLT-Modellwelt bildet das Objektverwaltungssystem ACPLT/OV.*

2.2 Darstellungsformen

Modellierung kommt in den unterschiedlichsten Domänen zum Einsatz, um komplexe Zusammenhänge für einen spezifischen Anwendungsfall aufzubereiten. Die Heterogenität der dabei zu beschreibenden Modelle bedingt die Entwicklung ebenso heterogener Sprachen für die Darstellung der Modelle. Diese oft hochspezialisierten domänenspezifischen Sprachen (engl. domain-specific languages, DSL) sind für einen beschränkten Anwendungsbereich ausgelegt und in ihrer Symbolik optimal an die Domäne angepasst.

Die Hierarchie und die Architektur der Modelle macht eine Aussage darüber, WAS modelliert wird. Stachowiak [Sta73] macht bei der Formulierung des Pragmatischen Merkmals seines Modellbegriffs deutlich, dass auch das FÜR WEN, WANN und WOZU eine zentrale Rolle bei der Modellbildung spielen. In diesem Zusammenhang kommen zwei weitere Aspekte der Modellierung ins Spiel, zum einen die Form der Darstellung, zum anderen der Grad der Formalisierung. Diese beiden Aspekte sollen in den nächsten Abschnitten näher beleuchtet werden.

Definition 2.5 (Darstellungsform) *Die Darstellungsform eines Modells wird bestimmt durch die verwendeten Symbole (Diagramme, Fließtext, . . .) und die verwendete Beschreibungsmethode. Die Beschreibungsmethoden lassen sich unterteilen in deskriptive und konstruktive Beschreibung.*

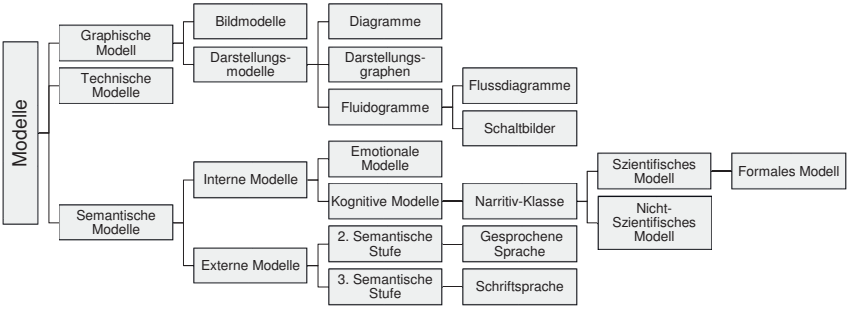


Abbildung 2.2: Ausschnitt aus den Modellkategorien nach Stachowiak [Sta73]

Auch Stachowiak geht in seiner Allgemeinen Modelltheorie auf verschiedene Darstellungsformen ein. Er kategorisiert die Modelle dabei nur nach der verwendeten Symbolik (vgl. Abbildung 2.2). Den Aspekt der Beschreibungsmethoden lässt er dabei außen vor. Durch seine Sicht auf Modelle als „Attributklassen“ schränkt er sich sogar auf die beschreibende/deskriptive Modellierung ein. Dies ist insofern nicht weiter verwunderlich, da seine Arbeit den Bereich der Soziologie fokussiert. Die eher aus den Sprachwissenschaften stammende Methode der konstruktiven Modellierung ist für die Anwendung in der Soziologie zu formal und daher wenig geeignet.

In der Informatik unterscheidet man in diesem Zusammenhang zwischen der abstrakten und der konkreten Syntax eines Modells respektive einer Sprache.

Definition 2.6 (Syntax) *Die Syntax beschreibt die erlaubten Symbole (das Alphabet) eines Modells sowie die daraus generierbaren erlaubten Modellinstanzen (Wörter). Die Abstrakte Syntax stellt die Konzepte zur Verfügung, die durch das Modell abgebildet werden sollen. Die konkrete Syntax beschreibt die Darstellungsform, die für die Erstellung von Modellinstanzen zur Verfügung steht. Es können mehrere konkrete Syntaxen zu einem Modell existieren.*

Beispiel 2.2 *Die in Abbildung 2.1 verwendete Assoziation „repräsentiert“ besagt nichts anderes, als dass hier der gleiche Aspekt mit Hilfe mehrere konkreter Syntaxen modelliert werden kann. Modellierungswerkzeuge verwenden intern ggf. eine ganz andere Darstellung der Modelleigenschaften z.B. in Form von C-Funktionen oder Arrays. Dies ist die jeweilige abstrakte Syntax.*

In diesem Abschnitt stehen die konkrete Syntax, ihre möglichen Ausprägungen und die damit einhergehenden Vor- und Nachteile im Fokus der Betrachtung.

2.2.1 Deskriptiv vs. Konstruktiv

Bei der deskriptiven Modellierung steht die Beschreibung von Objekten und Eigenschaften im Vordergrund unter einem konkreten Aspekt (z.B. Vererbungshierarchie, Interaktion zwischen Objekten, etc.) im Vordergrund. Diese, auch der UML zu Grunde liegende Modellierung findet insbesondere in der Beschreibung von (technischen) Systemen ihren Einsatz. Der deskriptiven Modellierung steht die konstruktive Modellierung entgegen, die auf einem Alphabet und Konstruktionsregeln basiert. Ein typischer Vertreter konstruktiver Modellierung sind Grammatiken. Ausgehend von einer initialen Konstruktionsregel, die das zu beschreibende Modell als Ganzes wiedergibt, kann durch wiederholte Anwendung der Konstruktionsregeln der gesamte Raum der möglichen Modellinstanzen konstruiert werden.

Zwar gilt auch für die konstruktive Modellierung, dass nur ein Ausschnitt aus der Realität modelliert wird, für konstruktive Modelle gilt aber im Allgemeinen die Annahme, dass Objekte, Eigenschaften und Strukturen, die nicht durch die Konstruktionsregeln hergeleitet werden können auch nicht Teil eines validen Modells sein können. Anders verhält es sich bei den deskriptiven Modellen, bei denen diese Weltabgeschlossenheit (engl. Closed world assumption) explizit formuliert werden muss. Der beschreibende Charakter der deskriptiven Modellierung fokussiert eher die lockere Spezifikation (engl. Loose specification), bei der nur einzelne Aspekte valider Modelle spezifiziert sind.

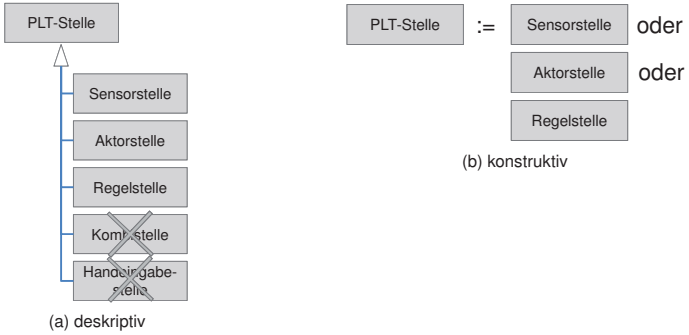


Abbildung 2.3: Modellierung einer PLT-Stelle

Beispiel 2.3 *Abbildung 2.3a [PandIX] beschreibt die Eigenschaft „ist eine PLT-Stelle“ der drei Modellelemente „Sensorstelle“, „Aktorstelle“ und „Regelstelle“ und die Eigenschaft „ist keine PLT-Stelle“ der Modellelemente „Kombi-Stelle“ und „Handeingabestelle“. Ob eine „Pumpe“ die Eigenschaft „ist eine PLT-Stelle“ oder die Eigenschaft „ist keine PLT-Stelle“ besitzt, lässt sich anhand dieser Modellierung nicht eindeutig klären. Abbildung 2.3b hingegen leitet aus dem Vorhandensein einer PLT-Stelle in der Modellinstanz ab, dass diese entweder eine „Sensorstelle“, „Aktorstelle“ oder „Regelstelle“ sein kann. Die Frage, ob eine „Pumpe“ eine PLT-Stelle ist, lässt sich eindeutig klären. Da die Konstruktionsregel für PLT-Stelle keine Möglichkeit bietet, nach „Pumpe“ aufzulösen, ist „Pumpe“ keine PLT-Stelle.*

2.2.2 Textuell vs. Graphisch

Bei der Modellierung der PLT-Stelle aus Beispiel 2.3 wurde eine graphische Repräsentation gewählt. Die wohl bekannteste grafische Modellierungssprache ist die UML. Neben Klassendiagrammen (vgl. Abbildung 2.3a) stellt die UML auch grafische Modellierungselemente für Abläufe, Zustände und vieles mehr zur Verfügung. Eine besondere Herausforderung bei der Verwendung grafischer Sprachen stellt deren flexible Interpretierbarkeit dar. Schüller und Epple [SE12] identifizieren insbesondere drei Aspekte bei der Verwendung von Grafiken zur Modellierung als kritisch, die hier nicht unerwähnt bleiben sollen:

Vollständigkeit Insbesondere manuell erstellte Grafikmodelle enthalten teilweise implizite über das Modell hinausgehende Informationen. So werden Farben, Positionierungen von Objekten zueinander und Kommentare genutzt, um Aspekte abzubilden, die das Modell nicht vorsieht. Für eine automatisierte Verarbeitung der grafischen Modelle sollten alle verwendeten informationstragenden Elemente möglichst vollständig im Modell dokumentiert sein.

Verschiedenartigkeit Das Aussehen einzelner Modellelemente kann durch die Verwendung verschiedener Stile von Implementierung zu Implementierung variieren ohne das Modell zu verletzen.

Komplexität Grafiken vereinigen oft mehrere zu modellierende Aspekte in kompakter Form. Die Betrachtung einer Modellinstanz unter einem konkreten Aspekt erfordert in diesen Fällen eine stark selektive Betrachtung der Grafik.

Bei den textuellen Modellierungssprachen ist die erweiterte Auszeichnungssprache (engl. extended markup language, XML) der am weitesten verbreitete Modellierungsansatz. Mit Hilfe von XML-Schemata lassen sich Hierarchien von XML-basierten Modellen erstellen und für die rechnergestützte Validierung nutzbar machen.

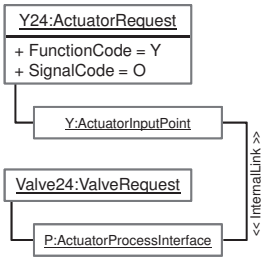
Zu einer abstrakten Syntax können parallel mehrere grafische als auch textuelle konkrete Syntaxen vorliegen. Dies kommt unter anderem bei der Bereitstellung von XML-basierten Austauschformaten für grafische Modelle zum Tragen.

Beispiel 2.4 *In Abbildung 2.4 ist ein Ventil mit einer entsprechenden PLT-Stelle grafisch als R&I-Symbol nach [DIN10628] und textuell nach [PandIX] dargestellt. In diesem Fall sind die enthaltenen Informationen nahezu identisch, da PandIX als Austauschformat für R&I-Fließbilder definiert wurde. In der textuellen Repräsentation ist eine zusätzliche Information über den sicheren Zustand (SafeState) enthalten, die nicht in der grafischen Repräsentation abgebildet werden kann.*

Die von Schüller und Epple erwähnten Aspekte grafischer Modellierung stellen insbesondere bei der Umwandlung einer grafischen Modellinstanz in ein textuelles Modell und umgekehrt eine besondere Hürde dar. Allgemein gilt jedoch, je formaler (im Sinne von eindeutig interpretierbar) die beiden Modelle beschrieben sind, desto effektiver kann ein Informationsaustausch zwischen den Instanzen der Modelle erfolgen. Im Folgenden soll daher auf die Bewertung von Modellen anhand ihres Formalisierungsgrades eingegangen werden.

```
<InternalElement Name="Y24">                % PLT-Stelle
   <Attribute Name="FunctionCode">
      <Value>Y</Value>
   </Attribute>
   <Attribute Name="SignalCode">
      <Value>O</Value>
   </Attribute>
   <RoleRequirements
    RefBaseRoleClassPath="PCE@ActuatorRequest"/>
</InternalElement>
<InternalElement Name="Valve24">            % Ventil
   <Attribute Name="Position">
      <Value>S01P2|1.30</Value>
   </Attribute>
   <Attribute Name="SafeState">
      <Value>close</Value>
   </Attribute>
   <RoleRequirements
    RefBaseRoleClassPath="PPE@ValveRequest"/>
</InternalElement>
<InternalLink Name="Valve24_Y24"            % Verbindung
              RefPartnerSideA="Valve24:Y"
              RefPartnerSideB="Y24:P"/>
```

(a) Textuell nach PandIX



(b) UML-Darstellung

(c) Grafisch nach DIN 10628

Abbildung 2.4: Modellierung eines Ventils und zugehöriger PLT-Stelle

2.3 Formalisierungsgrad

In seiner Kategorisierung der Darstellungsformen (vgl. Abbildung 2.2) ordnet Stachowiak die formalen Modelle als eine Unterkategorie den Semantischen Modellen zu. Doch schon durch

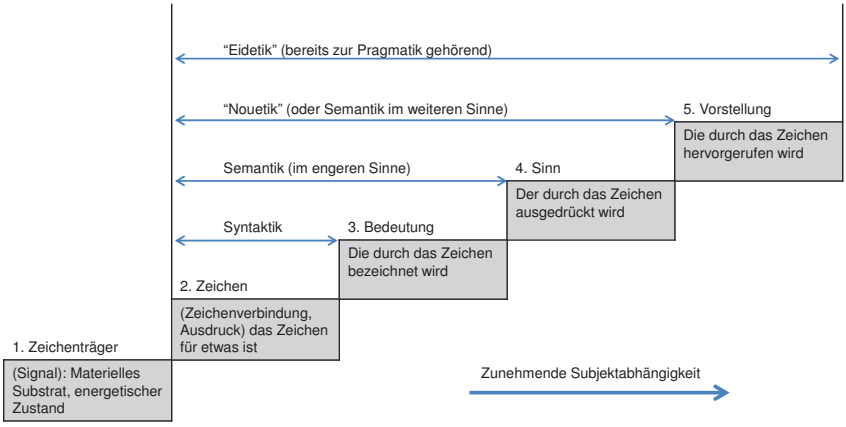


Abbildung 2.5: Die Kategorien der Bedeutung und des Sinnes nach G. Frege [Fre92] vereinfachte Form der Darstellung von Stachowiak [Sta73]

seine an Frege [Fre92] angelehnte „Kategorien von Sinn und Bedeutung“ (vgl. Abbildung 2.5) deutet er an, dass der Grad der Formalisierung (als Umkehr des Grades der Subjektivität) orthogonal zur Darstellungsform ist. Auch die in den vorangegangenen Abschnitten aufgeführten Beispiele legen nahe, dass die Eigenschaft „Formales Modell“ nicht in Korrelation mit der Wahl der Symbolik steht.

Beispiel 2.5 *Die in Abbildung 2.3b aufgeführte Modellierung einer PLT-Stelle ist zwar grafisch formuliert, aber sehr formal beschrieben. Auch die in Abbildung 2.4 aufgeführten Modelle sind unabhängig von der Darstellungsart als formale Modelle einzuordnen.*

Woran lässt sich nun aber messen, ob ein Modell formal beschrieben ist oder nicht? Um dieser Frage auf den Grund zu gehen, soll zunächst eine informelle Definition des Begriffs erfolgen, der im Laufe des Abschnitts verfeinert und formalisiert wird:

Definition 2.7 (Formale Modellbeschreibung) *Angelehnt an die „Kategorien von Sinn und Bedeutung“ [Sta73] ist eine Sprache umso formaler, je weniger subjektabhängig sie interpretiert werden kann, d.h. je eindeutiger ihre Semantik beschrieben ist.*

Definition 2.8 (Semantik) *Die Semantik ordnet einzelnen Symbolen oder Symbolgruppen eine Bedeutung zu. Je nach syntaktischem Aufbau bzw. Kontext kann der gleichen Gruppe von Symbolen auch unterschiedliche Semantik zugeordnet sein.*

Beispiel 2.6 *Die Semantik beschreibt, dass die Symbole aus Abbildung 2.4c ein Ventil in der Anlage und seine Anknüpfung an die Informationswelt des Leitsystems darstellen.*

Der Verein Deutscher Ingenieure (VDI) verwendet in seiner Richtlinie zum Thema „Einordnung und Bewertung von Beschreibungsmitteln aus der Automatisierungstechnik“ [VDI3681] eine ähnlich Definition von formal:

- *Formales Beschreibungsmittel*
  Besitzt eine mathematische Basis und eine präzise und vollständige Syntaxdefinition sowie eine eindeutige semantische Interpretation.
- *Semiformales Beschreibungsmittel*
  Besitzt eine definierte vollständige Syntax sowie eine eindeutige semantische Interpretation aber keine mathematische Basis.
- *Informales Beschreibungsmittel*
  Besitzt eine Syntax, die nicht grundsätzlich vollständig definiert ist, sowie eine Semantik, die nicht eindeutig sein muss.

In der Informatik erfolgt die Einordnung nach etwas anderen Maßstäben. So wird UML allgemein als semi-formal eingestuft, obwohl nicht in allen Details eine eindeutige semantische Interpretation vorliegt. Die OMG geht daher einen anderen Weg. Statt zu definieren, was „Formal“ bedeutet, werden in [UML] fünf Eigenschaften aufgeführt, die durch eine möglichst formale Beschreibung forciert werden sollen:

Korrektheit Bei dieser Eigenschaft steht die Validierbarkeit des zu beschreibenden Modells im Vordergrund. Voraussetzung dafür ist, dass das Modell so aufgebaut ist, dass nur korrekte Modellinstanzen aus den Regeln herleitbar sind.

Präzision Doppeldeutigkeiten in der Beschreibung der Syntax oder der Semantik sind zu vermeiden. Die OMG erlaubt dabei jedoch explizit beschriebene alternative semantische Auslegungen.

Konsistenz Das Modell muss so formuliert sein, dass es keine Widersprüchlichkeiten enthält.

Prägnanz Es sind nur die Eigenschaften des Modells zu modellieren, die wirklich von Interesse sind. Überflüssige Ergänzungen sind wegzulassen.

Verständlichkeit Das Modell muss so gestaltet sein, dass es leicht lesbar und leicht zu verstehen ist.

Die Eigenschaften Prägnanz und Verständlichkeit sind wichtige Bewertungskriterien für die Einordnung von Modellen nach ihrer Anwendbarkeit. Sie sollen hier aber nicht weiter als Eigenschaften formaler Modelle verwendet werden, da sie zum einen subjektiver Einschätzung unterliegen und im allgemeinen Verständnis von „formal“ nicht zum Tragen kommen, wie das folgende Beispiel zeigt:

Beispiel 2.7 *Ein strikt formales Modell wird nicht dadurch weniger formal, dass ihm redundante Informationen hinzugefügt werden. Ein komplexes mathematisches Modell ist für den Experten trivial, während es für einen Laien völlig unverständlich ist. Es wird dadurch nicht formaler oder weniger formal für die einzelnen Personengruppen.*

Im letzten Abschnitt dieses Kapitels werden Modellierungsverfahren vorgestellt, die die Erstellung formaler Modelle unterstützen.

2.4 Formale Modellierung

Im vorangegangenen Abschnitt sind einige Bewertungskriterien für den Grad der Formalisierung aufgestellt worden. Um dem Leser geeignete Mittel für die Erstellung formaler Engineeringmodelle an die Hand zu geben, werden in diesem Abschnitt konkrete Herangehensweisen für die formale Modellierung vorgestellt. Insbesondere sollen die Ideen der formalen Sprachen und formaler Grammatiken im Vordergrund stehen. Zuvor wird jedoch die für die deskriptive Modellierung maßgebliche Sprache UML und ihre Grundlagen skizziert.

2.4.1 Deskriptive, grafische Modellierung

UML ist die bekannteste deskriptive Modellierungssprache, die auch außerhalb der Softwareentwicklung weite Verbreitung findet. Dies liegt zum einen daran, dass UML intuitiv verständlich und auf der anderen Seite mächtig genug ist, um komplexe Zusammenhänge zu beschreiben. Zudem wird durch den objektorientierten Grundgedanken der UML eine „natürliche“ und wiederum intuitive Strukturierung des Modells ermöglicht. Die von der OMG als Standard heraus-

gegebene aktuelle Modellbeschreibung von UML - UML 2.4.1 [UML; UML2], ist mit Hilfe der M3-Sprache MOF spezifiziert.

Als diagrammorientierte Modellierungssprache stellt UML eine Reihe von Sprachelemente für unterschiedliche Modellierungsanforderungen zur Verfügung. An dieser Stelle wird jedoch nur auf die Elemente eingegangen, die für das Verständnis der weiteren Kapitel von Interesse sind (Abbildung 2.6). Klassen von Objekten mit gleichen Eigenschaften werden in UML durch

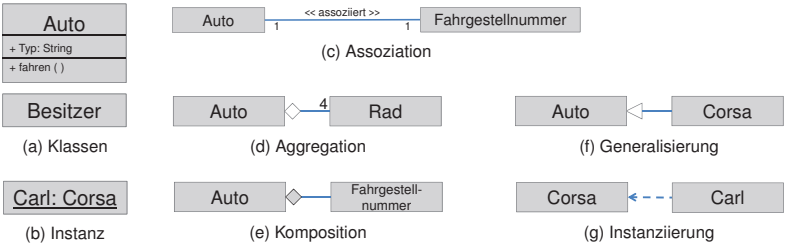


Abbildung 2.6: UML Sprachelemente

Rechtecke abgebildet (Abbildung 2.6a). Besitzt eine Klasse Attribute oder Operationen, so werden diese separiert unter dem Klassennamen angegeben. Die Kurzschreibweise einer Klasse beinhaltet nur den Klassennamen. Instanzen einer Klasse werden durch Angabe des Instanznamens und des Klassennamen getrennt durch ein „:“ beschrieben (Abbildung 2.6b). Instanzen können konkrete Ausprägungen der Attribute besitzen, diese werden äquivalent zur Darstellung in Klassen angegeben. Zwischen Klassen können Assoziationen gekennzeichnet werden (Abbildung 2.6c). Assoziationen können optional eine Richtung, einen Namen sowie Angaben zur Multiplizität besitzen. Ersteres dient der näheren Beschreibung der Assoziation, letzteres gibt an wie viele Instanzen der beiden Klassen jeweils miteinander durch die Assoziation verlinkt werden. Die Generalisierung ist eine spezielle «ist ein»-Assoziation (Abbildung 2.6f), die den Zusammenhang zwischen einer Klasse und ihrer Verallgemeinerung darstellt. Die «besteht aus»-Assoziation wird auch als Aggregation bezeichnet und durch eine leere Raute gekennzeichnet (Abbildung 2.6d). Die Komposition oder auch starke Aggregation ist eine spezielle Aggregation, bei der die Einzelteile ohne das Ganze nicht existieren. Zudem ist die Anzahl der Elternteile auf eins beschränkt. Die Komposition wird durch eine ausgefüllte Raute gekennzeichnet (Abbildung 2.6e). Die Instanziierung bzw. «Instanz von»-Assoziation (Abbildung 2.6g), erbt UML von MOF. Diese Assoziation kann unter anderem zwischen Klassen und von „Klasse“ abgeleiteten Modellelementen wie zum Beispiel Modellen und Instanzen verwendet werden [MOF].

2.4.2 Konstruktive, textuelle Modellierung

Der Bereich der konstruktiven Modellierung wurde maßgeblich durch die Arbeiten von Noam Chomsky [Cho65] geprägt. Sein Ziel war es, eine mathematisch präzise Modellierung natürlicher Sprache zu formulieren. Dies ist zwar bis heute nicht gelungen, aber seine Ideen zu formalen Grammatiken und formalen Sprachen bilden die Grundlage zur Modellierung vieler

heutiger künstlicher, maschinenauswertbarer Sprachen. Der Exkurs in die Welt der formalen Grammatiken und Sprachen scheint zunächst etwas weit ab vom Thema der formalen Modellierung. Doch wenn man genauer hinschaut, so finden sich eine Reihe von Modellierungen, die auf eben diesen formalen Grammatiken basieren. Auch die, der Arbeit zugrunde liegenden Tripel-Graph-Grammatiken (Kapitel 5.1) basieren auf den formale Sprachen nach Chomsky.

Die Grundidee von Chomsky ist dieselbe, wie sie bei der Beschreibung natürlicher Sprachen auch zum Einsatz kommt. Eine Grammatik, die Regeln bereitstellt, nach denen sich syntaktisch korrekte Zeichenfolgen der Sprache bilden lassen. Eine gute und allgemeinverständliche Einführung in formale Grammatiken und formale Sprachen nach Chomsky bietet das „Skript zur Vorlesung Formale Sprachen und Berechenbarkeit“ von Prof. Dr. Georg Schnitger [Sch11]. Das besondere an Schnitgers Skript ist, dass er XML-basierte Sprachen bzw. die sie erzeugenden XML-Grammatiken in den Kontext der formalen Sprachen einordnet. Die Wichtigkeit von XML in der Automatisierungstechnik, sowie der bereits erwähnte Zusammenhang mit Tripple Graph Grammatiken, bedingen eine kurze Zusammenfassung der im Skript eingeführten Ideen und Definitionen.

Definition 2.9 (Alphabet, Worte, Sprachen [Sch11])

(a) Ein Alphabet Σ ist eine endliche, nicht-leere Menge. Die Elemente von Σ werden Buchstaben genannt.

(b) $\Sigma^n = \{(a_1, \ldots, a_n) |\ a_i \in \Sigma\}$ ist die Menge aller Worte der Länge n über Σ. Wir werden im folgenden $a_1 \cdots a_n$ statt $(a_1, \ldots, a_n)$ schreiben. [. . .]

(d) $\Sigma^* = \bigcup_{n=0}^{\infty} \Sigma^n$ ist die Menge aller Worte über dem Alphabet Σ. [. . .]

(f) Für $w \in \Sigma^*$ bezeichnet $|w|$ die Länge von w, also die Anzahl der Buchstaben von w.

(g) Eine (formale) Sprache L (über Σ) ist eine Teilmenge von Σ^*.

Eine formale Sprache ist somit eine Menge Zeichenketten endlicher Länge, die sich aus den Buchstaben eines Alphabets, den sogenannten Terminalen erzeugen lassen (Wörter). Im Allgemeinen sind allerdings nur Sprachen von Interesse, deren Wörter bestimmten Regeln folgen. Formale Sprachen bedienen sich bei der Beschreibung dieser Regeln sogenannter Termersetzungssysteme (oder Semi-Thue-Systeme).

Definition 2.10 (Termersetzungssysteme (TES)) *Ein Termersetzungssystem hat die folgenden Komponenten:*

- *ein endliches Alphabet Σ,*
- *eine endliche Menge P von Produktionen mit $P \subseteq \Sigma^* \times \Sigma^*$*

Produktionen werden auch Konstruktionsregeln genannt. Sie sind Tupel (LHS, RHS) aus Zeichenketten. Eine Produktion kann angewendet werden, wenn in einem Wort eine der beiden Zeichenketten LHS oder RHS vorkommt. Diese kann dann durch die andere Seite der Produktion ersetzt werden, um ein neues gültiges Wort zu erzeugen.

Einen Spezialfall der Termersetzungssysteme bilden Grammatiken. Bei ihnen erfolgt die Ersetzung immer von LHS nach RHS (monotone TES). Zudem besitzen Grammatiken neben dem Alphabet (von Terminalsymbolen) eine Menge von Variablen (Nichtterminale). Nichtterminale dienen als Platzhalter komplexerer Terme. Als gültige Worte der durch die Grammatik erzeugten Sprache gelten nur solche, die keine Nichtterminale mehr enthalten. Produktionen von Grammatiken haben auf der linken Seite immer mindestens ein Nichtterminal. Diese Nichtterminale werden durch die Produktion ersetzt.

Eine formale Grammatik ist definiert als:

Definition 2.11 (Grammatik [Sch11]) *Eine Grammatik G hat die folgenden Komponenten:*

- *ein endliches Alphabet* Σ*,*
- *eine endliche Menge V von [...] Nichtterminalen [...] mit* $\Sigma \cap V = \emptyset$*,*
- *das Startsymbol* $S \in V$ *und*
- *eine endliche Menge P von Produktionen*

Definition 2.12 (Produktion) *Seien* LHS *(engl. left-hand side) und* RHS *(engl. right-hand side) zwei Zeichenketten. Produktionen sind Tupel* (LHS, RHS) *für die gilt:*

$$\begin{aligned} RHS &\in (V \cup \Sigma)^* \textit{ und} \\ LHS &= xyz \textit{ mit } x, z \in (V \cup \Sigma)^* \textit{ und } y \in V \end{aligned}$$

Ausgehend von einer Zeichenkette aus Terminalen und Nichtterminalen können durch Anwendung der Produktionen neue Zeichenketten abgeleitet werden.

Definition 2.13 (Ableitung) *Sei* (LHS, RHS) *eine Produktion von G und* $w_1, w_2 \in (V \cup \Sigma)^*$ *zwei Zeichenketten. Dann gilt* $w_1 \to w_2$ *genau dann, wenn* w_2 *durch die einmalige Ersetzung von* LHS *durch* RHS *in* w_1 *erzeugt wird.*[1]

Gibt es eine Sequenz von Zeichenketten $w_1, w_2, \ldots, w_n$ *mit* $n \in \mathbb{N}$*, so dass gilt*

$$w_1 \to w_2 \to \cdots \to w_n,$$

dann ist w_n *eine Ableitung von* w_1 *(kurz* $w_1 \overset{*}{\to} w_n$*).*

Definition 2.14 (Erzeugte Sprache) *Sei* $G = (\Sigma, V, S, P)$ *eine Grammatik. Die Menge aller vom Startsymbol S ableitbaren Worte*

$$\{w \in \Sigma^* | \; S \overset{*}{\to} w\}$$

ist die durch G erzeugte Sprache L(G).

Im folgenden Abschnitt soll beispielhaft anhand von XML gezeigt werden, wie die Definition von Modellierungssprachen mit Hilfe formaler Sprachen aussieht.

[1] Im Weiteren wird dem Skript folgend diese Schreibweise auch für die Produktionen selbst genutzt ($LHS \to RHS$).

2.4.2.1 XML-Grammatiken

Die erweiterbare Auszeichnungssprache (engl. eXtensible Markup Language, XML) ist die Basis einer ganzen Reihe von Modellierungssprachen. XML-basierte Sprachen werden häufig dazu verwendet, grafische Modelle textuell abzubilden. Insbesondere Modelle, die in UML beschrieben sind, besitzen oftmals auch eine textuelle Repräsentation in Form eines XML-Dokuments. Auch in der Automatisierungstechnik sind verschiedene XML-basierte Sprachen als Modellaustauschsprachen im Einsatz.

Die so beschriebenen Sprachen bestehen aus geschachtelten Elementen, wie sie aus HTML oder PandIX [PandIX] bekannt sind:

```
% Element mit Name "InternalElement und Attribut mit Name "Name"
<InternalElement Name="Y24">
  <Attribute Name="FunctionCode">        % Element mit Name "Attribute"
    <Value>Y</Value>                     % Element mit Name "Value" ohne Attribut
  </Attribute>
  <Attribute Name="SignalCode">
    <Value>O</Value>
  </Attribute>
  <RoleRequirements RefBaseRoleClassPath="PCE@ActuatorRequest"/>
</InternalElement>
```

Aufbauend auf der Basisdefinition für XML bilden die XML-basierten Sprachen durch schrittweise Verfeinerung der Syntax eine Hierarchie von Sprachen. Während die Basisgrammatik allgemein beschreibt, dass Elemente der Sprache aus eine Starttag (ggf. mit Attributen), Inhalt und einem dem Starttag zugehörigen Endtag bestehen, schränken die darauf aufbauenden Sprachen die erlaubten Tags und Attribute mit Hilfe von XML-Grammatiken entsprechend ihren Anforderungen ein. Für die Beschreibung von XML-Grammatiken gibt es verschiedene Möglichkeiten. Abbildung 2.7 zeigt zum Beispiel einen Ausschnitt einer Grammatik für PandIX, der dem Ausschnitt aus der XML-Schema-Definition aus Abbildung 2.8 entspricht.

```
$\Sigma$ = (A..Z, a..z, 0..9, =, :, ")          % Sequenz aus $\Sigma$ durch
                                                %          '' gekennzeichnet
V = (CAEXdoc, IEType, AType,  RRType, RPType, Name, Value, String, AnyType)
S =  CAEXdoc

P = { CAEXdoc ::= {IEType}                      % EBNF-Notation: beliebilg viele
                                                % EBNF-Notation: optional RRType
      IEType  ::= '<InternalElement' Name '>' {AType} [RRType] '</InternalElement>'
      AType   ::= '<Attribute ' Name '>' {Value} '</Attribute>'
      RRType  ::= '<RoleRequirements ' [RPType] '/>'
      RPType  ::= 'RefBaseRoleClassPath = "' String '"'

      Name    ::= 'Name = "' String '"'          % String definiert in XML
      Value   ::= 'Value = "' AnyType '"'        % AnyType definiert in XML
    }
```

Abbildung 2.7: Grammatik für PandIX

2.4.3 Deskriptive, textuelle Modellierung

Im Allgemeinen sind für die Beschreibung XML-basierter Sprachen jedoch deskriptive Methoden im Einsatz. Zudem können durch die Hierarchie der XML-basierten Sprachen Produktionen aus der allgemeineren Modellbeschreibung in die spezielle übernommen werden. Eine der gebräuchlichsten Formen sind XML-Schema-Definitionen (kurz XSD). Abbildung 2.8 zeigt eine stark verkürzte Variante der XML-Schema-Definition für CAEX, die der Grammatik aus Abbildung 2.7 entspricht.

```
...
<xs:complexType name="AttributeType">
  <xs:attribute name="Name"  type="xs:string"  use="required">
  <xs:element   name="Value" type="xs:anyType" minOccurs="0"/>
</xs:complexType>

<xs:complexType name="InternalElementType">
  <xs:attribute name="Name"      type="xs:string" use="required">
  <xs:element   name="Attribute" type="AttributeType" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="RoleRequirements" minOccurs="0">
    <xs:attribute name="RefBaseRoleClassPath" type="xs:string" use="optional"/>
  </xs:element>
</xs:complexType>

<xs:element name="InternalElement" type="InternalElementType" minOccurs="0"
            maxOccurs="unbounded"/>
...
```

Abbildung 2.8: XSD für PandIX

2.4.4 Konstruktive, grafische Modellierung

In den ersten beiden Abschnitten wurde zum einen die deskriptive, graphische Sprache UML und zum anderen die konstruktive textuelle Herangehensweise an Sprachen über Grammatiken vorgestellt. Während die grafischen Sprachen meist eingänglicher sind, besitzt die konstruktive Herangehensweise den Vorteil einer impliziten Weltabgeschlossenheit (vgl. Abschnitt 2.2.1). In diesem Abschnitt sollen nun die Vorteile beider in einer konstruktiven, grafischen Modellierung, den sogenannten Graph-Grammatiken zusammengeführt werden. Zunächst werden jedoch die Begriffe Graph und Teilgraph eingeführt.

Definition 2.15 (Graph) *Ein Graph G ist ein Quadrupel $(V, E, targets, label)$ von Knoten V (engl. vertex), Kanten E (engl. edges) mit $E \subset V \times V$ und den darauf definierten Funktionen targets und label über den Labelalphabeten Ω_E und $\Omega_V \cup \Omega_N$. Die durch eine Kante $e \in E$ miteinander verbundenen Knoten $v_1, v_2 \in V$ können mit Hilfe der Funktion $targets$ ermittelt werden:*

$$\forall e \in E. \quad targets(e) = \quad (v_1, v_2)$$

Die Funktion label liefert zu jedem Knoten und jeder Kante die Knoten- respektive Kantenbeschriftung

$$\begin{aligned} \forall e \in E. \quad label(e) = \quad & \omega e \in \Omega_E \\ \forall v \in V. \quad label(v) = \quad & \omega v \in \Omega_V \cup \Omega_N \end{aligned}$$

Abkürzend gelte außerdem die folgende Schreibweise für Knoten und Kanten in G

$$\begin{aligned} \forall e \in E. \quad & e \in G \\ \forall v \in V. \quad & v \in G \end{aligned}$$

Definition 2.16 (Teilgraph) *Sei G = (V, E, targets, label) ein Graph. Eine Menge von Knoten V', eine Menge von Kanten E' und die Funktionen targets' und label' mit*

$$\begin{aligned} targets' = \quad & targets \quad | E' \\ label' = \quad & label \quad | V' \cup E' \end{aligned}$$

bilden einen Teilgraph G' = (V', E', targets', label') von G ($G \subseteq G'$), wenn folgendes gilt:

- *V' ist eine Teilmenge von V*
- *E' ist eine Teilmenge von E*

Für jede Kante e aus E' gilt, dass auch die beiden Knoten, die sie verbindet in V' sind.

Definition 2.17 (Graph-Grammatik) *Eine Graph-Grammatik GG = ($\Omega_E, \Omega_V, \Omega_N, S, GP$) hat die folgenden Komponenten:*

- *ein endliches Alphabet Ω_E terminaler Kantenbeschriftungen*
- *ein endliches Alphabet Ω_V terminaler Knotenbeschriftungen*
- *eine endliche Menge Knotenbeschriftungen Ω_N (Nichtterminalen) mit $\Omega_V \cap \Omega_N = \emptyset$,*
- *das Startsymbol $S \in \Omega_N$ und*
- *eine endliche Menge GP von graphbasierten Produktionen*

Knoten, die mit Elementen aus Ω_V beschriftet sind, heißen Terminal-Knoten; solche, die mit Elementen aus Ω_N beschriftet sind Nichtterminal-Knoten.

Definition 2.18 (Kanten-/Knotenbeschriftung) *verschoben nach Definition 2.15*

Definition 2.19 (Graphbasierte Produktion) *Sei $GG = (\Omega_E, \Omega_V, \Omega_N, S, GP)$ eine Graph-Grammatik und $LHS_G = (V_L, E_L, targets, label)$, $RHS_G = (V_R, E_R, targets, label)$ zwei Graphen mit*

$$\begin{aligned} \forall v \in V_L \cup V_R: \quad & label(v) \quad \in \Omega_V \cup \Omega_N \\ \forall e \in E_L \cup E_R: \quad & label(e) \quad \in \Omega_E \end{aligned}$$

Graphbasierte Produktionen sind Tupel (LHS_G, RHS_G). Für (LHS_G, RHS_G) kommt als alternative Schreibweise auch $LHS_G ::= RHS_G$ zum Einsatz.

Beispiel 2.8 *Abbildung 2.3b zeigt eine graphbasierte Produktion basierend auf dem Alphabet* Ω_V *= „Sensorstelle“, „Aktorstelle“, „Regelstelle“,* $\Omega_E = \emptyset$, Ω_N *= „PLT-Stelle“.*

Bei der Anwendung einer graphbasierten Produktion wird ähnlich wie bei einer Produktion zunächst ein Vorkommen von LHS_G im zu transformierenden Graph (Match) gesucht. Ist ein Match gefunden, wird der gemeinsame Teilgraph (auch Klebegraph) K von LHS_G und RHS_G bestimmt. Die Anwendung der graphbasierten Produktion erfolgt dann durch Löschen all jener Knoten und Kanten des Matches, die zwar in LHS_G aber nicht in RHS_G enthalten sind ($LHS_G - K$). Anschließend werden alle Knoten und Kanten, die nur in RHS_G nicht aber in LHS_G enthalten sind ($RHS_G - K$) hinzugefügt. Eine Anwendung der graphbasierten Produktion darf jedoch nur durchgeführt werden, wenn sowohl die Kontakt- als auch die Identifikationsbedingung erfüllt sind. Erstere besagt, dass beim Anwenden der Regel keine hängenden Kanten entstehen dürfen, also Kanten deren Quelle oder Ziel gelöscht wurden. Die Identifikationsbedingung schränkt die Wahl des Matches ein. Ein Match darf einen Knoten in G prinzipiell auch mehreren Knoten von LHS_G zuordnen. Dabei muss allerdings gewährleistet sein, dass dies nicht LHS_G-Knoten betrifft, von denen einer in K und der andere in $LHS_G - K$ liegt, da ansonsten der zugeordnete Knoten in G sowohl gelöscht als auch nicht gelöscht werden müsste.

2.5 Fazit

Während Modelle noch vor wenigen Jahren insbesondere für die Kommunikation zwischen Menschen entwickelt wurden, bedarf die fortschreitende maschinelle Auswertung Anpassungen an den gewählten Modellierungsmethoden. Wie in den vergangenen Abschnitten deutlich wurde, ist dabei neben der Darstellungsform auch der gewählte Grad der Formalisierung ausschlaggebend. Ersteres sollte so gestaltet sein, dass es dem natürlichen Sprachgebrauch der Domänenexperten entgegenkommt. Die Analyse der verschiedenen Darstellungsformen hat gezeigt, dass eine generelle Festlegung auf eine optimale Form nicht möglich ist. Hier muss fallspezifisch zwischen grafisch/textuell und deskriptiv/konstruktiv entschieden werden. Für eine geplante maschinelle Auswertung sollte, unabhängig von der Wahl der Darstellungsform, auf ein möglichst formales Modell gesetzt werden.

Zwar besitzen konstruktive Modelle den Vorteil der Weltabgeschlossenheit, einem wichtigen Kriterium bei der Bewertung des Formalisierungsgrades, aber auch ein deskriptives Modell kann durch explizit formulierte Abgeschlossenheit dieses Kriterium erfüllen. So werden deskriptive Modelle in Form von XML-Schema-Definitionen als vollständig angesehen. Nicht im Schema formulierte Zusammenhänge werden damit auch als nicht valide angesehen. Da in dieser Arbeit die maschinelle Auswertung von Modellen eine zentrale Rolle spielt, soll im nächsten Kapitel der Formalisierungsgrad vorhandener Modelle aus der Anwendungsdomäne der Automatisierungstechnik im Fokus stehen.

3 Modelle in der Automatisierungstechnik

Alleine in der deutschen Gesellschaft für Mess- und Automatisierungstechnik (GMA) des VDI beschäftigen sich Experten in 75 Fachausschüssen damit, ein gemeinsames Verständnis für Fragen der Automatisierungstechnik zu erarbeiten. Im Jahr 2011 resultierte dies in über 50 VDI/VDE-Richtlinien [New12]. Auch die NAMUR, das DIN und der DKE beschäftigen sich mit Modellen für die deutschsprachige Automatisierungsbranche. International sind unter anderem Normungsgremien wie ISO, IEC und ANSI damit beschäftigt, die Modellvorstellungen in weltweit standardisierter Form bereitzustellen. Diese enorme Anzahl an Gremien und daraus resultierenden Normen, Standards und Richtlinien verdeutlicht, welchen Stellenwert Modelle in der Automatisierungstechnik einnehmen.

Die Gleichsetzung von normativen Schriften (Normen, Standards und Richtlinien) mit Modellen ist angelehnt an die Definitionen aus Kapitel 2.1. So beschäftigen sich alle normativen Schriften damit, einen dem Titel der Schrift entsprechenden Teil der Realität zu beschreiben (Abbildungsmerkmal). Es wird dabei jeweils nur auf die Eigenschaften des Originals eingegangen, die für die Schrift von Interesse sind (Verkürzungsmerkmal). Zudem sind normative Schriften nicht *per se* für alle Zeit gültig. Vielmehr muss eine regelmäßige Überprüfung stattfinden, ob die Schrift dem Original noch gerecht wird. Neben den normativen Schriften finden sich insbesondere die Modelle aus dem wissenschaftlichen Bereich auch als Zeitschriften-Artikel [ERD11], Konferenzbeiträge [Yu+12] und Bücher [Mey02; Sch10; Qui11] oder anderen, nicht-normativen Veröffentlichungen wieder. Auch hier wird der geneigte Leser im Allgemeinen die drei Merkmale von Modellen wiederfinden können.

Im Folgenden soll beispielhaft gezeigt werden, welche Anforderungen die Automatisierungstechnik an Modelle stellt und welche Probleme bei der aktuell gewählten Beschreibung auftreten. Mit den hier beispielhaft aufgeführten Modellen ist die Modellwelt der Automatisierungstechnik bei weitem nicht vollständig repräsentiert, die typischen Probleme lassen sich aber gut erkennen.

3.1 Stand der Technik

Einen umfassenden und detaillierten Überblick über Anwendungsmodelle, also M1-Modelle, in der Automatisierungstechnik bietet Meyer in [Mey02]. Meyer konzentriert sich dabei auf die Modelle, deren konkrete Maßnahmen unmittelbare Auswirkungen auf einen Prozess haben. Dies schließt alle reinen Planungsmodelle aus. Ergänzend zu der Arbeit von Meyer empfiehlt sich für einen umfassenden Einblick die VDI/VDE 3681 - „Einordnung und Bewertung von Beschreibungsmitteln aus der Automatisierungstechnik“ [VDI3681], die eine Auflistung gängiger

Beschreibungsmodelle (M2- bzw. M3-Modelle) der Automatisierungstechnik liefert und eine Bewertung nach Kriterien wie „Formalisierung“, „Determinismus“ und „Darstellungsart“ gibt.

Wie einleitend erwähnt, soll an dieser Stelle keine vollständige Analyse der Modellwelt der Automatisierungstechnik durchgeführt werden, sondern beispielhaft die Modelle der in Abschnitt 1.2 identifizierten Anwendungsszenarien und ihrer potentiellen Realisierung in der ACPLT-Modellwelt untersucht werden.

Bei der Aufarbeitung von Planungsdaten zur Weiterverarbeitung im Laufzeitsystem (Szenario S0) sind folgende Modelle von besonderem Interesse:

R&I-Fließbild Das Rohrleitungs- und Instrumentenfließbild [DIN10628; DIN19227] dient zur Darstellung der Instrumentierung und Verrohrung einer verfahrenstechnischen Anlage. Dabei steht die funktionale Struktur mit Flusswegen, Reaktionsorten, Stell- und Messgeräten im Vordergrund. Rückschlüsse auf die Lage der Elemente im Raum lassen sich aus dem Modell nicht ableiten.

CAEX Das XML-basierte Austauschformat CAEX [IEC62424] ist zum Austausch hierarchischer Strukturen konzipiert worden. Im Laufe der Zeit haben sich verschiedene Spezialisierungen entwickelt, um konkrete Strukturen besser beschreiben zu können.

PandIX Viele Informationen für die Basisautomatisierung verfahrenstechnischer Anlagen lassen sich direkt aus dem R&I-Fließbild ableiten. Um diesen Informationsgewinn zu unterstützen, wurde das Austauschformat PandIX [PandIX] entwickelt. Basierend auf CAEX können damit die funktionalen Zusammenhänge, die in einem R&I-Fließbild beschrieben sind, in die Implementierung übernommen werden.

ACPLT/PandIX ACPLT/PandIX [SE13] ist eine Realisierung von PandIX auf Basis der Objektverwaltung ACPLT/OV. Sie ermöglicht es, die Struktur einer Anlage im Laufzeitsystem als erkundbare Objekte und Assoziationen zwischen diesen Objekten bereitzustellen.

AutomationML Basierend auf CAEX als strukturbeschreibendes Kernstück ermöglicht AutomationML [AML1; AML2; AML3; AML4] die Einbettung domänenspezifischer Modelle und Verlinkungen zwischen den Modellinstanzen. Dadurch lassen sich verschiedene Facetten einer Anlage wie die Struktur (PandIX), die Geometrie und Kinematik (COLADA) und das Verhalten (PLCopen XML) in einem Modell zusammenfassen.

Neben den für S0 aufgeführten Modellen sind für die Analyse von Flusswegen in verfahrenstechnischen Anlagen (Szenario S1) insbesondere Modelle von Interesse, die sich mit der Modellierung der Anlagenstruktur und dem Anlagenzustand befassen. Zudem benötigt die Präsentation der Analyseergebnisse entsprechende Darstellungsmodelle. Diese Arbeit fokussiert auf die am Lehrstuhl für Prozessleittechnik der RWTH Aachen University entwickelten Modelle aus diesen Bereichen, da diese auch in der Referenzimplementierung zum Einsatz kommen:

ACPLT/FlowPath Das Flusswegmodell ACPLT/FlowPath [Qui11] ermöglicht die Überwachung von Stoffströmen in einer verfahrenstechnischen Anlage. Basierend auf der als ACPLT/PandIX vorliegende Anlagenstruktur sowie dem Wissen über den Aktualzustand der Anlageteile (Ventilstellungen, etc.) werden Leckagen erkannt oder Auswirkungen von Aktorsteuerungen vorhergesagt.

ACPLT/csHMI Das ClientSide HMI [JE12; Sch10] ist ein Meta-Modell für Benutzerschnittstellen, deren Realisierung ausführbare, erkundbare Objekte in der ACPLT-Laufzeitumgebung

sind. Es basiert auf der Idee, dass für die grafische Darstellung komplexer Inhalte lediglich eine kleine, klar definierte Menge an Basiselementen wie Linien, Kreise und Text notwendig sind. Diese werden durch spezielle ACPLT/OV-Objekte repräsentiert, die entsprechend ihrer grafischen Ausprägung SVG- bzw. HTML-Code erzeugen. Ein Server stellt diese grafischen Repräsentationen zur Ansicht mit beliebigem Browser zur Verfügung. Aufbauend auf den Basiselementen lassen sich Templates definieren, die eine vordefinierte Kombination an Basiselementen zur Wiederverwendung bereitstellen. So stellen die Template-Datenbanken ACPLT/csHMI$_{PF}$ und APCLT/csHMI$_{R\&I}$ grafische Elemente für die Prozessführung respektive für die Darstellung eines R&I-Fließbildes zur Verfügung.

ACPLT/PF Die Kaskadierung der Prozessführung durch eine Einzelsteuerebene und eine oder mehrere Gruppensteuerebenen ist eine gängige Methode, um komplexe Prozessansteuerungen zu strukturieren. In [Ens01; WE15] wird mit der kommandoorientierten Prozessführungen ein Konzept vorgestellt, dass diese Art der kaskadierten Prozessführung in der Funktionsbausteintechnik zugänglich macht. Dies bildet die Basis von ACPLT/PF, einer Funktionsbausteinbibliothek für die Einzelsteuerebene mit Ansteuerbausteinen für prozessleittechnische Aktoren. Die Bausteine der ACPLT/PF stellen neben der eigentlichen Ansteuerfunktionalität auch die Handhabung des Belegungszustandes (Hand, Automatik, ...) sowie eine Fehlerüberwachung zur Verfügung. ACPLT/PF ist nur durch lehrstuhlinterne Technologiepapiere auf informeller Ebene dokumentiert.

Für die Generierung des Bedienbildes (Szenario S2) sowie für die Konsistenzanalyse zwischen Bedienbild und Prozessführung und die darauf aufbauende Modellreparatur (Szenario S3) kommen die bereits genannten Modelle ACPLT/csHMI und ACPLT/PF zum Einsatz. Neben den genannten M1-Modellen sollen zudem die folgenden M2- bzw. M3-Modelle näher betrachtet werden:

SPS-Sprachen Die IEC 61131 [IEC61131] beschreibt den Aufbau, die Programmierung und die Kommunikation Speicherprogrammierbarer Steuerungen. Von besonderem Interesse für diese Arbeit ist dabei der dritte Teil der Norm, der sich mit den Programmiersprachen der Automatisierungstechnik beschäftigt. Neben den textuellen Sprachen Instruktionsliste (IL) und Strukturierter Text (ST) werden drei grafische Sprachen beschrieben. Der Kontaktplan (LD) orientiert sich in seiner Darstellung an Stromlaufplänen und eignet sich besonders, um Informationsflüsse zu beschreiben. Eine spezielle Art von Zustandsautomaten bietet die Ablaufsprache (SFC) und die Funktionsblock-Diagramme (FBD) fokussieren auf der Darstellung von Funktionsschnittstellen mit Parametern und Rückgabewerten. Mit der dritten Fassung der Norm erhielten die Sprachen 2013 eine objektorientierte Erweiterung.

ACPLT/OV Die ACPLT-Modellwelt umfasst eine Reihe von Modellen zu verschiedenen automatisierungstechnischen Fragestellungen. Das MOF-ähnliche Modell ACPLT/OV bildet den objektorientierten Kern aller ACPLT-Modelle. Eine elementare Eigenschaft dieses Objekt-Modells ist die Introspektion. Leitgedanke aller ACPLT-Modelle ist die Bereitstellung des Modells in drei Facetten:

1. ein Implementierungsunabhängiges Modell,
2. ein auf ACPLT/OV und davon abgeleiteten Modellen basierendes M1-Modell und

3. die eigentliche Realisierung als M0-Modell

Die ACPLT-Grundidee, alle Modelle mit einer objektorientierten, ausführbaren Repräsentation zu versehen, macht die auf ACPLT/OV aufbauenden Modelle so interessant für diese Arbeit.

ACPLT/FB ACPLT/FB ist ein objektorientiertes Bausteinsystem auf Basis von ACPLT/OV. Funktionsbausteinklassen kapseln Basisfunktionalität und die dazugehörigen Daten. Der Datenaustausch erfolgt dediziert über Bausteingänge und -ausgänge (Ports). Durch Instanziierung und Verknüpfung der Bausteininstanzen können komplexe Funktionalitäten realisiert werden. Die Bearbeitung der Bausteininstanzen erfolgt zyklisch. Die Reihenfolge der Abarbeitung ist dabei festgelegt durch die Reihenfolge der Bausteine in der Taskliste.

Die für diese Arbeit betrachtete Hierarchie der Modelle ist in Abbildung 3.1 dargestellt. Ausgegraute Modelle werden dabei nicht weiter auf ihre formalen Eigenschaften hin untersucht. Sie dienen lediglich der Einordnung der anderen Modelle in das Gesamtbild.

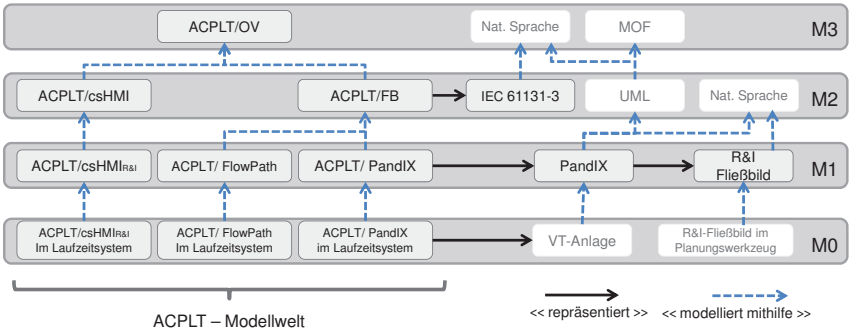


Abbildung 3.1: Modellhierarchie der Anwendungsszenarien

3.2 Bewertung der Modelle

Für die Bewertung der Modelle werden die Kriterien aus Abschnitt 2.3 zu Grunde gelegt. Zusätzlich werden die Modelle dahingehend untersucht, ob das Modell eine maschinenlesbare Syntax besitzt.

3.2.1 Fließschemata für verfahrenstechnische Anlagen

Die DIN EN 10628 [DIN10628] stellt eine Empfehlungen für die Erstellung verschiedener verfahrenstechnischer Fließbilder bereit. Das Grundfließbild dient zur Strukturierung von verfahrenstechnischen Anlagen oder Verfahren mit Hilfe von „besteht aus“- Beziehungen, während

das Verfahrensfließbild die Zerlegung eines Verfahrens in einzelne Verfahrensschritte illustriert. Von besonderem Interesse für diese Arbeit ist allerdings das dritte Fließbild, das R&I-Fließbild. Es stellt eingebaute Aktoren, Sensoren, Verrohrungen und andere für die Automatisierung der Anlage interessante Bauteile grafisch dar. Zudem sind Regelungs- und Steuerungsaufgaben gekennzeichnet.

Der Fokus der Norm liegt jedoch nicht auf der durch die Fließbilder abgebildeten Struktur, sondern auf der Bereitstellung der Grafikelemente. Der normative Teil befasst sich insbesondere mit Layout-Hinweisen wie Linienstärken oder Mindestabständen. Diese sind explizit als Vorschläge formuliert und beinhalten viele Freiheitsgrade. So lassen Aussagen wie, dass die „Hauptfließrichtung [...] im allgemeinen von links nach rechts und von oben nach unten“ verläuft, keine Rückschlüsse über die tatsächliche Fließrichtung ziehen. Auch die Verortung der Elemente lässt sich aus dem R&I-Fließbild nicht ableiten. So ist ein Temperatursensor, der am oberen Ende eines Behälters eingezeichnet ist nicht zwangsweise auch im oberen Bereich des Behälters verbaut.

Den bei weiten größten Teil der Norm bilden die nicht-normativen Anhänge mit Beispielen für Fließbilder. Auch die Symbole für die R&I-Fließbilder werden erst im Anhang C eingeführt. Die Darstellung der Symbole ist zwar mittlerweile Quasi-Standard, Hinweise wie der, dass die Spitze des Pumpensymbols in Förderrichtung zeigt, finden jedoch nicht zwingend Anwendung. Eine eingehende Analyse der Norm bieten Schmitz et al. [SSE08].

Zusammenfassend lässt sich feststellen, dass insbesondere die fehlende Präzision und Validierbarkeit eine maschinelle Modellauswertung auf Basis dieser Norm unmöglich macht. Der Fokus der Norm liegt klar bei der Interpretierbarkeit durch den Menschen und nicht durch eine Maschine. In ihrem Rahmen ist sie verständlich und konsistent. Symbole und ihre Bedeutung können allerdings unterschiedliche implementierungsspezifische Ausprägungen besitzen. Um Fließschemata für die Modelltransformation nutzbar machen zu können, wird eine ergänzende formale Beschreibung der strukturellen Zusammenhänge benötigt.

3.2.2 CAEX

Die DIN EN 62424 [DIN62424] widmet sich der Aufgabe, zumindest einen Teil der DIN EN 10628, die dargestellten Steuerungs- und Regelungsaufgaben eines R&I-Fließbildes für die maschinelle Verarbeitung aufzuarbeiten. Die Norm stellt zunächst eine normierte grafische Darstellung von prozessleittechnischen Funktionen in Form von PLT-Stellen zur Verfügung. Der Kern der Norm ist jedoch das XML-basierte Meta-Modell CAEX für den Austausch von objektorientierten Strukturdaten. CAEX erweitert die, für objektorientierte Sprachen typischen Elemente Klasse und Schnittstelle durch Rollenklassen. Während Schnittstellen syntaktische Anforderungen an eine Instanz beschreiben, enthalten Rollenklassen semantische Festlegungen wie z.B. Mindestdurchsätze einer Pumpe. Die drei Elemente lassen sich jeweils zu Klassen-, Instanz- und Rollen-Bibliotheken zusammenfassen und zusammen mit der Instanzhierarchie in einem CAEX-Dokument abbilden. Der Natur eines Meta-Modells entsprechend ist CAEX für sich erstmal nicht für den Datenaustausch anwendbar. Hierzu müssen konkrete CAEX-Bibliotheken entworfen werden.

Ein dritter Aspekt der Norm ist daher die Abbildung der für den Austausch mit der Prozessleittechnik relevanten Daten des R&I-Fließbildes auf die CAEX-Elemente, im Folgenden PLT-CAEX genannt. Zu diesen Daten gehören Signal-, Prozess- und Produktverbindungen sowie Steuerungsfunktionen und zusätzliche physikalische Prozessparameter. Mechanische Zusammenhänge sowie Angaben zu Betriebsmitteln und grafische Attribute werden hingegen nicht abgebildet. Die Norm beschreibt, mit Hilfe welcher CAEX-Elemente einzelne Aspekte der grafischen Darstellung abgebildet werden müssen. Eine konkrete Umsetzung erfolgt jedoch nur beispielhaft.

Da die drei Aspekte der Norm grundverschiedene Zielsetzungen und Einsatzgebiete verfolgen, wird an dieser Stelle eine getrennte Bewertung anhand der Kriterien formaler Modellierung vorgenommen.

PLT-Stelle Die Beschreibung der grafischen Repräsentation ist zumeist prägnant und leicht verständlich. Das Modell ist konsistent und eindeutig beschrieben, jedoch auf Grund seiner grafischen Natur nur schwer maschinenauswertbar.

CAEX Die Modellbeschreibung von CAEX erfolgt in drei Stufen. Zunächst wird eine auf Verständnis und Semantik ausgerichtete Beschreibung durch Fließtext, hinterlegt mit Beispielen, gegeben. In einem zweiten Schritt werden die einzelnen Elemente der XML-Grammatik in knapper Form dargestellt und ihre Bedeutung zusammengefasst. Ergänzt wird dies durch eine XML-Schema-Definition. Es werden alle Anforderungen an eine formale Modellierung erfüllt. Kleinere Inkonsistenzen in den gelieferten Beispielen wie die Benennung der External Interfaces [DIN62424, S. 51] haben keinen Einfluss auf die Konsistenz des eigentlichen Modells.

PLT-CAEX In der Norm wird bereits auf die semiformale Natur der Modellbeschreibung hingewiesen. Diese Einschätzung kann hier durchaus unterstützt werden. Auch wenn der Name der Norm sowie die einleitenden Worte vermuten lassen, dass dieser Aspekt den Kern der Norm darstellt, ist das Modell weder vollständig noch eindeutig beschrieben. Es dient vielmehr beispielhaft zur Verdeutlichung der CAEX-Sprachelemente. Eine weiterführende Analyse der daraus entstehenden Probleme und möglicher Lösungsansätze bieten Theurich et. al [The+14].

Die drei sehr unterschiedlichen Facetten der Norm, grafische Repräsentation von Prozessleittechnischen Informationen, Austauschformat für Strukturdaten und Abbildung der R&I-Informationen in das Austauschformat, machen die Norm unnötig komplex. Eine klare Fokussierung auf CAEX und die Referenzierung vorhandener Normen sowie die Ergänzung durch nicht-normative Veröffentlichungen hätten hier für ein insgesamt kompakteres und formaleres Modell gesorgt.

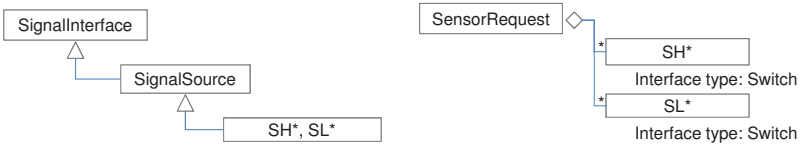
3.2.3 PandIX

PandIX greift den fehlenden formalen Aspekt der DIN EN 62424 bei der Abbildung von PLT-Stellen in CAEX auf. Die PandIX-Modellbeschreibung liegt als, mit UML-Diagrammen angereicherter Fließtext, vor. Dieser benennt und formalisiert die für die Automatisierungstechnik relevanten Informationen, die in R&I-Fließbildern enthalten sind, in einer objektorientierten Be-

trachtungsweise. Die darauf aufbauende CAEX-Repräsentation umfasst sowohl eine Klassen-, eine Schnittstellen- sowie eine Rollenbibliothek, so dass Syntax und Semantik von PLT-Stellen in einer austauschbaren Form beschrieben werden.

Die Modellbeschreibung für PandIX wird ergänzt durch eine CAEX-Musterbibliothek, die weiterführende CAEX-Klassen bereitstellt. Während der Fokus bei PandIX auf Informationsflüssen liegt, erweitert die Musterbibliothek das Repertoire auf die Elemente des Materialflusses.

Die Top-Down Betrachtungsweise von den Objekten der Anlagenwelt bis hinunter zum Aufbau einer Sensorstelle macht die Idee und den Grundaufbau allgemein verständlich. Kleinere Inkonsistenzen bzw. missverständliche UML-Diagramme bieten jedoch bei einer konkreten Realisierung viel Interpretationsfreiraum. So lassen die - nicht ganz UML-konformen - Darstellungen in Abbildung 3.2 zwei widersprüchliche Interpretationsmöglichkeiten offen. SH* und SL* können entweder Klassen sein [PandIX, S. 26] oder Instanznamen [PandIX, S. 11] von Objekten der Klasse Switch. Die CAEX-Repräsentation beantwortet diese Frage ebenfalls nicht eindeutig, da SH* und SL* in ihr nicht definiert sind. Das Vorhandensein der Klasse Switch und ihre Beschaffenheit lassen aber vermuten, dass die Interpretation als Instanzen gemeint ist. Auch die wechselnde Verwendung englischer und deutscher Begriffe ohne explizite Zuordnung zueinander (z.B. Prozessanlagenelement →PPERequest) erschweren die eindeutige Interpretation und vor allem die Validierbarkeit von PandIX-Modellen.



(a) S.11: SH*, SL* als Klassen

(b) S.25: SH*, SL* als Instanzen der Klasse „Switch“

Abbildung 3.2: Mehrdeutige Repräsentation von Elementen in [PandIX]

Eine weitere Herausforderung bei der rechnergestützten Auswertung von PandIX-Modellen ist die fehlende Vollständigkeit der Modellbeschreibung. Die Musterbibliothek stellt nur eine der möglichen Erweiterungen dar. Die letztendlich anzuwendende Modellbeschreibung kann von Implementierung zu Implementierung oder im schlimmsten Fall innerhalb einer Implementierung gar von Modell zu Modell variieren.

3.2.4 Sprachen für die SPS-Programmierung

Die Beschreibung von Programmiersprachen für die SPS-Programmierung steht im Fokus des dritten Teils der IEC 61131-3 [IEC61131]. Die Norm fokussiert bei der Sprachbeschreibung zunächst den Strukturierten Text (ST). Der Hauptteil der Norm entwickelt schrittweise die Grammatik der Sprache und reichert diese anhand von Beispielen mit Semantik an. Annex B bietet anschließend nochmal die formale Spezifikation der Syntax in kompakter Form. Die Instruktionsliste (IL) wird ähnlich formal, bedingt durch ihren deutlich geringeren Umfang

aber in knapperer Form beschrieben. Auch die grafischen Sprachen werden, soweit möglich, formal beschrieben. So wird für die Funktionsblock-Diagramme (FBD) festgelegt, dass als Darstellung ein Rechteck mit Namen des Bausteins im oberen Teil gewählt werden soll. Zudem werden Position, Verbindungsmöglichkeiten und die Darstellung von Negation von Bausteinparametern beschrieben. Hier bleibt es natürlich nicht aus, dass ein gewisser Freiheitsgrad in der Darstellung besteht. Die Semantik der einzelnen in FBD vorhandenen Bausteine wird zumeist in ST beschrieben oder ergibt sich aus den zugehörigen logischen Operatoren. Die Norm lässt, bis auf kleine Ausnahmen in der grafischen Darstellung, keinen Platz für Doppeldeutigkeiten oder Widersprüchlichkeiten. Auch für mögliche implementierungsspezifische Erweiterungen der Norm wird klar geregelt, wie diese sich konsistent, prägnant und korrekt in das Gesamtbild einbetten lassen.

3.2.5 AutomationML

Die Modellbeschreibung von AutomationML (kurz AML) ist aufgeteilt in das Kernmodell [AML1], vorgefertigten Rollenbibliotheken [AML2] und Einbindungsvorschriften für gängige Modelle [AML3; AML4]. Allen drei Komponenten der Modellbeschreibung ist die starke Verquickung von Semantik- und Syntaxbeschreibung gemein. Dies macht die Modellbeschreibung unnötig komplex.

Bei der Beschreibung des Kernmodells wird zunächst eine Beschreibung auf Basis von Beispielen geliefert. Dies sollte an sich hilfreich sein, um die Grundgedanken des Modells dem Leser näher zu bringen. Im Fall von AML fehlt allerdings die Referenz auf die konkrete Syntax, so dass der Abgleich der beispielhaften Beschreibung mit der formalen Beschreibung schwer fällt. Auch der formalere gehaltene Teil der Modellbeschreibung mischt Syntax und Semantik und erschwert zusätzlich durch widersprüchliche Formulierungen die eindeutige Interpretation. Ein Beispiel hierfür ist die Interface-Klasse „Order“. Sie dient als Basisklasse für alle Arten von sortierten Listen. Das enthaltene Attribut „Direction“ gibt dabei die Sortierreihenfolge an. Als valide Werte für „Order“ werden allerdings nur „In“, „Out“ und „InOut“ angegeben, was der semantischen Beschreibung widerspricht, die „Descending“ als Möglichkeit vorsieht.

> The interface class “Order” is an abstract class that shall be used for the description of orders, e.g. a successor or a predecessor. [. . .] The attribute “Direction” shall be used in order to specify the direction. Permitted values are “In”, “Out” or “InOut”. [AML1, S. 29]

Auch die mitgelieferte CAEX-Repräsentation liefert an dieser und anderen Stellen keine Konkretisierung sondern noch mehr Flexibilität. Hier müssen vorgefertigte Modellinstanzen zu Rate gezogen werden, um die beabsichtigte Interpretation zu erkennen.

AutomationML ist explizit nicht vollständig beschrieben, sondern offen für die Einbettung beliebiger Teilmodelle. Es eignet sich daher als Basis von *SpeedStandardisierung* [Sch+15; Mer16], die darauf setzt, dass das Grundgerüst standardisiert ist, einzelnen Facetten aber als Pseudo-Standard oder *Request for Comment* schnell und unkompliziert umgesetzt werden können.

Das Kernmodell von AML besitzt eine starke Typisierung und eine für die maschinelle Auswertung optimierte Darstellung. Mögliche Einbettungen müssen getrennt bewertet werden.

3.2.6 ACPLT-Modelle

Der Formalisierungsgrad der ACPLT-Modelle ist sehr heterogen. Für die Kernmodelle existiert eine formale Spezifikation in Form einer Grammatik [Alb97b; Mey02] sowie ausführliche beschreibende Dokumentation [Alb96; Alb97a]. Anders sieht es bei den Anwendungsmodellen aus. Sie basieren zwar meist auch auf einer formalen Beschreibung, diese ist aber auf Grund ihrer Natur als Forschungsmodelle meist nur in Auszügen veröffentlicht. Nur vereinzelt sind die M1-Modelle bis zur formalen Reife vorangetrieben. Neben einem implementierungsunabhängigen Modell besitzen ACPLT-Modelle ein auf ACPLT/OV basierendes M1-Modell und eine Repräsentation in der automatisierungstechnischen Laufzeitumgebung. Dank dieser drei Facetten bieten die ACPLT-Modelle eine optimale Grundlage für die rechnergestützte Verarbeitung modellierter Zusammenhänge.

Im Einzelnen ergibt sich für die ACPLT-Modelle folgendes Bild:

ACPLT/OV Das Basismodell der ACPLT-Modellwelt ist umfassend spezifiziert und dokumentiert [Mey02]. Auszüge aus der Arbeit fassen die Kernpunkte in Form von Technologiepapieren zusammen. Das für diese Arbeit interessante Technologiepapier 3 [Alb97b] beschreibt eine vollständige Grammatik von ACPLT/OV. Ergänzt wird es durch weitere Technologiepapiere [Alb96; Alb97a], die verschiedene Facetten des Modells beleuchten und mit Hilfe von UML-Diagrammen und Fließtext mit Semantik versehen. Die Trennung von Grammatik und Semantik ermöglicht ein kompaktes, konsistentes, validierbares und leicht verständliches Modell.

ACPLT/FB Das Funktionsbausteinsystem ACPLT/FB ist umfänglich, wenn auch nicht vollständig, in der Softwaredokumentation zum Tool iFBspro [NN04] beschrieben. Des Weiteren existieren Veröffentlichungen, die das Konzept und einige Details näher erläutern [GKE12; Yu+12]. Die vollständige Syntax liegt in Form einer nichtöffentlichen ACPLT/OV-Modelldatei (.ovm) sowie einer zugehörigen ACPLT/OV-Funktionsdatei (.ovf) vor. Alles in allem treffen für ACPLT/FB die gleichen Eigenschaften wie für ACPLT/OV zu: kompakt, konsistent, validierbar und leicht verständlich.

ACPLT/PandIX Da es sich bei ACPLT/PandIX weitestgehend um eine 1-zu-1 Übertragung der XML-Elemente aus PandIX in Funktionsbausteine und Verknüpfungen handelt (vgl. Abbildung 4.2a), leiten sich die meisten Punkte aus der Bewertung der PandIX-Modellbeschreibung ab. In den Punkten „Konsistenz“ und „Präzision“ wurde bei der Implementierung eine mögliche Interpretation umgesetzt bzw. bei Inkonsistenzen eine Variante gewählt.

ACPLT/FlowPath Eine vollständige Grammatik sowie eine formale Semantik lassen bei diesem Modell keine Interpretationsfreiheit zu. Eine vorliegende Implementierung sowie umfangreiche Dokumentation [Qui11] ergänzen die formale Modellierung.

ACPLT/csHMI Das Modell ist durch eine vollständige, aber unveröffentlichte Modellierung als lehrstuhlinternes Technologiepapier definiert. Auszüge aus dem Modell wurden in wissenschaftlichen Arbeiten und auf einschlägigen Fachtagungen präsentiert [Sch10; JE13;

JE12]. Die Basis bildet ein Klassendiagramm, das alle erlaubten Klassen beinhaltet. Ausgehend davon werden die einzelnen Klassen und ihre Parameter beleuchtet. Dies geschieht in einer semiformalen Kombination aus UML und Fließtext. Der Forschungscharakter dieses Modells kommt hier stark zum Tragen. Die niedergeschriebene Modellbeschreibung lässt viel Interpretationsspielraum zum Beispiel bei der Umsetzung von Nutzerinteraktionen. Umfangreich vorhandene Beispielanwendungen zeigen aber, welche Umsetzung der Autor im Sinn hatte.

Auf Grund seiner auf Basiselemente fokussierten Art eignet sich das Modell als Grundlage für grafische Modellierung verschiedenster Art [SE13]. Aufbauend auf ACPLT/csHMI ist eine Reihe von Template-Bibliotheken entstanden. Nicht in allen Fällen besitzen diese ein vollständig niedergeschriebenes Modell. Hierzu gehören das Bedienmodell zur Prozessführung ACPLT/csHMI$_{PF}$. In [JE12] führt Jeromin Teile von ACPLT/csHMI$_{PF}$ als Bestandteil des Meta-Modells ACPLT/csHMI auf. An dieser Stelle soll es jedoch als eigenständiges Modell gesehen werden, da es nicht wie ACPLT/csHMI grundlegende HMI-Primitiva modelliert, sondern Anlagenstrukturen. Da für ACPLT/csHMI$_{PF}$ keine weiter Dokumentation besteht, kann eine Einordnung hierzu nicht stattfinden.

ACPLT/PF In [WE15] beschreiben die Autoren die Grundidee sowie abstrakte und drei konkrete Syntaxen für ACPLT/PF. Basis dazu bildet die kommandoorientierte Prozessführung, die umfassend und formal fundiert in [Ens01] vorgestellt wird.

3.3 Gewonnene Erkenntnisse

In dieser Arbeit wird nicht weiter auf die Optimierung des Formalisierungsgrades vorhandener Normen, Richtlinien oder nicht-normativer Modellbeschreibungen eingegangen. Dies ist Aufgabe entsprechender Gremien und Arbeitskreise. Ein Aufruf zur formalen Beschreibung von Modellen soll an dieser Stelle trotzdem nicht fehlen, da sie nicht nur der automatisierten Modelltransformation zugutekommt, sondern auch den eigentlichen Hintergrund der Modellierung fördert, eine einheitliche Verwendung der Modelle.

In den vorgestellten Modellen wird vielfach auf mehrdeutigen Fließtext zurückgegriffen. Ob diese Mehrdeutigkeit politisch beabsichtigt ist, um allen an der Erstellung beteiligten Parteien gerecht zu werden, oder weil das Bewusstsein für die Mehrdeutigkeit nicht vorhanden ist, sei an dieser Stelle dahin gestellt. Fest steht, dass dies unweigerlich zu unterschiedlichen Interpretationen und Implementierungen führt, was den Zweck eines Modells ad absurdum führt. Für die zukünftige Beschreibung von Automatisierungsmodellen wird an dieser Stellen daher folgende Empfehlung gegeben:

Trennung von Syntax und Semantik Bei der Analyse der verschiedenen automatisierungstechnischen Modelle hat sich gezeigt, dass eine getrennte Beschreibung von Syntax und Semantik ein wesentliches Kennzeichen der formal beschriebenen Modelle darstellte. Insbesondere die dadurch entstandene Kompaktheit der Syntax vereinfacht die Entwicklung konsistenter und vollständiger Modelle.

Semi-Formale Semantik Der Einsatz von formalen Semantiken in der Automatisierungstechnik als Alternative zur natürlichsprachlichen Beschreibung ist von Fall zu Fall abzuwä-

gen. Der formale Aspekt steht hierbei der allgemeinen Verständlichkeit entgegen. Bei der Verwendung der natürlichsprachlichen Beschreibung muss diese genauestens auf missverständliche oder zweideutige Formulierungen untersucht werden. Verwendete Bilder müssen im Text genau beschrieben werden, da gerade unkommentierte Bilder viel Interpretationsspielraum zulassen. Für die Beschreibung der Semantik empfiehlt sich zudem UML als unterstützendes Werkzeug. UML hat sich mittlerweile in vielen Domänen als Quasistandard zur Modellierung herauskristallisiert und wird daher auch von Domänenfremden leichter verstanden.

Abstrakte Syntax Unabhängig von der Syntax, die der Entwickler später für die Modellentwicklung nutzt, empfiehlt sich, zunächst die Konzepte des Modells allgemein verständlich mittels einer abstrakten Syntax zu beschreiben. Es empfiehlt sich ein UML-Klassendiagramm zur Beschreibung des Modellalphabets und weitere Klassendiagramme zur Beschreibung der modellinternen Zusammenhänge bereitzustellen:

Alphabet Zur Auflistung der vorhandenen Modellelemente inklusive aller modellspezifischen Assoziationen, empfiehlt sich ein Klassendiagramm, das lediglich die Generalisierung als Assoziation erlaubt. Auch Variablen und Methoden der Modellelemente sollten in diesem Diagramm außer Acht gelassen werden. Durch eine solche Auflistung entsteht ein erster, umfassender Überblick über das Alphabet des Modells.

Konstruktionsregeln Weitere Assoziationen zwischen Modellelementen, wie „Besteht-Aus“ und die Verwendung modellspezifischer Assoziationen, sowie wichtige Methoden und Variablen sollten in davon unabhängigen Klassendiagrammen beschrieben werden. Jedes dieser Klassendiagramme sollte einen klar umrissenen und im Text ausführlich erläuterten Aspekt des Modells abbilden. Bei der Verwendung der Assoziationen sollte jeweils die Multiplizität angegeben werden.

Die Modellbeschreibung sollte abschließend formuliert sein. Dadurch wird eindeutig festgelegt, welche Konstrukte ein valides Modell bilden und welche nicht erlaubt sind. Ist eine flexible Erweiterbarkeit des Modells unabdingbar, so sollte dies an klar definierten Schnittstellen im Modell erfolgen.

Konkrete Syntax Für den Entwickler von Modellen stellt die konkrete Syntax das Handwerkszeug dar. Die konkrete Syntax hat daher weniger die allgemeine Verständlichkeit als Ziel, sondern vielmehr die Eignung für die konkrete Modellerstellung. Die konkrete Syntax muss unabhängig von der gewählten Darstellungsform konsistent zur abstrakten Syntax sein. Für graphische/diagrammartige Modelle bietet sich die Syntaxdefinition mittels Metamodellierung auf Basis von UML (bzw. MOF) an, für textuelle Sprachen eignet sich hingegen eher eine EBNF-basierte Grammatik. Ist ein Datenaustausch oder eine textuelle Repräsentation des Modells das Ziel, empfiehlt sich die Erstellung einer XML-Schema-Definition.

Starke vs. schwache Typisierung [SK12] Von starker Typisierung spricht man, wenn Objekte mit unterschiedlichen Eigenschaften auch unterschiedlichen Klassen zugeordnet sind, bei der schwachen Typisierung werden diese Eigenschaften durch Attribute einer gemeinsamen Klasse repräsentiert. Die Verwendung von Attributen zur Unterscheidung von Elementen verschiedenen Typs hat durchaus seine Vorteile, da das Modell zunächst weniger Klassen besitzt. So wird im PandIX nur die Klasse „ActuatorRequest“ benötigt, um Pumpen, Ventile und elektrische Aktuatoren abzubilden. Eine Unterscheidung erfolgt anhand des Attributs „FunctionCode“. Trotzdem sollte diese schwache Typisierung sehr

sparsam und bewusst eingesetzt werden, da nur die starke Typisierung eine typgerechte Verwendung und Parametrierung eines Objektes garantieren kann. Insbesondere beim Austausch von Modelldaten zwischen verschiedenen Werkzeugen kann die schwache Typisierung zu Fehlinterpretationen führen.

Dieses Vorgehen ist sicher nicht für alle Modelle der Automatisierungstechnik vollständig umsetzbar, für viele bietet es aber eine einfach zu realisierende und nachvollziehbare Möglichkeit der Beschreibung. Zudem ermöglicht dieses Vorgehen ein frühzeitiges Erkennen von potentiellen Missverständnissen oder Widersprüchen in den Modellen und stellt eine solide Grundlage für die maschinelle Auswertung der Modelle dar.

4 Modelltransformation in der Automatisierungstechnik

Während des gesamten Lebenszyklus einer verfahrenstechnischen Anlage ist eine Vielzahl von Modellen im Einsatz. Dabei können die meisten beteiligten Modelle nicht losgelöst betrachtet werden. Eine Änderung in einem Modell erzeugt oftmals eine Kaskade von Modellanpassungen. Diese starke Korrelation sowie die mit der langen Lebensdauer einer verfahrenstechnischen Anlage einhergehende Dynamik der Modelle bedürfen eines gut funktionierenden Änderungsmanagements. Hier ist das größte Potential für den Einsatz rechnergestützter Modelltransformation in der Automatisierungstechnik zu sehen: der Abgleich von Modelländerungen über alle beteiligten Modelle hinweg. Dies kann zum einen durch das Erzeugen und Ändern von Modellen erfolgen, zum anderen sind auch Konsistenzprüfungen denkbar. Weiteres Potential bieten die starken Regularien bei der Planung und dem Bau einer verfahrenstechnischen Anlage. Neben umfangreichen Dokumentationspflichten und Sicherheitsbestimmungen bieten auch Zertifizierungsvorschriften Einsatzmöglichkeiten für rechnergestützte Modelltransformationen. So kann ein Teil der Dokumentation anhand der Planungsdaten generiert, das Vorhandensein von sicherheitsrelevanten Programmteilen überprüft oder die Zertifizierung der resultierenden Automatisierungsfunktion durch einmalige Zertifizierung der Transformation vereinfacht oder vollständig ersetzt werden. Nicht zuletzt bietet der in dieser Arbeit angestrebt Weg hin zu Automatisierungsfunktionen als Serienprodukt die Möglichkeit, auf Änderungen in den Planungsdaten so flexibel wie nie zuvor reagieren zu können. Zur Realisierung dieses Ansatzes bedarf es einer anlagenneutralen Beschreibung der Modellzusammenhänge. Zur Laufzeit erfolgt eine Auswertung der aktuell gültigen Planungsdaten mittels Modelltransformation und als direkte Folge davon die Bereitstellung der Automatisierungsfunktion.

In diesem Kapitel werden zunächst die aus der Informatik stammenden Grundbegriffe der Modelltransformation eingeführt und bisherige Ansätze aus dem Bereich der Automatisierungstechnik vorgestellt, Modellgrenzen für die enthaltenen Informationen durchlässig zu machen. Um eine Einordnung dieser Ansätze vornehmen zu können, erfolgt im Vorfeld eine Analyse der durch die Automatisierungstechnik gestellten Anforderungen.

4.1 Allgemeine Begriffsbestimmung

Wir sprechen von korrelierenden Modellen, wenn ein Zusammenhang zwischen den Daten oder Strukturen zweier (oder mehrerer) Modelle besteht und daher eine widerspruchsfreie Modellierung der jeweiligen Modellinstanzen beachtet werden muss. Traditionell haben Modelltransformationen zum Ziel, diese Widerspruchsfreiheit (Konsistenz) zu gewährleisten, in dem

sie automatisiert die Instanz eines Quellmodells in eine Instanz des korrelierenden Zielmodells überführen. Das Quellmodell einer Modelltransformation ist jenes, aus dem die Daten gewonnen werden, das Zielmodell ist das Modell, welches erstellt bzw. verändert wird.

Basis einer Modelltransformation ist ein Satz an Transformationsregeln und ein Kontrollalgorithmus. Transformationsregeln beschreiben, wie Strukturen eines Quellmodells in Strukturen eines Zielmodells abzubilden sind [KWB03]. Der Kontrollalgorithmus ist für die Auswahl und Anwendung geeigneter Regeln zuständig.

Die in den folgenden Kapiteln verwendeten Beispiele basieren auf dem in Kapitel 1.2 vorgestellten Szenario S2, der Erstellung eines Bedienbildes anhand der PandIX-Daten. Der vollständige Regelsatz zu diesem Szenario findet sich in Anhang B. Für die beispielhafte Demonstration der vorgestellten Ideen beschränken wir uns an dieser Stelle zunächst auf die Regeln, die die Abbildung eines Ventils und der zugehörigen PLT-Stelle beschreiben. Abbildung 4.1verdeutlichen den Zusammenhang zwischen der PandIX-Darstellung von Ventil und PLT-Stelle und der grafischen Repräsentation. Die angestrebte Modelltransformation findet zwischen den entsprechenden ACPLT-Modellen ACPLT/PandIX und ACPLT/csHMI statt (vgl. Abbildung 4.2).

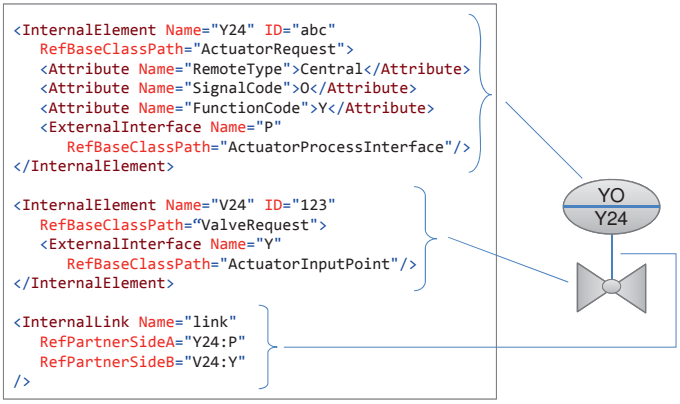


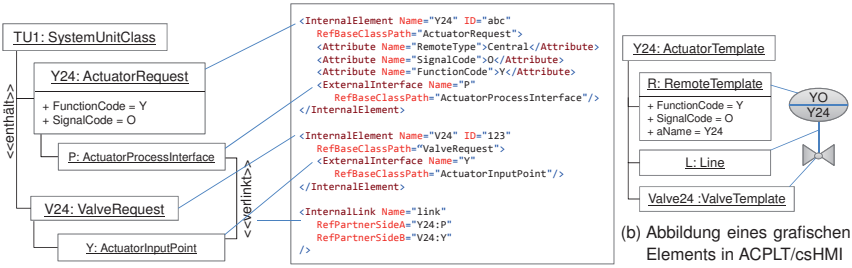
Abbildung 4.1: Korrelation zwischen PandIX-Modell einer Anlage und zugehörigem Bedienbild[1]

Dieses Szenario beinhaltet typischer Weise folgende Aktionen:

1. Entwurf der Anlage mit Hilfe eines CAD-Werkzeugs
2. Generieren eines Grobentwurfs für das Bedienbild anhand der CAD-Daten
3. Nachträgliche Änderungen am Entwurf im CAD-System
4. Manuelle Anpassungen am Bedienbild

Dadurch ergeben sich vielfältige Aufgaben für den Einsatz von Modelltransformationen:

[1] Die PandIX-Darstellung ist der Übersichtlichkeit halber leicht vereinfacht. Unter anderem wurden die Value-Tags der Attribute entfernt und die Pfadangabe für die Basisklassen ist in verkürzter Form angegeben.



(a) Abbildung eines PandIX-Modells nach ACPLT/PandIX

(b) Abbildung eines grafischen Elements in ACPLT/csHMI

Abbildung 4.2: ACPLT-Repräsentationen von PandIX und Bedienoberfläche

Initiale Batch-Transformation Der bestehende CAD-Entwurf wird als PandIX-Modell bereitgestellt und daraus das zugehörige Bedienbild „in einem Rutsch“ automatisch erzeugt.

Inkrementelles Update Änderungen am CAD-Entwurf werden inkrementell in das zuvor schon erstellte Bedienbild nachgezogen.

Zurückpropagieren Änderungen am Bedienbild werden (soweit relevant) in den CAD-Entwurf übertragen. Dies kann zum Beispiel Bezeichner oder verwendete Sensor-/Aktortypen betreffen.

Bei den genannten drei Transformationen handelt es sich jeweils um Outplace-Transformationen, bei denen Quell- und Zielmodell disjunkt sind. Dabei bleibt das Quellmodell unverändert. Im Gegensatz dazu sind bei der Inplace-Transformation Quell- und Zielmodell identisch. Szenario S1 ist (je nach Implementierung) ein Kandidat für eine solche Inplace-Transformation, da das Einfärben von Rohleitungen anhand des aktuellen Anlagenzustandes das Bedienbild sowohl als Quell- als auch als Zielmodell benötigt.

4.2 Besondere Herausforderungen in der Automatisierungstechnik

Die Akzeptanz einer rechnergestützten Modelltransformation zur modellgetriebenen Entwicklung von Automatisierungsfunktionen ist nur zu erwarten, wenn folgende Grundeigenschaften zugesichert werden können:

Unidirektionale Batch-Transformation Aus einem oder mehreren Planungsmodellen muss per Batch-Transformation das Zielmodell generiert werden. Anpassungen des Zielmodells müssen durch den Applikateur in gewohnter Weise vorgenommen werden können.

Inkrementelle Änderungen Die Anforderungen an eine Anlage können sich über die Zeit ändern. Es muss daher möglich sein, Änderungen an den Planungsdaten vorzunehmen und die geänderten Informationen per Modelltransformation in die Implementierung zu übernehmen. Vom Applikateur vorgenommene Anpassungen des Zielmodells müssen, soweit nicht direkt von der inkrementellen Änderung betroffen, dabei unberührt bleiben.

Bidirektionale Auswertbarkeit Die Lehre des ordnungsgemäßen Baus und Betriebs einer Anlage [Pol94] postuliert, dass Änderungen immer zunächst in den Planungsdaten und danach in der Implementierung eingepflegt werden. Die Realität sieht jedoch meist anders aus. Es wird daher ein Mechanismus benötigt, der nicht nur die Informationsübernahme aus den Planungsmodellen in die Implementierung ermöglicht, sondern auch den umgekehrten Weg unterstützt. Dadurch können Änderungen in der Implementierung automatisiert in den Planungsdaten „dokumentiert“ werden.

Konsistenzüberprüfung Ergänzend zur bidirektionalen Auswertbarkeit muss ein Mechanismus vorhanden sein, der die Widerspruchsfreiheit (Konsistenz) von Instanzen prüft, ohne Änderungen vorzunehmen. Dies ist insbesondere dann von Interesse, wenn die verfahrenstechnische Anlage bereits in Betrieb ist. Automatisierte Änderungen an den beteiligten Automatisierungsfunktionen sind dann nur noch in festgelegten Wartungszeiträumen möglich. Eine passive Überprüfung kann aber jederzeit durchgeführt werden.

Modellreparatur Inkonsistente Modelle müssen wieder in einen konsistenten Zustand überführt werden. Da Inkonsistenzen sowohl durch Änderungen im Quell- als auch im Zielmodell entstehen können, muss der Nutzer aktiv in den Reparaturprozess eingebunden werden. Alternativ müssen frühere Modellzustände in den Reparaturprozess einbezogen werden, um identifizieren zu können, ob die Inkonsistenz durch Änderungen am Quell- oder Zielmodell verursacht wurden.

Nachvollziehbarkeit Die Transformation muss nachvollziehbar durchgeführt und dokumentiert werden. Dazu gehört auch, dass die Transformation deterministisch ist und die einzelnen Transformationsschritte dokumentiert werden.

Freie Werkzeugwahl In der Planungs- und Betriebsphase einer verfahrenstechnischen Anlage ist eine ganze Reihe hochspezialisierter Werkzeuge im Einsatz. Die Wahl dieser Werkzeuge soll aus zweierlei Gründen nicht eingeschränkt werden. Auf der einen Seite haben Planungsingenieure und Applikateure nicht immer Einfluss auf die Wahl des Werkzeugs. Dies ist insbesondere der Fall, wenn Planungsdaten von extern eingekauft werden oder wenn die Implementierung an ein Subunternehmen vergeben wird. Auf der anderen Seite sollen Planungsingenieure und Applikateure sich nicht an ein Werkzeug binden müssen, sondern das Werkzeug wählen können, das ihren Ansprüchen am besten genügt.

Wartbare, nachvollziehbare Regeln Für einen nachhaltigen Ansatz ist es wichtig, dass die formulierten Regeln über Jahrzehnte und von verschiedenen Anwendern leicht nachzuvollziehen und anzupassen sind. Komplexe, auf speziellen Sprachen basierende Regeln können daher nicht zum Einsatz kommen. Vielmehr muss die Formulierung der Zusammenhänge möglichst durch einfache Verknüpfung der beteiligten und somit bekannten Modelle erfolgen. Diese sollten minimalistisch angereichert werden mit entsprechenden Elementen für die Modelltransformation.

Hinweise auf weitere Anforderungen bieten die zuvor definierten Anwendungsszenarien.

S1. Einzelne Automatisierungsfunktion als Serienprodukt Das Besondere an diesem Anwendungsbeispiel ist die Verwendung von Informationen aus zwei Quellen. Zum einen wird das Anlagenstrukturmodell benötigt, um Zusammenhänge zwischen Aktoren und Rohrleitungen zu identifizieren, auf der anderen Seite wird der aktuelle Zustand der Prozessführung für die Auswertung der offenen Flusswege benötigt. Neben der Verwendung

zweier Quellmodelle ist auch die Transformation innerhalb der automatisierungstechnischen Laufzeitumgebung ausschlaggebend für dieses Szenario.

S2. Entwicklungsbegleitende Modelltransformation Das Bedienbild wird in diesem Szenario als R&I-Fließbild dargestellt. Die entsprechenden Abbildungsvorschriften sowie die Positionierung werden aus dem PandIX übernommen. Die Anforderungen an ein Bedienbild erfordern aber im Normalfall eine angepasste Positionierung und Darstellung. Das Bedienbild muss daher nach der Erstellung vom Applikateur änderbar sein. Es muss weiterhin möglich sein, spätere inkrementelle Änderungen durch erneute Transformation aus dem PandIX vorzunehmen. Bereits vorgenommene Parametrierungen oder Ergänzungen im Bedienbild müssen davon unberührt bleiben, sofern sie nicht direkt von den Änderungen betroffen sind.

S3. b) Modellreparatur bei aufgetretenen Inkonsistenzen Für die Realisierung dieses Szenarios muss eine Interaktion mit dem Benutzer möglich sein, da Inkonsistenzen nicht automatisiert in die eine oder andere Richtung aufgelöst werden können.

4.3 Stand der Technik

Ein erster Schritt zur Nutzung der Planungsdaten im weiteren Lebenszyklus der Anlage sind Austauschmodelle. Diese spielen insbesondere bei der Anforderung der freien Werkzeugwahl eine große Rolle. Nahezu alle gängigen Engineeringwerkzeuge bieten heutzutage einen Export/Import ihrer Daten als XML-Datei oder in einem anderen digitalen Format an. Die Modelltransformation zwischen Transportmodell und dem Datenmodell des Werkzeugs erfolgt hierbei bei der Interpretation bzw. der Generierung der Austauschformate durch die Import/Export Schnittstelle der Werkzeuge. Herstellerspezifische Modelle haben den Vorteil, dass sie die Daten eines Werkzeugs optimal abbilden können. Durch die Vielzahl an auf dem Markt befindlichen Werkzeugen impliziert dieser Ansatz einen erheblichen Implementierungsaufwand. Allein das Planungswerkzeug ePLAN P8 bietet Schnittstellen zu zehn verschiedenen Automatisierungssystemen sowie weitere ePLAN-spezifische Austauschformate an. Wie in Kapitel 3 gezeigt, gibt es auch in Normungsgremien und Interessensverbänden große Anstrengungen, Transportmodelle wie CAEX [IEC62424], PandIX [PandIX] oder AutomationML [AML1] herstellerneutral bereitzustellen. Solche Modelle kommen der Vielfalt der Werkzeuge auf dem Markt entgegen. Auch kleinere Hersteller können sich durch die Umsetzung eines solchen normierten Formates in die Werkzeuglandschaft integrieren. Bei dem Ansatz der Austausch- und Transportmodelle stellen inkrementelle Änderungen und der Abgleich nach dem initialen Datenaustausch noch immer eine große Herausforderung dar.

In [BS09] stellen die Autoren einen Ansatz vor, bei dem sich herstellerneutral Werkzeuge durch die Bereitstellung einer entsprechenden Schnittstelle an eine gemeinsame Datenbasis, dem Engineering Servicebus (ESB) ankoppeln können. Das Wissen über die im ESB vorhandenen Daten und ihre Semantik muss, wie bei den Austauschformaten, wieder jedes Tool für sich mitbringen. Erschwerend kommt hinzu, dass jedes Werkzeug seine Daten in werkzeugspezifischer Art ablegen kann. Eine Verknüpfung der Daten oder eine Interpretation der Daten durch andere Werkzeuge ist daher nur schwer zu realisieren. Dieser Ansatz des integrierten Engineerings, bei dem alle Gewerke auf einen gemeinsamen Datenhaushalt zugreifen, ist vermehrt

auch bei den großen auf dem Markt befindlichen Werkzeugherstellern zu finden. So hat Siemens im Jahr 2008 mit der Übernahme der Firma innotec GmbH sein Produktportfolio um die Planungswerkzeuge der Comos-Produktgruppe erweitert [Div08]. Eine gemeinsame Datenbasis, das Siemens Teamcenter, bildet nun die Grundlage für die Planung der Elektroverkabelung sowie der Mess-, Steuer- und Regelungstechnik mit Hilfe von Comos und die Programmierung der Automatisierung sowie die Erstellung von Anlagensimulationen. Auch namenhafte Hersteller von Planungswerkzeugen erweitern ihr Produktportfolio, um die Daten durchgängig für alle Gewerke bereitstellen zu können. Im Jahr 2013 übernimmt die Friedhelm Loh Group, Muttergesellschaft der Eplan Software und Service AG, den Hersteller mechanischer CAD-Systeme Kuttig [EPL13] und den Autodesk-Hersteller Cideon [Pre13]. Über die unterlagerte ePLAN-Plattform werden somit neben R&I-Daten, Daten zur Elektroplanung, Daten für die Mess-, Steuerungs- und Regelungsplanung, der virtuelle Schaltschrankbau und nun auch mechanische Planungsdaten aufeinander abgebildet. Der Vorteil des integrierten Engineering ist, dass im Hintergrund eine gemeinsame Datenbasis die Daten aller Gewerke akkumuliert und in gewerkespezifischen Sichten zur Bearbeitung bereitstellt. Änderungen werden dadurch automatisch über alle Gewerke hinweg publiziert. Der große Nachteil dieses Ansatzes ist, dass es kleine und mittelständische Unternehmen aus dem Markt drängt, da diese keine durchgängige Werkzeugkette bieten können. Der Anforderung nach freier Werkzeugwahl wird hier nicht Genüge getan.

Ein überwiegend im universitären Umfeld im Einsatz befindlicher Ansatz für die Nutzung der Planungsdaten im weiteren Lebenszyklus der Anlage ist die Automatisierung der Automatisierung [SSE09]. Unter diesem Begriff lassen sich all jene Arbeiten zusammenfassen, die sich mit der teil- oder vollautomatisierten Durchführung des Engineeringworkflows beschäftigen. Hierzu gehören unter anderem der Einsatz von Wissensbasierten Systemen, das Regelbasierte Engineering und agentengestützte Ansätze. Schon 1998 beschäftigten sich Viswanathan et al. [Vis+98a; Vis+98b] mit dem Einsatz Wissensbasierter Systeme für die automatische Generierung von Steuerungscode für Batch-Prozesse. Anlagenneutrale Verfahrensrezepte werden dabei mit Informationen aus der Anlagenplanung zu Steuerungsrezepten verfeinert. Anlage und Rezept besitzen innerhalb des Wissensbasierten Systems eine spezielle objektorientierte Repräsentation in Grafchart. Ebenfalls auf der Basis eines Wissensbasierten Systems wird in [Güt09] die Generierung von SPS-Programmen aus den Planungsdaten beschrieben. Ausgehend von der Voraussetzung, dass strukturbezogene Code-Schnipsel als Klassen bereits im Engineeringsystem vorliegen, wird anhand der Planungsdaten der benötigte Steuerungscode durch entsprechendes Instanziieren erstellt und verknüpft. Dieser Ansatz ist insbesondere für Serienanlagen oder Anlagen mit sehr ähnlichem Equipment und Verhalten konzipiert. Es wird allerdings ein angepasster Planungsprozess verlangt, da viele Informationen aus der Formalisierten Prozessbeschreibung nach [VDI3682] gewonnen werden. Diese kommt in ihrer reinen Form aber in realen Planungsprozessen von prozesstechnischen Anlagen heutzutage selten zum Einsatz. Eine automatische Parametrierung von Bedienbildern wird in [UOS12] vorgeschlagen. Basierend auf den Informationen des R&I-Fließbildes werden vorgefertigte, standardisierte Bedienbilder mit Instanznamen, Einheiten, Grenzwerten und weiteren anlagenspezifischen Daten parametriert. Auch können in Abhängigkeit des im Fließbild angegebenen Sensor- oder Aktortyps spezifische Bedienfelder aktiviert werden. Allen gemein ist die 1-zu-1 Transformation von Daten von einem Modell in genau ein anderes Modell. Zudem sind die Ansätze

inflexibel gegenüber Änderungen in den Planungsdokumenten. Die generierten Daten müssen erneut erzeugt werden und ggf. gemachte händische Änderungen oder Erweiterungen gehen verloren.

Grüner [GE14; GWE14] schlägt ein Regelbasiertes Engineering auf Basis von Graphabfragen vor. Dieses ist deutlich flexibler im Einsatz verschiedener Modelle. Wie bei Güttel [Güt09] wird bei diesem Ansatz das Expertenwissens in einer domänenfremden Sprache als Regeln abgelegt, was zu zusätzlichen Barrieren bei der Formulierung der Zusammenhänge führt. Einen agentenbasierten Ansatz verfolgt Wagner [Wag08] mit seiner aktiven Unterstützung des Applikateurs im Engineeringprozess. Die Agenten analysieren während des Engineeringprozesses fortlaufend Abhängigkeiten zwischen den vom Applikateur eingegebenen Modelländerungen und anderen Modellen. Durch Interaktion mit dem Applikateur können die eingesetzten Agenten aufgetretene Inkonsistenzen auch dann behoben werden, wenn sie sich nicht direkt aus den in den Agenten verankerten Regeln lösen lassen. Durch die Ansätze von Grüner und Wagner wird das Spektrum der Betrachtung auf eine m-zu-n-Beziehung zwischen Modellen erweitert und ein über den initialen Modellabgleich hinausgehende Modelltransformation ermöglicht. Die Liste von Ansätzen zum modellbasierten Engineering ließe sich beliebig fortführen. Die in dieser Arbeit betrachteten Automatisierungsfunktionen als Serienprodukte können von diesen Ansätzen allerdings nicht profitieren, da die Engineeringphase bereits vollständig abgeschlossen ist, wenn das auszuwertende Modell vorliegt.

Das von Schmitz [SE06; SE08] vorgestellte Regelbasierte System bietet die Basis für ein anlagenneutrales Engineering. Die WENN-DANN-Regeln des Regelsystems sind bei diesem Ansatz Objektstrukturen im Laufzeitsystem und können vorhandene Modelle auswerten und manipulieren. Das von Schmitz vorgestellte Konzept beschränkt sich jedoch auf die Formulierung von unidirektionalen Regeln. Eine Konsistenzprüfung oder gar eine Rückführung von Informationen in das Ausgangsmodell ist daher nicht möglich. Ein weiteres Problem stellt die konzeptbedingte rechenzeitintensive Bearbeitung des Regelsatzes dar. Das in dieser Arbeit vorgestellte Konzept stellt eine Weiterentwicklung der Vorarbeiten von Schmitz [SE06; SE08] dar und ergänzt diese unter anderem um eine formale Basis und die Möglichkeit der bidirektionalen Auswertung. Als einen weiteren Ansatz für das anlagenneutrale Engineering schlägt Mersch [Mer16] die Realisierung von Automatisierungsfunktionen als Dienste vor, die auf einer gemeinsamen Modellarchitektur operieren. Die vorgeschlagenen Dienste bearbeiten konkrete Anfragen selbständig durch Erkundung und Verändern der in der Laufzeitumgebung vorliegenden Modelle. Zur ressourcenschonenden Erhaltung der systemweiten Konsistenz wird eine Beobachterschnittstelle für die einzelnen Modellinstanzen vorgeschlagen. Abhängige Modelle können sich bei einem Quellmodell registrieren und werden anschließend bei Änderungen informiert. Entsprechende Dienste können dann mit Hilfe des überlagerten Relationsmodells Abhängigkeiten zwischen den Modellen erkennen und Anpassungen am Zielmodell vornehmen. Durch die Verwendung von Diensten wird der Ansatz insbesondere für verteilte Automatisierungssysteme interessant. Auf die Realisierung der einzelnen Dienste geht Mersch nicht weiter ein.

Einen weiteren anlagenneutralen Ansatz verfolgt Quirós [Qui11] mit der Flussweganalyse. Auf Basis der im R&I-Fließbild bzw. im zugehörigen PandIX-Modell enthaltenen Daten und dem aktuellen Anlagenzustand werden Vorhersagen zu Auswirkungen von Aktorsteuerungen und

Warnungen bei unerlaubten Zuständen oder bei Leckage generiert. Hier ist die Modelltransformation zwar anlagenneutral formuliert, allerdings ist sie auf genau ein Quell- und ein Zielmodell beschränkt.

Einen ganz anderen Ansatz verfolgt Schlereth [Sch14]. Ausgehend von einer plattformunabhängigen Beschreibung der Modellzusammenhänge werden plattformspezifische Modelltransformationen z.B. zur Ausführung in automatisierungstechnischen Laufzeitumgebungen durch Higher-Order-Transformation erzeugt. Dieser Ansatz ist in Domänen interessant, wo plattformspezifisch unterschiedliche Programmiersprachen zum Einsatz kommen. Für den Einsatz in der Automatisierungstechnik ist er allerdings weniger geeignet, da alle gängigen SPSen die Sprachen der IEC 61131 und das dazugehörige Austauschformat PLCOpen XML unterstützen. Eine plattform- und sprachunabhänge Beschreibung der Modellzusammenhänge bedeutet daher einen unnötigen Mehraufwand bei der Spezifikation. Ein durchaus auch für die vorliegende Arbeit interessanter Aspekt des von Schlereth vorgestellten Ansatzes ist, das für die IEC 61131 Sprache ST beschriebene Konzept der Modelltransformation zur Laufzeit. Wie bei Schmitz [SE06; SE08] wird dabei auf unidirektional auswertbare Bausteine gesetzt, die einzelne Objekte des Quellmodells bearbeiten. Die bei Schlereth beschriebenen Regeln sind allerdings so komplex, dass sie nicht mehr durch den Domänenexperten handhabbar sind. So muss für jede Klasse des Quellmodells und jede Klasse des Zielmodells ein eigener Baustein im Transformationsmodell bereitgestellt werden. Bei einer automatischen Generierung der plattformspezifischen Transformationsregeln mag dies hinnehmbar sein, eine Wartung oder Erweiterung der Regeln ist aber nur durch Anpassung des plattformunabhängigen Transformationsmodells möglich.

Zusammenfassend lässt sich feststellen, dass bereits eine ganze Reihe von Ansätzen Modellzusammenhänge in der Automatisierungstechnik und die automatische Auflösung der daraus resultierenden Abhängigkeiten bei Modelländerungen adressieren. Der Großteil der Ansätze beschränkt sich dabei jedoch alleine auf die Modelle zur Engineeringzeit. Oftmals sind die Transformationen beschränkt auf ein spezifisches Quell- und Zielmodell. Eine bidirektionale Auswertung von Modellzusammenhängen verfolgt keiner der bisherigen Ansätze. Die Wahl spezieller Regelsprachen für die Beschreibung der Modellzusammenhänge stört bei vielen Ansätzen die Integration in den bisherigen Engineeringprozessen, da die Domänenexperten neue Sprachen und Werkzeuge erlernen müssen. Eine Lösung für die zuvor identifizierten Anforderungen an Modelltransformationen in der Automatisierungstechnik kann keiner der bisherigen Ansätze liefern.

Im Folgenden werden vielversprechende Ansätze aus der Informatik vorgestellt und anschließend in Kapitel 6 gezeigt, wie diese sich nutzen lassen, um die bisherigen Beschränkungen aufzuheben und eine effiziente, bidirektional auswertbare Modelltransformation für die Laufzeitsysteme der Automatisierungstechnik zur Verfügung zu stellen.

5 Modelltransformation

Bei der Konzeptentwicklung für eine Modelltransformation in prozessleittechnischen Laufzeitumgebungen stand die Suche nach einem geeigneten, gut erforschten und fundierten Ansatz aus dem Bereich der Informatik als Basismodell im Vordergrund. Bei dieser Suche stellten sich die bidirektionale Auswertbarkeit der Regeln sowie die inkrementelle Anwendbarkeit und die freie Werkzeugwahl schnell als Schlüsselkriterien heraus. Während diese Eigenschaften tief im Transformationsmodell verankert sind, lassen sich andere Anforderungen wie eine gute Nachvollziehbarkeit durch Protokollierung der Transformationsschritte oder die Wiederverwendbarkeit von Regeln und Regelteilen vergleichsweise einfach nachrüsten.

Das Forschungsgebiet der Modelltransformation bietet ein breites Spektrum an potentiellen Basismodellen. Eine umfangreiche und gut strukturierte Übersicht zu diesem Thema bieten Czarnecki und Helsen [CH06]. Schlussendlich kristallisierten sich Tripel-Graph-Grammatiken als geeigneter Kandidat heraus. Nicht zuletzt durch das umfangreiche Erweiterungspotential [Kön08; Kla12; Lau12; Leb+15; Leb+16, ...], erste Ansätze für eine Modelltransformation zur Laufzeit [Vog+09a; Vog+09b; VG13] und Regeln, die auf den domänenspezifischen Sprachen der korrelierenden Modelle aufbauen konnte der Ansatz überzeugen. Dieses Kapitel widmet sich daher hauptsächlich den Triple-Graph-Grammatiken und darauf aufbauenden Forschungsansätzen.

Da Triple-Graph-Grammatiken jedoch nicht für die Modelltransformation in leittechnischen Laufzeitumgebungen entwickelt wurden, liegt es in der Natur der Dinge, dass nicht alle gestellten Anforderungen in Gänze mit diesem Ansatz erfüllt werden können. Ein kleiner Einblick in weitere Transformationsmodelle, die diese offenen Punkte thematisieren, schließt das Kapitel ab.

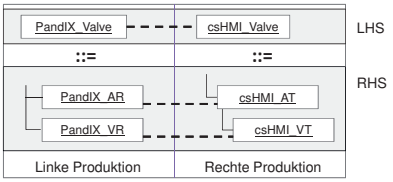
5.1 Tripel-Graph-Grammatiken

Die von Schürr entwickelten Tripel-Graph-Grammatiken (TGG) [Sch95], sind eine Weiterentwicklung der Paar-Grammatiken nach Pratt [Pra71]. Beide Ansätze gehen davon aus, dass für zwei konsistente Modellinstanzen M_1 und M_2 korrelierender Modelle gilt:

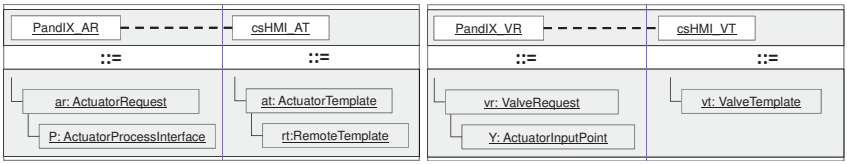
- für jede valide Änderung von M_1 gibt es mindestens eine valide Änderung von M_2, die die beiden Modellinstanzen konsistent hält
- für jede valide Änderung von M_2 gibt es mindestens eine entsprechende konsistenzerhaltende Änderung in M_1

Die Regeln der beiden Ansätze beschreiben solche konsistenzerhaltenden simultanen Änderungen von Modellen. Pratt erzeugt in seinem Ansatz zwei, mittels kontextfreier Grammatiken beschriebene, konsistente Modelle. Seine Regeln für die Modelltransformation setzen Paare von Produktionen in Beziehung (vgl. Abbildung 5.1). Jeder Zwischenschritt erzeugt durch Regelanwendung parallel zwei Graphen; einen in der linken und einen in der rechten Domäne. Diese enthalten Nichterminalsymbole, die miteinander über eine 1-zu-1 Nichtterminal-Paarung verbunden sind. Durch die Anwendung einer Regel wird beim simultanen Auffinden des LHS-Nichterminalsymbols im jeweiligen Modell dieses durch das entsprechende RHS-Pattern ersetzt. Die beiden Modelle werden dadurch simultan erzeugt. Jeder Zwischenschritt ersetzt genau ein Nichtterminal-Paar. Das finale Graphen-Paar enthält keine Nichtterminale und somit auch keine Nichtterminal-Paarungen mehr.

Ein Hauptproblem bei dieser Art der Regeln ist die fehlende Nachvollziehbarkeit. Nur zum Zeitpunkt der Transformation ist klar, welche Teile der beiden Modelle durch simultanes Erzeugen einander zuzuordnen sind. Eine inkrementelle Änderung von Modellen ist daher mit Paar-Grammatiken schwierig. Durch die Verwendung von kontextfreien Sprachen wird dieser Effekt noch verstärkt, da die LHS ihrer Produktionen jeweils Nichtterminale sind. Ein bestehendes valides Modell-Paar enthält aber keine Nichtterminale mehr, die durch die Produktionen ersetzt werden können. Eine starke Einschränkung für den Einsatz von Paar-Grammatiken stellen die 1-zu-1-Beziehungen zwischen den Nichtterminalen dar.



(a) Initiale Ventil-Regel



(b) Aktuator-Regel (c) Ventil-Regel

Abbildung 5.1: Paar-Grammatik zur simultanen Erstellung von PandIX und csHIM

Beispiel 5.1 *Die Regeln aus Abbildung 5.1 bilden einen Ausschnitt aus der Paar-Grammatik für die simultane Erstellung eines PandIX-Modells und einer Bedienoberfläche mittels ACPLT/cs-HMI. Nichtterminal-Paare sind durch gestrichelte Linien gekennzeichnet. Bei Anwendung der Regeln werden die Nichtterminale der LHS im Graph der jeweiligen Domäne gesucht und durch RHS ersetzt. Die starre 1-zu-1 Beziehung bei den Nichtterminal-Paaren führt dazu, dass Erweiterungen, die nur eine der beiden Modelle betreffen schwieriger zu realisieren sind. So muss eine Erweiterung der Bedienoberfläche um ein Detailbild entweder mit der Ersetzung in*

5.1c erfolgen oder in der initialen Ventil-Regel muss auf beiden Seiten ein weiterer Nichtterminalknoten erzeugt werden. Wobei bei der Auswertung dieses Nichtterminal-Paares nur auf der ACPLT/csHMI-Seite Elemente erzeugt werden.

Drei Erweiterungen sind daher nötig, die Verwendung von kontextsensitiven Sprachen, die Protokollierung der Modellbeziehungen über den Transformationsprozess hinaus und die Erweiterung auf n-zu-m-Beziehungen zwischen den beiden Domänen. Mit der Weiterentwicklung von Paar-Grammatiken zu Tripel-Graph-Grammatiken konnte Schürr diese Aufgaben erfolgreich lösen. Zum einen führt Schürr einen dritten, ebenfalls simultan erzeugten Korrespondenzgraph ein. Die Knoten dieses Graphs dokumentieren den Transformationsverlauf für den gesamten Lebenszyklus der beiden Modellinstanzen. Zudem verwendet Schürr als Basis seiner Regeln kontextsensitive Graph-Grammatiken. Diese erlauben die Referenz auf terminale Kontext-Elemente und ermöglichen so eine einfachere inkrementelle Modellentwicklung.

Durch Tripel-Graph-Grammatiken werden immer drei Teilgraphen parallel entwickelt: das Quellmodell, das Zielmodell und der Korrespondenzgraph. Abbildung 5.2 zeigt beispielhaft

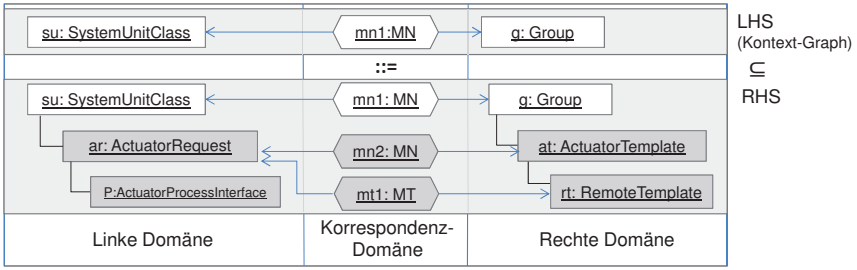


Abbildung 5.2: TGG-Regel zur simultanen Erstellung von PandIX und csHIM

ein TGG-Produktion für das Hinzufügen einer PLT-Stelle. Die Relation zwischen Elementen der linken und der rechten Domäne werden dabei mit Hilfe sogenannter Korrespondenzlinks beschrieben:

Definition 5.1 (Korrespondenzlink) *Seien $\Omega_{VR}, \Omega_{VL}, \Omega_{VK}$ drei endliche Alphabete terminaler Knotenbeschriftung und Ω_{ER}, Ω_{EL} zwei Alphabete terminaler Kantenbeschriftungen. Seien außerdem LD, KD, RD Graphen mit Knoten- und Kantenbeschriftungen aus den entsprechenden Domänen. Es gelte daher für alle Knoten v und alle Kanten e:*

$$\begin{aligned}
\forall v \in LD: \quad & label(v) \quad \in \Omega_{VL} \\
\forall e \in LD: \quad & label(e) \quad \in \Omega_{EL} \\
\forall v \in RD: \quad & label(v) \quad \in \Omega_{VR} \\
\forall e \in RD: \quad & label(e) \quad \in \Omega_{ER} \\
\forall v \in KD: \quad & label(v) \quad \in \Omega_{VK}
\end{aligned}$$

Außerdem gelte:

$$\nexists e \in KD$$

Die Knoten von KD werden Korrespondenzknoten genannt. Die Funktion $map(v_K)$ liefert zu jedem Korrespondenzknoten $v_K \in KD$ ein Tupel (γ_l, γ_r), wobei γ_l eine nichtleere Menge an Knoten und Kanten aus LD und γ_r eine nichtleere Menge an Knoten und Kanten aus RD sind.

Eine Menge $\{L_{KD} | L_{KD} = (\gamma_l, v_K, \gamma_r), map(v_K) = (\gamma_l, \gamma_r)\}$ mit folgenden Eigenschaften:

$$x \in \gamma_l \quad \Rightarrow \quad x \in LD$$
$$x \in \gamma_r \quad \Rightarrow \quad x \in RD$$

wird Korrespondenzlinks zwischen LD und RD genannt.

Die Definition zeigt, dass Korrespondenzlinks sowohl Kanten als auch Knoten der beiden Grammatiken miteinander in Relation setzten können. Korrespondenzlinks beschreiben das Mapping zwischen linker und rechter Domäne durch die Formulierung von Bedingungen (vgl. Abbildung 5.3). Zudem erzeugen sie durch Anlegen der entsprechenden Objekte aus der Korrespondenzdomäne eine Dokumentation des Transformationsverlaufes. Sie sind zentraler Bestandteil der TGG-Produktionen, deren genereller Aufbau in Abbildung 5.4 dargestellt ist.

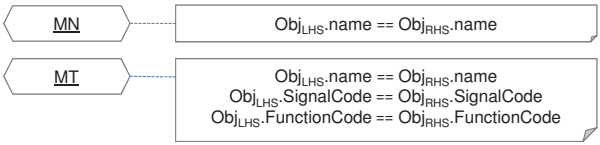


Abbildung 5.3: Korrespondenzobjekte der TGG-Produktion aus Abbildung 5.2

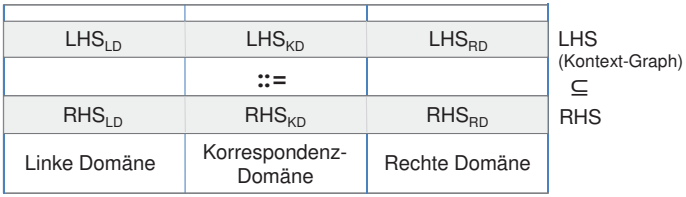


Abbildung 5.4: Aufbau einer TGG-Produktion

Definition 5.2 (Tripel-Graph-Grammatik)
Eine Tripel-Graph-Grammatik TGG = ($\Omega_E, \Omega_V, \Omega_N, S, P_{TGG}, map$) ist eine Graph-Grammatik mit folgenden speziellen Eigenschaften:

- *drei endliche Alphabete $\Omega_{VR}, \Omega_{VL}, \Omega_{VK}$ terminaler Knotenbeschriftungen mit*
$$\Omega_{VR} \cup \Omega_{VL} \cup \Omega_{VK} = \Omega_V$$
- *zwei endliche Alphabete Ω_{ER}, Ω_{EL} terminaler Kantenbeschriftungen mit*
$$\Omega_{ER} \cup \Omega_{EL} = \Omega_E$$
- *dem Startsymbol $S = \emptyset$*
- *eine endliche Menge von TGG-Produktionen P_{TGG}*

Definition 5.3 (TGG-Produktion) *Seien $\Omega_{VR}, \Omega_{VL}, \Omega_{VK}$ drei disjunkte endliche Alphabete terminaler Knotenbeschriftung und Ω_{ER}, Ω_{EL} zwei disjunkte endliche Alphabete terminaler Kantenbeschriftungen. Seien außerdem LHS_{LD}, RHS_{LD}, LHS_{RD}, RHS_{RD} Graphen mit Knoten- und Kantenbeschriftungen aus den entsprechenden Domänen:*

$$\begin{aligned}
\forall v &\in LHS_{LD} \cup RHS_{LD}: \quad label(v) \in \Omega_{VL} \\
\forall e &\in LHS_{LD} \cup RHS_{LD}: \quad label(e) \in \Omega_{EL} \\
\forall v &\in LHS_{RD} \cup RHS_{RD}: \quad label(v) \in \Omega_{VR} \\
\forall e &\in LHS_{RD} \cup RHS_{RD}: \quad label(e) \in \Omega_{ER} \\
\forall v &\in LHS_{KD} \cup RHS_{KD}: \quad label(v) \in \Omega_{VK} \\
\nexists e &\in LHS_{KD} \cup RHS_{KD}
\end{aligned}$$

Eine TGG-Produktion $P_{TGG} = (LHS, RHS, L_K)$ ist eine graphbasierte Produktion (LHS, RHS) für die gilt:

$$\begin{aligned}
LHS &= LHS_{LD} \cup LHS_{KD} \cup LHS_{RD} \\
RHS &= RHS_{LD} \cup RHS_{KD} \cup RHS_{RD}
\end{aligned}$$

$$\begin{aligned}
LHS_{LD} &\subseteq RHS_{LD} \\
LHS_{KD} &\subseteq RHS_{KD} \\
LHS_{RD} &\subseteq RHS_{RD}.
\end{aligned}$$

Für die Menge L_K von Korrespondenzlinks gelte zudem:

$$\begin{aligned}
\forall v_K \in LHS_{KD}. \forall l \in L_K. l = (\omega_l, v_K, \omega_r). \quad & x \in \omega_l \Rightarrow x \in LHS_{LD} \wedge \\
& x \in \omega_r \Rightarrow x \in LHS_{RD} \\
\forall v_K \in RHS_{KD}. \forall l \in L_K. l = (\omega_l, v_K, \omega_r). \quad & x \in \omega_l \Rightarrow x \in RHS_{LD} \wedge \\
& x \in \omega_r \Rightarrow x \in RHS_{RD}
\end{aligned}$$

Die Teilgraphen $LHS_{LD}, LHS_{KD}, LHS_{RD}, RHS_{LD}, RHS_{KD}, RHS_{RD}$ werden im Folgenden als Pattern der TGG-Produktion bezeichnet, die in drei Teilproduktionen

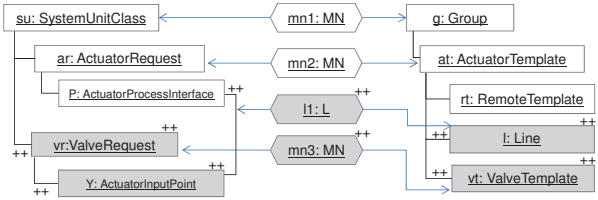
$$\begin{aligned}
LHS_{LD} &:= RHS_{LD} \\
LHS_{KD} &:= RHS_{KD} \text{ und} \\
LHS_{RD} &:= RHS_{RD}
\end{aligned}$$

organisiert sind.

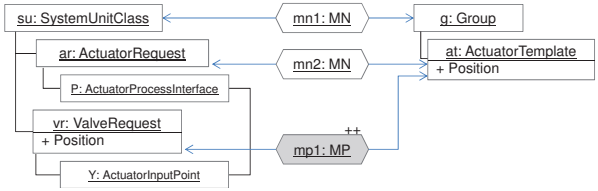
Königs [Kön08] führt in seinen Arbeiten eine verkürzte Schreibweise der TGG-Produktion ein, die auch in dieser Arbeit Verwendung findet. Dabei werden die Pattern der LHS und RHS aufeinander abgebildet. Elemente, die durch die Regel hinzugefügt werden, also nur in der RHS vorkommen, werden mit „++“ gekennzeichnet. Abbildung 5.5a stellt die verkürzte Form der Regel aus Abbildung 5.2 dar. Abbildung 5.5b demonstriert, wie durch die Verwendung von Kontextelementen und durch die Einführung des Korrespondenzgraphen der Aktor jeweils zur

entsprechenden Instanz der PLT-Stelle hinzugefügt werden kann. Selbiges gilt für die Positionierung der Elemente in Abbildung 5.5c. Die LHS der Regeln (helle Elemente) bilden nicht nur die bereits erstellte Elemente ab, sondern auch die zuvor angelegten und nun erwarteten Modellzusammenhänge. Das den Produktionen einer TGG zu Grunde liegende Transformationsmodell für zwei korrelierende Modelle wird kompakt durch ein TGG-Schema abgebildet. Dieses beschreibt die Zusammenhänge der beiden Modelle als Ganzes. Abbildung 5.5d zeigt den Ausschnitt aus dem TGG-Schema für das Beispielszenario.

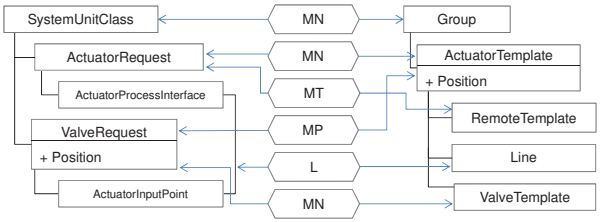
(a) Anlegen der PLT-Stelle



(b) Hinzufügen des Aktors



(c) Positionieren der Elemente



(d) Ausschnitt aus dem TGG-Schema

Abbildung 5.5: TGG-Produktionen und TGG-Schemata

Um die zu suchenden oder zu erstellenden Strukturen noch feingranularer beschreiben zu können, führt Königs [Kön08] die Attributwertweitergabe ein. Dadurch können Elemente mit Attributen versehen werden, die wiederum Attribute der Klassen und Assoziationen in den Modellen repräsentieren. Für das Anwendungsszenario ermöglicht dies unter anderem die Übertragung

der Position vom Quellmodell in das Zielmodell (vgl. Abbildung 5.5c)[1]. Auch Informationen aus schwach typisierten Modellen können somit für die Modelltransformation nutzbar gemacht werden.

Neben der Attributwertweitergabe beschreibt Königs in seiner Arbeit eine Reihe weiterer Erweiterungen von Tripel-Graph-Grammatiken, die auch für diese Arbeit von Interesse sind:

NACs Soll ein bestimmtes Element explizit nicht vorhanden sein, so kann dies durch sogenannte NACs (engl. negativ application condition) beschrieben werden. Ein solches Element wird durch ein Kreuz gekennzeichnet.

Optionale Elemente Optionale Elemente oder Teilgraphen dienen dazu, zwei Regeln zusammenzufassen, die bis auf das Vorhandensein des optionalen Elements/Teilgraphen identisch sind.

Optionale Erstellung Insbesondere für inkrementelle Modellentwicklung kann es wichtig sein, Elemente anzulegen, wenn sie noch nicht im Zielmodell vorhanden sind und unverändert zu lassen, wenn sie schon existieren. Dies ist wiederum eine Verschmelzung zweier separater Regeln, die das Vorhandensein des Elements als Kontext haben oder nicht.

Neuere Arbeiten zu Tripel-Graph-Grammatiken liefern weitere Ansätze zur Verbesserung der Les- und Wartbarkeit von TGG-Produktionen. So ermöglicht das Vererbungskonzept von Anjorin, [Anj14; Anj+15] Teile der Produktion durch Dekomposition als wiederverwendbare Oberklasse auszugliedern. Abbildung 5.6 zeigt die Dekomposition der Produktion aus Abbildung 5.5b. Die so geschaffene Oberklasse lässt sich für die Positionierung der Elemente aus Abbildung 5.5c wiederverwenden.

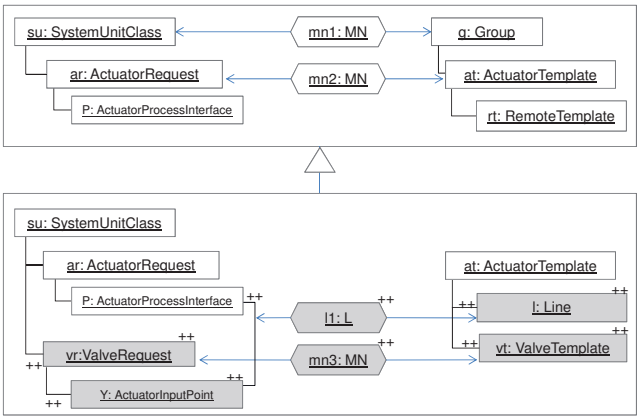


Abbildung 5.6: Vererbungskonzept für Tripel-Graph-Grammatiken

[1] Vereinfachend wird an dieser Stelle davon abstrahiert, dass Quell- und Zielmodell unterschiedliche Darstellungen der Positionsangaben verwenden.

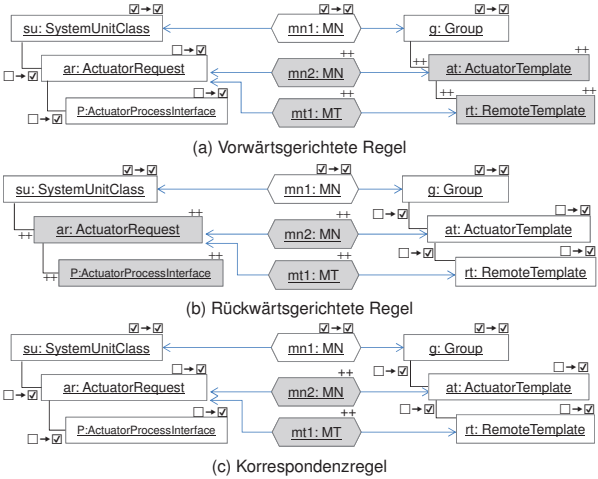
Die Anwendung von Tripel-Graph-Grammatiken auf nicht-bijektive Modellzusammenhänge steht bei der Amalgamierung [Leb+15; Leb+16] sowie dem Einsatz von Leerlauf-Regeln [Leb+16] im Vordergrund. Die Amalgamierung erlaubt, 1-zu-n Zusammenhänge zwischen Elementen des Quell- und Zielmodells zu beschreiben. So können zum Beispiel unterschiedlich detaillierte Ansichten einer Anlage (Ansicht je Reaktor, Übersicht Teilanlage, Übersicht gesamte Anlage, ...) im HMI existieren. Wird die Erstellung der einzelnen Ansichten optional bzw. durch einen Parameter in den Planungsdokumenten festgelegt, ist zum Zeitpunkt der Regelbeschreibung nicht klar, wie viele Ansichten bei der späteren Regelanwendung existieren. Dennoch soll in jeder Ansicht ein Navigationselement zu den jeweilig anderen Ansichten generiert werden. Um diese 1-zu-n Beziehung beschreiben zu können, werden mehrere konkrete TGG-Regeln zusammengefasst und auf Basis einer Kern-Regel beschrieben. Im aktuellen Beispiel umfasst die Kern-Regel zunächst die Erstellung einer Ansicht ohne Navigationselemente. Die Multi-Regeln erweitern diesen Kern um das Vorhandensein keiner/einer/zwei/... weiterer Ansichten zu denen Navigationselemente erstellt werden müssen. Die Kern-Regel enthält den Kontext-Teil, den alle zusammengefassten Regeln gemeinsam haben. Zusammen mit den Multi-Regeln, die die unterschiedlichen Kombinationen beschreiben, ergibt die Kern-Regel ein Interaktions-Schema. Die konkrete für ein Quell-/Zielmodell resultierende TGG-Produktion wird multiamalgamierte TGG-Regel genannt. Für ein konkretes Planungsdokument mit drei parametrierten Ansichten erstellt die multiamalgamierte TGG-Regel diese drei Ansichten mit jeweils zwei Navigationselementen zu den anderen beiden Ansichten.

Leerlaufregeln erlauben Modelländerungen im Quell- oder Zielmodell, die keinen Einfluss auf das jeweils andere Modell haben. So hat ein Kommentar im PandIX keine Entsprechung im HMI und Hintergrundbilder oder Markierungspfeile des HMIs finden sich nicht im PandIX wieder. Quell-Leerlauf-Regeln besitzen daher nur eine LD-Teilproduktion, Ziel-Leerlauf-Regeln nur eine RD-Teilproduktion.

5.1.1 Operationale Regeln

Die im vorangegangenen Abschnitt vorgestellten bidirektionalen deklarativen TGG-Produktionen können dafür genutzt werden, zwei Modelle simultan zu erstellen. Ein Blick auf die in Kapitel 4.2 identifizierten Anforderungen zeigt, dass dies nicht den Kern der Modelltransformation in der Automatisierungstechnik trifft. Schürr [Sch95] führt dazu eine Übersetzung der bidirektionalen deklarativen Regeln zu operationalen Regeln ein. Je nach Anwendungsfall werden dabei unidirektionale vorwärts- oder rückwärtsgerichtete Regeln erzeugt. Vorwärtsgerichtete Regeln werden genutzt, um Informationen aus dem Quellmodell in das Zielmodell zu übertragen. Auf der anderen Seite können die Informationen aus dem Zielmodell mittels rückwärtsgerichteten Regeln in das Quellmodell zurückpropagiert werden. Bei der Untersuchung von Quell- und Zielmodell auf Widerspruchsfreiheit werden bidirektionale Korrespondenzregeln eingesetzt. Die Darstellung der operationalen Regeln ist an die der TGG-Produktionen angelehnt. Wie bei Lauder [Lau12] werden Elemente, die im Quellmodell hinzugekommen sind mit $\square \rightarrow \boxtimes$ gekennzeichnet und Elemente, die bereits als Kontext bestanden mit $\boxtimes \rightarrow \boxtimes$.

Die plattformunabhängigen operationalen Regeln werden in einem weiteren Schritt kompiliert und dadurch in plattformspezifische imperative Anweisungen zur Modelltransformation über-



(a) Vorwärtsgerichtete Regel

(b) Rückwärtsgerichtete Regel

(c) Korrespondenzregel

Abbildung 5.7: Aus Abbildung 5.5a abgeleitete operationale Regeln

setzt. Dieser Ansatz ermöglicht es, bei der Übersetzung zu optimieren und eine effiziente Modelltransformation bereitzustellen. Einen alternativen Ansatz bietet der TGG-Interpreter [KW07; GK07]. Hierbei werden keine operationalen Regeln erzeugt und es erfolgt auch keine Übersetzung in imperative Anweisungen. Stattdessen bildet der TGG-Interpreter die plattformspezifische zentrale Einheit, die die bidirektionalen TGG-Regeln auswertet und anwendet. Eine Optimierung der Modelltransformation ist durch den fehlenden Übersetzungsschritt bei diesem Ansatz nur sehr eingeschränkt möglich.

Beiden Ansätzen gemein ist, dass bei einer Vorwärtstransformation[2] eine Regel nur dann angewendet werden kann, wenn eine Entsprechung für RHS_{LD} in der linken Domäne gefunden wurde und alle Kontextelemente bereits übersetzt wurden. Zudem muss in der rechten Domäne eine Entsprechung für LHS_{RD} zu finden sein. Sind all diese Anforderungen erfüllt, so können die neuen Elemente in der rechten Domäne erzeugt werden. Die Objekte der linken Domäne, die durch die Regel auf Nicht-Kontextelemente abgebildet wurden, werden anschließend als übersetzt markiert und können für weitere Regelanwendungen als Kontextelemente genutzt werden. Sie dürfen jedoch im weiteren Verlauf der Transformation nicht durch eine weitere Regel als Nicht-Kontextelemente verwendet werden, da jedes Objekt nur einmal übersetzt werden darf.

5.1.2 Kontrollalgorithmus

Die Regeln dienen als Basis für den TGG-Kontrollalgorithmus. Königs unterscheidet beim Kontrollalgorithmus zwischen der Strategie und dem Scheduling. Während die Strategie den Teil

[2] Äquivalentes gilt für eine Rückwärtstransformation sowie eine Konsistenzalanyse.

des Modells identifiziert, auf den eine Regel als nächstes angewendet werden soll, beschreibt das Scheduling in welcher Reihenfolge die Regeln ausgewertet werden. Die Bestimmung der Reihenfolge kann unabhängig vom Nutzer geschehen (implizites Scheduling) oder unter Einflussnahme des Nutzers (explizites Scheduling). Zweiteres kann zum Beispiel durch die Vergabe von Prioritäten oder durch direktes Einbeziehen des Nutzers in den Übersetzungsprozess geschehen. Sind Kontrollstrukturen und Regeln strikt voneinander getrennt, spricht man von externem Scheduling. Im Gegensatz dazu kann beim internen Scheduling die Auswertung von Regeln auch durch andere Regeln angestoßen werden.

5.1.3 Modelltransformation zur Laufzeit

Unter dem Begriff *models@runtime* untersuchen Wissenschaftler seit einigen Jahren die Möglichkeit, Modelltransformation nicht nur im Engineering-Prozess sondern auch im Laufzeitsystem anwendbar zu machen. An dieser Stelle sollen insbesondere die Arbeiten von Vogel et al. [Vog+09a; Vog+09b; VG13] Erwähnung finden, da sie sich bei der Umsetzung der Modelltransformation auf die in dieser Arbeit fokussierten Tripel-Graph-Grammatiken stützen (vgl. Abbildung 5.8). Die Modelltransformation findet bei diesem Ansatz außerhalb der automatisierungstechnischen Laufzeitumgebung statt. Zudem sind Laufzeitsystem und Engineeringsystem strikt voneinander getrennt. Dies hat zur Folge, dass ein aufwändiger Synchronisationsmechanismus zwischen Engineering-, Transformations- und Laufzeitsystem implementiert werden muss. Das Quellmodell wird dabei um eine Sensor-/Effektor-Schnittstelle erweitert. Über Sensoren werden Änderungen am Quellmodell nach außen bekanntgegeben. Änderungen am Zielmodell können über Effektoren in das Quellmodell zurückgespielt werden. Insbesondere die Quellmodelle müssen dazu speziell angepasst und um die Sensor-/Effektor-Schnittstelle erweitert werden. Um Änderungen effizient erkennen zu können, schlagen Vogel et al. einen Ereignismechanismus vor. Beim Auftreten eines Ereignisses, also einer Änderung am Quell- oder Zielmodell wird zunächst überprüft, ob die Modelle weiterhin konsistent sind. Ist dies nicht der Fall, wird eine Modellanpassung vorgenommen.

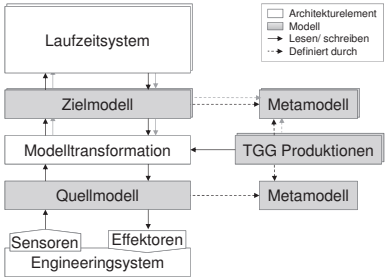


Abbildung 5.8: Architektur von *models@runtime* nach [Vog+09b]

Unabhängig vom konkreten Ansatz beschreiben Bencomo et al. [Aßm+14] die Fähigkeit zur Introspektion und Selbstmodifikation als einen Kernpunkt beim Einsatz von Modellen für die Modelltransformation zur Laufzeit.

5.2 Alternative Ansätze

In vorangegangenen Untersuchungen [KQ11; SK12; Kra+12] hat sich gezeigt, dass Tripel-Graph-Grammatiken eine gute Grundlage für die Modelltransformation in der Automatisierungstechnik bieten. Insbesondere die Verwendung von Graphen als Basis für die Produktionen kommt dem Einsatz in der Automatisierungstechnik sehr entgegen. Die oftmals als UML-Diagramme und/oder Grammatiken vorliegenden Automatisierungsmodelle können durch ein Tripel-Graph-Schema direkt miteinander in Beziehung gesetzt werden. Die daraus abgeleiteten Produktionen sind für den Automatisierungstechniker gut verständlich. Auch die in Kapitel 4 identifizierten Anforderungen an Modelltransformationen für die Automatisierungstechnik können durch Tripel-Graph-Grammatiken bereits weitestgehend erfüllt werden. So kommt der TGG-Ansatz ohne Anpassungen an den Quell- und Zielmodellen aus. Dies ist eine wichtige Voraussetzung dafür, dass die im Lebenszyklus der Anlage verwendeten Werkzeuge ohne nennenswerte Einschränkungen frei gewählt werden können, sofern sie ihre Daten in einem allgemein lesbaren Austauschformat zur Verfügung stellen. Zudem stellen Tripel-Graph-Grammatiken basierend auf einem Regelsatz ein breites Spektrum von Einsatzmöglichkeiten von Batch-Transformation über Rückwärtspropagieren, Konsistenzanalyse und inkrementelle Modelländerungen zur Verfügung.

An anderen Stellen zeigen sich jedoch auch die Schwächen des TGG-Ansatzes. Mit dem Dekompositionsansatz von Anjorin ist zwar ein erstes Konzept zur Wiederverwendung von Regelbestandteilen geschaffen, weiterführende Möglichkeiten der Wiederverwendung stehen jedoch noch nicht zur Verfügung. Auch ist die Mächtigkeit von Tripel-Graph-Grammatiken gegenüber anderen Ansätzen noch verhältnismäßig eingeschränkt. Die Abbildbarkeit von nicht-bijektiven Zusammenhängen mittels Leerlauf-Regeln und multiamalgamierten Regeln bieten zwar auch hier schon eine gute Basis, allerdings fehlen unter anderem praktikable Ansätze für die Modelltransformation zwischen mehrere Quell- und Zielmodellen. Wiederverwendbarkeit von Regelteilen und multiple Quell- und Zielmodelle sind wichtige Voraussetzungen, um Tripel-Graph-Grammatiken fit für reale Einsatzszenarien zu machen. Im Folgenden sollen zwei Transformationsansätze angerissen werden, die in ihrem Rahmen Lösungen für diese offenen Punkte anbieten.

Das *VIsual Automated model TRAnsformations Framework* - kurz VIATRA [Ber+15; RL07] ist eine Graftransformationserweiterung für Eclipse. Es ermöglicht eine ereignisgetriebene Modelltransformation, die auf Änderungen im Modell direkt reagiert. Zwei Arten von Ereignissen werden dabei unterschieden; explizite Aufrufe der Regelbearbeitung, sogenannte kontrollierte Ereignisse und beobachtete Ereignisse wie zum Beispiel Modelländerungen. VIATRA bietet ein breites Spektrum an Wiederverwendungskonzepten bei der Gestaltung der Regeln. So können Regeln die Bearbeitung anderer Regeln anstoßen. Dies ermöglicht ein Kaskadieren von Regeln und ein Auslagern mehrfach benötigter Regelbestandteile in separate Regeln. Eine Regel

kann dabei nicht nur andere Regeln sondern auch die eigene Bearbeitung anstoßen, wodurch eine rekursive Regelbeschreibung ermöglicht wird. Auch erlaubt VIATRA die Definition von alternativen Regelbestandteilen, die auf Basis einer booleschen Bedingung nach dem Schema WENN-DANN-SONST angewendet werden. Hierdurch lassen sich, ähnlich einer Amalgamierung von TGG-Produktionen, zwei nahezu identische Regeln zu einer zusammenfassen. Neben der reinen Wiederverwendbarkeit spielt hier auch die Zeitersparnis bei der Regelauswertung eine große Rolle, da der Regelteil vor der Alternative nur einmal gebunden werden muss und anschließen sowohl für die WENN-Variante als auch für die SONST-Variante zur Verfügung steht. Die Bereitstellung von Graphmustern stellt eine weitere Möglichkeit zur Wiederverwendung dar. Graphmuster sind prototypische Ausschnitte aus einem Instanzmodell, die bei der Regelanwendung injektiv auf Modellstrukturen abgebildet werden, d.h. die Elemente des Musters werden auf disjunkte Elemente im Modell abgebildet. Ist eine solche Abbildung erfolgreich, werden die Modellobjekte für die Regelanwendung gebunden und es kann innerhalb der Regel darauf zugegriffen werden. Besonders effizient sind die Graphmuster, wenn sie in mehreren Regeln zur Anwendung kommen oder wenn sie als Teile in anderen Mustern angewendet werden. Auch bei der Beschreibung von Mustern sind Rekursion und Alternativen erlaubt.

Auch die OMG stellt mit der sogenannten QVT - Query/View/Control Sprachen für die Modelltransformation zur Verfügung [MOF-QVC]. QVT besteht aus den deklarativen Sprachen QVT-Relations und QVT-Core, sowie der imperativen Sprache QVT-Operational. QVT-Core stellt einen kleinen, sehr kompakten Sprachumfang für die Beschreibung von Modellzusammenhängen zur Verfügung. Der verringerte Sprachumfang vereinfacht die spezifikationsgetreue Realisierung von QVT-Core. Dies geschieht jedoch zu Lasten der Benutzerfreundlichkeit, da die Beschreibung von Modellzusammenhängen vergleichsweise kompliziert ist. Für den Entwickler einer QVT-basierten Modelltransformation spielt QVT-Relations mit seinem benutzerfreundlicheren Sprachumfang die größer Rolle. Hiermit lassen sich Relationen zwischen Modellen beschreiben. Diese können mit imperativen Elemente aus QVT-Operational angereicht werden. Für die Anwendung einer Modelltransformation, werden die mit Hilfe von QVT-Relations beschriebenen Modellzusammenhänge zunächst in QVT-Core übersetzt. QVT-Core bildet daher den „Bytecode“ für die „Hochsprache“ QVT-Relations. Für diese Arbeit von besonderem Interesse sind die Unterstützung von multiplen Quell- und Zielmodellen, sowie die Möglichkeit der Inplace-Transformation. Im Bereich der Wiederverwendbarkeit sticht insbesondere das Templatekonzept von QVT hervor. Templates beschreiben einfache oder komplexe Muster im Quell- oder Zielmodell und versehen diese mit einem Namen. Dadurch lassen sich diese Muster in verschiedenen Relationen über das Schlüsselwort „Domain“ und den Namen des Templates wiederverwenden.

Beide Ansätze, sowohl VIATRA als auch QVT erlauben das Löschen von Modellelementen. Dies ist bisher bei Tripel-Graph-Grammatiken noch nicht in zufriedenstellender Weise möglich. Zwar laufen hierzu erste Untersuchungen, jedoch sind die bisher dazu veröffentlichten Ergebnisse noch sehr vage [KW07], es fehlen Beweise zum Erhalt der formalen Eigenschaft von Tripel-Graph-Grammatiken und es kommt zu Performanceproblemen [GPR11].

6 ACPLT/MT - Modelltransformation für die Automatisierungstechnik

In den vorangegangenen Kapiteln wurden das Potential sowie die Voraussetzungen für eine Modelltransformation in der Automatisierungstechnik beleuchtet und gezeigt, dass Tripel-Graph-Grammatiken den an eine solche Transformation gestellten Anforderungen besonders gut gerecht werden. Um Tripel-Graph-Grammatiken innerhalb prozessleittechnischen Laufzeitumgebungen nutzbar zu machen, bedarf es allerdings einer Integration in die gängigen Sprachen der Automatisierungstechnik. Die ACPLT-Modelltransformation (ACPLT/MT) bietet diese Integration. In diesem Kapitel wird zunächst ein Überblick über die getroffenen Designentscheidungen gegeben, die die Entwicklung von ACPLT/MT beeinflusst haben. Das daraus entstandene plattformunabhängige Modell sowie seine plattformspezifische Realisierung auf Basis von ACPLT/FB werden anschließend näher beleuchtet.

6.1 Grundlegende Design-Entscheidungen

In der Automatisierung von Anlagen kommen meist Speicherprogrammierbare Steuerungen (kurz SPS) zum Einsatz. Diese zeichnen sich durch vergleichsweise niedrige Speicher- und Performance-Werte aus. Zudem stehen zur Programmierung von SPSen ausschließlich die Sprachen der IEC 61131-3 [IEC61131] zur Verfügung. Insbesondere die streng zyklische Bearbeitung einzelner Softwarekomponenten unterscheiden diese Sprachen von gängigen Programmiersprachen. Taktzyklen von 10ms bis runter zu 10μs (in der Fertigungstechnik) sowie die von SPSen erwartete Echtzeitfähigkeit erfordern eine modulare und unterbrechbare Programmierung. Um die zu realisierende Automatisierungsfunktion auf einer SPS zur Verfügung zu stellen, muss das zu entwickelnde Konzept daher eine Portierung von Tripel-Graph-Grammatiken in die Welt der IEC 61131-3 beinhalten.

Der gravierendste Unterschied von ACPLT/MT zu anderen TGG-Ansätzen zeigt sich im Aufbau und der Anwendung der TGG-Produktionen. Neben der Verwendung der verkürzten Schreibweise von Tripel-Graph-Grammatiken setzt ACPLT/MT auf sogenannte aktivierbare Regeln. Diese werden, weder wie beim Standardansatz in operationale Regeln übersetzt, noch werden die Produktionen mittels Interpreter ausgewertet. Stattdessen sind die einzelnen Elemente einer ACPLT/MT-Regel Objekte in der SPS und enthalten imperativen Code, der sie befähigt bestimmte Teilaufgaben der Transformation wie das Suchen oder Erstellen von Objekten und Links selbstständig durchzuführen. Um dies realisieren zu können, sind die Elemente der linken und rechten Domäne, anders als bei Tripel-Graph-Grammatiken üblich, keine Elemente aus den jeweiligen domänenspezifischen Sprachen, sondern modellneutrale Repräsentanten,

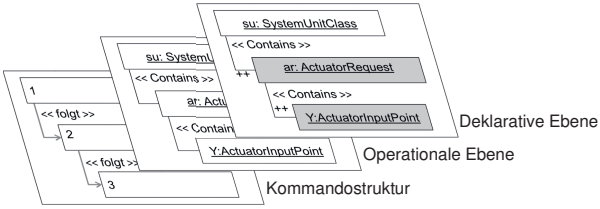


Abbildung 6.1: Funktionsebenen eines ACPLT/MT-Pattern

die mit den erwarteten Eigenschaften, wie Klasse, vorhandene Attribute oder Verlinkungen parametriert werden können. Diese schwache Typisierung über Repräsentanten ermöglicht es, dass die Repräsentanten Code zur Durchführung der Modelltransformation enthalten.

ACPLT/MT-Regeln liegen zunächst als passive Objektstrukturen vor, werden also nicht zyklisch ausgeführt. In diesem passiven Zustand werden die enthaltenen imperativen Anweisungen nicht ausgeführt und die MT-Regel steht als bidirektionale deklarative Regel zur Verfügung. Wird eine ACPLT/MT-Regel aktiviert, wird sie in die zyklische Bearbeitung eingebunden und dadurch zur operativen Regel. Dieses Design hat den Vorteil, dass keine operationalen Kopien der deklarativen Regeln erstellt werden müssen. Auch ein Interpreter wird hierbei nicht benötigt. Die Aktivierung einer Regel erfolgt durch Übergabe eines Richtungsparameters und eines Aktivierungskommandos. Ersteres legt fest, ob ein Vergleich, eine Vorwärts- oder eine Rückwärtstransformation der beiden Modelle durchgeführt werden soll. Das Aktivierungskommando versetzt die Objekte der ACPLT/MT-Regel in den aktiven Zustand und löst die Bearbeitung der imperativen Anweisungen aus. Hierbei werden jedoch nicht alle Objekte einer Regel gleichzeitig aktiviert. Vielmehr sind die einzelnen Objekte über zusätzliche Assoziationen zu einer Kommandostruktur miteinander verknüpft. Zur Laufzeit werden zunächst die Objekte angestoßen, die in der Kommandostruktur führend sind. Diese bearbeiten ihre interne Logik und stoßen anschließend die Objekte an, die ihnen in der Kommandostruktur untergeordnet sind. Wir sprechen im weiteren Verlauf von einer Auftraggeber-Auftragnehmer-Beziehung, in der das Objekt, das in der Kommandostruktur übergeordnet ist als Auftraggeber und das untergeordnete Objekt als Auftragnehmer bezeichnet wird. Eine ACPLT/MT-Regel besitzt daher drei verschiedene Funktionsebenen (vgl. Abbildung 6.1). Neben der Repräsentation der Modellzusammenhänge zwischen linker und rechter Domäne, der deklarativen Ebene, repräsentieren die gleichen Elemente auch die Ausführungslogik in Form der operationalen Regel sowie die Kommandostruktur.

Neben den Anpassungen aus Platz- und Performancegründen, spielen aber auch strukturelle Besonderheiten eine große Rolle für die Integration von Tripel-Graph-Grammatiken in die prozessleittechnische Laufzeitumgebung. Da innerhalb der SPS alle Modelle in einer baumartigen Struktur mit gemeinsamen Wurzelknoten organisiert sind, ist jede Transformation zwischen zwei dieser Modelle per se erstmal eine potentielle Inplace-Transformation. Ein weiteres Problem stellt die Anforderung der Automatisierungstechnik nach Transformationen zwischen mehreren Quell- und Zielmodellen dar. Im Anwendungsszenario S1 treten beide Problemfälle zu Tage.

Beispiel 6.1 *Anwendungsszenario S1 benötigt für die Auswertung des Anlagenzustandes die Struktur der Anlage und den aktuellen Anlagenzustand. Ersteres wird durch PandIX, zweiteres durch die Prozessführung, in unserem Fall ACPLT/PF, bereitgestellt. Der Stand dieser beiden Modellinstanzen wird abgeglichen mit der Anzeige auf der Bedienoberfläche und in Szenario S1.c) zusätzlich mit der Prozessführung selbst zum Sperren oder Freigeben eines Aktors.*

Sowohl Inplace-Transformationen als auch multiple Quell- und Zielmodelle sind mit Tripel-Graph-Grammatiken zunächst nicht zu vereinbaren. Um die Integration von Tripel-Graph-Grammatiken in die Welt der Automatisierungstechnik zu ermöglichen, bedarf es einer genaueren Analyse, unter welchen Bedingungen Tripel-Graph-Grammatiken trotzdem zum Einsatz kommen können und wo die Grenzen der Einsatzfähigkeit erreicht sind. So muss der gemeinsame Wurzelknoten äquivalent zum leeren Modell behandelt werden. Er ist nicht Teil der einzelnen Modelle und darf als Kontextelement in allen drei Domänen verwendet werden. Es darf jedoch keine TGG-Produktion geben, die den Wurzelknoten erzeugt. Zudem müssen die Modelle zusätzliche Eigenschaften erfüllen, um Inplace-Transformationen zu vermeiden.

Definition 6.1 (Klassendisjunkte Modelle) *Zwei objektorientierte Modelle MM_1 und MM_2 sind klassendisjunkt, wenn die Menge der Klassennamen disjunkt sind. Klassendisjunkte Modelle können zum Beispiel durch die Angabe von vollständig qualifizierte Klassennamen forciert werden.*

Um Tripel-Graph-Grammatiken auf Modelle anzuwenden, die unter einem gemeinsamen Wurzelknoten organisiert sind, müssen die Modelle klassendisjunkt sein. Neben der Verwendung von klassendisjunkten Modellen muss verhindert werden, dass Verlinkungen zwischen den beiden Modellen erzeugt werden. Assoziationen sollen allerdings nicht dahingehend eingeschränkt sein, dass die Assoziationsnamen der beiden Modelle disjunkt sind. Dadurch würden zwar Querverbindungen unterbunden, allerdings wären elementare Assoziationen wie „enthält“ oder auch Verbindungen zwischen Aus- und Eingangsvariablen der Funktionsbausteine nicht mehr domänenübergreifend einsetzbar. Um Links zwischen den beiden Modellen dennoch unterbinden zu können, wird die Verwendung von Assoziationen in den TGG-Produktionen dahingegen eingeschränkt, dass sie nur gemeinsam mit allen durch sie verbundenen Klassen verwendet werden dürfen. Mit dieser Einschränkung kann folgende Eigenschaft zugesichert werden:

Behauptung 6.1 *Eine Transformation mittels Tripel-Graph-Grammatiken zwischen zwei Modellen klassendisjunkter Modelle kann als Outplace-Transformation behandelt werden, auch wenn die Modelle über einen gemeinsamen modellexternen Wurzelknoten verfügen.*

Anders ausgedrückt heißt das, dass Objekte und Links, die durch Pattern der linken Seite erzeugt werden, weder durch Pattern der rechten Seite erzeugt werden können noch als Kontextelemente des rechten Patterns gefunden werden können. Das Gleiche gilt sinngemäß für Objekte und Links, die durch Pattern der rechten Seite erzeugt werden.

Beweis 6.1 *Angenommen, es gibt eine Objekt $\mathcal{O}$, das sowohl durch die linke Teilproduktion einer TGG-Produktion als auch durch die rechte Teilproduktion der gleichen oder einer anderen TGG-Produktion gefunden oder erzeugt werden kann. Da die linke Teilproduktion nur Klassen des Modells der linken Domäne enthalten, muss $\mathcal{O}$ eine Instanz einer solchen Klasse sein. Mit der gleichen Begründung muss es aber auch Instanz einer Klasse des Modells der rechten Domäne sein. Da die Menge der Klassennamen der linken und rechten Domäne jedoch klassendisjunkt sein sollen, kann ein solches Objekt $\mathcal{O}$ nicht existieren.*

Angenommen, es gibt einen Link $\mathcal{L}$, der sowohl durch eine linke Teilproduktion einer TGG-Produktion als auch durch eine rechte Teilproduktion derselben oder einer anderen TGG-Produktion gefunden oder erzeugt werden kann. Da eine Assoziation immer im Zusammenhang mit den durch sie verknüpften Klassen in einer TGG-Produktion verwendet werden muss, müssen die durch den Link verbundenen Objekte $\mathcal{O}_1$, $\mathcal{O}_2$ Instanzen der entsprechenden Klassen sein. Der obigen Argumentation folgend können $\mathcal{O}_1$ und $\mathcal{O}_2$ aber nur entweder durch linke oder rechte Teilproduktionen gefunden oder erzeugt werden. Auch wenn der Link an sich in beiden Domänen gefunden oder erzeugt werden könnte, in Kombination mit den verlinkten Objekten ist er eindeutig einer Domäne zuzuordnen. Ein solcher Link $\mathcal{L}$ kann daher nicht existieren.

Auch Querverbindungen zwischen den Modellen sind dadurch ausgeschlossen, da eine Querverbindung Objekte verbinden würde, von denen eines eine Instanz einer Klasse des linken Modells wäre und das andere eine Instanz einer Klasse des rechten Modells. Angenommen ein solcher Link würde durch eine rechte Teilproduktion erzeugt. Dann müsste diese auch eine Klasse enthalten, die das Objekt erzeugt oder findet, das zum linken Modell gehört. Dies ist aber nicht möglich, da die Modelle klassendisjunkt sind. Gleiches gilt sinngemäß für das Erzeugen des Links in einer linken Teilproduktion.

Für den Nachweis, dass Tripel-Graph-Grammatiken unter bestimmten Umständen auch für n-zu-m Transformationen zum Einsatz kommen können, werden sogenannte Kombinationsmodelle eingeführt:

Definition 6.2 (Kombinationsmodell) *Seien $M_1 \ldots M_n$ Modelle mit den durch sie erzeugbaren Modellinstanzen $m_{1.1} \ldots m_{1.a}, \ldots, m_{n.1} \ldots m_{n.b}$. Ein Kombinationsmodell $KM_{1..n}$ ist ein Modell, das alle Kombinationen von Modellinstanzen der kombinierten Modelle erzeugen kann.*

Bei der Verwendung von Graph-Grammatiken wird ein Kombinationsmodell durch die Verschmelzung der einzelnen Grammatiken zu einer gemeinsamen Grammatik erzeugt. Diese Verschmelzung umfasst folgende Punkte

- *die Menge der Knoten-/Kantenbeschriftungen der kombinierten Grammatik ist die Vereinigung der Knoten-/Kantenbeschriftungen aller verschmolzenen Grammatiken*
- *die Menge der Produktionen der kombinierten Grammatik ist die Vereinigung der Produktionen aller verschmolzenen Grammatiken*

Mit Hilfe dieser Kombinationsmodelle lässt sich zeigen, dass unter der Voraussetzung der klassendisjunkten Modelle für die linke und rechte Domäne Tripel-Graph-Grammatiken auch auf n-zu-m Transformationen angewendet werden können:

Behauptung 6.2 *Tripel-Graph-Grammatiken können auf n-zu-m Transformationen angewendet werden, wenn die Modelle der linken und der rechten Domäne klassendisjunkt sind.*

Beweis 6.2 *Die Modelle der linken, respektive rechten Domäne können zu Kombinationsmodellen KM_L und KM_R verschmolzen werden. Dadurch kann die Transformation auf eine 1-zu-1 Transformation abgebildet werden. Durch die klassendisjunkten Modelle können unerwünschte Inplace-Transformationen auch im Setting der SPS ausgeschlossen werden.*

Sind die Modelle der linken und der rechten Domäne klassendisjunkt, bleiben daher auch bei einem gemeinsamem Wurzelknoten oder der n-zu-m Transformation alle für Tripel-Graph-Grammatiken zugesicherten Eigenschaften erhalten.

Schwieriger gestaltet sich die Situation, wenn Modelle domänenübergreifend verwendet werden. Dies bringt nicht absehbare Probleme mit sich und ist mit Tripel-Graph-Grammatiken nicht realisierbar. Bei ACPLT/MT wird jedoch auch die Beschränkung auf klassendisjunkte Modelle nicht per se gefordert. Stattdessen wird bei der domänenübergreifenden Verwendung von Modellen eine Warnung an den Benutzer ausgegeben. Diesem steht es dann frei, bewusst die Welt der Tripel-Graph-Grammatiken zu verlassen und die damit verbundenen Risiken einzugehen. Das Sperren und Freigeben eines Aktors aus dem Anwendungsszenario S1 ist ein solcher Fall, bei dem die Modelle der linken und rechten Domäne nicht klassendisjunkt sind, da ACPLT/PF in beiden Domänen zum Einsatz kommt. Der Rest des Szenarios hingegen kann mittels Tripel-Graph-Grammatiken realisiert werden.

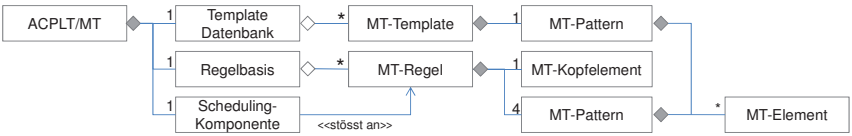
Die Amalgamierung wurde nicht realisiert, da sich diese nur schwer mit dem Konzept der aktivierbaren Tripel-Graph-Grammatiken vereinbaren lässt.

Abbildung 6.2 zeigt eine Übersicht über die Struktur des ACPLT/MT-Frameworks und den zur Verfügung stehenden Klassen. Hier zeigt sich eine weitere Besonderheit von ACPLT/MT. Zusätzlich zu den drei üblichen Teilproduktionen für die linke, die rechte und die Korrespondenzdomäne besitzen ACPLT/MT-Regeln noch ein weiteres, sehr kompaktes Triggerpattern. Dieses erzeugt bei der Ausführung keinerlei Objekte. Es dient dazu, ähnlich wie bei VIATRA, die Bearbeitung einer Regel gezielt anzustoßen oder auf ein Ereignis zu reagieren. Dieses Design verringert die Anzahl der aktiven Objekte, die in der SPS bearbeitet werden müssen. Zudem kann so zugesichert werden, dass zu einem beliebigen Zeitpunkt immer nur eine Regel aktiv ist. So werden zwar alle Triggerpattern parallel ausgewertet, schlägt aber ein Trigger an, wird die Triggerauswertung unterbrochen und die zum Trigger gehörige Regel angewendet. Im Abschnitt 6.5 wird auf das Konzept der Triggerpattern nochmals im Detail eingegangen.

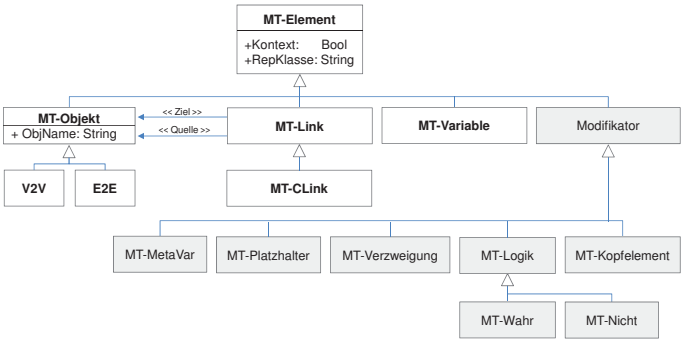
Die Klassen von ACPLT/MT sowie ihre Ausprägung in der deklarativen Ebene, der operationalen Ebene und der Kommandostruktur werden im Folgenden im Detail vorgestellt.

6.2 Deklarative Ebene

ACPLT/MT-Regeln orientieren sich in ihrem Grundaufbau an der verkürzten Schreibweise von TGG-Produktionen, die jeweils ein Pattern für die linke und rechte Domäne sowie ein



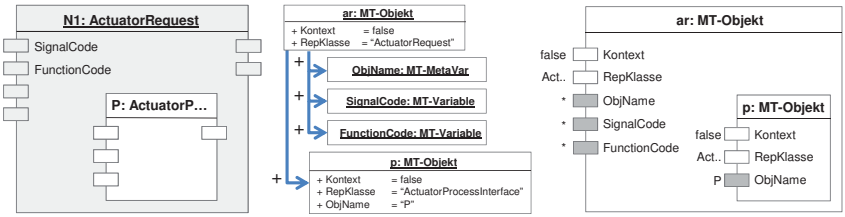
(a) Struktur des ACPLT/MT-Frameworks



(b) ACPLT/MT-Elemente

Abbildung 6.2: ACPLT/MT-Komponenten im Überblick

Pattern für die Korrespondenzdomäne besitzen. So lange ACPLT/MT als Realisierung von Tripel-Graph-Grammatiken zum Einsatz kommen, wird synonym zum Begriff Pattern der Begriff Teilproduktion verwendet. Wie auch beim zu Grunde liegende Konzept der Tripel-Graph-Grammatiken, sind die Pattern Graphen, die die Strukturen der korrelierenden Modelle sowie die der Korrespondenzdomäne abbilden. Wie in Abbildung 6.2b ersichtlich, stehen spezielle ACPLT/MT-Elemente für die Repräsentation von Objekten, Links und Variablen zur Verfügung. Abbildung 6.3 zeigt, wie diese genutzt werden, um Suchmuster zu beschreiben. Die gesuchten Modellobjekte sind in diesem Fall zwei geschachtelte Funktionsbausteine `N1` und `P` (vgl. Abbildung 6.3a), wobei für `N1` verlangt wird, dass es zum einen eine Instanz der Klasse `ActuatorRequest` ist und zum anderen mindestens zwei Eingangsvariablen `SignalCode` und `FunctionCode` zur Verfügung stellt. Typ und Wert dieser Variablen sind in diesem Beispiel nicht von Interesse. Das gesuchte Modellobjekt `P` muss in `N1` eingebettet und eine Instanz der Klasse `ActuatorProcessInterface` sein. Für die Musterbeschreibung stehen in ACPLT/MT drei Darstellungen zur Verfügung, die an die übliche Darstellung der deklarativen TGG-Produktionen angelehnte Objektdarstellung, die an FBDs angelehnte Funktionsblockdarstellung und die gemischte Darstellung, in der Elementen der Objektdarstellung und der Funktionsblockdarstellung gemeinsam verwendet werden. In der Objektdarstellung werden MT-Links zur besseren Lesbarkeit durch Pfeile repräsentiert, die mit dem Namen der Assoziation beschriftet sind. Ist kein Name angegeben, so handelt es sich implizit um eine *enthält*-Beziehungen (`RepKlasse="contains"`). Falls ein entsprechender Link durch die Regel im Modell erzeugt wird, ist der Pfeil mit einem „+“markiert. Quelle und Ziel des repräsentierten Links sind durch die Pfeilrichtung gekennzeichnet.



(a) Gesuchte Modellobjekte (b) Objektdarstellung (c) Funktionsblockdarstellung

Abbildung 6.3: Verwendung von MT-Objekt und MT-Variable zur Musterbeschreibung

MT-Variablen dienen zur Repräsentation von Variablen im Modell. Eine Ausnahme stellt die MT-Variable `ObjName` dar, die den Namen des Modellobjektes repräsentiert. MT-Variablen können in der Objektdarstellung entweder als eigenständiges Objekt oder als Variable des übergeordneten MT-Objektes dargestellt werden. Gleiches gilt auch für die Metavariablen, wie `Kontext` und `RepKlasse`. Metavariablen besitzen keine Entsprechung in den zu durchsuchenden Modellen, sondern dienen der Parametrierung der MT-Elemente. So bedingt die Verwendung von speziellen MT-Elementen an Stelle von Elementen aus den domänenspezifischen Sprachen eine schwache Typisierung über die Metavariable `RepKlasse`. Für die Kennzeichnung von Kontextelementen bzw. Elementen, die durch die Regel erzeugt werden, steht die Metavariable `Kontext` zur Verfügung, wobei `Kontext=false` der Markierung „++“ entspricht. Bei MT-Variablen wird zudem über die Metavariable `Typ`, der erwartete Datentyp und über die Metavariable `Value` der erwartete Variablenwerte angegeben. Für Metavariablen ist eine Parametrierung mit Jokerzeichen möglich, um eine entsprechende Variabilität zu kennzeichnen (vgl. Abbildung 6.3c: `ObjName`, `SignalCode` und `FunctionCode`). Die Parametrierung einer Metavariablen durch ein Asterisk kann in beiden Darstellungen auch durch Weglassen der entsprechenden Metavariablen verdeutlicht werden. Auf MT-Variablen und Metavariablen kann innerhalb

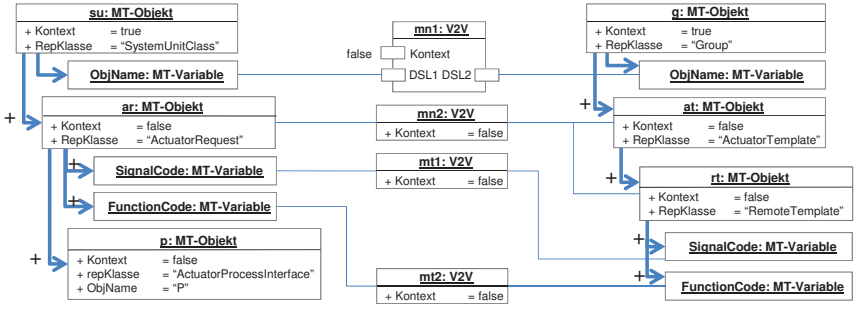


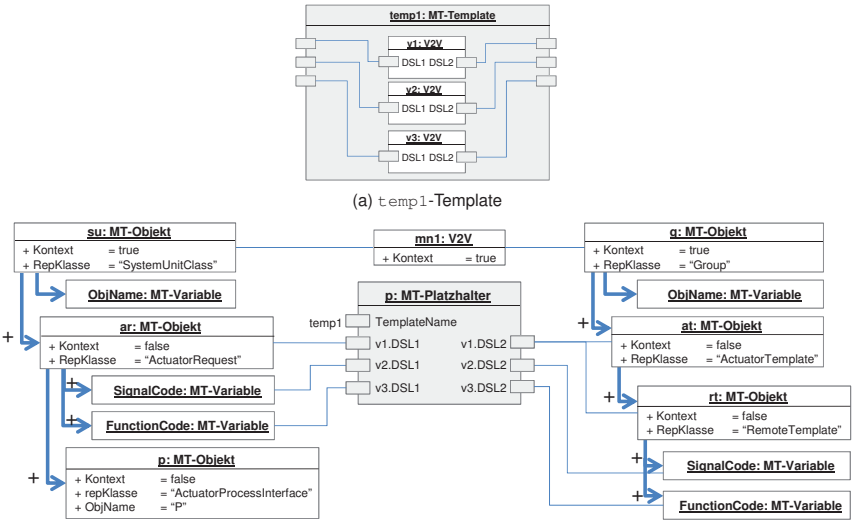
Abbildung 6.4: ACPLT/MT-Regel für das Anlegen einer PLT-Stelle

der Regel zugegriffen werden. Dies wird durch einfache Linien gekennzeichnet. Abbildung 6.4 verdeutlicht dies anhand der Einbettung des vorgestellten Suchmusters im Gesamtzusammenhang einer ACPLT/MT-Regel. Die Abbildung zeigt beispielhaft die ACPLT/MT-Repräsentation der TGG-Produktion aus Abbildung 5.5a, bei der unter anderem das Objekt `mn2` sicherstellt,

dass der `ActuatorRequest` im Modell der linken Domäne den gleichen Namen hat, wie das `ActuatorTemplate` der rechten Domäne. Dazu wird entlang der Korrespondenzlinks eine Weitergabe des Objektnamens in Richtung der Regelauswertung realisiert. Zudem kommen in der Teilproduktion für die Korrespondenzdomäne spezielle Unterklassen von MT-Objekt zum Einsatz, die mit je mindestens einem MT-Element aus der Quell- und aus der Zieldomäne über spezielle MT-Links assoziiert sind. Anders als bei Tripel-Graph-Grammatiken üblich, sind im Basismodell von ACPLT/MT genau zwei Klassen für die MT-Objekte der Korrespondenzdomäne, `E2E` und `V2V` vorgesehen. Ersteres verbindet Elemente der beiden Domänen miteinander ohne Einschränkungen zu Klasse, Elementname oder Typ des Elements. Existiert das durch das Korrespondenzobjekt verlinkte Objekt in der linken Domäne, so muss eine Entsprechung für das MT-Objekt in der rechten Domäne gefunden bzw. erstellt werden. Korrespondenzobjekte vom Typ `V2V` verbinden zwei MT-Variablen miteinander, die den gleichen Datentyp und den gleichen Wert besitzen. In der Objektdarstellung ist auch eine Verknüpfung zweier MT-Objekte mittels `V2V` möglich. Dies bezieht sich jeweils auf die MT-Variable `ObjName`, die aus Platzgründen und zur besseren Lesbarkeit nicht immer einzeln aufgeführt wird. Die Beschränkung auf die beiden Klassen von Korrespondenzobjekten `E2E` und `V2V` ermöglicht den flexiblen, modellunabhängigen Einsatz von ACPLT/MT.

Wie am Anfang des Kapitels erwähnt, ist ACPLT/MT auf Performance und Platzoptimierung ausgelegt. Die bisher vorgestellten Elemente wirken durch ihre Granularität diesem Ziel jedoch eher entgegen. Während zum Beispiel in der TGG-Produktion für PLT-Stellen (Abbildung 5.5a) ein Korrespondenzobjekt vom Typ `MT1` reichte, benötigt die ACPLT/MT-Variante drei Korrespondenzobjekte und entsprechend viele Links, um die gleiche Relation darzustellen. Zusätzlich zu den Repräsentanten für Objekte, Links und Variablen gibt es daher noch eine Reihe weiterer MT-Elemente, die unter dem Begriff MT-Modifikatoren zusammengefasst werden. Diese haben keinen direkten Bezug zum Quell- oder Zielmodell. Stattdessen modifizieren sie die Abarbeitung der Regel. Einer dieser Modifikatoren, die Metavariablen (MT-MetaVar) wurden bereits vorgestellt.

Auch die Wiederverwendung von Regelteilen (MT-Platzhalter) und Verzweigungen in den deklarativen Regeln (MT-Verzweigung) werden mit Hilfe von Modifikatoren realisiert. Diese beiden Konzepte stellen eine Erweiterung der bisherigen Tripel-Graph-Grammatik-Ansätze dar. Platzhalter beschreiben, ebenso wie QVT-Templates, komplexe Objektnetzwerke, die in verschiedenen Regeln oder in einer Regel mehrfach verwendet werden können. Sie werden während der Ausführung der Transformation durch die entsprechenden Objektnetzwerke ersetzt, so dass zum Ausführungszeitpunkt zeitweise die komplette Regel im System vorliegt. Dies ermöglicht eine TGG-konforme Verwendung von Platzhaltern. Verzweigungen ermöglichen in deklarativen ACPLT/MT-Regeln die Beschreibung von mehreren alternativen Regelvarianten. Dadurch können Regeln mit gemeinsamer Oberklasse zusammengeführt und Anjorins Vererbungskonzept umgesetzt werden. Verzweigungen werden dabei immer paarweise verwendet, eine in der linken und eine in der rechten Domäne. Paare von Verzweigungen sind jeweils gleichgeschaltet, d.h. wenn in der Verzweigung der linken Teilproduktion die erste Alternative zum Einsatz kommt, so muss dies auch für die Verzweigung in der rechten Teilproduktion gelten. Zum Zeitpunkt der Transformation ist immer maximal ein Paar an Alternativen aktiviert und es liegt somit eine TGG-konforme Regel vor. Beide Erweiterungen, die Einführung von Platzhaltern und



(a) `temp1`-Template

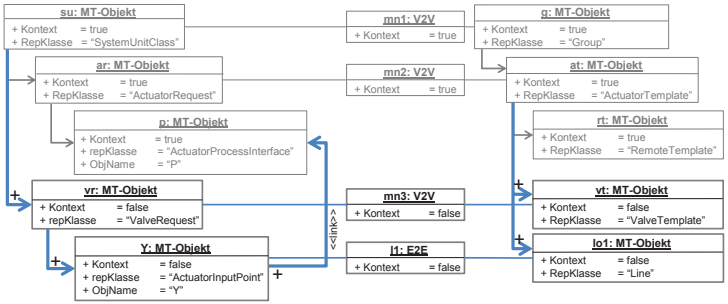
(b) Regel mit `temp1`-Platzhalter

Abbildung 6.5: Template-/Platzhalterkonzept von ACPLT/MT

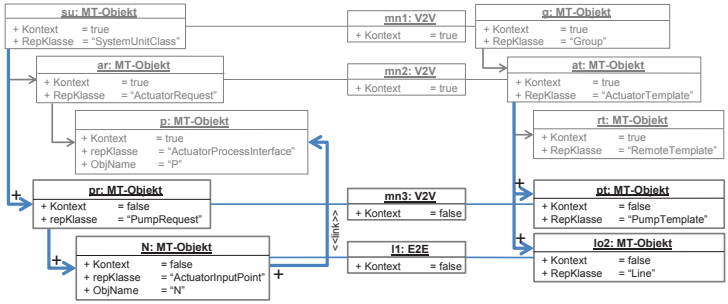
die Verwendung von Verzweigungen dienen neben der Effizienzsteigerung auch der besseren Nachvollziehbarkeit und Wartbarkeit der Regeln.

Platzhalter Beim Einsatz von ACPLT/MT konnte immer wieder beobachtet werden, dass Teilmuster sich in verschiedenen Regel wiederholen. Das Auslagern von Teilmustern in Templates und die spätere Referenzierung mittels Platzhaltern ist in anderen Modell- bzw. Graphtransformationsansätzen wie VIATRA und QVT längst Standard. Bei Tripel-Graph-Grammatiken kommt ein solches Konzept bisher nicht zum Einsatz. Zwar erlaubt das Vererbungskonzept von Anjorin, Teile der Regel wiederzuverwenden, die Vererbung ist jedoch auf eine Oberklasse beschränkt. Platzhalter lassen sich in ACPLT/MT für beliebige Teile einer TGG-Regel erstellen. Abbildung 6.5a zeigt die Kapselung dreier `V2V`-Objekte in einem Template. Templates müssen alle später in der Regel benötigten Anknüpfungspunkte für Links nach außen als Ein- bzw. Ausgänge sichtbar machen. Die Möglichkeit der Wiederverwendung von Templates und die erhöhte Lesbarkeit der Regeln durch Verwendung aussagekräftiger Templatenamen machen das Template-/Platzhalterkonzept zu einem äußert sinnvollen Hilfsmittel bei der Gestaltung von Regeln. Für Anbieter von Automatisierungssoftware bietet das Template-/Platzhalterkonzept zudem die Möglichkeit vorgefertigte Regelbestandteile als Template-Datenbank bereitzustellen und Applikateuren dadurch die Erstellung der Regeln weiter zu vereinfachen.

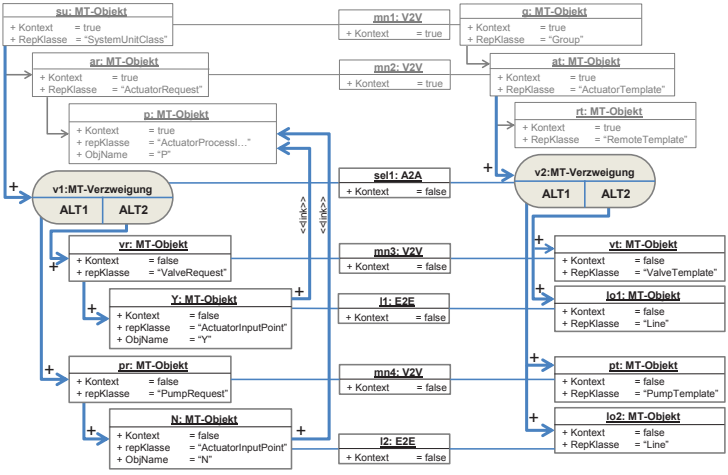
Verzweigung Eine ACPLT-MT Umsetzung des Vererbungskonzeptes von Anjorin ist die Verwendung von MT-Verzweigungen. Anstatt Klassen zu bilden, werden hierbei jedoch alle Unterklassen in eine gemeinsame Regel abgebildet. Dies ist nötig, um dem Konzept der aktivierbaren Regeln gerecht zu werden. MT-Verzweigungen basieren auf der Beschrei-



(a) ACPLT/MT-Regel zum Anlegen eines Ventils



(b) ACPLT/MT-Regel zum Anlegen einer Pumpe



(c) Kombinierte Regel

Abbildung 6.6: Zusammenführung zweier ähnlicher Regeln mittels MT-Verzweigung

bung von alternativen Regelvarianten mit der Besonderheit, dass MT-Verzweigungen immer paarweise in der linken und rechten Teilproduktion auftreten. Die Wahl der Alternative ist bei Paaren von MT-Verzweigungen gleichgeschaltet. Dies wird durch eine Korrespondenzobjekt vom Typ `A2A` zugesichert. Abbildung 6.6c zeigt, wie die beiden Einzelregeln zur Erstellung eines Ventils in ACPLT/HMI und ACPLT/PF aus Abbildung 6.6a und der in Abbildung6.6b gezeigten Erstellung einer Pumpe in den beiden Domänen mittels MT-Verzweigung zusammengeführt werden können. Der ausgegraute Teil der Regeln entspricht der Anjorins Oberklasse und ist in allen drei Regeln gleich.

Da die MT-Verzweigung eine Unterklasse von MT-Objekt ist, unterbricht sie einen MT-Link. Ausschlaggebend bei der Anwendung der Regel sind die Metavariablen des von der MT-Verzweigung in die jeweilige Alternative führenden MT-Links. MT-Links, die Quelle oder Ziel innerhalb einer Alternativen besitzen, gehören zu dieser Alternative, MT-Links zwischen zwei Alternativen sind nicht erlaubt. Auch die Verwendung der Korrespondenzobjekte ist eingeschränkt. Sie können nur Elemente von einander zugeordneten Alternativen der linken und rechten Teilproduktion miteinander verbinden.

Die gemeinsame Nutzung einer „Oberklasse“ ermöglicht deutliche Einsparungen von Ressourcen sowohl in Bezug auf den Platzbedarf als auch beim Zeitbedarf während der Transformation. Zudem ergibt sich ein deutlicher Zugewinn für die Wartbarkeit der Regeln.

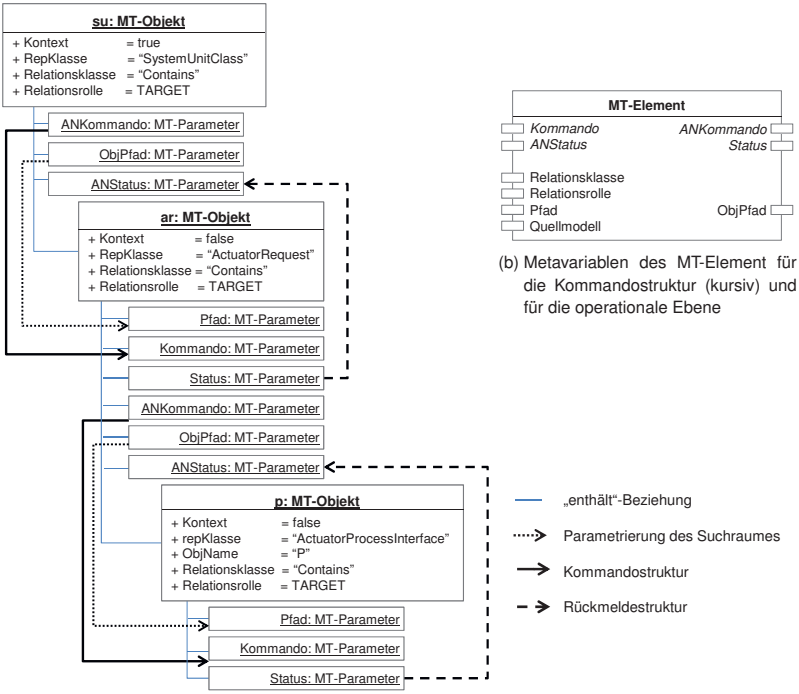
Logische Operatoren Für die Realisierung der NACs, wie sie von Königs [Kön08] vorgeschlagen wurden, wird der Operator `NICHT` eingeführt. Er negiert das Vorhandensein der ihm nachgeschalteten Struktur. `NICHT`-Elemente dürfen nur im Kontext einer Regel zum Einsatz kommen, um Interpretationsprobleme in den operationalen Regeln zu vermeiden. Der Operator `Wahr` wird bei der Einführung von Triggern im Abschnitt 6.5 genauer erläutert. Er erlaubt keine nachgeschalteten Kontrollstrukturen.

Kopfelement In der deklarativen ACPLT/MT-Regel dient das Kopfelement als gemeinsames Wurzelelement für die Pattern der Regel. Zudem ist das Kopfelement der Einstiegspunkt für die Bearbeitung der operationalen MT-Regel.

Metavariablen Daten, die zur Parametrierung von ACPLT/MT benötigt werden, werden in Metavariablen (MT-MetaVar) abgelegt. Diese sind wie MT-Variablen Ein-/Ausgänge oder lokale Variablen der MT-Objekte. Beispiele für Metavariablen sind `Kontext`, `RepKlasse` und die für die operationale Ebene und den Kontrollalgorithmus benötigten Daten.

6.3 Kommandostruktur

Jedes MT-Element besitzt eine eigene interne Logik, die bestimmte Teilaufgaben bei der Anwendung einer MT-Regel übernimmt. Die Bearbeitung der einzelnen Teilaufgaben wird über die Kommandostruktur zu einer operationalen Regel verknüpft. Abbildung 6.7 zeigt beispielhaft die Kommandostruktur der linken Teilproduktion für das Anlegen eines Ventils, sowie die Klasse MT-Element mit den für die Kommandostruktur relevanten Metavariablen. Die MT-Elemente agieren während der Regelbearbeitung entlang der Kommandostruktur als Hierarchie von Auftraggebern und Auftragnehmern. Für die Weitergabe benötigter Daten und Aufträge entlang dieser Hierarchie stehen verschiedene Metavariablen zur Verfügung. Über die Metavariable



(a) Ausschnitt Kommandostruktur der MT-Regel für PLT-Stellen

(b) Metavariablen des MT-Element für die Kommandostruktur (kursiv) und für die operationale Ebene

Abbildung 6.7: Kommandostruktur und operationale Ebene

`Kommando` erhält ein MT-Element Kommandos von seinem Auftraggeber. Neben dem Kommando `START`, das einen Auftragnehmer veranlasst, seine interne Logik zu bearbeiten, gibt es außerdem die Kommandos `RÜCKGÄNGIG`, `ZURÜCKSETZEN` und `HALTEN`. Ersteres kommt nur bei MT-Objekten zum Tragen, die Elemente in der Zieldomäne anlegen. Das zuletzt angelegte Objekt wird beim Befehl `RÜCKGÄNGIG` wieder gelöscht oder in den Ursprungszustand zurückversetzt, falls sich nur die Variablen geändert haben. Das Kommando `ZURÜCKSETZEN` wird genutzt, um die Werte der ausgehenden Metavariablen zu löschen und die Regel wieder in ihren Ursprungszustand zurück zu versetzen.

Während die Auftragnehmer ihre Logik bearbeiten, befindet sich der Auftraggeber im Zustand `ARBEITEND`.Das Kommando `HALTEN` unterbricht die Auswertung der internen Logik. Dies ist insbesondere dann von Interesse, wenn ein MT-Objekt ein Objekt der entsprechenden Domäne gefunden oder angelegt hat und mit der Suche bzw. dem Anlegen weiterer Objekte im Suchraum warten soll, bis die gesamte Regel abgearbeitet wurde. Durch die zyklische Bearbeitung wird die interne Logik von MT-Objekten in der Quelldomäne sonst ggf. die Suche fortsetzen bevor eine Auswertung durch die MT-Elemente der Zieldomäne stattgefunden hat.

Das Kommando `HALTEN` erzeugt nach Rückmeldung der Auftragnehmer einen Statuswechsel zu `HALTEND`.

Kommandostruktur und die deklarative Ebene überlagern sich in weiten Teilen. Zudem erfolgt die Verarbeitung der Informationen über einen MT-Link immer am Auftragnehmer. Diese beiden Eigenschaften ermöglichen es, Kommandostruktur und deklarative Ebene aufeinander abzubilden. Zu diesem Zweck erhalten MT-Elemente zusätzlich die Metavariablen `Relationsklasse` und `Relationsrolle`, um die Bedeutung des am`Kommando`-Eingang anliegenden eingehenden Links für die deklarative Regel anzugeben. Beide Metavariablen erlauben die Verwendung von WildCards, um unbestimmte Relationen oder Relationsrollen abzubilden. Der in Abbildung 6.7a dargestellte „enthält“-Link zwischen den MT-Objekten dient lediglich der strukturierten Datenhaltung. Sie hat keine Bedeutung für die Logik der MT-Regel.

Ausnahme von der Überlagerung der Kommandostruktur und der deklarativen Ebene bilden die Links zwischen zwei MT-Variablen. Wird die Kommandostruktur entlang einer solchen Verbindung weitergeführt, ergeben sich dadurch Probleme beim Anlegen der Zielvariablen, da das Elternobjekt noch nicht gefunden bzw. erstellt ist. Links zwischen MT-Variablen werden daher von beiden MT-Objekten, deren Variablen miteinander verbunden sind, bearbeitet. Die MT-Objekte überprüfen dazu, ob das über den Link verbundene MT-Objekt bereits bearbeitet wurde. Ist dies der Fall, so kann die Verbindung überprüft bzw. angelegt werden, ansonsten wird die Verbindung zunächst ignoriert.

Parallel zur Kommandostruktur existiert eine Rückmeldestruktur, über die der Auftragnehmer dem Auftraggeber seinen aktuellen Zustand mitteilen kann.

6.4 Operationale Ebene

Die Regelanwendung erfolgt in ACPLT/MT durch Bearbeitung des in den MT-Elementen enthalten Codes. Ausgangspunkt für die Regelbearbeitung ist das gemeinsame Wurzelelement der drei Teilproduktionen, das Kopfelement (vgl. Abbildung 6.8). Dieses wird mit der Richtung der Transformation (Vorwärts/Rückwärts/Konsistenz) sowie mit den Pfaden zum Wurzelknoten der beteiligten Modelle parametriert und stößt nacheinander die Bearbeitung der Teilproduktionen an. Wird ein Kopfelement aktiviert, stößt es zunächst die Kommandostruktur der Quelldomäne an und wartet auf Rückmeldung. Erfolgt eine positive Rückmeldung, so wird die Kommandostruktur der Zieldomäne angestoßen. Erfolgt auch durch diese eine positive Rückmeldung, so werden nacheinander die Elemente der Korrespondenzdomäne angestoßen, um die Transformation zu dokumentieren. Zudem werden alle durch ein Nicht-Kontext-Element des Quellmodell-Pattern gebundenen Objekte in einem zentralen Datenbankobjekt registriert und sind somit als übersetzt markiert. Dies erlaubt die Verwendung dieser Objekte als Kontextelemente in weiteren Regelanwendungen.

Dieser Vorgang wiederholt sich so lange, bis das Pattern der Quelldomäne ein `NICHT_GEFUNDEN` zurückliefert. In diesem Fall sind alle durch das Pattern repräsentierten Strukturen im Quellmodell übersetzt. Tritt während der Bearbeitung der drei Pattern ein Fehler auf, so wird an die

bereits ausgeführten Regelbestandteile das Kommando `RÜCKGÄNGIG` geschickt, wodurch die MT-Objekte ihren letzten Schritt rückgängig machen. An dieser Stelle sei auf Anhang C verwiesen, in dem anhand der Regel aus Abbildung 6.6c die Übersetzung eines Modells in ein anderes mit Hilfe von ACPLT/MT Schritt für Schritt erläutert wird.

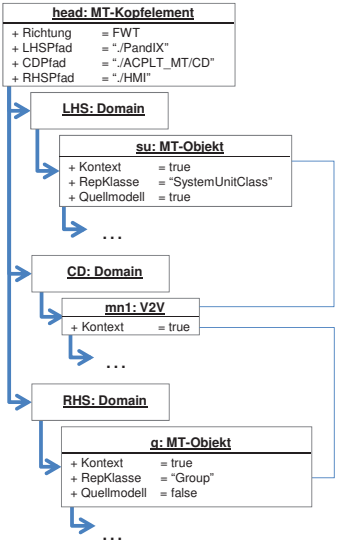


Abbildung 6.8: Operationale ACPLT/MT-Regel für die Vorwärtstransformation

Für die operationale Ebene stehen MT-Elementen die Eingangs-Metavariablen `Quellmodell`, `Relationsklasse`, `Relationsrolle` und `Pfad` sowie der Ausgang `ObjPfad` zur Verfügung (vgl. Abbildung 6.7b), auf die im Folgenden genauer eingegangen werden soll.

Während bei operationalen TGG-Produktionen für jedes Element individuell durch $\square \rightarrow \boxtimes$ bzw. $\boxtimes \rightarrow \boxtimes$ angegeben wird, wie es zu interpretieren ist, bleibt bei den operationalen MT-Regeln die Metavariable `Kontext` unverändert und gibt in Kombination mit der ergänzenden Metavariablen `Quellmodell` (`true|false`) an, ob ein entsprechendes Objekt im Modell gesucht oder erstellt werden soll. Dieses Design bringt insbesondere unter dem Gesichtspunkt der Integration in die IEC 61131 - Sprachen enorme Vorteile bei der Handhabung mit sich. Der Wert der Metavariablen `Quellmodell` ist für alle MT-Elemente eines Patterns identisch und kann daher ausgehend vom Kopfelement der MT-Regel durch Verbindungen zwischen den Variablen in alle MT-Elemente des Patterns propagiert werden. Der Wert der Metavariablen `Kontext` bleibt so für weitere Regelausführungen unverändert erhalten.

Wird die Regel zur Konsistenzanalyse eingesetzt, so werden sowohl die MT-Elemente der linken als auch die der rechten Domäne mit `Quellmodell = true` parametriert. Dies bewirkt, dass sie unabhängig davon, ob sie Teil des Kontext sind oder nicht nach passenden Modellobjekten suchen und keine Modellobjekte erstellen. Beim Starten einer Regel zur Konsistenzana-

lyse stößt das Kopfelement zunächst die Kommandostruktur der linken Domäne an. Für jede positive Rückmeldung der linken Domäne stößt sie die Kommandostruktur der rechten Domäne und anschließend der Korrespondenzobjekte an. Gibt es von der rechten Domäne oder von den Korrespondenzobjekten eine negative Rückmeldung, so wird der Fehler gemeldet. Fehlende Korrespondenzobjekte werden dabei nicht als Fehler sondern als Warnung ausgegeben. Es steht dem Anwender frei, die gemeldete Inkonsistenz zu beheben oder sie zu ignorieren. Gibt die Kommandostruktur der linken Domäne eine negative Rückmeldung, so werden die MT-Elemente der Regel zurückgesetzt und anschließend der gesamte Vorgang für die rechte Domäne wiederholt. Eine automatische Korrektur der Inkonsistenzen ist bisher nicht vorgesehen. Der Anwender hat nach einer Konsistenzanalyse jedoch die Möglichkeit, eine inkrementelle Vorwärts- oder Rückwärtstransformation anzustoßen oder die Inkonsistenzen manuell zu beheben.

6.4.1 MT-Objekt

Wird ein MT-Objekt während der Laufzeit durch einen Auftraggeber aktiviert, so sucht es innerhalb eines vorgegebenen Suchraums selbstständig ein der Parametrierung entsprechendes Modellelement oder legt es - je nach Richtung der Regel - an. Der Suchraum wird über die Metavariable `Pfad` festgelegt. Eine Parametrierung des Suchpfads mit Hilfe von Jokerzeichen erlaubt eine flexible Suche zum Beispiel im gesamten Modell oder nur in bestimmten Teilstrukturen. Die Metavariablen `Relationsklasse` und `Relationsrolle` geben an, über welche Assoziation das zu suchende Objekt mit dem durch den Auftraggeber gebundene Objekt verknüpft ist. Beide Metavariablen können wiederum mit Jokerzeichen parametriert werden, um beliebige Assoziationen zuzulassen.

War die Suche oder das Anlegen erfolgreich, wird der Auftragnehmer selbst zum Auftraggeber und stößt weitere Elemente der Regel an, ihrerseits passende Objekte zu suchen oder anzulegen. Haben alle angestoßenen Elemente ihre Aufgabe erfolgreich erledigt, wird eine Rückmeldung an den Auftraggeber erzeugt. Beim Auftreten eines Fehlers wird eine Fehlermeldung an den Auftraggeber zurückgeliefert und gegebenenfalls durchgeführte Änderungen am Zielmodell rückgängig gemacht. Zusammenfassend besteht die interne Logik der MT-Objekte aus folgenden Schritten:

Starten der Bearbeitung Zunächst wird die Metavariable `Status` auf `ARBEITEND` gesetzt, um dem Auftraggeber eine Rückmeldung über den Erhalt des Kommandos zu geben. Zudem wird anhand der Metavariablen `Kontext` und `Quellmodell` ausgewertet, ob sich das MT-Objekt im Modus `SUCHEN`, `VERGLEICHEN` oder `ANLEGEN` befindet.

Suchen/Erstellen eines Objektes Es wird ein Objekt entsprechend der Parametrierung unter Berücksichtigung des Modus, des Suchraumes (`Pfad`) und der von da ausgehenden Assoziation (`Relationsklasse`, `Relationsrolle`) gesucht oder erstellt. Nacheinander werden bei jedem Regeldurchlauf alle auf die Beschreibung zutreffende Objekte gefunden/erstellt. Hierbei kommt, wie von Königs [Kön08] vorgeschlagen, das optionale Erstellen zum Einsatz. Objekte die bereits im Zielmodell vorhanden sind werden dabei erkannt und nicht erneut angelegt.

Auswerten der MT-Variablen In Abhängigkeit des Modus werden alle MT-Variablen des MT-Objektes auf Typ und Wert überprüft (SUCHEN, VERGLEICHEN) oder die Werte der Variablen werden über den Korrespondenzgraph vom MT-Element im Quellmodell geholt und am Zielobjekt gesetzt (ANLEGEN).

Auswerten von Verbindungen Alle Verbindungen zwischen MT-Variablen werden dahingehend untersucht, ob das verbundene MT-Objekt bereits bearbeitet wurde. Ist dies der Fall, so wird das Vorhandensein der Verbindung überprüft (SUCHEN/VERGLEICHEN) oder die Verbindung wird angelegt (ANLEGEN).

Setzen des Objektnamens Der Pfad des gefundenen/erstellten Objektes wird in der Metavariablen ObjPfad gespeichert und dem Kopfelement bekanntgegeben.

Anstoßen der Auftragnehmer Über die Metavariable ANKommando wird der Befehl START abgesetzt und somit die Auftragnehmer angestoßen, ihre Logik abzuarbeiten. Diese suchen oder erstellen ihrerseits einen Teil der durch die Regel beschriebenen Struktur.

Warten auf Rückmeldung Das MT-Objekt wartet auf eine Rückmeldung von den angestoßenen Auftragnehmern.

Rückmeldung an den Auftraggeber Über die Metavariable Status wird eine Rückmeldung an den Auftraggeber abgesetzt. War die Bearbeitung erfolgreich und die eigenen Auftragnehmer haben eine positive Rückmeldung gegeben, wird der Status OK zurückgegeben. Bei aufgetretenen Fehlern können ein allgemeiner Fehler oder spezielle Fehlermeldungen wie NICHT_GEFUNDEN oder im Modus VERGLEICHEN der Fehler INKONSISTENT bei fehlenden Objekten oder inkonsistenten Variablenwerten zurückgegeben werden.

6.4.2 Modifikatoren

Die Modifikatoren besitzen eine angepasste Ausführungslogik, die ebenfalls die komplette Bearbeitung der für das Element benötigten Funktionalität umfasst.

Platzhalter/Template Wird ein Platzhalters aktiviert, erstellt er eine lokale Kopie des referenzierten Templates und verknüpft diese mit den eigenen Ein- und Ausgängen (vgl. Abbildung 6.9). Diese temporäre Instanziierung des Template ist notwendig, da Templates mehrfach in der Regel referenziert werden können. Bedingt durch die zyklische Bearbeitung in der SPS, muss jede Instanz ihre Ein- und Ausgänge über mehrere Zyklen halten. Eine bloße Verlinkung zum Template und somit eine Mehrfachbenutzung der templateinternen MT-Elemente würde ein Überschreiben der Variablenwerte zur Folge haben. Nach dem Anlegen der lokalen Kopie des Templates stößt das Platzhalterobjekt die im Template enthaltene Struktur zur Bearbeitung ihrer Logik an. Rückmeldungen von der Kommandostruktur der lokalen Kopie werden direkt an den Auftraggeber weitergereicht. Erhält der Platzhalter das Kommando ZURÜCKSETZEN, wird die lokale Kopie des Templates wieder gelöscht und der Speicherplatz freigegeben.

Verzweigungen Die Verzweigung bedingt eine doppelte Kommandostruktur und somit jeweils zwei Metavariablen ANKommando und ANStatus. Der ObjPfad kann für beide Alternativen verwendet werden. Wird eine Verzweigung im Modus SUCHEN aktiviert, so stößt sie zunächst die Bearbeitung der ersten Alternative an. Erst wenn diese die Rückmeldung NICHT_GEFUNDEN zurückliefert, erhält die nächste Alternative das Kommando zur Be-

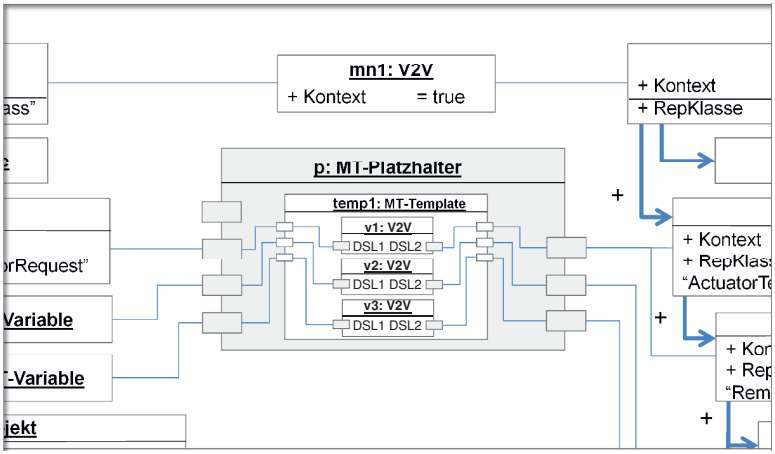


Abbildung 6.9: Eingesetztes Template für den in Abbildung 6.5b gezeigten Platzhalter

arbeitung ihrer Logik. Über eine zusätzliche Metavariable zeigt die MT-Verzweigung an, welche Alternative gerade aktiv ist. In den Modi `VERGLEICHEN` und `ANLEGEN` wird die aktive Alternative des Partners ausgelesen und nur die entsprechende eigene Alternative zur Bearbeitung angestoßen. Durch die gemeinsame Nutzung von Regelbestandteilen halbiert sich die Anzahl der Suchvorgänge für diesen gemeinsamen Teil. Dadurch kann mit der Verwendung von Verzweigungen eine verringerte Systembelastung bei gleicher Wirkung erzielt werden.

Die mit dem Einsatz von MT-Verzweigungen einhergehende Verzweigung der Kontrollstruktur wird zudem genutzt, wenn ein Pattern aus zwei nicht miteinander verbundenen Teilgraphen besteht. Dies kann insbesondere bei der Verwendung von multiplen Quellmodellen auftreten. Um die Kontrollstruktur trotzdem abbilden zu können wird die MT-Verzweigung in diesem Fall als `UND` statt als `ODER` genutzt. Hierzu wird eine weitere Metavariable eingeführt, die eine entsprechende Umschaltung zwischen den beiden Varianten ermöglicht. Ein passender Korrespondenzknoten vom Typ `V2V` synchronisiert diese Wahl zwischen linker und rechter Teilproduktion. Wird eine als `UND` parametrierte Verzweigung durch seinen Auftraggeber angestoßen, so gibt er diesen Auftrag direkt an alle Alternativen weiter. Erst wenn alle Alternativen eine positive Rückmeldung liefern, wird dies an den Auftraggeber weitergeleitet. Als `UND` parametrierte MT-Verzweigungen werden ungepaart verwendet. Sie benötigen daher kein Gegenstück in der anderen Domäne.

Logik-Operatoren Der `NICHT`-Operator liefert eine positive Rückmeldung, wenn der nachgeschaltete Auftragnehmer ein `NICHT_GEFUNDEN` meldet und umgekehrt. Fehlermeldungen von den Auftragnehmern werden entlang der Kommandostruktur nach oben durchgereicht. Der Operator `Wahr` liefert immer ein `GEFUNDEN` zurück, sobald er angestoßen wird.

6.4.3 Korrespondenzgraph

Die MT-Elemente des Korrespondenzgraphen haben zwei Aufgaben. Zum einen dienen sie dazu, durch Verlinkung der repräsentierenden MT-Elemente eine Verknüpfung zwischen den MT-Objekten, MT-Links und MT-Variablen der linken und der rechten Teilproduktion herzustellen. Die Auswertung dieser Verlinkung erfolgt in den verlinkten MT-Elementen. So überprüft jede MT-Variable entlang der mit ihr verknüpften Korrespondenzobjekte `V2V`, ob die Gegenseite bereits gebunden ist. Ist dies der Fall, so passt es seine Metavariablen `Type` und `Value` an, um die Gleichheit von Wert und Typ zuzusichern. Die zweite Aufgabe der Korrespondenzobjekte ist die Dokumentation der Transformation. Hierzu sind verschiedene Realisierungen denkbar. So können, wie bei Tripel-Graph-Grammatiken üblich, Objekte angelegt werden. Dies widerspricht aber dem Ansatz, einer ressourcenschonende Implementierung. Eine weitere Möglichkeit ist die Speicherung in einem Datenbankobjekt, so dass jede Instanziierung eines Korrespondenzobjektes ein Datenbankeintrag erzeugt, der unter anderem Daten über die durch das Korrespondenzobjekt verlinkten Modellelemente sowie die Richtung und den Zeitpunkt der Transformation enthält. Durch diesen Ansatz können auch die korrelierenden Modelle unverändert bleiben, da kein Anknüpfungspunkt für den Link zum Korrespondenzobjekt benötigt wird.

6.5 Kontrollalgorithmus

Der Kontrollalgorithmus besteht aus zwei Komponenten, der Strategie und dem Scheduling. Die Strategiekomponente ist in den MT-Regeln enthalten. Insbesondere MT-Objekte und die Modifikatoren besitzen eine eigene Logik, die zur Laufzeit die Funktionalität des entsprechenden Elements selbständig bearbeitet. Das Scheduling dient als übergeordnete Steuerungskomponente für die Bearbeitung der verschiedenen MT-Regeln.

Durch die beschränkten Ressourcen in einer SPS wäre ein fortlaufendes Ausführen aller in der Regelbasis enthaltenen Regeln eine enorme Belastung für das Gesamtsystem. Erfahrungsgemäß sind es aber bestimmte Ereignisse, wie das Speichern eines Modells nach erfolgter Modellanpassung oder die nutzergesteuerte Anforderung eines Zustandswechsels bei einem Ventil, die einem benötigten Modellabgleich vorausgehen. Aus diesem Grund eignet sich eine ereignisgesteuerte Regelaktivierung wie bei VIATRA besonders gut für eine Modelltransformation in prozessleittechnischen Laufzeitumgebungen. Zur Ereignisüberwachung kommt ein viertes relativ kompaktes Pattern, das Triggerpattern zum Einsatz. Anders als die übrigen Pattern der MT-Regel müssen Trigger ihre gesamte Logik innerhalb eines Zyklus bearbeiten, da ansonsten Ereignisse, die nur einen Zyklus lang vorliegen, verloren gehen können. Zudem darf es keine zusätzlichen Zyklen benötigen, um die Trigger zurückzusetzen. Zu diesem Zweck wird ein weiteres Kommando `TRIGGER` eingeführt. Dieses veranlasst das angestoßene MT-Objekt zustandslos zu arbeiten und in jedem Zyklus nur in Abhängigkeit von den Eingängen und nicht auf Basis von Ergebnissen vorheriger Zyklen zu arbeiten.

Die Schedulingkomponente nutzt die Trigger, um die Regel zum richtigen Zeitpunkt anzustoßen. Folgende Schritte werden dabei durchlaufen:

Aktivieren aller Trigger Die Schedulingkomponente aktiviert zunächst die Trigger aller MT-Regeln der Regelbasis und überprüft anschließend zyklisch ihren Status.

Regel-Bearbeitung Hat mindestens ein Trigger eine positive Rückmeldung erzeugt, so wird die zugehörige Regel aktiviert. Haben mehrere Trigger eine positive Rückmeldung erzeugt, so werden ihre Regeln nacheinander aktiviert, so dass zu jedem Zeitpunkt maximal eine aktive Regel ihren Teil der Modelltransformation durchführt. Ist eine Regel einmal aktiviert, ermittelt sie nacheinander alle relevanten Modellstrukturen und führt entsprechend oft eine Transformation durch. Dieser sehr einfache Ansatz ist sicher zunächst nicht sehr performant. Die Suche nach einem optimalen Algorithmus kann Ausgangspunkt von weiterführenden Forschungen sein, soll an dieser Stelle aber nicht weiter thematisiert werden. Der Status der Regel wird zyklisch von der Schedulingkomponente überwacht.

Trigger Zurücksetzen Meldet die aktivierte Regel zurück, dass sie ihre Transformationsschritte abgeschlossen hat, so wird ihr Trigger sowie alle zuvor in einen Fehlerzustand geratenen Trigger zurückgesetzt.

Trigger aktivieren Anschließend wird die zuvor angehaltene Bearbeitung der Trigger weitergeführt. Trigger die vor dem Anhalten bereits eine positive Rückmeldung geliefert haben, werden direkt erkannt und Schritt 2 wird ausgelöst.

Trigger können unter anderem auch durch die erfolgreiche Abarbeitung einer anderen MT-Regel anschlagen. Dadurch können Transformationsschritte gezielt nacheinander ausgeführt werden. Auch ein simples Prioritäten-Konzept lässt sich durch die Trigger abbilden, indem die Schedulingkomponente jeweils die höchstpriore MT-Regel mit positivem Trigger zur Bearbeitung anstößt. Sollen bestimmte Regeln dauerhaft aktiv sein, so kann ein Trigger realisiert werden, der nur ein `Wahr`-Objekt enthält und somit immer positive Rückmeldung liefert.

Für die Aktorregel aus Abbildung 6.6c bietet sich ein Trigger an, der die Regel zur Übersetzung der `SystemUnitClass` auf erfolgreiche Bearbeitung überwacht. Schlägt dieser Trigger an, hängt die Schedulingkomponente die Regel zusammen mit allen anderen Regeln deren Trigger angeschlagen haben in eine Warteschlage. Anschließend wird die erste Regel der Warteschlange aktiviert, d.h. ihre MT-Elemente werden in das Tasking der SPS eingehängt und somit ihre interne Logik zur Ausführung gebracht.

6.6 Referenzimplementierung

Die Referenzimplementierung von ACPLT/MT erfolgte auf Basis von ACPLT/FB-Funktionsbausteinen (kurz FB) [NN04]. Als Laufzeitsystem kam iFBspro [NN04] zum Einsatz. Durch Ähnlichkeit von ACPLT/FB-Funktionsbausteinen zu den Funktionsblockdiagrammen der IEC 61131 und die für ACPLT-Modelle typischen Eigenschaften der Introspektion und Selbstmanipulation bieten FBs eine geeignete Basis für die Umsetzung von ACPLT/MT. Analog zu den Funktionsblockdiagrammen der IEC 61131 wird die interne Logik der FBs mit einer textuellen Sprache, in diesem Fall C, programmiert. Einzelne FBs werden über ihre Ein- und Ausgänge miteinander verknüpft und stellen dadurch komplexere Automatisierungsfunktionen zur Verfügung.

Um die Echtzeitfähigkeit der Automatisierungsfunktionen gewährleisten zu können, wird die Bearbeitung des gesamten Programmcodes in Zyklen mit fester Zykluszeit durchgeführt. Das Tasking der einzelnen FBs erfolgt dabei anhand einer Taskliste, die pro Zyklus einmal durchlaufen wird und jedem FB die Möglichkeit gibt, einen Teil seiner Logik zu bearbeiten. FBs mit zeitaufwändiger Funktionalität müssen diese in Teilaufgaben zerlegen, um die Echtzeitfähigkeit des Systems nicht zu gefährden. Die Teilaufgaben müssen dabei so gestaltet sein, dass eine Unterbrechung und eine Bearbeitung der anderen FBs der Taskliste keine ungewollten Seiteneffekte auf die Gesamtfunktionalität haben.

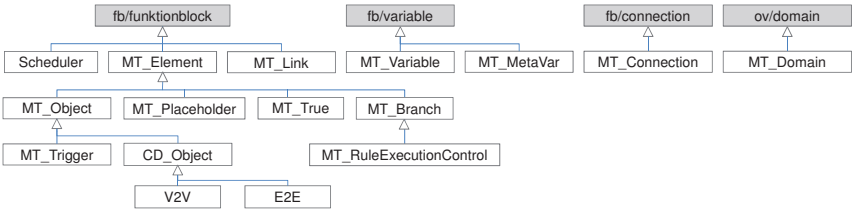


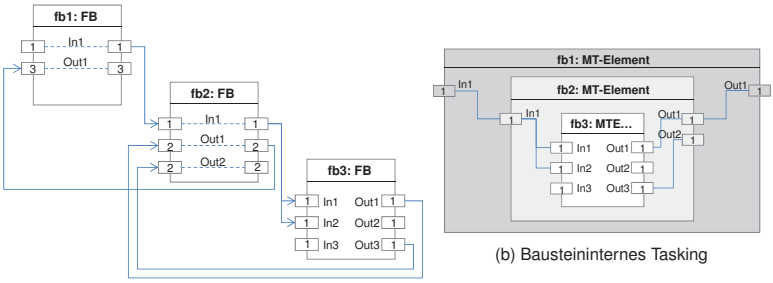
Abbildung 6.10: Klassendiagramm ACPLT/MT

Die Referenzimplementierung von ACPLT/MT umfasst die in Abbildung 6.10 aufgeführten Klassen. Die Klassen `MT_Element` und `MT_Link` sind als Funktionsblöcke realisiert. Variablen dieser beiden Klassen können daher als Ein-/Ausgänge oder lokale Variablen (`fb/variable`) deklariert werden, so dass sie vom iFBspro als miteinander verknüpfbare Ports dargestellt werden. Es bietet sich daher an MT-Variablen und Metavariablen als Ports zu implementieren. Um eine besser Unterscheidung während der Regelbearbeitung zu ermöglichen wird hier eine starke Typisierung mittels eigener Unterklassen vorgenommen. Auch für die Verbindung zwischen zwei Ports (`fb/connection`) wird eine spezielle Unterklasse bereitgestellt um sicherzustellen, dass nur jeweils zwei Objekte von Typ `MT_Variable` oder zwei Objekte vom Typ `MT_MetaVar` über sie miteinander verbunden werden können. Zudem wurde eine Klasse `Trigger` eingefügt, die eine Unterklasse von `MT_Objekt` bildet. Sie erlaubt keine Kaskadierung und sichert eine zustandslose Ausführung der internen Logik zu. Zur besseren Strukturierung des ACPLT/MT-Frameworks im Laufzeitsystem wird die Klasse `MT_Domain` bereitgestellt. Sie dient als „Ordner“ in der Baumstruktur und besitzt keine eigene Logik.

Bevor die einzelnen Klassen und ihre Funktionsweise im Detail vorgestellt werden, soll zunächst auf die Limitierungen von ACPLT/FB und die Auflösung dieser Limitierung eingegangen werden.

6.6.1 Taskingkonzept

Anders als bei FBDs ist es in ACPLT/FB nicht möglich, Bausteine in die Ausführungslogik eines anderen Bausteins einzubetten, so dass dieser vor- und nachher eigenen Code ausführen kann. Stattdessen werden alle FBs baumartig in eine Taskliste eingehängt und erhalten pro Zyklus genau einmal die Kontrolle.



(a) Globales Tasking

(b) Bausteininternes Tasking

Abbildung 6.11: Tasking ohne und mit bausteininternem Tasking

Erhält ein FB einen Zeitslot zum Bearbeiten seiner internen Logik, so wird zunächst das Prozessabbild der Eingangsvariablen mit Daten versorgt, anschließend wird die Bausteinlogik abgearbeitet und die Werte ins Prozessabbild der Ausgangsvariablen geschrieben. Dies bringt insbesondere bei kaskadierten FBs zeitliche Verzögerungen mit sich. Abbildung 6.11a zeigt eine solche Kaskadierung. Die interne Logik der dargestellten Bausteine sei der Einfachheit halber so implementiert, dass die Daten der Eingänge unverändert an die jeweiligen Ausgänge weitergegeben werden. Während die Daten von `fb1` über `fb2` nach `fb3` innerhalb des ersten Zyklus durchgereicht werden, benötigt die Datenweitergabe in die andere Richtung drei Zyklen. Im ersten Zyklus stehen die Werte für den zweiten und dritten Eingang von `fb2` noch nicht zur Verfügung, da `fb3` noch nicht ausgeführt wurde und somit die Werte noch nicht bereitstellen konnte. Erst im zweiten Zyklus kann `fb2` auf diese Werte zugreifen und sie als Ausgänge zur Verfügung stellen. Da zum Zeitpunkt der Bearbeitung von `fb2` der Baustein `fb1` jedoch schon bearbeitet wurde, stehen ihm diese Daten erst im dritten Zyklus zur Verfügung. Pro Verbindung zwischen FBs entgegen der Reihenfolge in der Taskliste werden die Daten somit um einen Zyklus verzögert. Dies kann gewollt sein, wenn zum Beispiel ein Speicher implementiert werden soll, bei der Weitergabe von Stati wie sie für ACPLT/MT benötigt wird, geht eine solche Verzögerung jedoch deutlich zu Lasten der Performance.

Als Alternative steht in vielen Leitsystemen die Möglichkeit der Schachtelung von Funktionsbausteinen zur Verfügung (vgl. Abbildung 6.11b). Erhält nun `fb1` einen Zeitslot zur Bearbeitung seiner internen Logik, so wird wiederum zunächst das Prozessabbild der Eingangsvariablen mit Daten versorgt und anschließend wird die Bausteinlogik, genauer gesagt das `PreTasking` abgearbeitet. Bevor jedoch das Prozessabbild der Ausgangsvariablen geschrieben wird, wird die Kontrolle an `fb2` abgegeben, der seinerseits seine Logik abarbeiten kann. Erhält `fb1` die Kontrolle von `fb2` zurück, kann weiterer `fb1`-Code, das `PostTasking` ausgeführt werden, bevor abschließend das Prozessbild der Ausgänge geschrieben wird. Durch diese Unterbrechung in der Ausführung der Bausteinlogik können Daten von außen nach innen und von innen nach außen innerhalb eines Zyklus weitergegeben werden.

Um gekapselte Funktionsbausteine auch für ACPLT/MT einsetzen zu können, wurde das Taskingkonzept von ACPLT/FB in der Unterklasse `MT_Element` überschrieben. Jedes `MT_Element` erhält eine eigene interne Taskliste und ein eigens rudimentäres Scheduling, dass nach der

Bearbeitung des internen Codes die Teilnehmer der Taskliste nacheinander zur Ausführung bringt. Außerdem wurde die Tatsache ausgenutzt, dass sowohl die Taskliste als auch die Kaskadierung von FBs mit Hilfe von Links realisiert sind. Wird ein neues `MT_Element` in einer Kaskade angelegt, so löscht der Konstruktor den Link zur globalen Taskliste und erstellt einen neuen Link zur Taskliste des in der Kaskade vorgelagerten MT_Element-Objektes. Eine Kaskade von MT_Element-Objekten wird dadurch automatisch in geschachtelte FBs umgewandelt.

Beispiel 6.2 *Abbildung 6.11b zeigt das bei ACPLT/MT realisierte interne Tasking. Beim Anlegen von `fb3` löscht dieser selbstständig seine Zuordnung zur globalen Taskliste und hängt sich in die Taskliste von `fb2` ein. Während der Bearbeitung von `fb2` versorgt dieser im PreTasking unter anderem die Eingänge von `fb3` mit Daten. Anschließend wird die Kontrolle an `fb3` übergeben und nach dessen Bearbeitung werden die Ausgänge von `fb3` im PostTasking verarbeitet und ggf. die Werte an die Ausgänge von `fb2` weitergereicht.*

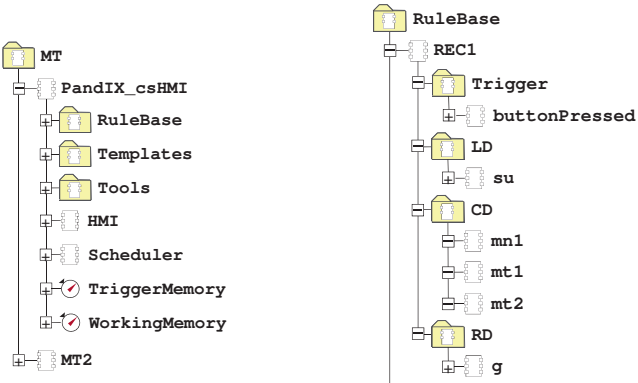
Die in dieser Arbeit gewonnenen Erfahrungen mit der Kapselung von Funktionsbausteinen sind in die Entwicklung von ACPLT/FC und ACPLT/SCC eingeflossen. Mit diesen Modellen stehen mittlerweile auch in der ACPLT-Modellwelt schachtelbare Funktionsblockdiagramme und Sequenzdiagramme für das Engineering zur Verfügung.

6.6.2 ACPLT/MT-Framework im Laufzeitsystem

Abbildung 6.12 zeigt die Umsetzung des ACPLT/MT-Framework im iFBSpro. Generell erlaubt ACPLT/MT parallel mehrere Modelltransformationen im Laufzeitsystem, von denen aber maximal eine gleichzeitig aktiv sein darf.

Jede Modelltransformation besteht aus einer Regelbasis, der Template-Datenbank und einem Satz an Werkzeugen, wie zum Beispiel Funktionsbausteine für das Anlegen und Löschen von Objekten bzw. Links. Außerdem besitzt jede Modelltransformation ihre eigene Schedulingkomponente und eine zugehörige Visualisierung, die den Zustand des Schedulers in Form eines Sequenzdiagramms zugänglich machen (vgl. Abbildung 6.13). Schlussendlich stehen noch zwei getrennte Tasklisten für die Trigger und die operationalen Regeln zur Verfügung. Den einzelnen Komponenten übergeordnet ist das Datenbankobjekt (vgl. `PandIX_csHMI` bzw. `MT2` in Abbildung 6.12a), das unter anderem die Datenbankeinträge zu den Korrespondenzobjekten enthält. Diese Informationen können auch zwischen verschiedenen Transformationen ausgetauscht werden, so dass aufeinander aufbauende Transformationen entstehen. Ein Beispiel dafür ist Anwendungsszenario S1, das Einfärben der Rohrleitungen, die von einer Aktorbedienung betroffen sind. Die dazu gehörige Modelltransformation findet zwischen ACPLT/PandIX und dem ACPLT/csHMI statt. Diese werden durch die Transformation aus Anwendungsszenario S2 erstellt und durch Korrespondenzobjekte dokumentiert. Anwendungsszenario S1 ist daher auf die Informationen aus Anwendungsszenario S2 angewiesen.

Die Regelbasis von ACPLT/MT enthält alle MT-Regeln der Transformation. Abbildung 6.12b zeigt den Aufbau einer solchen Regel im Laufzeitsystem. Das Kopfelement wird durch ein Objekt vom Typ `RuleExecutionControl` (kurz `REC`) realisiert. Dieses ist während der Regelausführung dafür zuständig, die Teilproduktionen mit Daten zu versorgen und ihre Bearbeitung



(a) Domainstruktur ACPLT/MT

(b) MT-Regel: PLT-Stelle anlegen

Abbildung 6.12: ACPLT/MT im Laufzeitsystem

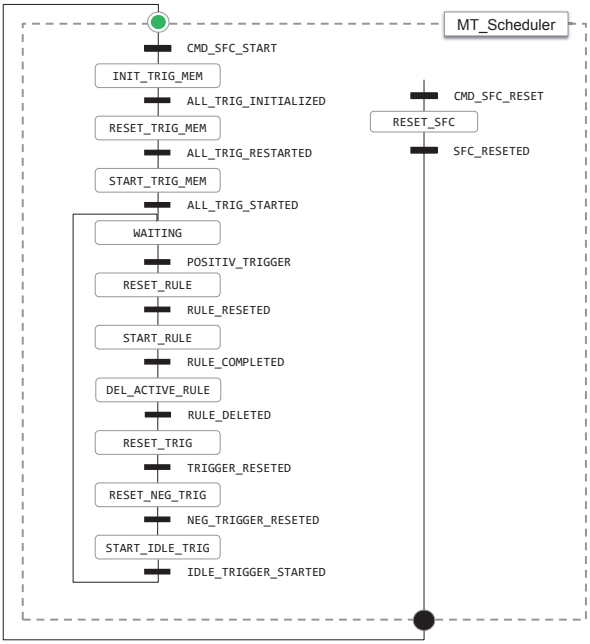


Abbildung 6.13: ACPLT/MT-Scheduling

anzustoßen. Zudem registrieren alle MT-Elemente der Regel die Pfade zu gebundenen oder erstellten Modellobjekten beim REC. Wurde ein Übersetzungsschritt erfolgreich durchgeführt, meldet der REC anhand der registrierten Modellobjekte dem Datenbankobjekt die neu übersetzten Objekte.

Zunächst liegen die MT-Regeln jedoch als inaktive Objektnetzwerke im System vor, d.h. sie hängen in keiner Taskliste und ihre interne Logik kommt nicht zur Ausführung. Wird eine Transformation durch das Setzen des Kommandos CMD_SFC_START am Kommandoeingang des Scheduler aktiviert, so hängt dieser die Triggerobjekte aller Regeln der Regelbasis in die Taskliste WorkingMemory, setzt die Eingänge der Trigger zurück und aktiviert die zyklische Bearbeitung der Trigger über deren Kommandoeingang (vgl. Abbildung 6.13). Sobald ein Trigger eine positive Rückmeldung liefert, wird der zugehörige REC in die Taskliste WorkingMemory eingehängt und erhält vom Scheduler den Befehl CMD_RESET. Dieser wird weitergereicht an alle MT_Elemente der Regel und bewirkt das Rücksetzen aller Ausgangsvariablen auf ihren Ursprungswert. Anschließend wird die Bearbeitung der Regel durch das Kommando CMD_START gestartet. Ist die Regel vollständig abgearbeitet, gibt der REC die Rückmeldung Ready. Daraufhin kann der Scheduler die Regel aus dem WorkingMemory löschen. Außerdem wird der zugehörige Trigger im TriggerMemory zurückgesetzt und mit einer neuen Überwachung beauftragt. Auch Trigger, die zuvor einen Fehler zurückgegeben haben, werden zurückgesetzt und neu gestartet.

Wird zu einem beliebigen Zeitpunkt ein Zurücksetzen der Transformation (CMD_SFC_RESET) angefordert, werden zunächst das TriggerMemory sowie das WorkingMemory geleert und anschließend die Modelltransformation wieder deaktiviert.

6.6.3 MT_Element

Um die in Abschnitt 6.6.1 thematisierte unverzögerte Rückmeldung realisieren zu können, wurde die zyklisch aufgerufenen Methode typemethode der Basisklasse fb/functioblock wie in Abbildung 6.14 gezeigt erweitert. Die eigentliche Bausteinfunktionalität wird in Pre- und Posttasking-Funktionalität unterteilt und in die Methoden mtLib_MTElement_preintask und mtLib_MTElement_postintask ausgelagert. Dadurch können das Pre- und Posttasking in abgeleiteten Klassen gezielt überschrieben werden. Zwischen den beiden Methodenaufrufen erfolgt die Bearbeitung der internen Taskliste, die die eingebetteten Funktionsbausteine enthält.

Die eigentliche Funktionalität von MT_Element und davon abgeleiteter Klassen wird mit Hilfe eines Sequenzdiagramms realisiert. Einen ausführlichen Einblick in die Funktionalität von MT_Element und davon abgeleiteten Klassen sowie eine Schritt-für-Schritt-Erläuterung dieses Sequenzdiagramms bietet Anhang C.

Da die ACPLT-Modellwelt zum Implementierungszeitpunkt von ACPLT/MT noch keine Realisierung für Sequenzdiagramme besaß, wurde die interne Logik mittels CASE-Anweisung in C implementiert und aufbauend darauf mit ACPLT/csHMI eine Visualisierung als SFCs erstellt (vgl. Abbildung 6.15). Die einzelnen Schritte und Transitionen wurde jeweils als eigene Metho-

```
OV_DLLFNCEXPORT void mtLib_MTElement_typemethod
(OV_INSTPTR_fb_functionblock pfb, OV_TIME * pltc)
{
    ...

    /* Execute own logic */
    mtLib_MTElement_preintask(pfb,pltc);

    /* Execute internal tasks */
    Ov_Call1 (fb_task, intask, execute, pltc);

    /* Execute another part of own logic */
    mtLib_MTElement_postintask(pfb,pltc);

    ...
}
```

Abbildung 6.14: Realisierung geschachtelter FBs

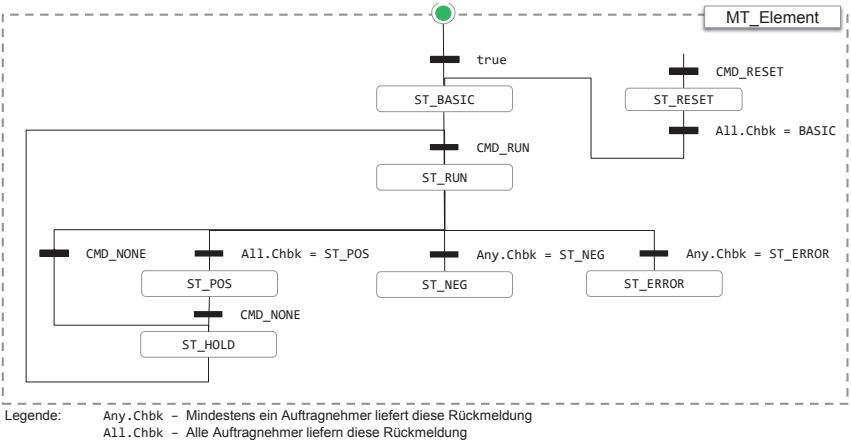


Abbildung 6.15: SFC des MT-Element

den realisiert, um sie in abgeleiteten Klassen gezielt überschreiben zu können. Die Bearbeitung der Schritte, sowie der Transitionen, die sich auf Kommandos beziehen erfolgt im `PreTasking` und die Auswertung der Transitionen die sich auf Rückmeldungen von Auftragnehmer beziehen im `PostTasking`. Dadurch können innerhalb eines Zyklus Rückmeldungen von Auftragnehmern direkt im `PostTasking` und somit in der Auswertung der Transitionen berücksichtigt werden. Wird die Bedingung einer Transition erfüllt, so führt dies direkt zu einem Schrittwechsel, was sich in der Metavariablen `State` widerspiegelt. Dadurch können Rückmeldungen von den eingebetteten Auftragnehmern noch im gleichen Zyklus an den eigenen Auftraggeber weitergereicht werden.

Insgesamt besteht die Funktionalität der Basisklassen `MT_Element` aus sieben Schritten. Im Basisschritt wird keine Logik ausgeführt, der Baustein ruht und wartet auf eingehende Kom-

mandos von Auftraggeber. Kommt der Befehl CMD_RUN wird mit der Bearbeitung der eigentlichen Bausteinfunktionalität begonnen. Die abgeleitete Klasse MT_Object führt in diesem Schritt zum Beispiel das Suchen bzw. Erstellen von Objekten durch. Im Normalfall wird die Logik des ST_RUN innerhalb eines Zyklus abgearbeitet. In den nachfolgenden Zyklen wird lediglich auf die Rückmeldung von eingebetteten Auftragnehmern oder das Eintreffen eines neuen Kommandos vom Auftraggeber gewartet. Erfolgt ersteres, dann wird je nach Art der Rückmeldung in die Schritte ST_POS, ST_NEG oder ST_ERROR gewechselt, was dem eigenen Auftragnehmer in Form der Metavariablen Status als Rückmeldung weitergegeben wird. Kommt vom Auftraggeber das Kommando CMD_NONE wird in den Schritt ST_HOLD gewechselt, der den aktuellen Zustand des MT_Element einfriert bis entweder erneut ein CMD_RUN oder ein Kommando zum Zurücksetzen vom Auftraggeber eingeht. Im Schritt ST_RESET, der jederzeit durch das Kommando CMD_RESET ausgelöst werden kann, werden alle Ein- und Ausgänge wieder auf den Ursprungszustand zurückgesetzt. Sobald auch alle eingebetteten Auftragnehmer ihre Werte zurückgesetzt haben, wird in den Basiszustand gewechselt.

Im Detail besitzen die Zustände folgende Ausführungslogik:

	PreTasking	PostTasking
ST_BASIC	Alle Ausgänge sind zurückgesetzt, der Baustein befindet sich im Basiszustand. Beim Aktivieren dieses Zustands wird an die nachfolgenden MT_Elemente der Befehl CMD_NONE gesendet. In den weiteren Zyklen wird auf einen Befehl vom Auftraggeber gewartet.	Der Zustand wird verlassen, sobald der Befehl CMD_RUN vom Auftraggeber kommt.
ST_RUN	In diesem Zustand wird die eigentliche Funktionalität des MT_Element ausgeführt. Es werden je nach Parametrierung Objekte in den Modellen gesucht oder erstellt. Anschließend erhalten die Auftragsnehmer ebenfalls das Kommando CMD_RUN.	In diesem Zustand wird verblieben, bis die Auftragsnehmer eine Rückmeldung über den Verlauf ihrer Bearbeitung liefern. Der rückgemeldete Zustand wird direkt an den eigenen Auftraggeber weitergereicht.
ST_POS	Alle Ausgänge bleiben in diesem Zustand unverändert. Beim Aktivieren dieses Zustands wird an die eingebetteten MT_Element der Befehl CMD_NONE gesendet. In den weiteren Zyklen wird auf einen Befehl vom Auftraggeber gewartet.	Der Zustand kann nur durch den Befehl CMD_RESET oder CMD_HOLD verlassen werden.

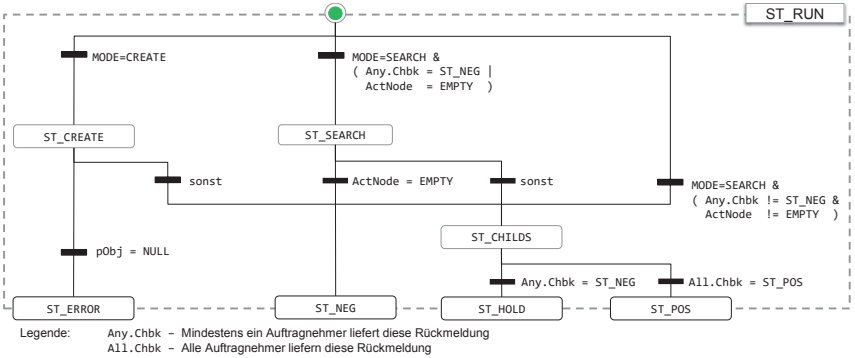


Abbildung 6.16: Verfeinerter SFC des Schrittes ST_RUN im MT_Object

	PreTasking	PostTasking
ST_HOLD	Die Bearbeitung des Bausteins wird unterbrochen. Die historischen Daten bleiben jedoch erhalten. Die eingebetteten Auftragnehmer bekommen den Auftrag `CMD_NONE` und gehen dadurch ebenfalls in den Zustand `ST_HOLD`.	Bekommt der Baustein den Befehl `CMD_RUN` so wird in den Zustand `ST_RUN` gewechselt und im nächsten Zyklus die Bearbeitung unter Berücksichtigung der historischen Daten wieder aufgenommen.
ST_NEG	siehe `ST_POS`	Der Zustand kann nur über den Befehl `CMD_RESET` verlassen werden.
ST_ERROR	siehe `ST_POS`	siehe `ST_NEG`
ST_RESET	Die Bearbeitung des Bausteins wird unterbrochen. Die historischen Daten des Bausteins werden zurückgesetzt. Eine erneute Suche startet wieder am Anfang des Suchraums. Alle Auftragnehmer erhalten ebenfalls den Befehl `CMD_RESET`.	Wurden alle Auftragnehmer erfolgreich zurückgesetzt, wird in den Ausgangszustand `ST_BASIC` gewechselt.

6.6.4 MT_Object

Die Klasse `MT_Object` überschreibt den Schritt `ST_RUN` der Basisklasse und unterteilt ihn in drei Teilschritte (vgl. Abbildung 6.16). Im Schritt `ST_SEARCH` wird nach einem passenden Modellobjekt gesucht. Die Suche ist dabei als Tiefensuche implementiert. Den Startpunkt der Suche erhält das `MT_Object` über die Metavariable `Pfad`. Diese stellt einen absoluten Pfad zu einem Modellobjekt als Zeichenkette bereit und wird dazu genutzt, einen Link auf das entspre-

chende Objekt zu generieren. Anschließend wird das Modellobjekt dahingehend untersucht, ob es entsprechend dem Wert der Metavariable Assoc verlinkt ist. Gefundene Links dieses Typs werden verfolgt und das über das Ziel bzw. die Quelle des Links als potentieller Kandidat für die aktuelle Suche in Betracht gezogen. Stimmen Klasse und Objektname mit den Werten der Metavariablen RepClass und ObjName überein und können auch die MT_Variable vom Typ und Wert her auf die Variablen des potentiellen Kandidaten abgebildet werden, so muss abschließend noch überprüft werden, ob das gefundene Objekt schon übersetzt wurde. Dies erfolgt durch eine Abfrage beim Datenbankobjekt, das alle bereits übersetzten Modellobjekte nachhält. Der Metavariablen-Ausgang ActNode wird mit dem Pfad des gefundenen Objektes belegt und dient damit aus Startpunkt für eingebettete MT_Object. Liefern diese eine positive Rückmeldung, so wurde eine Entsprechung im Modell gefunden. Da weitere Modellstrukturen auf die eingebetteten Objekte passen können, werden sie mit gleicher Parametrierung erneut mit der Bearbeitung ihrer internen Logik beauftragt. Erst wenn die eingebetteten Objekte eine negative Rückmeldung geben, kann keine weitere Objektstruktur im Modell gefunden werden und es wird der nächste potentielle Kandidat für das lokale MT_Object gesucht. Die Metavariable ActNode dient dabei als Speicher, welches Modellobjekt als letztes gefunden wurde und die Suche nach weiteren über RepAssoc verbundenen potentiellen Kandidaten kann fortgeführt werden. Dabei wird ausgenutzt, dass Links in ACPLT als geordnete Listen an den verlinkten Objekten, also auch an dem durch Pfad referenzierten Modellobjekt zur Verfügung stehen.

Im Schritt ST_CREATE wird zunächst wie im Schritt ST_SEARCH nach einem passenden Modellobjekt gesucht. Ist ein solches Objekt vorhanden, werden nur ggf. abweichende Variablenwerte angepasst. Ist ein solches Modellobjekt nicht vorhanden, wird ein Objekt der Klasse RepClass angelegt und mit Hilfe eines Links vom Typ RepAssoc mit dem Modellobjekt verknüpft, das über die Metavariable Pfad referenziert wird. Außerdem werden die Variablen mit den Werten entsprechend der MT_Variablen des MT_Object belegt. Gehen vom aktuellen MT_Object oder dessen MT_Variablen Links aus, so wird überprüft, ob das Gegenstück des Links bereits gebunden wurde. Ist dies der Fall, so wird ein Link entsprechend der Parametrierung im Modell angelegt. Auch im Schritt ST_CREATE gibt die Metavariable ActNode Informationen über das gebundene Modellobjekt.

War die Suche bzw. das Erstellen erfolgreich, so werden die eingebetteten MT_Object im Zustand ST_CHILDS bearbeitet und auf Rückmeldung von ihnen gewartet.

Zusammenfassend besitzen die Schritte und Transitionen des MT_Object folgende Logik:

	PreTasking	PostTasking
ST_RUN	Anhand der Metavariablen Context und SourceDom wird der Modus des MT_Object auf suchend (SEARCH) oder erstellend (CREATE) gesetzt.	Je nach aktuellem Modus wird in den Zustand ST_CREATE oder ST_SEARCH gewechselt

	PreTasking	PostTasking
ST_CREATE	Das durch die Metavariable `Path` referenziere Objekt wird als Startpunkt für die Suche nach einem entsprechend der `AssocClass` verlinkten Modellobjekt genutzt, das den Anforderungen nach Klasse (`RepClass`), Variablentypen und Variablenwerten entspricht. Kann ein solches Modellobjekt nicht gefunden werden, so wird es angelegt, der Wert der Variablen gesetzt und ggf. benötigte Verbindungen erstellt.	Trat bei der Erstellung ein Fehler auf, wird in den Zustand `ST_ERROR` gewechselt. Ansonsten wird der Auftrag an die Auftragnehmer weitergeleitet und in den Zustand `ST_CHILDS` gewechselt.
ST_SEARCH	Ausgehend von dem durch den Parameter `Path` referenzierten Objekt wird die nächste, noch nicht bearbeitete Instanz der Assoziation `AssocClass` gesucht und ihr Gegenpart dahingehend überprüft, ob er dem repräsentierten Objekt entspricht. Wurde ein passendes Objekt gefunden, wird es beim `REC` überprüft und registriert, um zu vermeiden, dass ein Objekt doppelt gebunden wird. Der Ausgang `ActNode` wird mit dem Pfad des gefundenen Objektes belegt.	War die Suche erfolglos und der Ausgang `ActNode` enthält keinen Objektpfad, wird in `ST_NEG` gewechselt, wird in den Zustand `ST_CHILDS` gewechselt.
ST_CHILDS	An alle Auftragnehmer wird der Befehl `CMD_RUN` geschickt, um ihre Bearbeitung anzustoßen.	Kommt eine negative Rückmeldung vom Auftragnehmer, wird die Bearbeitung unterbrochen. Eine positive Rückmeldung erlaubt einen Wechsel in den Zustand `ST_POS`.
ST_RESET	Zusätzlich zu der Funktionalität aus der Basisklasse wird bei gesetztem `ActNode` das Objekt noch beim `REC` de-registriert.	keine Änderung zur Basisklasse

Die Logik der übrigen Bausteine hält sich weitestgehend an die der Basisklasse `MT_Element`.

True Erhält der Baustein in den Schritten `ST_BASIC` oder `ST_HOLD` den Befehl `CMD_RUN`, so wird direkt in `ST_POS` gewechselt. Der negative Fall sowie der Fehlerfall können bei diesem Baustein nie erreicht werden.

Not Erfolgt durch den Auftragnehmer die Rückmeldung `ST_POS`, so liefert der Baustein an seinen eigenen Auftraggeber die Rückmeldung `ST_NEG` und umgekehrt. Alle anderen Schritte und Transitionen bleiben unberührt.

Placeholder Im Schritt ST_RUN wird eine Kopie des referenzierten Templates aus der Template-Datenbank der Modelltransformation als eingebetteter Funktionsbaustein des Platzhalters erstellt und die Ein- und Ausgänge verknüpft. Da die Ersetzung eines Platzhalters erst erfolgt, wenn der Kontrollfluss den entsprechenden Modifikator aktiviert, gibt es Platzhalter, die während einer Regelbearbeitung nie ersetzt werden. Dies ist zum Beispiel der Fall, wenn vor Erreichen des Platzhalters ein Fehler auftritt oder wenn bei einer Verzweigung sich der Platzhalter in der inaktiven Alternative befindet. Diese Lazy-Evaluation verringert die Anzahl der Objekte und Kopiervorgänge im Laufzeitsystem und ist daher platz- sowie ressourcensparend. Dies kommt dem Gesamtkonzept in doppelter Hinsicht entgegen.

Im Zustand ST_RESET wird die Kopie des Templates wieder gelöscht.

Trigger Statt dem Zustandswechsel nach ST_NEG werden die Ausgänge direkt gelöscht und im Zustand ST_RUN verblieben. Dies erlaubt die Erkennung von einmaligen Events, da kein Zyklus durch das Zurücksetzten des Bausteins verloren geht. Alle anderen Schritte und Transitionen werden unverändert von MT_Object übernommen.

Branch Die Verzweigung kann sowohl als ODER als auch als UND fungieren. Die Parametrierung nach ODER/UND muss in Quell- und Zieldomäne identisch sein. Bei einer ODER-Verzweigung erfolgt die Tiefensuche zunächst in der ersten Alternative und erst wenn diese eine negative Rückmeldung liefert, in der zweiten Alternative. Es wird daher zunächst der Auftragnehmer in der ersten Alternative angestoßen. Liefert dieser ST_NEG zurück, so erhält der Auftragnehmer in der zweiten Alternative den Befehl CMD_RUN. Erst wenn auch dieser eine negative Rückmeldung gibt, wechselt die Verzweigung in den Schritt ST_NEG. Fehlermeldungen werden weiterhin direkt an den eigenen Auftraggeber weitergereicht. In der Zieldomäne erhält die ODER-Verzweigung mit Hilfe des Korrespondenzgraphs Auskunft über die in der Quelldomäne aktive Alternative und stößt seinerseits nur die Bearbeitung des entsprechenden Auftragnehmers an. Unabhängig von der Domäne werden bei einer UND-Verzweigung beide Alternativen gleichzeitig angestoßen. In den Zustand ST_POS wird erst gewechselt, wenn beide Auftragnehmer diese Rückmeldung geben. Für den Wechsel in ST_NEG reicht die entsprechende Rückmeldung einer der beiden Auftragnehmer.

CD_Object Die Klasse für die Korrespondenzobjekte schreibt im Zustand ST_RUN die Transformationsdaten in ein Array des Kopfelements. Dabei werden neben den gebundenen Objekten der Quell- und Zieldomäne auch den Korrespondenztyp (E2E/V2V) sowie der Zeitstempel protokolliert. Nach erfolgreicher Bearbeitung eines kompletten Transformationsschrittes, werden diese Daten vom Kopfelement in das Datenbankobjekt übertragen, wo sie dauerhaft nachgehalten werden.

6.6.5 Metavariablen, Variablen und Links

Die Ports von ACPLT/FB-Funktionsbausteinen sind eigenständige Objekte und erfüllen alle Anforderungen an die Realisierung von Metavariablen. Aus diesem Grund werden Metavariablen direkt durch Ports realisiert und sind daher in der Signatur von MT_Element sichtbar (vgl. Abbildung 6.17). Die Verwendung von Ports zur Realisierung von MT-Parametern bringt eine

```
CLASS MT_Object : CLASS mtLib/MT_Element IS_INSTANTIABLE;
  COMMENT = "Represents an object of any domain in an ACPLT/MT-Pattern";
  VARIABLES
     RepLib                 :   STRING    HAS_ACCESSORS    FLAGS = "i"
        COMMENT = "Represented Library Element"
        INITIALVALUE = "*";
     RepClass               :   STRING    HAS_ACCESSORS    FLAGS = "i"
        COMMENT = "Represented Class Element"
        INITIALVALUE = "*";
     RepIdent               :   STRING    HAS_ACCESSORS    FLAGS = "i"
        COMMENT = "Represented Class Element"
        INITIALVALUE = "*";
     AssocLib               :   STRING    HAS_ACCESSORS    FLAGS = "i"
        COMMENT = "Lib of connection from principal"
        INITIALVALUE = "ov";
     AssocClass             :   STRING    HAS_ACCESSORS    FLAGS = "i"
        COMMENT = "Class of connection from principal"
        INITIALVALUE = "containment";
     AssocRole              :   UINT      HAS_ACCESSORS    FLAGS = "i"
        COMMENT = "Role of current Object 1 = Child, 2 = Parent, 3 = *"
        INITIALVALUE = 1;
    ...
    END_VARIABLES;
  ...
END_CLASS;
```

Abbildung 6.17: Signatur der Klasse `MT_Element`

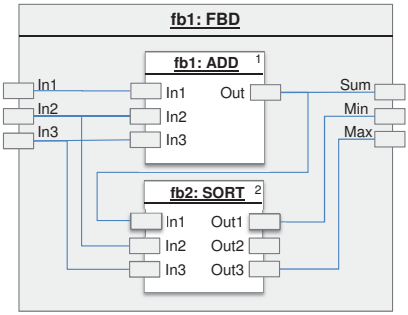
übersichtliche Darstellung aller Metavariablen als Ein- und Ausgänge eines `MT_Element` in iFBspro mit sich.

Im Vergleich zu Metavariablen benötigen `MT_Variablen` weitergehende Parametrierungsmöglichkeiten für den Datentyp und den erwarteten Wert der repräsentierten Variablen. Aus diesem Grund wurde für `MT_Variablen` eine entsprechende Erweiterung auf Basis der ACPLT/FB-Ports vorgenommen und eine spezielle Assoziation `MT_Connection` bereitgestellt, mit Hilfe derer man zwei `MT_Variable`n gerichtet miteinander verbinden kann. Eine `MT_Connection` repräsentiert eine entsprechende Verbindung zwischen Variablen.

Assoziationen, die entlang der Kommandostruktur verlaufen, werden wie in Abbildung 6.17 zu sehen ist, über die Metavariablen `AssocLib`, `AssocClass` und `AssocRole` repräsentiert. Alle anderen Assoziationen, die nicht durch die Kommandostruktur abgedeckt sind, werden mit Hilfe eines `MT_Assoc`-Objektes repräsentiert. Mittels einer speziellen Assoziation werden die `MT_Assoc`-Objekte mit den entsprechenden MT-Objekten verknüpft.

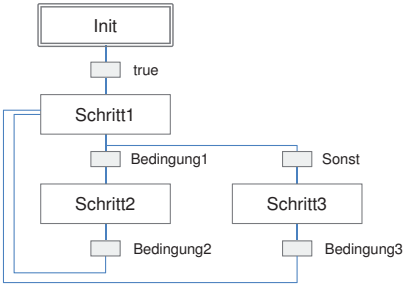
6.7 IEC 61131 basierte Modelltransformation

Die Sprachen der IEC 61131 sind imperative Sprachen, deren Funktionalität in zyklisch aufgerufenen Programmblöcken (engl. program organization unit, POU) gekapselt wird. Die interne Logik von POUs wird mit Hilfe der in der IEC 61131 beschriebenen Sprachen realisiert. In dieser Arbeit kommen dabei insbesondere die pascalähnliche Sprache des Strukturierten Text (engl. Structured Text, ST), die an Zustandsautomaten angelehnten Sequenzdiagramme (engl.




```
doSomethingSpecial();

Sum := fb1( In1 := THIS^.In1,
            In2 := THIS^.In2,
            In3 := THIS^.In3 );

fb2.In1 := Sum.Out;
fb2.In2 := THIS^.In2;
fb2.In3 := THIS^.In3;

fb2();

THIS^.Minimum := fb2.Out1;
THIS^.Maximum := fb2.Out3;

doSomethingSpecial();
```

(a) FBD vs. ST




```
CASE AktuellerSchritt OF
   1: Schritt1();
      IF Bedingung1 THEN
         AktuellerSchritt := 2;
      ELSE
         AktuellerSchritt := 3;
      END_IF
   2: Schritt2();
      IF Bedingung2 THEN
         AktuellerSchritt := 1;
      END_IF
   3: Schritt3();
      IF Bedingung3 THEN
         AktuellerSchritt := 1;
      END_IF
END_CASE
```

(b) SFC vs. ST

Abbildung 6.18: Sprachen der IEC 61131 im Vergleich

Sequential Function Chart, SFC) und die Funktionsblockdiagramme (engl. Function Block Diagram, FBD) zum Einsatz. In Abbildung 6.18 sind die grafischen Sprachen der entsprechenden ST-Umsetzung gegenübergestellt. Die Darstellung in ST in Abbildung 6.18a zeigt, dass vor und nach dem Aufruf der internen POUs weitere Bearbeitungsschritte erlaubt sind. Zudem ist in ST die Reihenfolge der Aufrufe klar ersichtlich. Diese Reihenfolge muss auch bei FBDs beachtet werden. Wird `fb2` vor `fb1` aufgerufen werden, so kann das Ergebnis von `fb1` erst einen Zyklus später durch `fb2` ausgewertet und in diesem Fall mit den neuen Werten der Eingänge `In2` und `In3` verglichen werden. Die Sprachen der IEC 61131 können gemeinsam in einem Programm eingesetzt werden. So kann, unabhängig von der aufrufenden POU, zur Realisierung der Funktionalität von `ADD` und `SORT` in Abbildung 6.18a, eine beliebige Sprache der IEC 61131 genutzt werden. In der IEC 61131 gibt es nur Verbindungen zwischen Variablen als Assoziationen. Andere Assoziationen müssen durch Zeigertypen realisiert werden. Verbindungen zwischen Variablen wirken wie Zuweisungen.

Für die direkte Realisierung von ACPLT/MT in den Sprachen der IEC 61131 fehlen diesen Sprachen zwei elementare Eigenschaften. Zum einen ist in der IEC 61131 keine Introspektion vorgesehen, so dass zur Laufzeit Modelle nicht durch MT-Elemente erkundet werden können. Zudem stellt die IEC 61131 keine dynamische Speicherverwaltung zur Verfügung. Das Anlegen von Objekten in der Zieldomäne lässt sich dadurch nicht realisieren. Auch wenn die Norm diese beiden Eigenschaften nicht vorsieht, so werden sie von immer mehr Automatisierungssystemen bereitgestellt. Insbesondere die Introspektion wird zurzeit durch einen anderen Trend geradezu forciert. Immer mehr Automatisierungssysteme setzen auf OPC/UA als Kommunikationsschnittstelle zwischen dem Automatisierungssystem und Komponenten außerhalb der Echtzeitumgebung. OPC/UA basiert aber auf dem Grundgedanken, dass ein Laufzeitsystem erkundbar ist. Eine Konsistenzanalyse auf Basis von ACPLT/MT lässt sich auf solchen Systemen bereits realisieren. Ein Ansatz für die Realisierung der Selbsterkundbarkeit ist die Ablage der Strukturinformationen als erkundbare Datei [TC3]. Alternativ können Bausteine mit einem Interface versehen werden, das eine Referenz auf den instanziierenden Baustein sowie Methoden zur Abfrage der eigenen Struktur beinhalten. Dieser interfacebasierte Ansatz lässt sich auch in Systemen realisieren, die sich strikt an die Vorgaben der IEC 61131 halten. Auch das dynamische Ändern und Anlegen von Funktionsbausteininstanzen zur Laufzeit wird von den ersten Automatisierungssystemen bereits angeboten [TC3]. Es ist also zu erwarten, dass sich die beschriebenen Konzepte in naher Zukunft in allen gängigen Automatisierungssystemen umsetzen lassen, auch wenn dies nicht explizit durch die IEC 61131 unterstützt wird.

Bei der Umsetzung von ACPLT/MT mittels eines IEC 61131 basierten Automatisierungssystems mit entsprechenden Möglichkeiten kann die interne Logik der MT-Elemente und der Scheduling-Komponente durch einen SFC realisiert werden. Oftmals wird jedoch auch eine `CASE`-Anweisung in ST für solche Zwecke genutzt (vgl. Abbildung 6.18b). Für die Beschreibung der MT-Regeln bietet sich ein Funktionsblockdiagramm an. Dies ist nur möglich, da die Sprachen der IEC 61131 frei gemischt werden können und so die interne SFC oder ST-Realisierung nach außen als ein FBD repräsentiert werden können. Das kommt auch zum Tragen bei der Realisierung der einzelnen Schritte und Transitionen der erstellten SFC. Hier bietet sich wiederum die Umsetzung in ST an.

7 Validierung

ACPLT/MT wurde bereits in mehreren Projekten zur Anwendung gebracht und konnte dabei gute Ergebnisse vorweisen. Insbesondere bei der Realisierung von Automatisierungsfunktionen für die Praktikumsanlage des Lehrstuhl für Prozessleittechnik der RWTH Aachen University und der Automatisierung eines modularen Elektro-Reduktionsofen (MERF) wurden verschiedenen Einsatzszenarien realisiert. Die dabei entstandenen Modelltransformationen spiegeln sich in den Anwendungsszenarien aus Kapitel 1.2 wieder. Im Folgenden soll erläutert werden, wieweit sich ACPLT/MT zur Realisierung der verschiedenen Szenarien eignete und an welchen Stellen das Konzept an seine Grenzen gestoßen ist. Anschließend wird die Passgenauigkeit von ACPLT/MT entlang der zuvor identifizierten informationstechnischen und automatisierungstechnischen Anforderungen an eine Modelltransformation analysiert.

7.1 S0 – Bereitstellung von Planungsdaten im Laufzeitsystem

Anwendungsszenario S0 realisiert den Datenabgleich zwischen den XML-basierten Planungsdaten und der ACPLT-Modellwelt. Als korrelierende Modelle kommen hierbei PandIX [PandIX] und ACPLT/PandIX zum Einsatz. Neben der bidirektionalen Auswertung von ACPLT/MT-Regel steht dabei die Anwendbarkeit auf Modelle im Vordergrund, die nicht in der ACPLT-Modellwelt beheimatet sind.

Die Verwendung von ACPLT/MT anstelle einer in das Automatisierungssystem integrierten Import-/Exportlösung bringt insbesondere bei nachträglichen Anpassungen der beteiligten Modelle sowie bei Erweiterungen der Funktionalität auf weitere Austauschmodelle signifikante Vorteile mit sich. Eine integrierte Import-/Exportfunktion bedingt bei Änderungen am Modell eine Anpassung des Laufzeitsystems und somit einen zeitweisen Ausfall der Anlage. In einer Anlage im Produktiveinsatz ist dies nicht realisierbar. Bei der Verwendung von ACPLT/MT müssen lediglich die Regeln angepasst werden. Dies kann ohne Unterbrechung der Basisfunktionalität und somit ohne Ausfallzeiten erfolgen. Die Performance-Nachteile gegenüber integrierten Import-/Exportfunktionen sind hingegen vernachlässigbar. Dies liegt insbesondere daran, dass ein Datenimport respektive -export vergleichsweise selten durchgeführt wird. Zudem ist die Bearbeitung eines solchen Datenaustauschs im Regelfall zeitunkritisch.

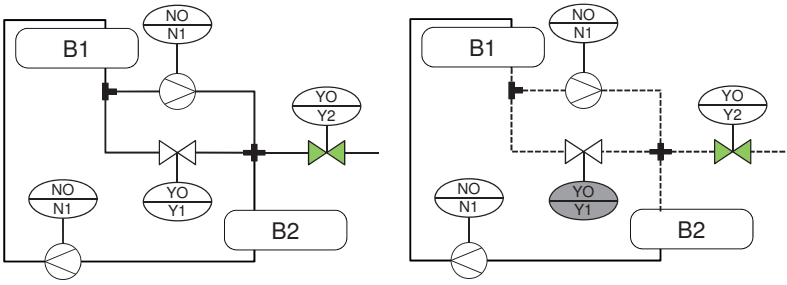
Durch die Fokussierung von ACPLT/MT-Elementen auf die Repräsentation von Objektnetzwerken war dieses Szenario nicht mit dem vorgestellten Basismodell realisierbar. Zur Umsetzung des Datenimports wurden daher spezielle ACPLT/MT-Objekte und ACPLT/MT-Variablen entwickelt, die den Inhalt von XML-Tags bzw. XML-Attributen je nach Transformationsrichtung auswerten oder schreiben. Die angepassten ACPLT/MT-Objekte erhalten als Eingabe einen

XML-Baum und suchen entsprechend ihrer Parametrierung darin Elemente mit den passenden Eigenschaften. Die identifizierten Teilbäume werden zur Weiterverarbeitung an die Auftragnehmer weitergereicht. Umgekehrt verläuft bei der erstellenden Abarbeitung die Generierung eines XML-Baums, der durch den eigenen Auftraggeber in eine größere XML-Struktur eingebettet wird. MT-Assoziationen und MT-Verbindungen kommen bei der Repräsentation von XML-Kontexten nicht zum Einsatz. Die vollständige TGG für dieses Anwendungsszenario findet sich in Anhang B. Es hat sich gezeigt, dass durch diese Erweiterung der ACPLT/MT-Bausteine beliebige XML-basierte Modelle in ACPLT/MT verarbeitet werden können. Durch die Verwendung weiterer modellspezifischen MT-Elemente kann auch eine generelle Unabhängigkeit von den verwendeten Planungswerkzeugen und den darin verwendeten domänenspezifischen Sprachen realisiert werden.

Trotz der nötigen Anpassungen an den MT-Objekten ist dieses Szenario prädestiniert für den vorgestellten Transformationsansatz mittels Tripel-Graph-Grammatiken. Änderungen in den Planungsdaten konnten zeitnah in die prozessleittechnische Laufzeitumgebung weitergegeben werden, Änderungen im Laufzeitsystem konnten in den Planungsdaten dokumentiert werden.

7.2 S1 – Einzelne Automatisierungsfunktion als Serienprodukt

Bei Anwendungsszenario S1 steht die Realisierung einer Flusswegkontrolle auf Basis der PandIX-Daten im Vordergrund. In einem ersten Teilszenario werden die Auswirkungen einer Aktorbedienung auf der Bedienoberfläche für den Anwender durch Einfärben der betroffenen Rohrleitungen sichtbar gemacht. So werde in der Anlage aus Abbildung 7.1a alle in Abbildung 7.1b gestrichelt gezeichneten Rohrleitungen eingefärbt, da sie beim Öffnen des durch den Anwender angefragten Ventils Y1 betroffen wären. Dieses Teilszenario stellt einen Sonderfall dar,



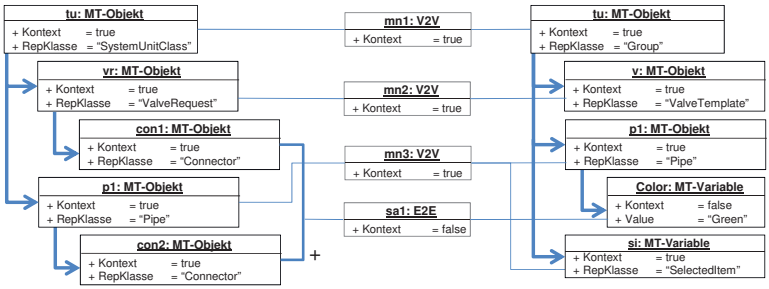
(a) Beispielanlage mit geöffnetem Ventil Y2 (b) Auswirkungen der Aktorsteuerung an Y1

Abbildung 7.1: Auswirkungen einer Aktorsteuerung

da die Modelltransformation dauerhaft aktiv bleibt und selbst Teil der Automatisierungsfunktion ist. Es verwendet daher das Prinzip „Parametrieren statt Implementieren“ [Sch12] in idealisierter Form. Dadurch verlässt dieses Teilszenario allerdings gleich aus mehreren Gründen die Welt der Tripel-Graph-Grammatiken. Zum einen handelt es sich hierbei nicht um einen einmali-

gen Übersetzungsprozess. Wird die Produktion S1-R0 angewendet, so wird der Link zwischen den beiden Flanschen (`Connector`) als übersetzt markiert. Ein Entfärben bzw. Schwärzen der Rohrleitung oder gar ein erneutes Färben sind mit einer TGG nicht realisierbar.

S1-R0: Einfärben



Um das Anwendungsszenario dennoch umsetzen zu können, kann das vorgestellte Framework ACPLT/MT jedoch auch als Basis für eine „herkömmliche“ unidirektionale Modelltransformation genutzt werden. Die MT-Elemente des linken Pattern sind dabei alle mit `Kontext = true` und `Quellmodell = true` parametriert. Die Regel lässt sich dann wie eine `WENN-DANN`-Regel lesen. Hierbei entfällt die Überprüfung, ob ein Objekt bereits übersetzt wurde. Anders als bei Leerlaufregeln kann dabei jedoch auf Informationen aus dem Quellmodell zugegriffen werden. Dies bringt natürlich neue Probleme mit sich. So steht eine bidirektionale Auswertbarkeit für solche Regelsätze nicht mehr zur Verfügung. Durch die erlaubte mehrfache Anwendung der Regel auf die gleichen Modellelemente entfällt zudem die Zusicherung, dass die Transformation terminiert. Wird beispielsweise in der Regel S1-R0 der Link im Pattern der linken Domäne als Kontext markiert und somit in eine `WENN-DANN`-Regel verwandelt, so wird sie immer und immer wieder ausgeführt werden, auch wenn die Rohrleitung bereits grün gefärbt ist. `WENN-DANN`-Regeln müssen daher so formuliert sein, dass sie dennoch ressourcenschonend ausgewertet werden können. Dies kann entweder durch einen geeigneten Trigger oder durch gezielte Abfrage des Aktualzustandes geschehen. Im aktuellen Beispiel schlägt der Trigger einmalig an, sobald eine Anfrage vom Benutzer erfolgt.

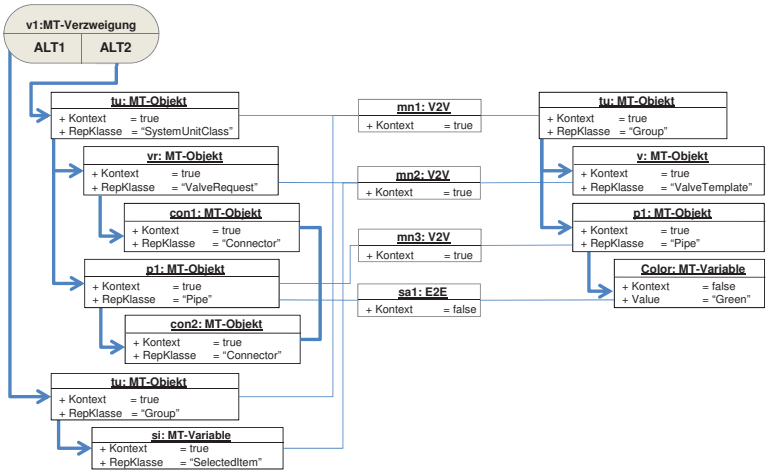
ACPLT/MT bietet für dieses Szenario noch weitere über die Mächtigkeit von Tripel-Graph-Grammatiken hinausreichende Möglichkeiten für die Realisierung an. Durch die Überprüfung des ausgewählten HMI-Objektes in der linken Domäne (`SelectedItem`) ist die Regel S1-R0 insgesamt sehr ineffizient, da das linke Pattern alle Kombinationen von Ventilen und Rohrleitungen findet und erst ganz zuletzt bei der Auswertung der rechten Domäne ein Vergleich mit dem eigentlich selektierten Element stattfindet. Um den Regelsatz effizienter zu gestalten, wurde die Regel umgestellt und eine Inplace-Transformation mit multiplen Quellmodellen formuliert. Dadurch wird die Welt der Tripel-Graph-Grammatiken endgültig verlassen.

Die resultierende Regel S1-R1 verwendet für die Umsetzung multipler Quellmodelle die als `UND` parametrierte MT-Verzweigung. Sie dient damit lediglich als Multiplikator für die Kommandostruktur und stößt die MT-Elemente der ersten und der zweiten Alternative parallel an. Da die

MT-Elemente der ersten Alternativen in der Taskliste vor denen der zweiten eingeordnet sind, wird zunächst das `SelectedItem` ausgewertet und in der zweiten Alternative nur das Ventil gefunden, das vom Anwender ausgewählt wurde.

Das massive Verlassen der TGG-Welt bringt eine Reihe von Problemen und Gefahren mit sich, für die im Einzelfall eine Kosten-Nutzen-Abwägung durchgeführt werden muss. So besteht die Gefahr eines nicht terminierenden Regelsatzes durch Inplace-Transformation und mehrfache „Übersetzung“ ein und desselben Objektes. Zudem fehlt die formale Zusicherung, dass die Regelanwendungen konsistenzerhaltend wirken. Die Mächtigkeit der Ausdruckskraft geht an dieser Stelle auf Kosten der Zusicherbarkeit von formalen Eigenschaften. Der Anwender muss daher jeden Regelsatz, der außerhalb der TGG-Spezifikation läuft, explizit auf die benötigten Eigenschaften hin überprüfen.

S1-R1: Auswirkung eines Stellbefehls am Ventil



Auf Basis dieser flexibleren nicht TGG-konformen Anwendung von ACPLT/MT färbt der in [KQ11; Kra+12] vorgestellte Regelsatz die im HMI dargestellten Rohrleitungen, schrittweise ausgehend vom angewählten Aktor, ein bis ein geschlossenes Ventil, eine ausgeschaltete Pumpe oder ein Reaktor erreicht ist. Die Anzahl der für die Einfärbung benötigten Zyklen entspricht der maximalen Anzahl der Rohrabschnitte in den gefundenen aktiven Flusswegen. Bei einer für die Prozessindustrie üblichen Zykluszeit von 10ms ist dieses Vorgehen auch für große Anlagen mit bis zu 40 oder 50 Rohrabschnitten bis zum nächsten Reaktor ausreichend schnell.

Die fünf für das Teilszenario benötigten Regeln werden in Anhang B als ACPLT/MT-Regeln detailliert aufgeführt. An dieser Stelle soll die schon in [Kra+12] verwendete beschreibende Form verwendet werden, um die Grundidee zu illustrieren. Die Regeln beschreiben den Zusammenhang von Ventil- (V), Pumpenzuständen (P), Vorhandensein von Kreuzungspunkte (K) und

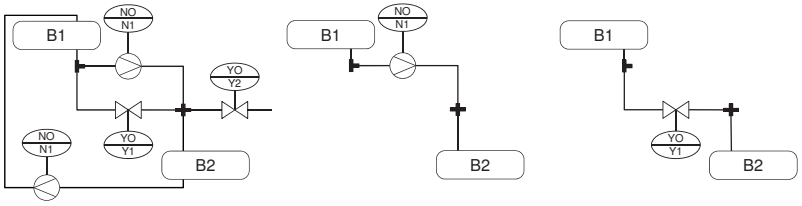
der Färbung der angeschlossenen Rohrleitungen (R). Indizes geben an, ob es sich um das entsprechende Objekt in der Domäne ACPLT/csHMI (HMI) und ACPLT/PandIX (R&I) handelt.

R1.	WENN	$V1_{HMI}$	angefragt wurde	UND
		$V1_{HMI}$	entspricht $V1_{R\&I}$	UND
		$V1_{R\&I}$	verbunden ist durch $R1_{R\&I}$	UND
	DANN	überprüfe $R1_{R\&I}$ entspricht $R1_{HMI}$		UND
		färbe $R1_{HMI}$ ein.		
R2.	äquivalente Regel für Pumpen $P1_{HMI}$ bzw. $P1_{R\&I}$			
R3.	WENN	$R1_{HMI}$	eingefärbt ist	UND
		$R1_{R\&I}$	verbunden ist mit $V1_{R\&I}$	UND
		$V1_{R\&I}$	offen ist	UND
		$V1_{R\&I}$	verbunden ist durch $R2_{R\&I}$	
	DANN	wird $R2_{HMI}$ eingefärbt		
R4.	äquivalente Regel für Pumpen $P1_{HMI}$ bzw. $P1_{R\&I}$			
R5.	WENN	$R1_{HMI}$	eingefärbt ist	UND
		$R1_{R\&I}$	verbunden ist mit $K1_{R\&I}$	UND
		$K1_{R\&I}$	verbunden ist durch $R2_{R\&I}$	
	DANN	wird $R2_{HMI}$ eingefärbt		

Während das erste Teilszenario inkrementelle Änderungen, multiple Quellmodelle und eine Inplace-Transformation fokussiert, steht bei der Erkennung von Leckagen und unerwünschter Vermischung von Medien die Umsetzung einer unidirektionalen Batch-Transformation im Vordergrund. Daher erlaubt sich an dieser Stelle die Frage, ob hier die Verwendung von Tripel-Graph-Grammatiken nicht über das Ziel hinausschießt. Da jedoch die Gefahr besteht, dass ein Anwender die Bausteinnetzwerke für die Flussweganalyse beabsichtigt oder unbeabsichtigt verändert und auch die Anlagenstruktur über die Zeit ggf. angepasst wird, ist die Möglichkeit zur Konsistenzanalyse und zum inkrementellen Update generell wünschenswert.

Als Basis für die Teilszenarien b) und c) dient die von Quirós [Qui11] formulierte Logik zur Analyse von Flusswegen in einer verfahrenstechnischen Anlage. Quirós beschreibt darin alle möglichen Flusswegproblematiken in Form von Funktionsbausteinnetzen. Abbildung 7.3 zeigt die anlagenspezifischen Bausteinnetze zur Realisierung der Flussweganalyse von Behälter B1 nach Behälter B2 über die Pumpe N1. Für die in Abbildung 7.3a gezeigte Anlage existieren zwei Flusswege von Behälter B1 nach B2 (vgl. Abbildungen 7.2b und 7.2c). Ein Vermischen kann potentiell dann auftreten, wenn ein Medium über das an den Flussweg angrenzende Ventil Y2 zufließt. Ein offenes Ventil Y2 bedeutet aber ebenso ein potentielles Leck.

Zur Auswertung dieser ungewünschten Zustände stehen entsprechende Bausteinnetze zur Überwachung von geöffneten Flusswegen 7.3c, die Erkennung von Leckagen 7.3d und die



(a) Beispielanlage (b) 1. Flussweg von B1 nach B2 (c) 2. Flussweg von B1 nach B2

Abbildung 7.2: Beispielanlage für die Flusswegkontrolle

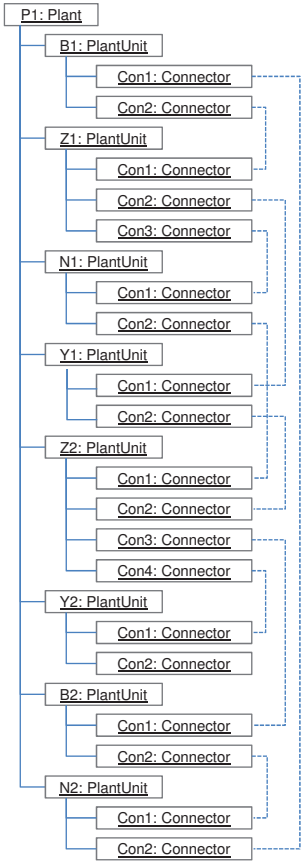
Erkennung von unerwünschtem Mischen von Medien 7.3e zur Verfügung. Wird eine Leckage oder ein Mischen außerhalb der dafür vorgesehenen Reaktoren identifiziert, so wird der Nutzer darüber informiert 7.3f. Die zu realisierenden Bausteinnetzwerke sind anlagenspezifisch realisiert. Bei flexiblen Anlagen wie M4P.AC [M4P] ist eine Flussweganalyse daher nur einsetzbar, wenn die entsprechenden Bausteinnetze automatisiert anhand der aktuellen Anlagenstruktur erstellt werden.

Bei der Realisierung dieser beiden Teilszenarien stößt das vorgestellte Konzept schnell an seine Grenzen. Zwar lässt sich die Darstellung der Anlage mittels ACPLT/Flowpath-Objekten noch relativ einfach in ACPLT/MT realisieren, doch schon die Suche nach möglichen Flusswegen ist nicht mehr möglich. Ursache hierfür ist die fehlende Möglichkeit, rekursive Zusammenhänge zu beschreiben. Die Suche nach einem Flussweg kann generell ähnlich angegangen werden wie das Einfärben der Rohrleitungen im ersten Teilszenario, sogar ohne die Verwendung von multiplen Quellmodellen, Inplace-Transformation und wiederholtem „Übersetzen“ ein und desselben Objektes. Anders als beim ersten Teilszenario reicht es aber nicht, Elemente die sich mehrere Flusswege teilen einmalig zu bearbeiten. Die Kreuzungspunkte verhindern daher den Einsatz von ACPLT/MT zur Realisierung des Szenarios. Beim Erreichen eines solchen Kreuzungspunktes müsste der bis dahin generierte Flussweg vervielfältigt werden. Dies ist mit ACPLT/MT jedoch nicht möglich. Die gleiche Problematik tritt auch bei der Generierung der übrigen Bausteinnetzwerke auf.

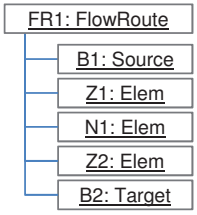
An diesem Anwendungsszenario wurde ersichtlich, dass das vorgestellte Konzept von ACPLT/MT zum einen flexibel genug ist, auch Modelltransformationen abzubilden, die über die Mächtigkeit von Tripel-Graph-Grammatiken hinaus gehen. Dennoch gibt es zunächst einfach anmutende Modellzusammenhänge, die nicht durch ACPLT/MT realisiert werden können.

7.3 S2 – Entwicklungsbegleitende Modelltransformation

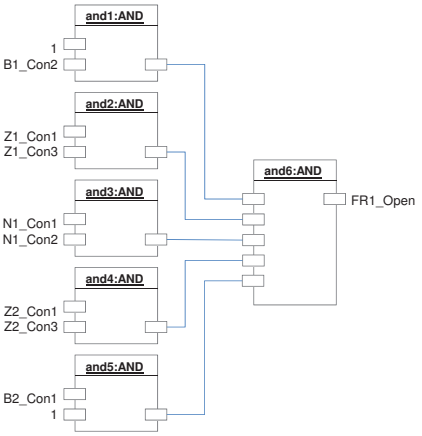
Das R&I-Fließbild enthält grafische Elemente für alle Sensoren und Aktoren und stellt die Anlagenstruktur in einer für den Menschen interpretierbaren Form dar. Eine 1-zu-1 Transformation in entsprechende Elemente des ACPLT/csHMI kann daher als ein erster Entwurf für die Bedienoberfläche angesehen werden. Neben der Abbildung entsprechender Symbole und der



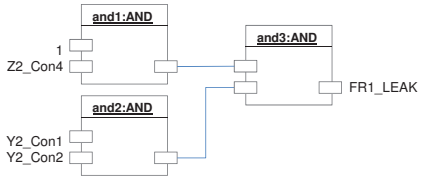
(a) ACPLT/Flowpath Darstellung der Anlage



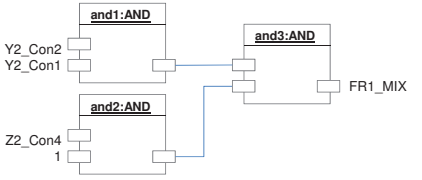
(b) 1. Flussweg



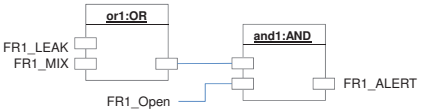
(c) Flussweganalyse für den ersten Flussweg



(d) Erkennen von Leckage



(e) Erkennen von Vermischung



(f) Generieren einer Warnung

Abbildung 7.3: Bausteinnetze zur Realisierung der Flussweganalyse nach [Qui11]

Verwendung der Struktur- und Positionsinformationen können auch angepasste Detailansichten zur Verfügung gestellt werden. So werden für analoge Aktoren Schieberegler oder Prozentangaben in der Detailansicht benötigt, für digitale Aktoren werden Schalter eingesetzt. Auch die Beschriftung innerhalb der Detailansichten ist abhängig vom Aktortyp. Bei digitalen Ventilen werden die Schalter mit „auf/zu“ beschriftet, bei Pumpen mit „an/aus“.

In [KQ11; Kra+12] sowie in den vorangegangenen Kapitel dieser Arbeit wurde bereits mehrfach auf den Regelsatz für die Realisierung einer solchen Transformation eingegangen. Der ausführliche TGG-Regelsatz findet sich in Anhang B und umfasst folgende Regeln:

R1. - Anlegen einer Domäne Das Axiom legt eine `ov/domain` als Wurzelelement für das PandIX bzw. das HMI an. Die `HMI`-Domäne erhält zudem zusätzliche strukturierende Elemente.

R2. - Anlage Für jede Anlage wird ein eigens R&I-Fließbild (PlantScheme) im PandIX angelegt bzw. eine Bedienoberfläche im ACPLT/csHMI.

R3. - Teilanlage Auch für Teilanlagen werden eigene Ansichten im ACPLT/csHMI bereitgestellt. Dies ermöglicht die spätere Umschaltung zwischen einem Gesamtüberblick und einer feineren Auflösung für eine Teilanlage.

R4. - Aktor Jeder Aktor ist im HMI durch ein Oval mit spezifischer Beschriftung gekennzeichnet. Diese entsprechen den `ActuatorRequest` im PandIX-Modell.

R5. - Ventile/Pumpen Pumpen und Ventile im PandIX erzeugen entsprechende Symbole in der Gesamtansicht.

R6. - Behälter Äquivalent zu R5 erstellt diese Regel Symbole für Behälter.

R7. - Rohrleitungen Äquivalent zu R6. erstellt diese Regel Rohrleitungen. Das Routing und somit den genauen Verlauf der Rohrleitungen werden durch das ACPLT/csHMI automatisch berechnet. Rohrleitungen erhalten lediglich ein Anfangs- und ein Endelement mit dem sie verknüpft sind. Dies sind die Flansche der Aktoren, Kreuzungspunkte und Behälter.

Mit diesem Anwendungsszenario konnte gezeigt werden, dass das entwickelte Konzept für den Einsatz praxisrelevanter Automatisierungsfunktionen geeignet ist. Anpassungen an der Anlagenstruktur können schnell und konsistent in die Bedienoberfläche übernommen werden. Zuvor bereits vorhandenen Aktoren, Behälter und Rohrleitungen bleiben bei einem solchen inkrementellen Updaten unverändert und behalten ihre durch den Apllikateur ggf. angepassten Grafikeigenschaften, wie die Position von Grafikelementen bei. Softsensoren und Gruppensteuerungen, die eine Kaskade von Aktoren ansteuert, können in der Bedienoberfläche erstellt und in die Planungsdaten rückdokumentiert werden. Auch die regelmäßige Konsistenzanalyse von Planungsdaten und HMI stellt während der Projektierung einer Anlage eine enorme Arbeitserleichterung dar und kann abschließend zur Zertifizierung des HMI genutzt werden.

Dieses Anwendungsszenario hat jedoch auch gezeigt, dass die TGG-Regeln selbst bei einfachen Modellzusammenhängen schnell sehr umfangreich und komplex werden.

7.4 S3 – Konsistenzanalyse und Modellreparatur

Die Konsistenzanalyse und eine bidirektionale Auswertung der Regeln zur Modellreparatur sind Kernthemen des letzten Anwendungsszenarios. Im ersten Teilszenario soll die Konsistenz zwischen den in der Bedienoberfläche dargestellten Aktoren und der Prozessführung sowie das Vorhandensein der entsprechenden Verknüpfung überprüft werden.

Die Prozessführung ist im Allgemeinen zu komplex, um sie automatisiert aus den Planungsdaten generieren zu können [KSY10]. Dennoch sind bestimmte Teile, wie die Verriegelungslogik von Pumpen und Ventilen, immer ähnlich. Wird die Verriegelungslogik vergessen oder kann ein aktives Element wegen einer fehlenden Verknüpfung zur Prozessführung nicht wie erwartet über die Bedienoberfläche angesteuert werden, so kann dies schwerwiegende Folgen für Mensch, Maschine und Umwelt nach sich ziehen. In diesem Szenario steht deshalb die Aufgabe im Vordergrund, zu jedem in der Bedienoberfläche vorhandenen Aktor eine entsprechende Prozessführungskomponente zu identifizieren und umgekehrt. Hierbei können einzelne Aktoren auch mehrfach in der Bedienoberfläche repräsentiert sein. Außerdem muss überprüft werden, ob die Prozessführungskomponenten mit den zugehörigen Bedienkomponenten verknüpft sind, da nur so die Aktoren angesteuert werden können.

Eine direkte Umsetzung dieses Szenarios als TGG ist nicht möglich, da die Überprüfung von Verknüpfungen zwischen Bedienoberfläche und Prozessführung einer Inplace-Transformation bedürfen. Hierzu werden die Transformation aus Anwendungsszenario S2 um die Komponenten der Prozessführung und die Verlinkungen zwischen Bedienbild und Prozessführung erweitert.

Zur Realisierung der Konsistenzanalyse wurde eine Vorwärts- und anschließende Rückwärtstransformation durchgeführt. Statt allerdings Objekte in der jeweiligen Zieldomäne anzulegen, wird bei der Anwendung einer Regel im Modus Konsistenzüberprüfung lediglich untersucht, ob Objekte existieren, die den MT-Elementen des Patterns der Zieldomäne entsprechen. Ist dies der Fall, so werden die Objekte der Quelldomäne wie gehabt als übersetzt markiert und falls nicht vorhanden die Korrespondenzlinks zwischen Quell- und Zieldomäne erzeugt. Nach vollständig durchlaufener „Transformation“ wird die Existenz nicht übersetzter Objekte der Quelldomäne untersucht und ggf. protokolliert.

Die Modellreparatur konnte nicht realisiert werden. Angedacht, war eine Interaktion mit dem Anwender, der für die einzelnen, bei der Konsistenzanalyse aufgetretenen Inkonsistenzen entscheidet, in welcher Richtung die Änderung propagiert wird. Dieser naive Ansatz hatte jedoch gleich mehrere Schwachstellen. Wird eine einzelne Inkonsistenz behoben, so kann dies Auswirkungen auf weitere zuvor bestehende Inkonsistenzen haben, da diese ggf. im gleichen Zug mit behoben werden. Um solche Wechselwirkungen aufdecken zu können, müsste nach jeder Änderung erneut eine Konsistenzanalyse durchgeführt werden. Dieses Vorgehen belastet das System jedoch unverhältnismäßig stark. Auch muss der Regelsatz erneut nach passenden Regeln für die Übersetzung der konkreten Inkonsistenz durchsucht werden, was die Performance weiter herabsetzt.

7.5 Anforderungen an eine bidirektionale Modelltransformation

Die drei Anwendungsszenarien bieten einen ersten Einblick, bei welchen Problemstellungen ACPLT/MT zur Realisierung eingesetzt werden kann und wo die Grenzen des Ansatzes liegen. Im Folgenden soll die Passgenauigkeit von ACPLT/MT auf die in den Kapiteln 4.2 identifizierten generellen Anforderungen an eine Modelltransformation für die Automatisierungstechnik analysiert werden sowie ein Abgleich mit den von Klar [Kla12] beschriebenen Anforderungen an eine bidirektionale Modelltransformation erfolgen.

Gerade bei der informationstechnischen Sicht auf ACPLT/MT muss dabei klar zwischen dem Modell an sich und der Realisierung einer TGG mit Hilfe von ACPLT/MT unterschieden werden, da hier große Unterschiede in Bezug auf die Mächtigkeit auf der einen Seite und der Zusicherung formaler Eigenschaften auf der anderen Seite zu erwarten sind. Bei der Analyse der von Klar aufgestellten Kriterien steht daher ein kritischer Blick auf die mit der gewonnenen Flexibilität einhergehenden Risiken und Probleme im Vordergrund.

Flexibel einsetzbar Tripel-Graph-Grammatiken sind für die Beschreibung von Modellzusammenhängen zwischen zwei graphbasierten Modellen entwickelt worden. Durch die Bereitstellung von MT-Elementen zur Interpretation von XML-Bäumen ist das Anwendungsspektrum nochmals erweitert worden. Diese Bandbreite an möglichen Modellen steht sowohl in ACPLT/MT allgemein als auch bei der Realisierung einer TGG auf Basis von ACPLT/MT zur Verfügung.

Eindeutige Semantik Die vorliegende Arbeit konzentriert sich auf die Konzeptentwicklung für eine Modelltransformation in prozessleittechnischen Laufzeitumgebungen. Eine umfangreiche Einführung in die Semantik von ACPLT/MT finden sich in Kapitel 6. Weiterführende Arbeiten sind jedoch noch notwendig, um eine detaillierte eindeutige Semantik zur Verfügung zu stellen.

Mächtigkeit Die Mächtigkeit von Tripel-Graph-Grammatiken ist im Vergleich zu anderen Modelltransformationssprachen wie QVT und VIATRA zu Gunsten der zugesicherten formalen Eigenschaften deutlich eingeschränkt. Durch die Beschränkung auf ein Quell- und ein Zielmodell, auf Outplace-Transformation und nichtrekursive Modellzusammenhänge sind viele Anwendungsszenarien nicht mit Tripel-Graph-Grammatiken realisierbar. ACPLT/MT hebt zwar einige dieser Beschränkungen auf, gerät aber zum Beispiel bei den rekursiven Problemen ebenso an seine Grenzen. Wie an den Anwendungsszenarien zu sehen war, lässt sich trotz der eingeschränkten Mächtigkeit ein breites Feld an praxisrelevanten Anwendungen mit ACPLT/MT und vielfach auch mit einer darauf aufbauenden TGG realisieren.

Bijektivität Wie in Abschnitt 7.2 ersichtlich wird, unterstützt ACPLT/MT die Beschreibung nicht-bijektiver Modellzusammenhänge. Zum einen stehen dafür die `WENN-DANN`-Regeln zur Verfügung, zum anderen bieten Leerlaufregeln eine TGG-konforme Möglichkeit, eines der beiden Modelle zu modifizieren ohne das andere anzupassen. Um Leerlaufregeln in ACPLT/MT zu nutzen, wird das Pattern für das unveränderte Modell mit Hilfe eines MT-True-Objektes realisiert. Dieses gibt immer eine positive Rückmeldung, so dass die Regel effektiv nur aus einem Pattern besteht.

Wiederverwendbarkeit Die Verwendung von Template-Datenbanken ermöglicht es, beliebig komplexe Regelbestandteile für die Mehrfachverwendung bereitzustellen. Zudem bieten Verzweigungen die Möglichkeit, mehrere Regeln zusammenzufassen und so gemeinsam genutzte Regelbestandteile wiederzuverwenden. Beide Wiederverwendungskonzepte sind auch beim Einsatz von ACPLT/MT als Basis einer TGG nutzbar.

Terminierung Diese Eigenschaft kann für den Kontrollalgorithmus nicht generell zugesichert werden. Sie ist abhängig von den formulierten Regeln. Insbesondere bei Inplace-Transformationen kann es zu Zyklen in der Bearbeitung kommen, wenn zum Beispiel durch eine Regel die RHS jeweils ein Element hinzufügt wird, das durch die LHS gesucht wird. In diesem Fall wird in jedem Schritt ein neues Element angelegt und im nächsten Schritt als neu angelegt gefunden. Auch bei Transformationen mit disjunkten Quell- und Zielmodellen kann es zu nicht-terminierenden Regelsätzen kommen. Eine Transformation, die den Zustand einer zyklisch blinkenden Lampe aus der Prozessführung in eine Anzeige der Bedienoberfläche überführt, kann bei ungünstiger Wahl der Abbruchbedingung bei der Bearbeitung der Regeln zu nicht-terminierendem Verhalten führen. Wie in diesem Beispiel ersichtlich ist, kann dies aber durchaus gewollt sein. Terminierung kann lediglich für einzelne, auf ein während der Regelbearbeitung stabiles, endliches Quellmodell angewendet, zugesichert werden. Die Einschränkung bei der Verwendung von Templates auf Zyklenfreiheit sowie die verwendete Tiefensuche mit Backtracking verhindert hier ein nichtterminierendes Festhängen in Zyklen oder toten Enden.
Wird ACPLT/MT jedoch als Basis für eine TGG genutzt und auf ein Quellmodell angewendet, das über den Zeitraum der Transformation unverändert bleibt, so kann eine Terminierung zugesichert werden, da jedes Element nur einmal übersetzt wird und anschließend nur noch im Kontext von Regeln zum Einsatz kommt.

Inkrementelle Änderungen Das vorgestellte Konzept eignet sich auch für inkrementelle Modelländerungen, da die Zusammenhänge zwischen Quell- und Zielmodell dauerhaft nachgehalten und auch für spätere Durchläufe der Transformation zur Verfügung stehen. Regelsätze können auch explizit als inkrementelle Regelsätze konzipiert werden, indem sie auf die Ergebnisse und Korrespondenzobjekte vorangegangener Transformationen aufbauen.

Wie aus der Analyse des Konzeptes entlang der von Klar aufgestellten Anforderungen deutlich wird, sind einige Punkte nicht generell für das vorgestellte Konzept gültig. Dieser Verzicht auf die formalen Rahmenbedingungen ist bewusster Bestandteil des Konzeptes. Dadurch ist es möglich, Modelltransformation auch für Anwendungsszenarien zur Verfügung zu stellen, die den formalen Eigenschaften nicht gerecht werden. Diese erhöhte Flexibilität geht jedoch immer mit der Gefahr einher, die positiven Eigenschaften, die bei der Verwendung von Tripel-Graph-Grammatiken zugesichert werden können zu verlieren. Das vorgestellte Konzept sollte daher immer im Zusammenhang mit den theoretischen Vorarbeiten aus dem Bereich der Tripel-Graph-Grammatiken gesehen werden. Insbesondere die Arbeiten von Königs [Kön08], Klar [Kla12] und Lauder [Lau12] beschreiben und beweisen die formalen Eigenschaften von Tripel-Graph-Grammatiken. Zudem bieten sie eine Fülle von Optimierungsvorschlägen für die Regelbeschreibung und -verarbeitung von denen nur ein kleiner Teil in dieser Arbeit Beachtung gefunden hat.

7.6 Anforderungen an eine Modelltransformation für die Automatisierungstechnik

Bei der Übertagung von Konzepten aus der Informatik in eine Anwendungsdomäne müssen immer die Bedürfnisse und Kenntnisse der Domänenexperten in die Konzeptentwicklung einbezogen werden. Ein Ansatz, der den üblichen Arbeitsfluss auf Dauer eher behindert als ihn zu unterstützen, kann noch so innovativ sein, er wird keine Anwendung finden.

Auch das vorgestellte Konzept muss sich daher an den Maßstäben aus Kapitel 4.2 messen lassen:

Unidirektionale Batch-Transformation Nicht alle mit ACPLT/MT realisierbaren Regelsätze sind zur Ausführung als unidirektionale Batch-Transformation geeignet. Insbesondere solche, die auf `WENN-DANN`-Regeln basieren sind für die kontinuierliche inkrementelle Modelländerung ausgelegt. Generell stellen unidirektionale Batchtransformationen allerdings schon das Hauptanwendungsgebiet von ACPLT/MT dar, insbesondere wenn die Realisierung mit Hilfe von Tripel-Graph-Grammatiken auf Basis von ACPLT/MT erfolgt.

Inkrementelle Änderungen Wie bereits im vorangegangenen Abschnitt erläutert, eignet sich ACPLT/MT auch für das Propagieren inkrementeller Modelländerungen und ermöglicht des Weiteren rein inkrementelle Regelsätze.

Bidirektionale Auswertbarkeit Eine Zusicherung dieser Eigenschaft kann nur erfolgen, wenn die Transformation als TGG auf Basis von ACPLT/MT realisiert wurde. Durch Setzen des entsprechenden Parameters des `REC` lassen sich ACPLT/MT-Regeln in beliebiger Richtung zur Modelltransformation anwenden. Die einmal formulierten Modellzusammenhänge lassen sich dadurch entweder zur Realisierung der Automatisierungsfunktion anhand der Planungsdaten oder zur Rückdokumentation von Änderungen an der Automatisierungsfunktion nutzen. Eine Anpassung der Regel ist dazu nicht notwendig.

Konsistenzprüfung Auch die Konsistenzprüfung erfolgt durch Parametrierung der bidirektional formulierten MT-Regeln. In diesem Fall produzieren die Objekte der Korrespondenzdomäne eine entsprechende Dokumentation der erfolgten Analyse.

Modellreparatur Das bisher vorgestellte Konzept ermöglicht noch keine effiziente Modellreparatur. Ein erster naiver Ansatz ermöglichte zwar eine generelle Identifikation der Inkonsistenzen und deren Behebung, belastet jedoch das System in einem nicht tragbaren Umfang. Zudem erfordert die aktuelle zustandsbasierte Ausrichtung von ACPLT/MT, die nur den aktuellen Zustand der Modelle betrachtet, den aktiven Einbezug des Anwenders bei der Auflösung von Inkonsistenzen.

Nachvollziehbarkeit Ob diese Anforderung erfüllt ist, hängt stark von den formulierten MT-Regeln ab. Die Dokumentation und Nachvollziehbarkeit der Transformationsschritte muss durch die Logik der Korrespondenzobjekte abgedeckt werden. Auch kann mit dem bisherigen Konzept noch keine deterministische Auswertung zugesichert werden. Hierzu müssen die Regeln so gestaltet sein, dass die Reihenfolge der Auswertung irrelevant für das Ergebnis ist oder es muss eine feste Auswertungsreihenfolge vorgeschrieben werden. Jeder Regelsatz muss explizit auf Determinismus hin untersucht werden.

Freie Werkzeugwahl Das Konzept ist unabhängig von den für die Erstellung der Modelle verwendeten Werkzeugen. Es stellt lediglich an die verwendeten Modelle die Anforderung,

dass sie in einer objektorientierten, rechnerauswertbaren Form vorliegen und in das Zielsystem in Form von Objektnetzwerken importiert werden können. Dies ist bei vielen Planungsmodellen in der einen oder anderen Form mittlerweile der Fall.

Wartbare, nachvollziehbare Regeln ACPLT/MT-Regeln setzen Objektnetzwerke der korrelierenden Modelle in Beziehung und ermöglichen so Regeln, die für den Domänenexperten einfach nachvollziehbar sind. Durch die Wiederverwendung von Regelteilen mit Hilfe des Platzhalter/Template-Konzeptes kann die Lesbarkeit von Regeln weiter verbessert werden. Hierzu ist es ratsam, für die Templates sprechende Namen zu verwenden. Auch die Wartbarkeit der Regel profitiert von der Verwendung der Templates, da Anpassungen nur an einer Stelle durchgeführt werden müssen und anschließend an allen Platzhaltern direkt zur Verfügung stehen. Die Verwendung von Verzweigungen reduziert die Anzahl der redundanten Regelteile weiter und trägt ihren Teil zur besseren Wartbarkeit bei.

8 Zusammenfassung und Ausblick

Traditionell werden verfahrenstechnische Anlagen einmal geplant, errichtet und über Jahrzehnte nahezu unverändert betrieben. Diese traditionellen Anlagen werden in Zukunft immer häufiger durch flexible Anlagen ergänzt, die auftragsbezogen rekonfiguriert und kombiniert werden können. Eine solche Flexibilität verlangt aber nicht nur ein Umdenken beim Bau von verfahrenstechnischen Anlagen sondern auch bei ihrer Automatisierung. Nur die Kombination aus modularen Anlagen und effizienter Anpassung der benötigten Automatisierungsfunktionen können die Anlagen der Zukunft fit machen für die mit Industrie 4.0 einhergehenden Anforderungen.

Das erklärte Ziel dieser Arbeit war die Bereitstellung von anlagenneutralen Automatisierungsfunktionen, die als Serienprodukt verkauft und per Modelltransformation anhand der Planungsdaten an die konkrete Anlage und an aktuelle Anforderungen angepasst werden können. Zudem sollen Änderungen in der Automatisierungsfunktion in die Planungsdaten zurückgespielt und somit dokumentiert werden können. Dieses Kapitel fasst das zu diesem Zweck entwickelte Konzept in Kürze zusammen, reflektiert, inwieweit die gesetzten Ziele erreicht werden konnten und gibt einen Ausblick auf die zukünftig Entwicklung von ACPLT/MT.

8.1 Modelltransformation für prozessleittechnische Laufzeitumgebungen

Die angestrebte Problemstellung und ein Einblick in das Potential einer Modelltransformation in prozessleittechnischen Laufzeitumgebungen wurden dem Leser in Kapitel 1 näher gebracht. Zudem wurden drei ganz konkrete aus der Praxis stammende Anwendungsszenarien vorgestellt, die im Verlauf der Arbeit unter anderem zur Veranschaulichung der beschriebenen Inhalte und zur Verdeutlichung konkreter Problemstellungen herangezogen wurden. Die Basisbegriffe aus dem Bereich der Modellierung sowie verschiedene Modellierungsarten wurden in Kapitel 2 angerissen. Kapitel 3 analysiert eine Reihe von Modellen aus dem Bereich der Automatisierungstechnik hinsichtlich ihrer Eignung für eine Modelltransformation. Dabei dienten die zuvor bereitgestellten Anwendungsszenarien als Leitfaden für die Auswahl der betrachteten Modelle. Die bei der Analyse gewonnenen Erfahrungen flossen ein in eine Empfehlung für die Entwicklung künftiger Automatisierungsmodelle am Ende des Kapitels. Als Basis für die angestrebte Konzeptentwicklung für eine Modelltransformation in prozessleittechnischen Laufzeitumgebungen wurden in Kapitel 4 und 5 Anforderungen an eine solche Modelltransformation formuliert und relevante Ansätze aus der Automatisierungstechnik und der Informatik vorgestellt und auf ihre Eignung hin untersucht. Ein besonderer Fokus lag dabei auf den Tripel-Graph-Grammatiken.

Der aus der Informatik stammende Ansatz der Modelltransformation mittels Tripel-Graph-Grammatiken wurde in Kapitel 6 in die Sprachen der IEC 61131 überführt. Das dadurch entstandene Framework ACPLT/MT für eine Modelltransformation in prozessleittechnischen Laufzeitumgebungen bietet die Grundlage für die Bereitstellung von parametrierbaren anlagenneutralen Automatisierungsfunktionen und ermöglicht eine Anpassung der Anlagenautomatisierung an neue Gegebenheiten ohne erneuten Implementierungsaufwand. Die Integration von Tripel-Graph-Grammatiken in die Sprachen der IEC 61131 gewährleistet, dass sich Automatisierungsfunktionen, die mit ACPLT/MT realisiert werden, nahtlos in die automatisierungstechnische Laufzeitumgebung einbinden lassen. Zudem ermöglicht die Umsetzung in den domänenspezifischen Sprachen eine erhöhte Akzeptanz, da die Applikateure keine neue Sprache für die Regelerstellung erlernen müssen. Ergänzend zur Vorstellung des Konzeptes wurde die Referenzimplementierung auf Basis der ACPLT-Modellwelt im Detail erläutert. Im Fokus standen dabei insbesondere die benötigten Anpassung am Tasking-Konzept sowie die Umsetzung in den an die IEC 61131-3 angelehnten Sprachen SFC und ACPLT/FB. Anders als bei bisherigen TGG-Ansätzen wurde dabei sowohl auf eine explizite Übersetzung in operationale Regeln als auch auf einen Interpreter verzichtet. Stattdessen kamen aktivierbare Regeln zum Einsatz, die zunächst als passive Objektnetzwerke die deklarativen TGG-Produktionen repräsentieren. Zum Zeitpunkt der Regelanwendung werden die Objekte in die zyklische Bearbeitung eingebunden und agieren daher als operationale Regeln.

Anhand von umfangreichen Testszenarien konnte in Kapitel 7 gezeigt werden, dass sich das vorgestellte Konzept in der Praxis bewährt und auch komplexere parametrierbare Automatisierungsfunktionen ermöglicht. Als eine besondere Herausforderung bei der Realisierung der Anwendungsszenarien stellte sich die Komplexität der Modelle und Modellzusammenhänge heraus. Insbesondere die Einschränkung von Tripel-Graph-Grammatiken, dass jedes Modellelement nur einmal übersetzt werden darf, führte dabei schnell zu sehr komplexen und unübersichtlichen Regeln. Zwar kann durch Leerlaufregeln schon ein Teil der Problematik abgefangen werden, diese lassen jedoch keine Wiederverwendung von Objektnamen oder Variablenwerte aus dem Kontext für die Generierung der neuen Objekte zu. Auch an anderen Stellen erschwert die eingeschränkte Mächtigkeit von Tripel-Graph-Grammatiken die Realisierbarkeit von automatisierungstechnischen Problemstellungen. So schränken die fehlende Rekursivität und die fehlende Möglichkeit zum Löschen die beschreibbaren Modellzusammenhänge merklich ein. Zwar bietet ACPLT/MT zum Teil die Möglichkeit, auf die flexibleren `WENN-DANN`-Regeln auszuweichen, dies geht aber zu Lasten der zugesicherten formalen Eigenschaften, die bei der Verwendung TGG-konformer Regeln gelten. Die spezifischen Handschriften der Planungsingenieure und Applikateure sowie domänenspezifische Benennungen von korrelierenden Modellelementen stellen eine weitere Herausforderung für das Konzept der regelbasierten Modelltransformation zur Realisierung von anlagenneutralen Automatisierungsfunktion dar. Dies kommt verstärkt an den Stellen zum Tragen, bei denen die Anlage aus einem anderen Blickwinkel betrachtet wird. So sieht der Elektrokonstrukteur die Anlage aus Sicht der verbauten Remote I/O und der durch sie zur Verfügung gestellten Anschlusspunkte für die Hardware im Feld, bei der Erstellung eines R&I-Fließbildes steht hingegen die funktionale Sicht auf die Anlage im Fokus. Dementsprechend fällt die Benennung der Aktoren und Sensoren zwischen der Elektroplanung und einem R&I-Fließbild unterschiedlich aus.

Trotz all dieser Unwägbarkeiten hat sich gezeigt, dass auch praxisrelevante Problemstellungen mit ACPLT/MT realisiert werden können. Die einmal beschriebenen Regeln können anlagenneutral bereitgestellt werden, um die anlagenspezifischen Funktionen zu generieren, oder selbst als Automatisierungsfunktion zu agieren. Mit ACPLT/MT steht daher ein umfangreiches Framework zur Realisierung von Modelltransformationen in der prozessleittechnischen Laufzeitumgebung zur Verfügung. Durch die Fokussierung von ACPLT/MT auf Tripel-Graph-Grammatiken als Basis steht dem Anwender eine fundierte, gut erforschte Transformationssprache mit umfangreichen formalen Zusicherungen zur Verfügung. Zukünftige Anpassungen am vorgestellten Konzept bieten das Potential, ACPLT/MT noch anwenderfreundlicher und robuster zu gestalten und das Anwendungsspektrum von ACPLT/MT zu erweitern. Einige dieser Erweiterungsmöglichkeiten werden im Folgenden vorgestellt.

8.2 Erweiterte Einsatzszenarien und mögliche Spracherweiterungen

Möchte sich ein Anwender die durch Tripel-Graph-Grammatiken bereitgestellten Eigenschaften in ACPLT/MT zu Nutze machen, unterliegt er strengen Regularien bei der Erstellung der Regeln. Gerade bei komplexen Regeln und umfangreichen Regelsätzen ist dies für den Anwender ohne unterstützende Werkzeuge nur schwer zu realisieren. Die Bereitstellung entsprechender Werkzeuge bietet daher großes Potential, die Anwendbarkeit von ACPLT/MT in der Praxis voranzutreiben. Ein erster Schritt zur Unterstützung des Anwenders ist die Einführung eines Validierungswerkzeuges, das vom Anwender entwickelte Regelsätze auf Konformität zu den TGG-Regularien überprüft. Um schon bei der Erstellung der Regeln Fehler zu vermeiden, bedarf es jedoch eines Entwicklungswerkzeuges, das die Beschreibung von Modellzusammenhängen unterstützt und dabei auf Wunsch auf TGG-konforme Möglichkeiten einschränkt. Auch die Regelauswertung bietet noch umfangreiches Erweiterungspotential. So ist die Bereitstellung einer weitgehend automatisierten Modellreparatur für die Automatisierungstechnik von besonderem Interesse. Nur so lassen sich Anlagenzustand und Anlagendokumentation auf Dauer konsistent halten. Hierbei kann auf Arbeiten aus dem Forschungsbereich der Tripel-Graph-Grammatiken zurückgegriffen werden, die nicht nur den aktuellen Modellzustand sondern die zeitliche Entwicklung und damit die vorangegangenen Modellzustände mit in die Betrachtung einbeziehen. Dadurch ist es bei auftretenden Inkonsistenzen in vielen Fällen möglich, zu entscheiden welche Modelländerung für die Inkonsistenz verantwortlich ist und die ursächliche Änderung in das andere Modell zu übernehmen. Ein weiterer Ansatzpunkt aus dem Bereich der Regelauswertung ist die Laufzeitoptimierung durch parallele Bearbeitung mehrerer Modelltransformationen oder mehrerer Regeln einer Transformation. Hierbei empfiehlt sich eine Art Sandkastensystem, bei dem jeder in Ausführung befindlichen Regel ein klar definierter Modellteil zum Lesen und Ändern bereitgestellt wird. Ein überlagerter Kontrollmechanismus muss sicherstellen, dass die Modellteile, die von parallel aktiven Regeln genutzt werden sich nicht überschneiden. Neben der Modellreparatur und paralleler Regelbearbeitung bietet das Forschungsgebiet der Tripel-Graph-Grammatiken generell ein breites Spektrum bisher noch nicht in Betracht gezogener Erweiterungen. So kann unter anderem die Realisierung von Löschungen [Lau12], Amalga-

mierung [Leb+15; Leb+16] und Vererbung [Anj14] den Einsatzhorizont von ACPLT/MT deutlich erweitern.

Bei der Realisierung der Anwendungsszenarien und im Gespräch mit Kollegen und Mitstreitern ergaben sich weitere vielversprechende Anregungen zur Weiterentwicklung von ACPLT/MT. So wurde bereits in [KME12] die Einbettung von ACPLT/MT in eine verteilte Automatisierungslandschaft untersucht. Die Verteilung von Funktionalität auf verschiedene Rechner und Steuerungssysteme ist ein Grundprinzip heutiger Automatisierungssysteme. Der Trend geht immer weiter in Richtung autark agierender Einzelkomponenten, die sich benötigte Informationen über den Anlagenzustand eigenständig über das angeschlossene Netzwerk besorgen. Ein populärer Ansatz auf diesem Gebiet ist der Gedanke der Service-Orientierten Architektur, kurz SOA. Dabei wird gezielt auf kleine Softwarekomponenten mit dedizierter Aufgabe und klar definierter Schnittstelle gesetzt. Die einzelnen Softwarekomponenten, die so genannten Services, kapseln die durch sie bereitgestellte Funktionalität und stellen sie bei Dienstaufrufen zur Verfügung. Durch den losen Verbund der einzelnen Softwarekomponenten in verteilten Systemen ist ein zentrales Änderungsmanagement aller Komponenten schwer zu realisieren. Anpassungen an einer Komponente können daher nicht kalkulierbare Auswirkungen auf andere Komponenten haben. Jede Softwarekomponente muss für sich entscheiden, ob eine im Gesamtsystem aufgetretene Anpassung Auswirkungen auf die lokalen Modelle hat. Hierzu schlagen [KME12] vor, die durch eine Softwarekomponente bereitgestellte Automatisierungsfunktion (z.B. den Erkundungsservice, vgl. Abbildung 8.1) in eine Servicegruppe einzubetten. Diese beobachtet das Gesamtsystem und passt ggf. die Automatisierungsfunktion mittels Modelltransformation an die Systemänderungen an. Hierzu stehen zwei lokale Dienste zur Verfügung, der Änderungs- und der Engineeringservice. Wird eine Automatisierungsfunktion geändert, so stellt der Änderungsservice Informationen über Art der Änderung und Änderungszeitpunkt als Änderungsnachricht für andere Servicegruppen bereit. Zudem registriert sich der Änderungsservice bei relevanten anderen Servicegruppen, um so Informationen über Änderungen an deren Automatisierungsfunktion mittels Änderungsnachricht informiert zu werden. Der Engineeringservice wiederum reagiert auf die einkommenden Änderungsnachrichten und passt das lokale Modell via Modelltransformation an die Änderungen der anderen Automatisierungsfunktionen an. Erste Erweiterungen des Konzeptes der Servicegruppen von Grüner et. al [GWE14] erlauben sogar den Einsatz über die Prozessleittechnik hinaus in die Fertigungstechnik.

In eine ähnliche Richtung geht der potentielle Einsatz von ACPLT/MT in Self-X-Systemen. Die Modelltransformation kann beispielsweise dazu genutzt werden, Automatisierungsfunktionen in ein vorher unbekanntes Gesamtsystem per Plug-and-Play einzubinden. Anders als in dieser Arbeit betrachtet erfolgt dabei allerdings keine Parametrierung mit Hilfe der Planungsdaten. Vielmehr erkundet die Modelltransformation, unterstützt durch Services, selbstständig das bestehende Gesamtsystem und passt die Automatisierungsfunktion per Selbstkonfiguration entsprechend an. Auch selbstheilende Systeme sind denkbar, die Inkonsistenzen erkennen und - sofern möglich - beheben. Die dritte vielversprechende Self-X Eigenschaft, die durch ACPLT/MT realisiert werden kann ist die Selbstoptimierung. Anhand aktueller Performancedaten der Anlage kann die Prozessführung angepasst und über die Zeit optimiert werden. Der Einsatz von ACPLT/MT in Self-X-Systemen jeglicher Art bedarf jedoch einer Situationsanalyse der konkreten Einsatzszenarien und umfangreicher Weiterentwicklungen am hier vorgestellten Konzept.

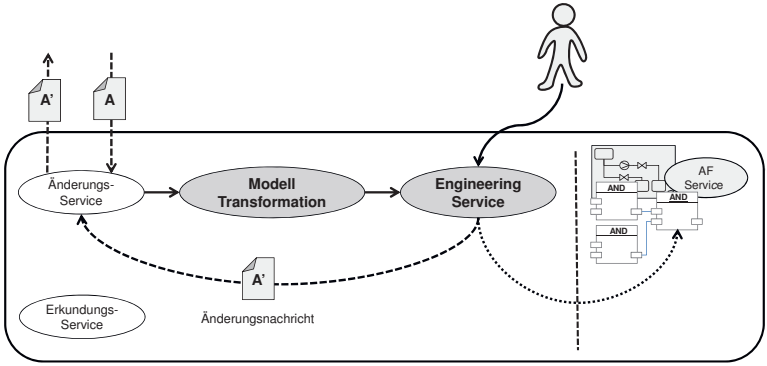


Abbildung 8.1: Automatisierungsfunktion als Service-Gruppe [KME12]

Auch zur Unterstützung des traditionellen Engineeringprozesses kann der Einsatz von ACPLT/MT sinnvoll sein. Werden die Ein- und Ausgänge eines Funktionsbausteins im Engineeringsystem verändert, so muss das Laufzeitsystem für gewöhnlich angehalten werden um den Baustein zu ersetzen und anschließen das Laufzeitsystem wieder zu starten. Diese Unterbrechung ist eine besondere Herausforderung für Applikateure, da die Anlage komplett leer gefahren werden muss, um nicht ungesteuert Produkte in der Anlage zu halten. Durch den Einsatz von ACPLT/MT können Funktionsbausteine im laufenden System ersetzt werden. Dazu muss zunächst ein Baustein mit den neuen Eigenschaften angelegt und mit den Eingängen des alten Bausteins verbunden werden. Sobald der neue Baustein sich in seinem Verhalten eingependelt hat, können die Ausgänge stoßfrei vom alten Baustein auf den neuen verlinkt werden.

Neben den noch nicht erschlossenen Anwendungsfeldern von ACPLT/MT bieten sich auch Weiterentwicklungsmöglichkeiten zur optimierten Verwendung in automatisierungstechnischen Laufzeitumgebungen. Mersch et al. [ME11] schlagen in ihren Arbeiten die Auslagerung von Komponenten für die Selbstkonfiguration, zu denen ACPLT/MT zu zählen ist, aus dem Echtzeitkontext vor. Dies ist dann möglich, wenn die Automatisierung mit einem Industrie-PC statt mit einer herkömmlichen SPS realisiert wird. Je nach Anwendungsszenario kann zur Entlastung der Echtzeit alternativ auch die Ausführung von ACPLT/MT in einem langsameren, niederprioren Task in Betracht gezogen werden. Beim Einsatz von Industrie-PCs mit Multi- oder ManyCore-Technologie ermöglicht ein separater ACPLT/MT-Task zudem die Auslagerung auf einen oder mehrere eigene Kerne.

Anhang A ACPLT/MT-Schema-Definition

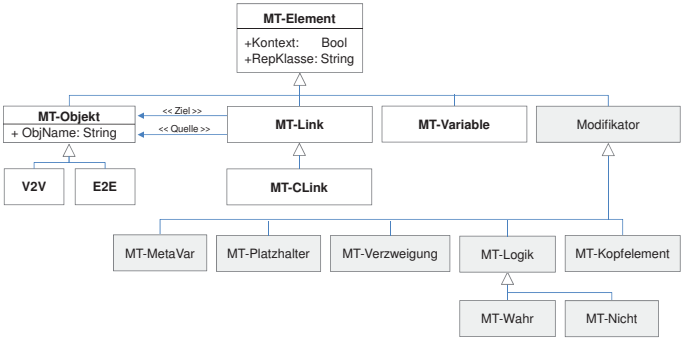


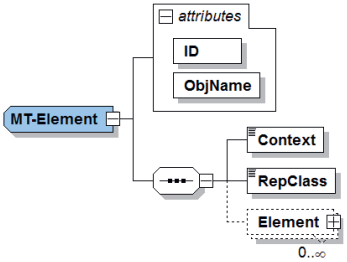
Abbildung A.1: ACPLT/MT-Elemente

Dem Konzept von ACPLT/MT folgend, sollten ACPLT/MT-Regelsätze nicht spezifisch für eine SPS entwickelt werden, sondern anhand von „Planungsdaten“ regelbasiert erstellt werden. Hierzu bietet sich an, bei der Bereitstellung von ACPLT/MT auf einer SPS einen initialen Regelsatz mitzuliefern, der ähnlich wie der Regelsatz für Anwendungsszenario S0 eine XML-Datei einlesen und daraus ACPLT/MT-Regeln erzeugen bzw. Änderungen an den Regeln zurückspielen kann. Das aktuelle Kapitel stellt zu diesem Zweck ein Austauschformat für ACPLT/MT als XSD bereit.

ACPLT/MT-Klassen

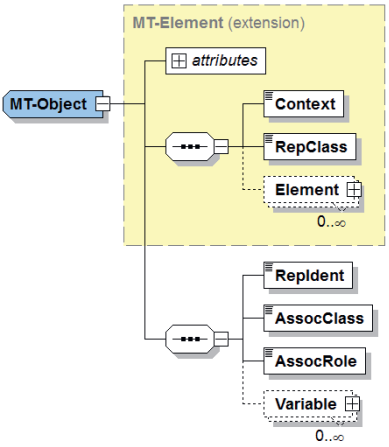
Abbildung A.1 zeigt nochmals die ACPLT/MT-Klassen aus Abbildung 6.2b im Überblick. In einem ersten Schritt werden Elemente für diese Klassen bereitgestellt.

Die Basisklasse `MT_Element` wird im XML mit einer ID versehen. Diese ist notwendig, um eine eindeutige Referenz für Korrespondenzlinks und MT-Links zu ermöglichen. Das Attribut `ObjName` gibt den Namen des MT-Elements im Laufzeitsystem an. Ob das MT-Element zum Kontext des jeweiligen Patterns gehört oder nicht wird über das Unterelement `Context` übermittelt.

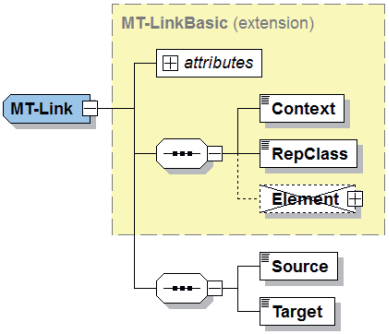


`RepClass` gibt die Klasse an, die das MT-Element in der jeweiligen Domäne repräsentiert. Für die Unterklasse MT-Modification wird dieses Feld genutzt, um den Typ der Modifikation kenntlich zu machen. Die in der Regel verankerte Kommandostruktur wird über eingebettete MT-Elemente repräsentiert.

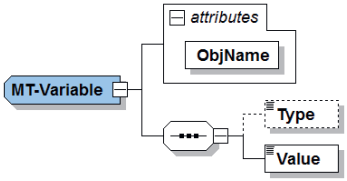
Für die Repräsentation von Modellobjekten, ergänzt das MT-Objekt die Struktur von MT-Element um Felder für den Namen des repräsentierten Objektes (`RepIdent`), um Angaben zur Assoziation über die das repräsentierte Objekt mit dem durch den Auftraggeber gebundene Objekt verlinkt ist (`AssocClass`, `AssocRole`) und um MT-Variablen.



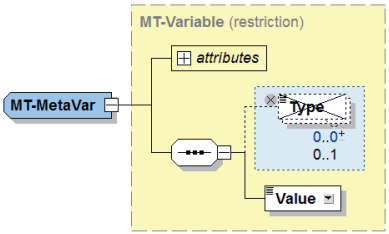
Ein MT-Link repräsentiert Links im Modell, die nicht entlang der Kommandostruktur verlaufen. Die Elemente `Source` und `Target` enthalten zu diesem Zweck Referenzen auf die MT-Elemente, die die Quelle bzw. das Ziel repräsentieren. MT-Link selbst besitzt keine Auftragnehmer und daher auch keine eingebetteten MT-Elemente.



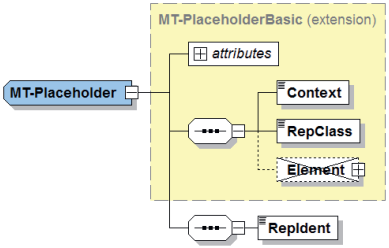
MT-Variablen repräsentieren eine Variablen im Modell und benötigen dazu Angaben zum Typ und ggf. zum Wert der jeweiligen Variablen.



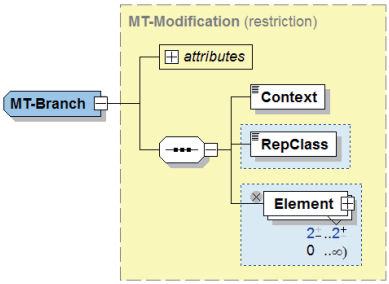
Metavariablen besitzen keine Entsprechung in den korrelierenden Modellen. Sie dienen lediglich zur Parametrierung der Modelltransformation. Aus diesem Grund kann bei ihnen auf die Typangabe verzichtet werden.



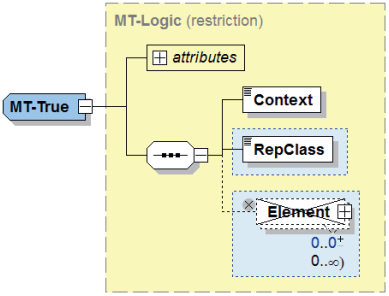
Platzhalter besitzen keine Auftragnehmer und daher auch keine eingebetteten MT-Elemente. Zudem ist die `RepClass` festgelegt auf den Wert „Placeholder“, um zu kennzeichnen, dass dieses MT-Element einen Platzhalter darstellt. Über das zusätzliche Element `RepIdent` wird der Name des Templates angegeben, für das der Platzhalter steht.



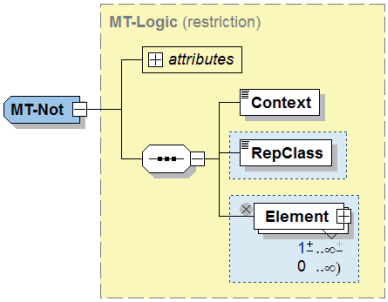
Auch die Verzweigung wird über einen festen Wert für `RepClass` identifiziert. Zudem ist die Anzahl der eingebetteten MT-Elemente auf genau zwei eingeschränkt, eines als Wurzelelement für die erste und eines als Wurzelelement für die zweite Alternative.



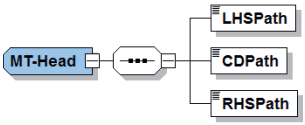
Der Modifikator MT-True ermöglicht die Umsetzung von Leerlaufregeln, indem sie als einziges MT-Element der linken oder rechten Domäne eingesetzt werden. MT-True besitzt keine eingebetteten MT-Elemente.



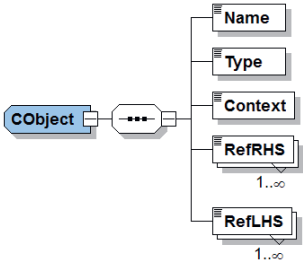
Der Modifikator MT-Not negiert die Rückmeldungen seiner Auftraggeber.



Das Kopfelement wird im Austauschformat vereinfacht repräsentiert, da lediglich die Parametrierung der drei Suchpfade für den Datenaustausch relevant ist.



Für die Elemente der Korrespondenzdomäne ist es wichtig, dass sie jeweils mindestens einen Link zu MT-Elementen der linken und der rechte Domäne besitzen. Um dies beim Datenaustausch zuzusichern, werden die MT-Elemente der Korrespondenzdomäne gesondert definiert.



ACPLT/MT

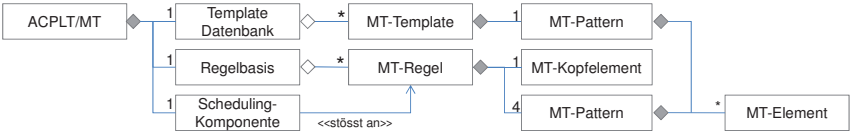
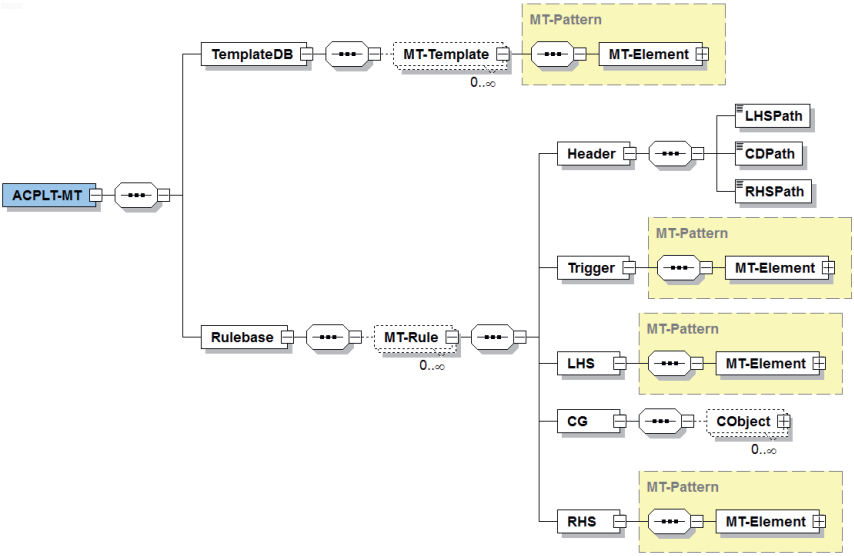


Abbildung A.2: Struktur des ACPLT/MT-Frameworks

Nun kann das Gesamtgebilde aufgebaut werden. Hierzu werden strukturierende Elemente hinzugefügt, die im Laufzeitsystem durch OV-Domänen realisiert sind und keine eigene Logik beinhalten. Abbildung A.2 zeigt nochmals die Gesamtstruktur des ACPLT/MT-Frameworks aus Abbildung 6.2a. Für den Austausch von Modelltransformationen zwischen zwei Systemen mit ACPLT/MT werden nicht alle Komponenten benötigt. Insbesondere der Scheduler besitzt keine Parametrierungsmöglichkeit und muss daher auch nicht in einem Austauschformat bereitgestellt werden.



Vollständige XSD für ACPLT/MT

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:vc="http://www.w3.org/2007/
    XMLSchema-versioning" elementFormDefault="qualified" attributeFormDefault="unqualified"
    vc:minVersion="1.1">
  <xs:element name="ACPLT-MT">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="TemplateDB">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="MT-Template" type="MT-Pattern" minOccurs="0" maxOccurs="
                  unbounded">
                <xs:annotation>
                  <xs:documentation>Der Name des MT-Element ist gleichzeitig der Name des
                      Templates, der durch die Platzhalter referenziert wird.
                  </xs:documentation>
                </xs:annotation>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="Rulebase">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="MT-Rule" minOccurs="0" maxOccurs="unbounded">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="Header" type="MT-Head" minOccurs="1" maxOccurs="1">
                      <xs:annotation>
                        <xs:documentation>Kopfelement der Regel mit Angaben zu Suchpfaden.
                        </xs:documentation>
                      </xs:annotation>
                    </xs:element>
                    <xs:element name="Trigger" type="MT-Pattern" minOccurs="1" maxOccurs="1">
                      <xs:annotation>
                        <xs:documentation>MT-Objekt, das die Regelbearbeitung anstößt.</xs:
                            documentation>
                      </xs:annotation>
                    </xs:element>
                    <xs:element name="LHS" type="MT-Pattern" minOccurs="1" maxOccurs="1">
                      <xs:annotation>
                        <xs:documentation>Pattern der linken Domäne.</xs:documentation>
                      </xs:annotation>
                    </xs:element>
                    <xs:element name="CG">
                      <xs:annotation>
                        <xs:documentation>Pattern der Korrespondenzdomäne.</xs:documentation>
                      </xs:annotation>
                      <xs:complexType>
                        <xs:sequence>
                          <xs:element name="CObject" type="CObject" minOccurs="0" maxOccurs="
                              unbounded"/>
                        </xs:sequence>
                      </xs:complexType>
                    </xs:element>
                    <xs:element name="RHS" type="MT-Pattern" minOccurs="1" maxOccurs="1">
                      <xs:annotation>
                        <xs:documentation>Pattern der rechten Domäne.</xs:documentation>
                      </xs:annotation>
                    </xs:element>
                  </xs:sequence>
                </xs:complexType>
```

```
            </xs:element>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<!-- Typen -->
<xs:complexType name="MT-Pattern" abstract="true">
  <xs:sequence>
    <xs:element name="MT-Element" type="MT-Element" minOccurs="1" maxOccurs="1"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="CObject">
  <xs:sequence>
    <xs:element name="Name" type="xs:string" minOccurs="1" maxOccurs="1"/>
    <xs:element name="Type" type="xs:string" minOccurs="1" maxOccurs="1"/>
    <xs:element name="Context" type="xs:boolean" minOccurs="1" maxOccurs="1"/>
    <xs:element name="RefRHS" type="xs:IDREF" minOccurs="1" maxOccurs="unbounded"/>
    <xs:element name="RefLHS" type="xs:IDREF" minOccurs="1" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="MT-Head">
  <xs:sequence>
    <xs:element name="LHSPath" type="xs:string" minOccurs="1" maxOccurs="1"/>
    <xs:element name="CDPath" type="xs:string" minOccurs="1" maxOccurs="1"/>
    <xs:element name="RHSPath" type="xs:string" minOccurs="1" maxOccurs="1"/>
  </xs:sequence>
</xs:complexType>
<!-- MT-Element -->
<xs:complexType name="MT-Element" abstract="true">
  <xs:sequence>
    <xs:element name="Context" type="xs:boolean" minOccurs="1" maxOccurs="1">
      <xs:annotation>
        <xs:documentation>Markiert, ob ein MT-Element Teil des Kontextes ist oder nicht.
        </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="RepClass" type="xs:string" minOccurs="1" maxOccurs="1">
      <xs:annotation>
        <xs:documentation>Klasse, die das MT-Element in der jeweiligen Domäne repräsentiert.
            Für die Erweiterung MT-Modification wird dieses Feld genutzt, um den Typ der
            Modifikation kenntlich zu machen.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="Element" type="MT-Element" minOccurs="0" maxOccurs="unbounded">
      <xs:annotation>
        <xs:documentation>Eingebettete MT-Elemente. Diese Schachtelung bildet die
            Kommandostruktur ab. </xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
  <xs:attribute name="ID" type="xs:ID" use="required"/>
  <xs:attribute name="ObjName" use="required"/>
</xs:complexType>
<xs:annotation>
  <xs:documentation>Definition der verschiedenen Sub-Typen von MT-Element</xs:documentation>
</xs:annotation>
<xs:complexType name="MT-Variable">
  <xs:sequence>
    <xs:element name="Type" type="xs:string" default="String" minOccurs="0" maxOccurs="1"/>
    <xs:element name="Value" type="xs:string" minOccurs="1" maxOccurs="1"/>
  </xs:sequence>
  <xs:attribute name="ObjName" use="required"/>
</xs:complexType>
```

```
<xs:complexType name="MT-Object">
  <xs:complexContent>
    <xs:extension base="MT-Element">
      <xs:sequence>
        <xs:element name="RepIdent" type="xs:string" minOccurs="1" maxOccurs="1"/>
        <xs:element name="AssocClass" type="xs:string" minOccurs="1" maxOccurs="1"/>
        <xs:element name="AssocRole" minOccurs="1" maxOccurs="1">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:enumeration value="parent"/>
              <xs:enumeration value="child"/>
              <xs:enumeration value="*"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
        <xs:element name="Variable" type="MT-Variable" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="MT-LinkBasic">
  <xs:complexContent>
    <xs:restriction base="MT-Element">
      <xs:sequence>
        <xs:element name="Context" type="xs:boolean" minOccurs="1" maxOccurs="1"/>
        <xs:element name="RepClass" type="xs:string" minOccurs="1" maxOccurs="1"/>
        <xs:element name="Element" type="MT-Element" minOccurs="0" maxOccurs="0"/>
      </xs:sequence>
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="MT-Link">
  <xs:complexContent>
    <xs:extension base="MT-LinkBasic">
      <xs:sequence>
        <xs:element name="Source" type="xs:IDREF" minOccurs="1" maxOccurs="1"/>
        <xs:element name="Target" type="xs:IDREF" minOccurs="1" maxOccurs="1"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<!-- MT-Modifikatoren -->
<xs:complexType name="MT-Modification" abstract="true">
  <xs:complexContent>
    <xs:extension base="MT-Element"/>
  </xs:complexContent>
</xs:complexType>
<xs:annotation>
  <xs:documentation>Definition der verschiedenen Sub-Typen von MT-Modification</xs:
      documentation>
</xs:annotation>
<xs:complexType name="MT-MetaVar">
  <xs:complexContent>
    <xs:restriction base="MT-Variable">
      <xs:sequence>
        <xs:element name="Type" type="xs:string" default="String" minOccurs="0" maxOccurs="0
            "/>
        <xs:element name="Value" type="xs:string" minOccurs="1" maxOccurs="1"/>
      </xs:sequence>
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="MT-PlaceholderBasic">
  <xs:complexContent>
    <xs:restriction base="MT-Modification">
```

```
        <xs:sequence>
          <xs:element name="Context" type="xs:boolean" minOccurs="1" maxOccurs="1"/>
          <xs:element name="RepClass" type="xs:string" fixed="Placeholder" minOccurs="1"
              maxOccurs="1"/>
          <xs:element name="Element" type="MT-Element" minOccurs="0" maxOccurs="0"/>
        </xs:sequence>
      </xs:restriction>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="MT-Placeholder">
    <xs:complexContent>
      <xs:extension base="MT-PlaceholderBasic">
        <xs:sequence>
          <xs:element name="RepIdent" type="xs:string" minOccurs="1" maxOccurs="1"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="MT-Branch">
    <xs:complexContent>
      <xs:restriction base="MT-Modification">
        <xs:sequence>
          <xs:element name="Context" type="xs:boolean" minOccurs="1" maxOccurs="1"/>
          <xs:element name="RepClass" type="xs:string" fixed="Branch" minOccurs="1" maxOccurs=
              "1"/>
          <xs:element name="Element" type="MT-Element" minOccurs="2" maxOccurs="2"/>
        </xs:sequence>
      </xs:restriction>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="MT-Logic">
    <xs:complexContent>
      <xs:extension base="MT-Modification"/>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="MT-True">
    <xs:complexContent>
      <xs:restriction base="MT-Logic">
        <xs:sequence>
          <xs:element name="Context" type="xs:boolean" minOccurs="1" maxOccurs="1"/>
          <xs:element name="RepClass" type="xs:string" fixed="True" minOccurs="1" maxOccurs="1
              "/>
          <xs:element name="Element" type="MT-Element" minOccurs="0" maxOccurs="0"/>
        </xs:sequence>
      </xs:restriction>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="MT-Not">
    <xs:complexContent>
      <xs:restriction base="MT-Logic">
        <xs:sequence>
          <xs:element name="Context" type="xs:boolean" minOccurs="1" maxOccurs="1"/>
          <xs:element name="RepClass" type="xs:string" fixed="Not" minOccurs="1" maxOccurs="1"
              />
          <xs:element name="Element" type="MT-Element" minOccurs="1" maxOccurs="unbounded"/>
        </xs:sequence>
      </xs:restriction>
    </xs:complexContent>
  </xs:complexType>
</xs:schema>
```

Anhang B TGG der Anwendungsszenarien

Die in Kapitel 1.2 definierten Anwendungsszenarien wurden im Rahmen mehrere Veröffentlichungen mit Hilfe des vorgestellten Ansatzes realisiert. Die benötigten Regelsätze werden soweit möglich in diesem Kapitel bereitgestellt und detailliert beschrieben. Der Umfang der Modelle wurde dabei zunächst auf die für die Beispielanlage aus Abbildung B.1 benötigten Elemente reduziert, da eine Erweiterung auf die vollständigen Modelle den Rahmen dieser Arbeit sprengt. Die dafür benötigten Regeln folgen im Allgemeinen jedoch dem gleichen Schema wie die bereitgestellten Regeln und können so im Bedarfsfall einfach ergänzt werden.

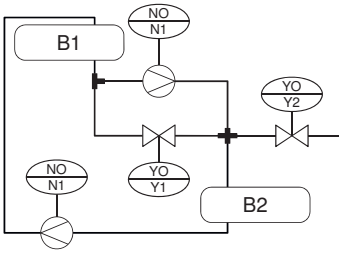


Abbildung B.1: R&I-Fließbild der Beispielanlage

S0 – Bereitstellung von Planungsdaten im Laufzeitsystem

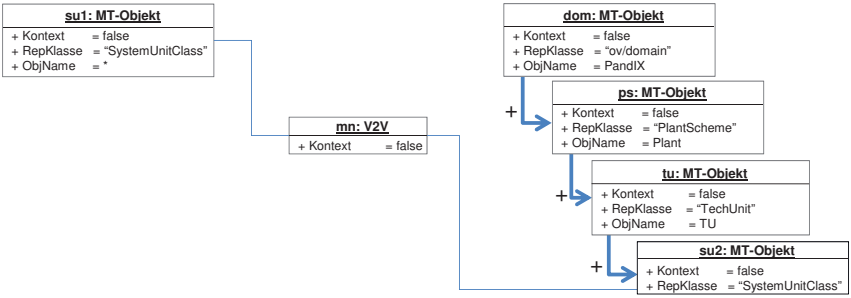
In diesem Anwendungsszenario steht der bidirektionale Datenaustausch zwischen den Planungstools und der prozessleittechnischen Laufzeitumgebung im Vordergrund. Es stellt damit einen Sonderfall für die Anwendung des vorgestellten Konzeptes dar, da das Modell der linken Domäne nicht auf den Basismodellen der Laufzeitumgebung – ACPLT/OV oder im allgemeinen Fall IEC 61131 – beruht sondern ein textuelles XML-basiertes Modell ist. Um diesen Anforderungen gerecht zu werden, wurden die Klasse `MT_Object` so erweitert, dass sie auch in XML-Bäumen nach Modellelementen sucht. Dabei wird nicht wie üblich die Metavariable `Pfad` zur Referenzierung des Suchraumes genutzt, sondern über die Metavariable `Input` ein XML-Baum übergeben. Innerhalb dieses Baumes wird nach entsprechenden Modellelementen gesucht oder es werden entsprechende Modellelemente angelegt. Bei erfolgreicher Suche wird ein Teilbaum des ursprünglichen XML als Suchraum für den Auftragnehmer zurückgeliefert, beim Anlegen entsprechend das neu generierte XML.

```
<SystemUnitClass Name="ExamplePlant">
   <InternalElement Name="Y1" RefBaseClassPath="ActuatorRequest">
      <ExternalInterface Name="Y1:P" RefBaseClassPath="ActuatorProcessInterface"/>
   </InternalElement>

   <InternalElement Name="V1" RefBaseClassPath="ValveRequest">
      <ExternalInterface Name="V1:Y"    RefBaseClassPath="ActuatorInputPoint"/>
      <ExternalInterface Name="V1:PIn"  RefBaseClassPath="ProductConnectionPoint"/>
      <ExternalInterface Name="V1:POut" RefBaseClassPath="ProductConnectionPoint"/>
   </InternalElement>

   <InternalLink Name="link" RefPartnerSideA="Y1:P" RefPartnerSideB="V1:Y"/>

   …
   <InternalElement Name="B1" RefBaseClassPath="VesselRequest">
      <ExternalInterface Name="B1:PIn"  RefBaseClassPath="ProductConnectionPoint"/>
      <ExternalInterface Name="B1:POut" RefBaseClassPath="ProductConnectionPoint"/>
   </InternalElement>

   <InternalElement Name="Z1" RefBaseClassPath="PipeJunctionRequest">
      <ExternalInterface Name="Z1:P1" RefBaseClassPath="ProductConnectionPoint"/>
      <ExternalInterface Name="Z1:P2" RefBaseClassPath="ProductConnectionPoint"/>
      <ExternalInterface Name="Z1:P3" RefBaseClassPath="ProductConnectionPoint"/>
   </InternalElement>

   <InternalElement Name="P1" RefBaseClassPath="PipeRequest">
      <ExternalInterface Name="P1:PIn"  RefBaseClassPath="ProductConnectionPoint"/>
      <ExternalInterface Name="P1:POut" RefBaseClassPath="ProductConnectionPoint"/>
   </InternalElement>

   …
   <InternalLink Name="pipe" RefPartnerSideA="B1:POut" RefPartnerSideB="P1:PIn"/>

   …
</SystemUnitClass>
```

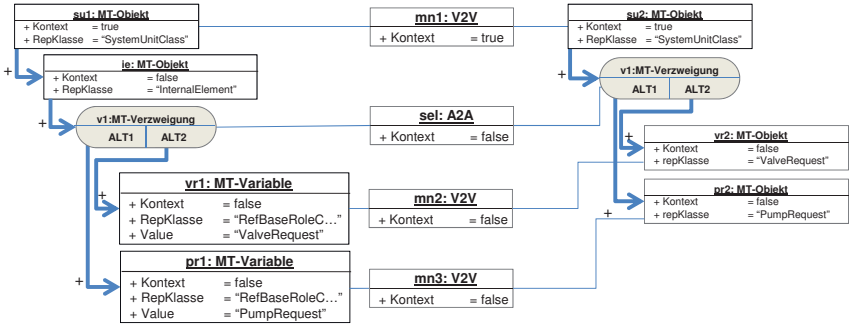
Abbildung B.2: PandIX-XML der Beispielanlage

Abbildung B.2 zeigt das PandIX-XML für die Beispielanlage. In einem ersten Schritt wird mit Hilfe der Regel S0-R1 die Rahmenstruktur in Form einer `ov/domain` und den für eine Anlage benötigten ACPLT/PandIX-Objekten angelegt. Die Hardwarekomponenten wie Pumpen, Ventile, Behälter, Verzweigungen und Rohleitungen werden alle auf die gleiche Weise generiert. Es wäre durchaus legitim dies mit Hilfe einer verschachtelten MT-Verzweigung in einer Regel abzubilden. Zur besseren Lesbarkeit wurde jedoch auf die Schachtelung verzichtet und drei getrennte Regeln S0-R2 für Pumpen und Ventile, S0-R3 für Verzweigungen und Behälter und S0-R4 für Rohrleitungen mit jeweils maximal einer Verzweigung genutzt. Mit Hilfe von Regel S0-R5 werden die Flansche und Anschlussstellen für die PLT-Stellen bereitgestellt. Die Auswertung der `InternalLinks`, die die Verknüpfung der Aktoren mit den PLT-Stellen und die Rohrleitungen symbolisieren erfolgt nach Typ getrennt. Zunächst erfolgt mit der Regel S0-R6 die Auswertung, der Verbindungslinien zwischen Aktoren und PLT-Stellen. In einem weiteren Schritt werden mit Hilfe der Regel S0-R7 die Rohleitungen über die entsprechenden Flansche mit den Aktoren, Verzweigungen und Behältern verbunden.

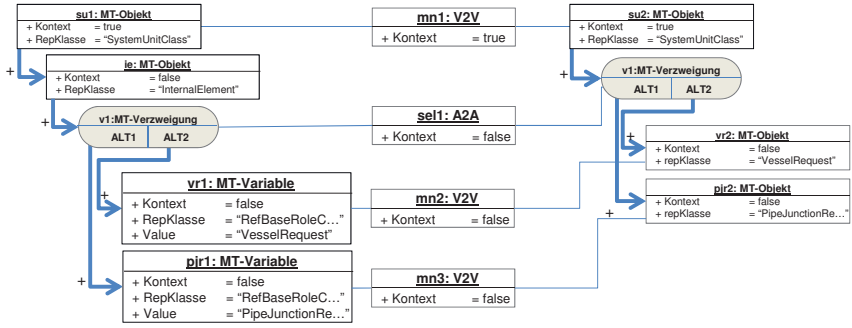
S0-R1: Rahmenstruktur



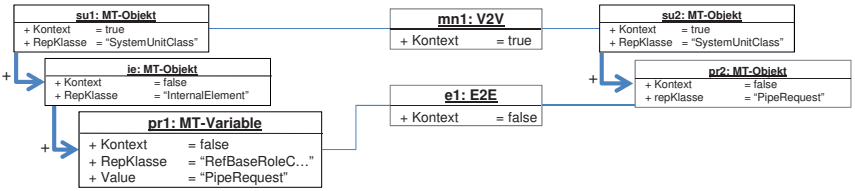
S0-R2: Pumpen und Ventile



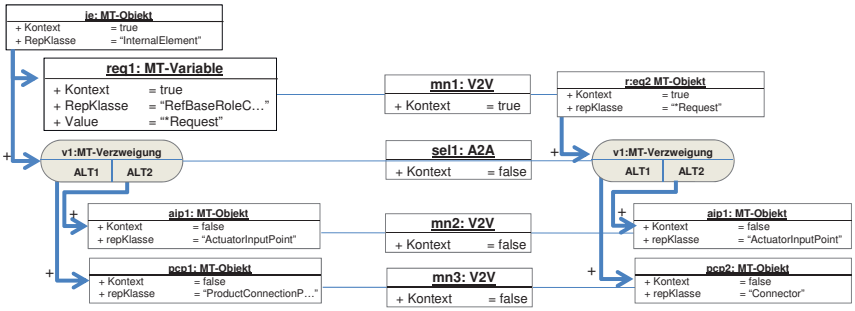
S0-R3: Verzweigungen und Behälter



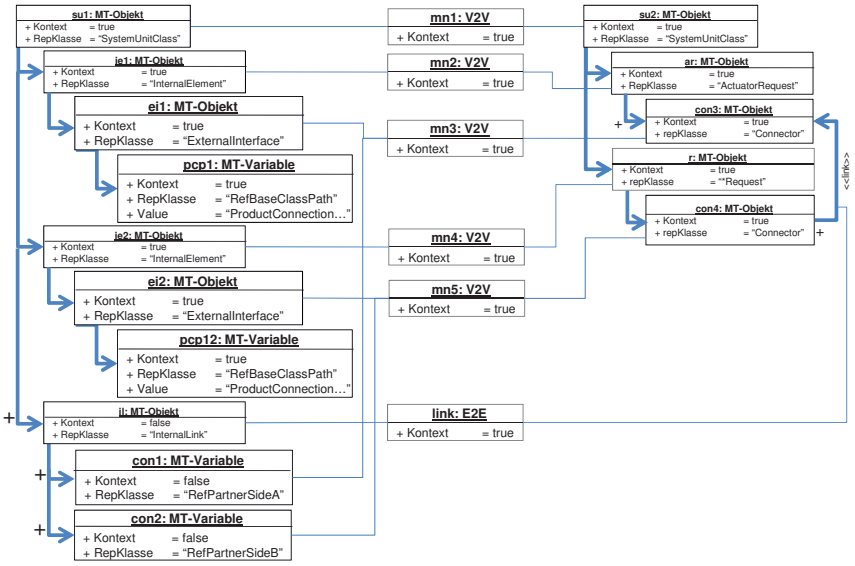
S0-R4: Rohrleitungen



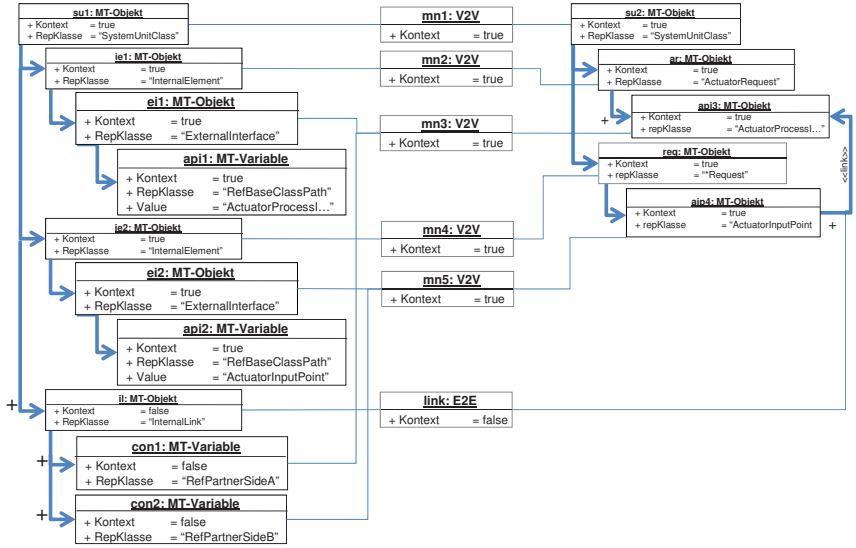
S0-R5: Flansche und Anschlussstellen



S0-R6: Zuordnung der PLT-Stellen



S0-R7: Verknüpfung der Rohrleitungen



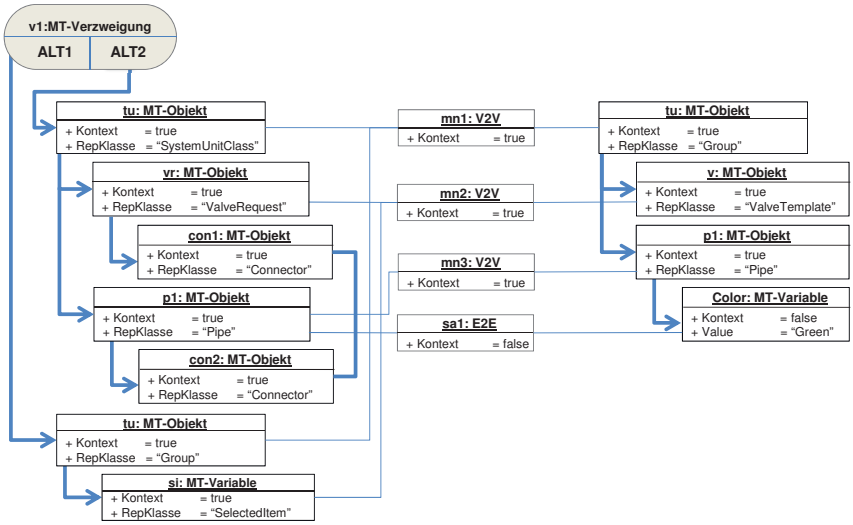
S1 – Einzelne Automatisierungsfunktion als Serienprodukt

Bei diesem Szenario soll der Anlagenfahrer eine Anfrage stellen können, welche Bereiche der Anlage durch das Öffnen eines konkreten Ventils oder durch das Anschalten einer Pumpe betroffen sind. Die Rohrleitungen, die ausgehend von diesem Aktor durch geöffnete Ventile und angeschaltete Pumpen miteinander verbunden sind, werden im Bedienbild farbig markiert.

Dieses Szenario kann nicht auf Basis von einer TGG realisiert werden. Stattdessen werden die ACPLT/MT-Regel so parametriert, dass die Regeln als `WENN-DANN`-Regeln agieren. Die Regel S1-R1 liest sich daher wie folgt: Wenn ein Link zwischen zwei Flanschen (`Connector`) existiert, dann färbe das zugehörige Rohrleitungssymbol im ACPLT/csHMI grün ein. Die MT-Objekte agieren dabei wie gewohnt und suchen bzw. erstellen die repräsentierten Modellobjekte oder setzten Variablenwerte. An dieser Regel wird deutlich, dass ACPLT/MT-Transformationen in gegenseitiger Abhängigkeit zueinander stehen können. So sind die im Kontext verwendeten MT-Objekte noch durch keine andere Regel des Regelsatzes übersetzt worden. Hier wird auf die Ergebnisse der Transformation aus Anwendungsszenario S2 zurückgegriffen, die eine TGG zwischen ACPLT/PandIX und ACPLT/csHMI beschreibt. Diese Korrelation von ACPLT/MT-Transformationen ermöglicht es, verschiedene Anwendungen auf ein und demselben Modellpaar aufzusetzen. Möchte man in der Welt der reinen Tripel-Graph-Grammatiken bleiben, so müssen die Regeln aus Anwendungsszenario S2 mit in den Regelsatz von S1 kopiert werden. Eine weitere TGG zwischen ACPLT/PandIX und ACPLT/csHMI

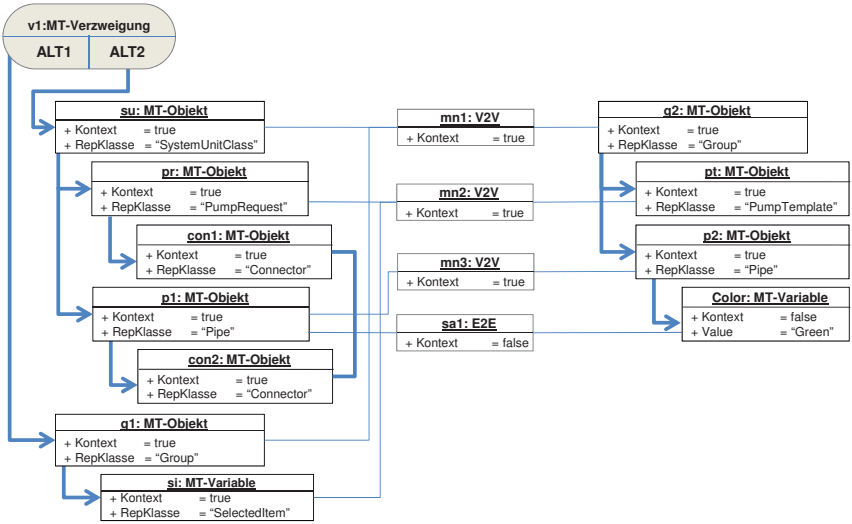
müsste ebenfalls eine Kopie der Regeln aus S2 beinhalten. Bei ein Vorwärtstransformation auf Basis dieser TGG würde dann allerdings eine neues ACPLT/csHMI-Modell erzeugt werden. Der alternative Ansatz, alle ACPLT/MT-Transformationen, die auf einer konkreten prozessleittechnischen Laufzeitumgebung zur Verfügung stehen in einer großen TGG zu vereinigen ist sehr inflexibel bezüglich Erweiterungen. Dennoch, im Großen gesehen funktioniert der Ansatz wie eine komplexe TGG, deren Regeln jedoch immer nur zu definierten Zeitslots – nämlich wenn die zugehörige ACPLT/MT-Transformation aktiv ist – angewendet werden können.

S1-R1: Auswirkung Stellbefehl Ventil als WENN-DANN-Regel



Der Anlagenfahrer kann jedoch nicht nur die Auswirkungen eines Fahrbefehls für ein Ventil abfragen, sondern auch für eine Pumpe. Die dazu gehörige Regel sieht wie folgt aus:

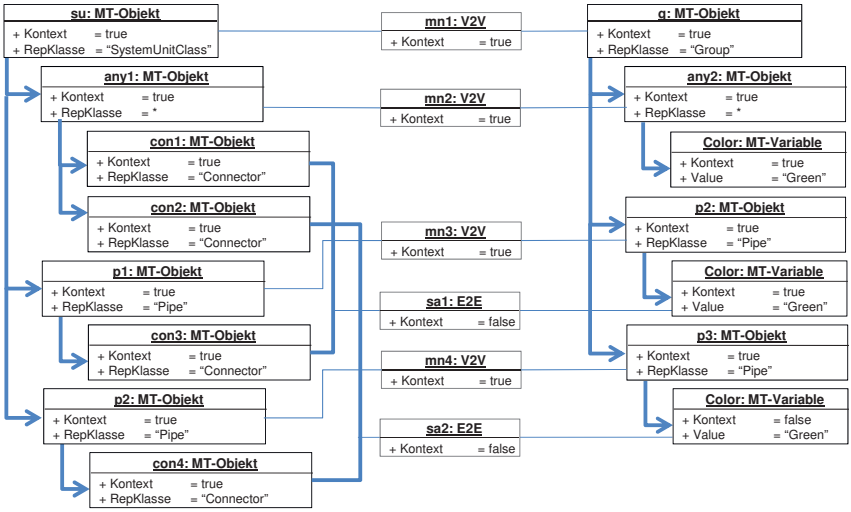
S1-R2: Auswirkung Stellbefehl Pumpe



Die beiden Regeln S1-R1 und S1-R2 ließen sich in eine gemeinsame Regel überführen, indem die Metavariablen `RepKlasse` der MT-Objekte `vr` der linken Domäne und `v` der rechten Domäne mit einem Asterisk parametriert werden.

Im nächsten Schritt werden, ausgehend von einer grün gefärbten Rohrleitung, weitere Rohrleitungen eingefärbt, die durch geöffnete Ventile, laufende Pumpen oder einfache Verzweigungen angebunden sind. Geöffnete Ventile oder laufende Pumpen können anhand ihrer grünen Farbe im HMI identifiziert werden. Die Regel S1-R3 zum Einfärben lautet daher: Wenn die Flansche eines Aktors über entsprechende Links mit zwei Rohrleitungen verbunden sind, dann überprüfe, ob für die entsprechenden Symbole der Rohrleitung im HMI eine der beiden Rohrleitungen bereits grün eingefärbt ist und färbe ggf. die andere ebenfalls grün ein.

S1-R3: Geöffnetes Ventil/ Gestartete Pumpe

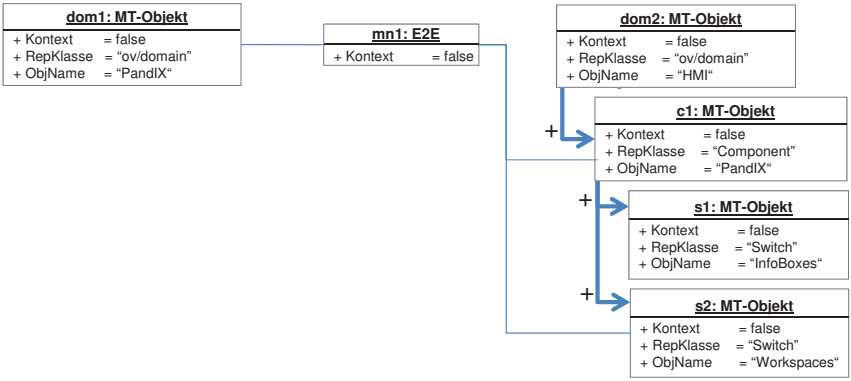


S2 – Modularisierte Teilfunktion

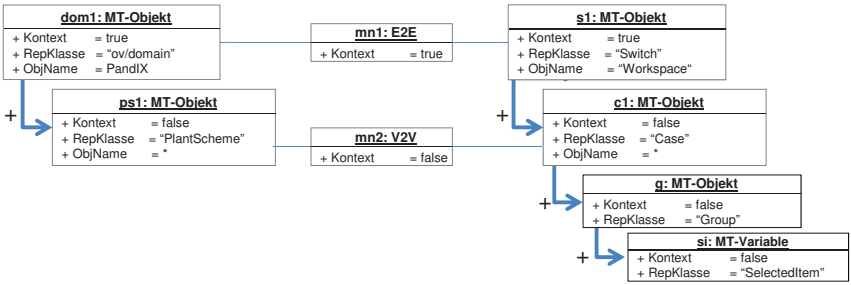
Das Szenario sieht die modellbasierte Erstellung eines einfachen Bedienbildes vor, auf dem für jeden Sensor und jeden Aktor im ACPLT/PandIX-Modell der Anlage ein repräsentierendes Symbol sowie eine Detailansicht für das Ablesen von Sensorwerten bzw. das Absetzen von Befehlen im Bedienbild erstellt wird. Eine anschließende Anpassung an die anlagenspezifischen Gegebenheiten durch den Applikateur muss dabei möglich bleiben.

In einem ersten Schritt werden mit Hilfe von Regel S2-R1 für die beiden Modelle eigene `ov/domain`-Objekte angelegt, die als Wurzelelement für alle weiteren Modellelemente dienen. Eine durch Regel S2-R2 erzeugte Gesamtübersicht zeigt das R&I-Fließbild (PlantScheme) als Ganzes. Hier können im Betrieb aufgetretene Fehler farblich markiert werden und somit eine schnelleren Problemlösung unterstützt werden. Für jede Teilanlage, die im R&I enthalten ist (TechUnit) wird durch Regel S2-R3 eine zusätzliche Ansicht im HMI erstellt, die die zugehörigen Modellobjekte kapselt und im ACPLT/csHMI-Modell den entsprechenden Ausschnitt anzeigt. Somit wird ein Hineinzoomen in einzelne Teilanlagen möglich. Zudem wird ein Objekt `SelectedItem` im ACPLT/csHMI-Modell angelegt, welches später Informationen über das aktuell selektierte Grafikobjekt einer Gruppe liefert. Diese Information wird für den aufbauenden Regelsatz aus Anwendungsszenario S1 benötigt.

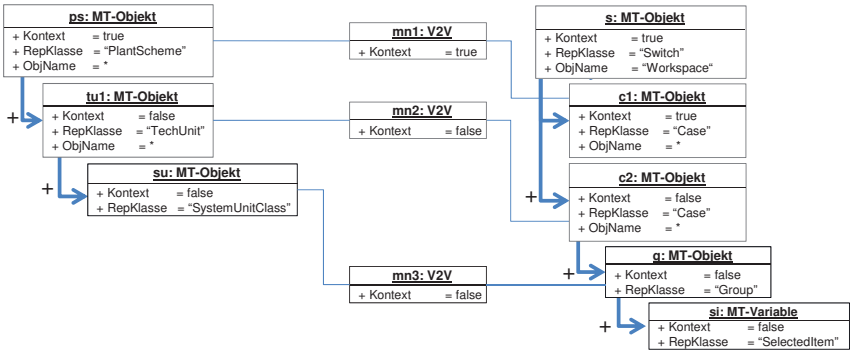
S2-R1: Axiom



S2-R2: Anlage

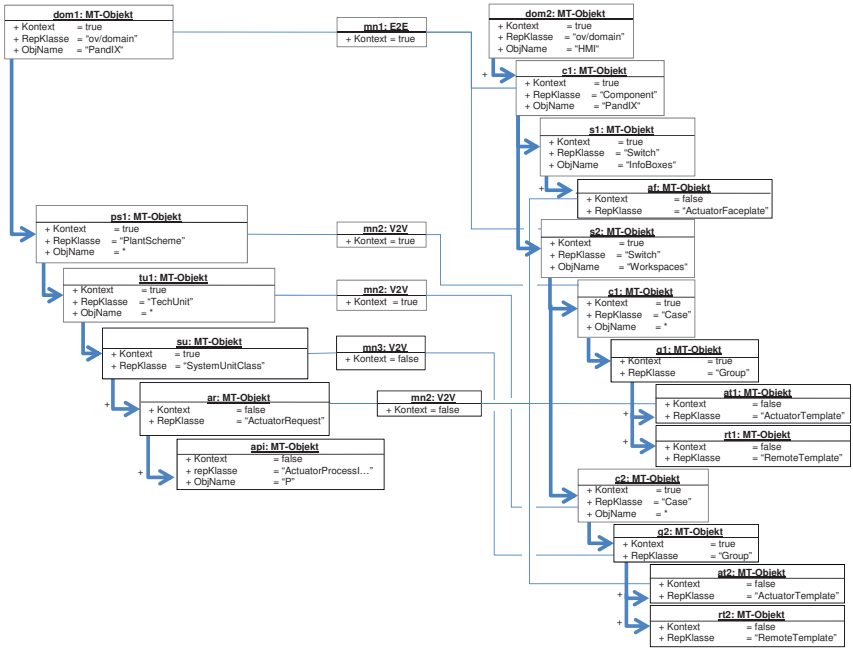


S2-R3: Teilanlage



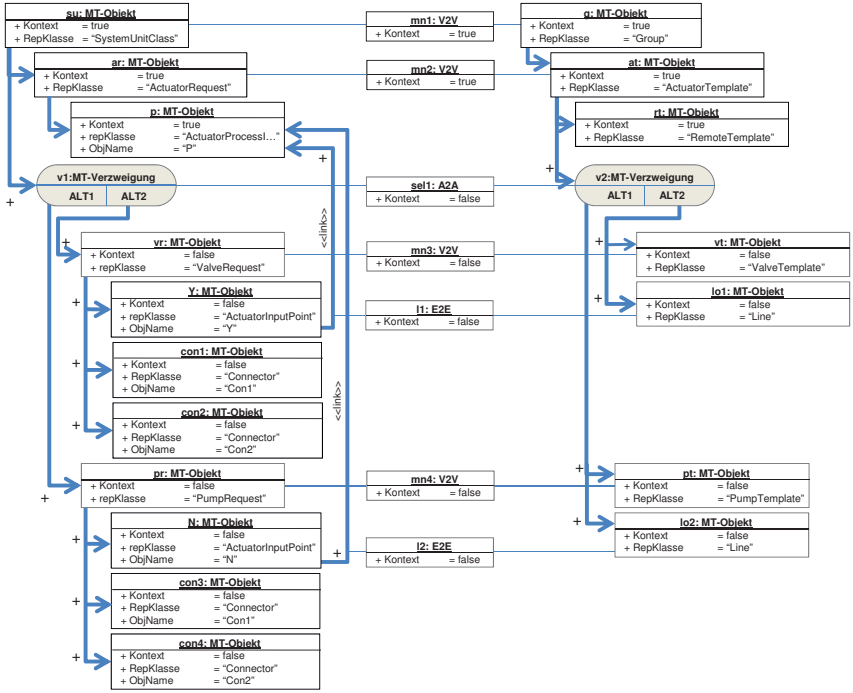
Aktoren (und auch Sensoren) werden im R&I durch eine PLT-Stelle – einem Oval mit entsprechenden Indizes – gekennzeichnet. Regel S2-R4 erzeugt entsprechende Darstellungen für die PLT-Stellen in der Gesamtansicht der Anlage und der Ansicht für die Teilanlage. Zudem wird ein Bedienbild zur Ansteuerung des Aktors in den Infoboxen generiert.

S2-R4: Aktor

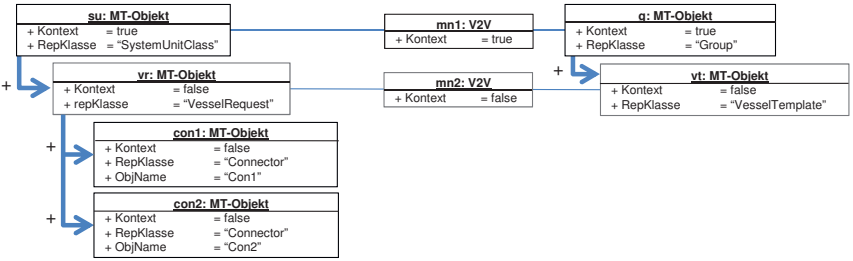


Zusätzlich zu den PLT-Stellen und dem Bedienbild besitzen Aktoren noch ein Icon, das den Typ repräsentiert. Dieses wird durch Regel S2-R5 erzeugt. Die Regel zur Bereitstellung der Icons für Pumpen und Ventile aus Abbildung 6.6c enthielt zur besseren Lesbarkeit noch keine Flansche. Diese sind jedoch für die anschließende Verrohrung elementar. Die Regel S2-R5 erweitert daher die ursprüngliche Version um eben diese Flansche (`Connector`). Auf der anderen Seite wurde auf die Generierung der Icons in der Gesamtdarstellung der Anlage verzichtet. Durch die Regel S2-R6 werden die Behälter und durch Regel S2-R7 die Rohleitungen erzeugt. Auch hierbei wurde auf die Generierung der entsprechenden Symbole in der Gesamtansicht der Anlage verzichtet. Im ACPLT/PandIX sind Rohleitungen durch Links zwischen Flanschen gekennzeichnet. Diese müssen im ACPLT/csHMI durch entsprechende Rohrleittungssymbole gekennzeichnet werden.

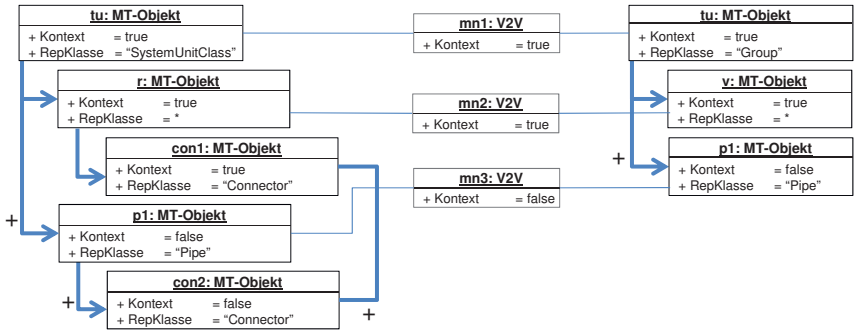
S2-R5: Pumpe/Ventil



S2-R6: Behälter



S2-R7: Rohleitung



Anhang C Schritt-für-Schritt-Anwendung einer ACPLT/MT-Regel

Dieses Kapitel beleuchtet beispielhaft die Anwendung der Regel aus Abbildung C.1a auf die in Abbildung C.1b gezeigte Modellsituation. Dabei sollen drei Detailstufen im Fokus stehen. Zunächst wird tief in die Regel hineingezoomt und die Funktionalität und Arbeitsweise einzelner MT-Elemente genauer betrachtet. Da die Umsetzung dieser Detailstufe stark implentierungsabhängig ist, beziehen sich die Erklärungen hierbei auf die Referenzimplementierung. In einem zweiten Schritt, wird das Zusammenspiel von MT-Elementen auf Basis der Kommandostruktur im Vordergrund stehen. Abschließend wird noch eine Stufe weiter herausgezoomt und die einzelne Regelanwendung im Gesamtbild der ACPLT/MT-Modelltransformation betrachtet. Dabei steht insbesondere das Schedulingverhalten im Fokus.

Bei der Beispielanwendung wird davon ausgegangen, dass durch vorangegangene Regelanwendungen der Kontext bereits vom ACPLT/PF-Modell (linke Domäne) in das ACPLT/csHMI-Modell (rechte Domäne) übersetzt wurde.

Funktionsweise `MT_Element`

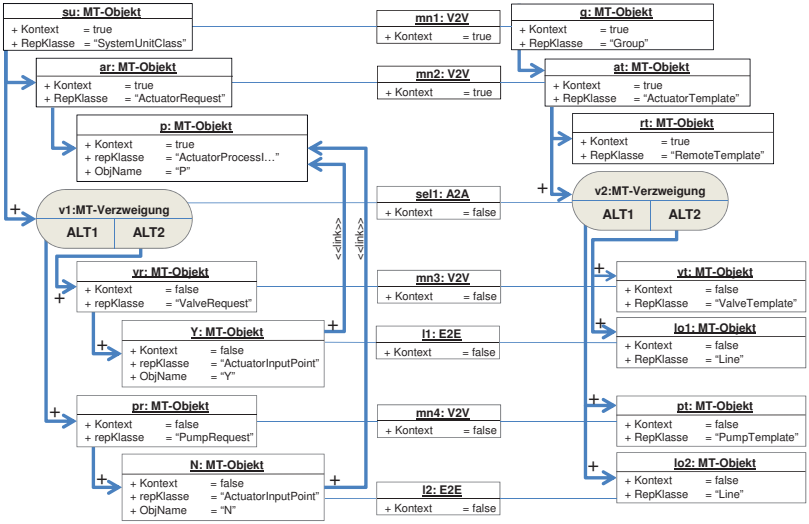
Die Bearbeitung von MT-Elementen wird beispielhaft an zwei MT-Objekten demonstriert. Das erste ist so parametriert, dass es ein entsprechendes Modellobjekt sucht, das zweite so, dass es ein Modellobjekt erstellt. Außerdem wird die Anwendung einer MT-Verzweigung detailliert beleuchtet. Dabei wird unterschieden zwischen der Anwendung einer Verzweigung in der Quelldomäne und einer Verzweigung in der Zieldomäne.

Zunächst soll jedoch die Bedeutung der einzelnen Metavariablen zusammengefasst werden:

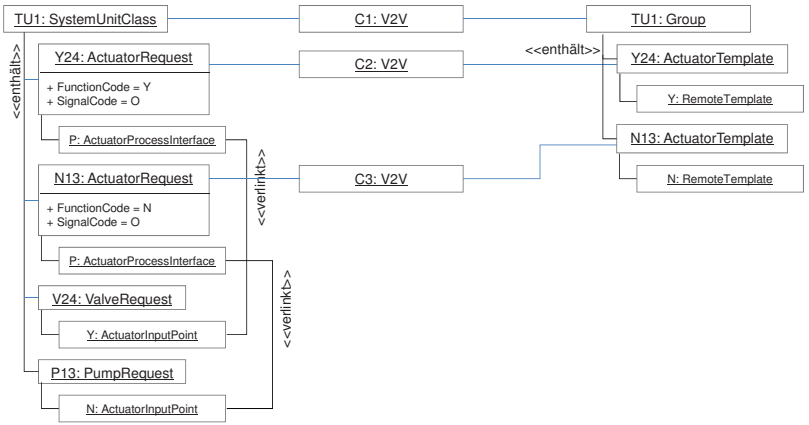
Objektname - `ObjName`
: Gibt an, wie das gesuchte Modellobjekt heißen muss, damit es als potentieller Kandidat in Frage kommt. Ist der Name irrelevant, so kann diese Metavariable mit Jokerzeichen parametriert werden.

Klasse - `RepClass`
: Gibt an, von welcher Klasse das gesuchte Modellobjekt sein soll. Auch hier ist eine Parametrierung mit Jokerzeichen möglich.

Kontext - `Context`
: Wenn das aktuelle MT-Objekt im Kontext der Regel steht, so ist diese Metavariable `true`.



(a) Kombinierte Regel aus Abbildung 6.6c



(b) Zustand der Modelle zum Aktivierungszeitpunkt der Regel

Abbildung C.1: Schritt-für-Schritt-Beispiel

Assoziation - `Assoc`

Gibt an, über welche Assoziation das gesuchte Modellobjekt mit dem über `Path` referenzierten Objekt verknüpft ist.

Rolle - `AssocRole`
Gibt an, ob das gesuchte Modellobjekt in der Rolle der Quelle oder des Ziels der durch `Assoc` bestimmten Assoziation auftritt.

Quellmodell - `SourceDom`
Ist das MT-Objekt in der Teilproduktion des Quellmodells, so ist diese Metavariable `true`. Das Quellmodell ist jenes Modell, von dem aus die Übersetzung stattfinden soll. Bei einer Rückwärtstransformation sind daher die MT-Objekte des linken Teilpattern mit `SourceDom = true` parametriert.

Pfad - `Path`
Die Metavariable `Path` gibt den Startpunkt der Suche an, von dem aus ein Link vom Typ `Assoc` und ein darüber verknüpftes Modellobjekt der Klasse `RepClass` gesucht wird.

Kommando - `Command`
Über diese Metavariable erhält das MT-Objekt seine Kommandos vom Auftraggeber.

ANStatus - `ChbkState`
Die Rückmeldung nachgeschalteter Auftragnehmer kann über die Metavariable ANStatus bzw. in der Referenzimplementierung `ChbkState` zugegriffen werden. Zum Zeitpunkt der ersten Ausführung eines MT-Objektes sind die Auftragnehmer noch im Zustand `ST_IDLE`.

Die Metavariablen-Ausgänge haben folgende Semantik:

ANKommando - `SendCommand`
Das Kommando für den Auftragnehmer wird mit Hilfe der Metavariablen ANKommando bzw. in der Referenzimplementierung `SendCommand` bereitgestellt. Dieser Ausgang ist mit dem Eingang `Command` des Auftragnehmers verknüpft.

Status - `State`
Gibt den Schritt zurück, indem sich das MT-Objekt befindet. Dies dient als Rückmeldung an den eigenen Auftraggeber.

Objektkennung - `ActIdent`
Wurde ein passendes Modellobjekt gefunden oder erstellt, so gibt diese Metavariable den Namen dieses Modellobjekt an.

Objektpfad - `ActPath`
Diese Metavariable gibt den Pfad zum gefundenen Modellobjekt an. Der Pfad dient für nachgeschalteten Auftragnehmern als Startpunkt für die Suche und wird daher mit der Metavariable `Path` der Auftragnehmer verknüpft.

`MT_Object` - Suche eines Modellobjekts

Die Suche eines Modellobjekts wird anhand des MT-Objekts `ar` aus der linken Domäne der Regel in Abbildung C.1a gezeigt. Dieses bekommt von seinem Auftraggeber `su` als Suchpfad „/TU1“ übergeben und erkundet ausgehend von dem dazugehörigen Modellobjekt `TU1` die Modellinstanz nach Objekten, die vom Typ `ActuatorRequest` sind und per `ov/containment` mit `TU1` verlinkt sind.

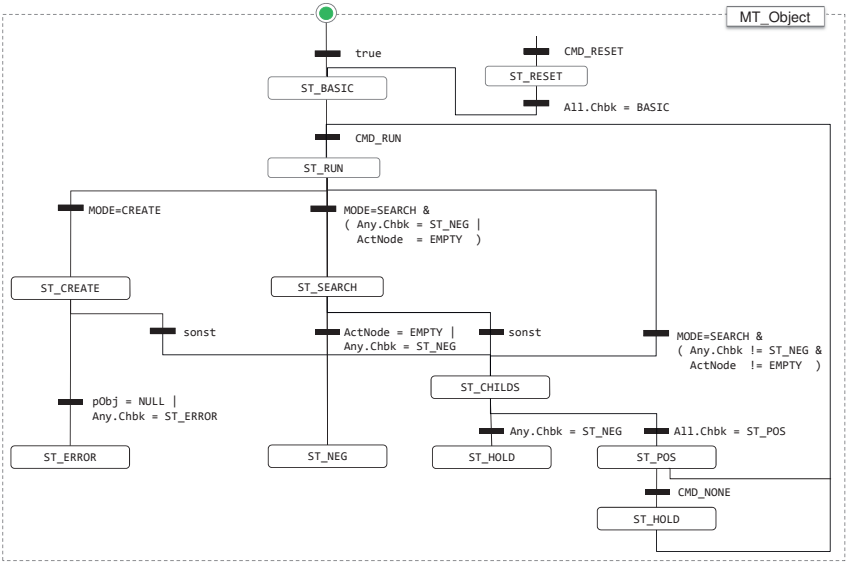


Abbildung C.2: SFC des MT_Object

MT_OBJECT - SUCHE EINES MODELLOBJEKTS ar: MT_Object			
Belegung der Metavariablen (Eingänge)			
Deklarative Ebene		*Operationale Ebene*	
ObjName	= *	Assoc	= „ov/containment“
RepClass	= „ActuatorRequest“	AssocRole	= CHILD
Context	= true	SourceDom	= true
		Path	= „/TU1“
Kommandostruktur			
Command	= CMD_RUN		
ChbkState	= ST_IDLE		
Belegung der Metavariablen (Ausgänge)			
Kommandostruktur		*Operationale Ebene*	
SendCommand	= CMD_RUN	ActIdent	= „Y24“
State	= ST_RUN	ActPath	= „/TU1/Y24“

Der SFC eines MT_Objekt ist in Abbildung C.2 nochmals dargestellt. Im **ersten Zyklus** befindet sich `ar` im Schritt `ST_IDLE`. Das anliegende Kommando `CMD_RUN` führt jedoch zu einem

Schrittwechsel nach ST_RUN.[1] In diesem Schritt wird anhand der Metavariablen Context und SourceDom festgestellt, ob ein Modellobjekt gesucht oder erstellt werden soll und die lokale Variable Modus des MT-Objektes entsprechend auf SEARCH oder CREATE gesetzt wird. Im aktuellen Beispiel ist ar Teil des Kontextes des Quellmodells, d.h. der Modus wird auf SEARCH gesetzt. Dies wiederum führt zu einem erneuten Schrittwechsel nach ST_SEARCH. Im Schritt ST_SEARCH wird zunächst ein Zeiger auf das durch die Metavariable Path referenzierte Objekt generiert. Existiert das Objekt nicht, so wird unmittelbar in den Schritt ST_NEG gewechselt. Das durch Path referenzierte Modellobjekt wird nach Links untersucht, die vom Typ her der Parametrierung der Metavariablen Assoc entsprechen. Diese Links sind in einer geordneten Liste organisiert. Das durch den ersten Link verknüpfte Modellobjekt ist ein erster potentieller Kandidat.

Im konkreten Beispiel wird zunächst ein Zeiger auf das Objekt TU1 erstellt. Dessen Linkliste für die Assoziation ov/containment enthält vier potentielle Kandidaten: Y24, N13, V24, P13. Diese werden nun nacheinander überprüft. Y24 erfüllt direkt die übrigen Bedingungen, da es eine Instanz der Klasse ActuatorRequest ist und Ziel bzw. CHILD des Links zu TU1 ist. Da keine weiteren Anforderungen an das Modellobjekt beschrieben sind, kann Y24 an ar gebunden werden und die Metavariablen ActIdent und ActPath von ar werden auf „Y24“ bzw. „/TU1/Y24“ gesetzt. Anschließend wird das Kopfelemente durch ar über diese Bindung informiert. Außerdem wird über die Metavariable SendCommand das Kommando CMD_RUN an den Auftragnehmer, in diesem Fall an das MT-Objekt p, gesendet. Da p auf das Modellobjekt P angewendet werden kann und selbst keine Auftragnehmer nachgeschaltet hat, meldet es einen positiven Verlauf (State = ST_POS) und gibt die Kontrolle an ar zurück. Dieser überprüft die Metavariablen-Ausgänge aller Auftragnehmer, also aller MT-Elemente, die per Link mit der Metavariablen SendCommand verknüpft sind und bildet daraus zusammenfassend eine Rückmeldung aller Auftragnehmer (ChbkState). Befindet sich mindestens ein Auftragnehmer im Schritt ST_ERROR, so wird auch der ChbkState = ST_ERROR gesetzt. Die zweithöchste Priorität hat die Rückmeldung ST_NEG, gefolgt von ST_BASIC. Nur wenn alle Auftragnehmer ST_POS melden, wird auch ChbkState auf ST_POS gesetzt. Durch die positive Rückmeldung von p erfolgt erneut ein Schrittwechsel über ST_CHILDS nach ST_POS.

Im **zweiten Zyklus** erhält das MT-Objekt ar erneut das Kommando CMD_RUN. Da die Suche im vorangegangenen Zyklus positiv verlaufen ist, erhält der Auftragnehmer p den Auftrag, nach weiteren potentiellen Kandidaten zu suchen. Da neben P jedoch kein weiteres Modellobjekt die Bedingungen des MT-Objektes p erfüllen kann, gibt es in diesem Zyklus eine negative Rückmeldung (ST_NEG) vom Auftragnehmer. Das MT-Objekt ar wechselt daraufhin in den Schritt ST_HOLD. Dieser Zwischenschritt ist notwendig, da die Auftragnehmer nur einmal pro Zyklus bearbeitet werden dürfen, um die Echtzeitfähigkeit nicht zu gefährden.

Im **dritten Zyklus** liegt wiederum das Kommando ST_RUN an. Es erfolgt ein Schrittwechsel über ST_RUN nach ST_SEARCH. Dadurch werden weitere potentielle Kandidaten gesucht, die die Anforderungen von ar erfüllen. Das Modellobjekt N13 erzeugt zusammen mit sei-

[1] Zur Erinnerung: Die Bearbeitung der internen Logik der Schritte, sowie die Auswertung der Transitionen, die sich auf Kommandos beziehen erfolgt im PreTasking. Die Auswertung der Transitionen die sich auf Rückmeldungen von Auftragnehmer beziehen im PostTasking. Es können daher mehrere Schritte im gleichen Zyklus durchlaufen werden.

nem ActuatorProcessInterface erneut eine positive Rückmeldung und anschließend im **vierten Zyklus** ein ST_HOLD.

Auch im **fünften Zyklus** liegt das Kommando CMD_RUN an. Das MT-Objekt ar kann jedoch keinen weiteren potentiellen Kandidaten finden, da V24 und P13 nicht den Anforderungen entsprechen. Es wird daher in den Schritt ST_NEG gewechselt, der nur durch das Kommando CMD_RESET wieder verlassen werden kann. Kommt ein solcher Befehl zum Zurücksetzten, wird es zum einem an die Auftragnehmer weitergeleiten und auf der anderen Seite werden ActIdent und ActPath zurückgesetzt. Dies bewirkt, dass beim nächsten Mal, wenn der Schritt ST_SEARCH durchlaufen wird, wieder am Anfang der Linksliste mit der Suche nach potentiellen Kandidaten begonnen wird.

MT_Object - Erstellen eines Modellobjekts

Das Erstellen eines Modellobjekts wird anhand des MT-Objekts vt der rechten Domäne der Regel aus Abbildung C.1a demonstriert.

MT_OBJECT - ERSTELLEN EINES MODELLOBJEKTS
vt: MT_Object

Belegung der Metavariablen (Eingänge)			
Deklarative Ebene		*Operationale Ebene*	
ObjName	= *	Assoc	= „ov/containment“
RepClass	= „ValveTemplate“	AssocRole	= CHILD
Context	= false	SourceDom	= false
		Path	= „/TU1“
Kommandostruktur			
Command	= CMD_RUN		
ChbkState	= ST_IDLE		
Belegung der Metavariablen (Ausgänge)			
Kommandostruktur		*Operationale Ebene*	
SendCommand	= CMD_RUN	ActIdent	= „Y24“
State	= ST_RUN	ActPath	= „/TU1/Y24“

Im **ersten Zyklus** befindet sich auch dieses MT-Objekt zunächst im Schritt ST_IDLE und wechselt durch das anliegende Kommando CMD_RUN nach ST_RUN. Anders als im vorangegangenen Beispiel wird dieses Mal jedoch der Modus auf CREATE gesetzt, da es sich um ein MT-Objekt der Zieldomäne handelt, das nicht zum Kontext gehört. Dies wiederum führt zu einem Schrittwechsel nach ST_CREATE. In diesem Schritt wird zunächst entlang des Korrespondenzlinks mn3 die Belegung der Metavariablen ObjName des MT-Objekts vt bestimmt. Dieser muss übereinstimmen mit dem Wert der Metavariablen ActIdent des MT-Elements vr der linken Domäne. Im konkreten Beispiel ist dies „V24“. Wie in ST_SEARCH versucht vt zunächst, ein passendes Modellobjekt zu finden, dieses Mal mit festgelegtem Objektnamen. Diese Suche

bleibt im aktuellen Beispiel erfolglos, da noch kein ValveTemplate im Zielmodell existiert. Es wird daher ein solches Objekt angelegt und über einen Link vom Typ ov/containment mit dem Modellobjekt Y24 verknüpft. Tritt dabei ein Fehler auf, so wechselt das MT-Objekt vt in den Schritt ST_ERROR. Ansonsten wird in den Schritt ST_CHILDS gewechselt, der jedoch direkt wieder verlassen wird, da vt keine eigenen Auftragnehmer besitzt. Da die Auswertung der Rückmeldungen im PostTasking erfolgt, wird diese positive Rückmeldung zunächst an den Auftraggeber weitergeleitet, so dass der Übersetzungsschritt durch das Kopfelement und die Korrespondenzobjekte dokumentiert werden kann.

Da in der linken Domäne keine weitere Instanz der Klasse ValveRequest existiert, wird das MT-Objekt vt im **zweiten Zyklus** das Kommando CMD_RESET erhalten. Anschließend verbleibt es im Schritt ST_BASIC.

MT_Branch - Quelldomäne

Die MT-Verzweigung hat mehrere Ausprägungen. Wird sie aktiviert, so wird zunächst anhand der Metavariablen SourceDom und Mode festgestellt, in welcher Ausprägung die MT-Verzweigung zum Einsatz kommt. Im hier vorliegenden Beispiel, tritt sie als v1 in der Quelldomäne als ODER auf und durchläuft den in Abbildung C.3 dargestellten SFC. Dieser überschreibt den Schritt ST_RUN der Basisklasse MT-Element und stellt zwei getrennte Schritte für die Bearbeitung der beiden Alternativen bereit.

MT_BRANCH - QUELLDOMÄNE
v1: MT_Branch

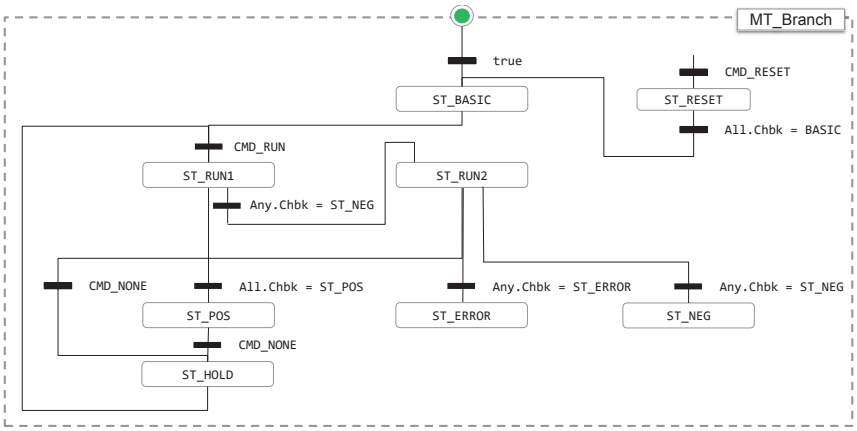
Belegung der Metavariablen (Eingänge)

Deklarative Ebene		*Operationale Ebene*	
Context	= false	SourceDom	= true
		Path	= „/TU1“
Kommandostruktur		Mode	= OR
Command	= CMD_RUN		
ChbkState	= ST_IDLE		
ChbkState2	= ST_IDLE		

Belegung der Metavariablen (Ausgänge)

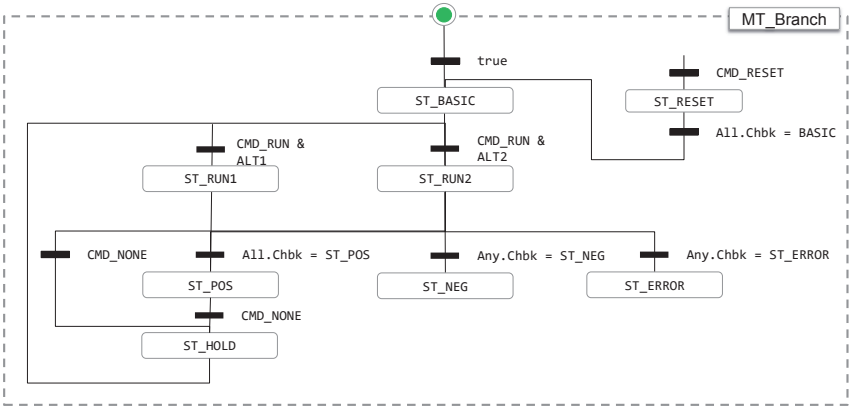
Kommandostruktur		*Operationale Ebene*	
SendCommand	= CMD_RUN	ActPath	= „/TU1/Y24“
SendCommand2	= CMD_RUN		
State	= ST_RUN1		

Im **ersten Zyklus** stößt sie die MT-Objekte der ersten Alternativen an. Liefern diese eine positive Rückmeldung, so wird zunächst ebenfalls eine positive Rückmeldung generiert. Im konkreten Beispiel sind im ersten Zyklus ar und p gebunden an die Modellobjekte Y24 und dem unterlagerten P, da diese in der Linkliste von TU1 vor N13 und dessen unterlagerten P stehen.



Legende: `Any.Chbk` – Mindestens ein Auftragnehmer liefert diese Rückmeldung
`All.Chbk` – Alle Auftragnehmer liefern diese Rückmeldung

(a) Quelldomäne



Legende: `Any.Chbk` – Mindestens ein Auftragnehmer liefert diese Rückmeldung
`All.Chbk` – Alle Auftragnehmer liefern diese Rückmeldung

(b) Zieldomäne

Abbildung C.3: SFC der MT-Verzweigung

Die erste Alternative von `v1` sucht nach einer Instanz der Klasse `PumpRequest` im Modell und identifiziert `P13` als potentiellen Kandidaten. Auch das MT-Objekt `N` passt noch auf das Modellobjekt `N`. Eine passende Verlinkung fehlt jedoch. Aus diesem Grund, liefert die erste Alternative im ersten Zyklus eine negative Rückmeldung. Die MT-Verzweigung wechselt daher in den Schritt `ST_RUN2` und stößt die Bearbeitung der zweiten Alternativen an. Diese findet

in V24 und Y zwei passende Modellobjekte, die auch über die verlangte Verlinkung verfügen. Die zweite Alternative gibt daher eine positive Rückmeldung, die die MT-Verzweigung an ihren Auftraggeber weiterreichen kann. Im **zweiten Zyklus** wird nach weiteren Treffern für die zweite Alternative gesucht. Da keine weiteren passenden Modellobjekte existieren, wird eine negative Rückmeldung an den Auftraggeber geliefert. Dies veranlasst dies Suche nach weiteren Modellobjekten, die der Parametrierung von der MT-Objekte ar und p entsprechen. Im **dritten Zyklus** erhält die MT-Verzweigung von seinem Auftraggeber das Kommando zum zurücksetzen (CMD_RESET). Die MT-Verzweigung leitet diese Kommando an beide Alternativen weiter, so dass diese ihre gespeicherten Werte löschen und wieder in den Ausgangszustand zurückkehren. Im **vierten Zyklus** sind die MT-Objekte ar und p an die Modellobjekte N13 und dessen unterlagertes P gebunden und die erste Alternative kann bereits eine positive Rückmeldung liefern. Im **fünften Zyklus** wird zunächst wieder versucht, weitere Modellobjekte zu finden, die auf die erste Alternative passen. Da dies fehlschlägt wird die zweite Alternative angestoßen. Auch diese liefert eine negative Rückmeldung, so dass die MT-Verzweigung diese an ihren Auftraggeber weiterleitet.

MT_Branch - Zieldomäne

In der Zieldomäne werden die beiden Schritte RUN1 und RUN2 nicht nacheinander durchlaufen. Vielmehr gibt die aktive Alternative der zugehörigen MT-Verzweigung in der Quelldomäne vor, welche der beiden Zweige aktiviert wird (vgl. Abbildung C.3b). Ist in der der Quelldomäne die erste Alternative aktiv, so wird auch in der Zieldomäne die erste Alternative aktiviert. Selbiges gilt sinngemäß für die zweite Alternative.

MT_BRANCH - ZIELDOMÄNE
v2: MT_Branch

Belegung der Metavariablen (Eingänge)

Deklarative Ebene		*Operationale Ebene*	
Context	= false	SourceDom	= false
		Path	= „/TU1“
Kommandostruktur		Mode	= OR
Command	= CMD_RUN		
ChbkState	= ST_IDLE		
ChbkState2	= ST_IDLE		

Belegung der Metavariablen (Ausgänge)			
Kommandostruktur		*Operationale Ebene*	
SendCommand	= CMD_RUN	ActPath	= „/TU1/Y24“
SendCommand2	= CMD_RUN		
State	= ST_RUN1		

In dem konkreten Beispiel aus Abbildung C.1, ist im **ersten Zyklus** die zweite Alternative im Quellmodell aktiv. Daher wird auch die zweite Alternative im Zielmodell aktiviert und es werden jeweils Instanzen der Klasse `ValveTemplate` und `Line` angelegt. Im **zweiten und dritten Zyklus** wird das rechte Teilpattern nicht aktiviert, da das linke eine negative Rückmeldung liefert und anschließend zurückgesetzt wird.

Im **vierten Zyklus** ist die erste Alternative der Quelldomäne aktiv, was sich so auch auf die MT-Verzweigung der Zieldomäne spiegelt. Dadurch werden im Zielmodell Instanzen der Klassen `PumpTemplate` und `Line` erstellt. Im **fünften Zyklus** wird das Teilpattern der Zieldomäne wiederum nicht aufgerufen.

Funktionsweise einer MT-Regel

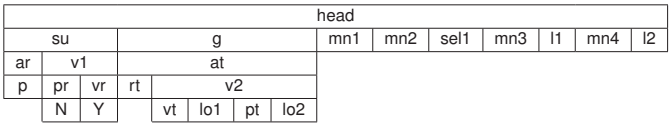
Bevor auf die schrittweise Bearbeitung der Regel eingegangen wird, soll die Bearbeitungsreihenfolge der MT-Elemente einer Regel in den Fokus gerückt werden. Diese bestimmt maßgeblich das Ergebnis eines Übersetzungsschritts. Ausschlaggebend für die Bearbeitungsreihenfolge ist die Schachtelung der MT-Elemente. Die folgende Tabelle stellt dies für die Regel aus Abbildung C.1a strukturiert dar. Das Kopfelement der Regel (`head: REC`) ist das einzige MT-Element, das in der globalen Taskliste `WorkingMemory` registriert ist. Es enthält `su`, `g`, `mn1` - `mn4`, `sel1`, `l1`, `l2` als eingebettete MT-Elemente und ist damit für das Scheduling dieser Bausteine zuständig. Durch grau hinterlegte Zellen markiert, welche MT-Elemente jeweils aktiv sind.

head												
su			g				mn1	mn2	sel1	mn3	l1	mn4	l2
ar	v1		at										
p	pr	vr	rt	v2									
	N	Y		vt	lo1	pt	lo2						

Im obigen Beispiel ist die Bearbeitung der internen Logik von `vr` aktiv. Da dieses MT-Element jedoch eingebettet ist in `v1` und dieses wiederum über `su` in `head` sind diese Bausteine ebenfalls aktiv und führen gerade ihren `Ov_Call1` aus (vgl. Abbildung 6.14).

Vor ihrer Aktivierung liegt die Regel als inaktives Objektnetzwerk vor und ist nicht in das Tasking eingebunden.

ZYKLUS 1
Zustand vor Schritt 1



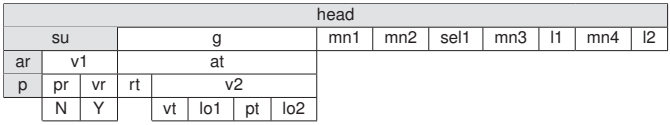
Schritt 1: `head` – MT-Kopfelement
Wird das Kopfelement aktiviert, so bestimmt es zunächst anhand seiner Metavariablen die Transformationsrichtung. In dem konkreten Beispiel liegt eine Vorwärtstransformation vor. Das MT-Kopfelement setzt daher die Metavariable `SourceDom` des linken Teilpatterns auf `true` und die des rechten auf `false` anschließend stößt es zunächst die MT-Objekte der linken Domäne an.

Schritt 2: `su` – MT-Objekt
Die Metavariable `Path` des MT-Objekt `su` ist so parametriert, dass es überall im Quellmodell nach einem potentiellen Kandidaten sucht („/*“). Bei der Suche wird `TU1` gefunden. Eine Abfrage beim Datenbankobjekt stellt sicher, dass `TU1` bereits übersetzt ist und `su` seine Auftragnehmer `ar` und `v1` aktivieren kann. Diese bearbeiten „parallel“ ihre Suche.

Schritt 3: `ar` – MT-Objekt Da Funktionsbausteine sequentiell entsprechend ihrer Reihenfolge in der Taskliste bearbeitet werden, startet zunächst `ar` mit der Suche nach einem potentiellen Kandidaten und findet `Y24`. Auch hier wird überprüft, ob das Modellobjekt bereits übersetzt ist.

Schritt 4: `p` – MT-Objekt Angestoßen von seinem Auftraggeber findet `p` das Modellobjekt `P` unterhalb von `Y24`. Es wird überprüft, ob das MT-Objekt, dass über den MT-Link mit `p` verbunden ist bereits gebunden ist. Dies ist jedoch nicht der Fall. Deshalb, und weil `p` keine eigenen Auftragnehmer besitzt, gibt `p` eine positive Rückmeldung.

ZYKLUS 1
Zustand vor Verlassen von Schritt 4



Schritt 5: `ar` – MT-Objekt
Nachdem `p` die Kontrolle zurückgegeben hat, wertet `ar` die Rückmeldung aus und gibt seinerseits eine positive Rückmeldung.

Schritt 6: `su` – MT-Objekt

Nachdem `su` die Kontrolle zurück erhalten hat, aktiviert es den nächsten Funktionsbaustein in seiner internen Taskliste - `v1`.

Schritt 7: `v1` – MT-Verzweigung

Die MT-Verzweigung aktiviert zunächst ihre erste Alternative.

Schritt 8: `pr` – MT-Objekt

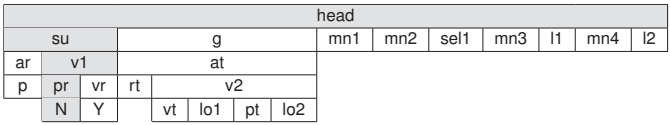
Das Modellobjekt `P13` wird als potentieller Kandidat für `pr` gefunden. Die Überprüfung beim Datenbankobjekt bestätigt, dass das gefundene Modellobjekt noch nicht übersetzt wurde.

Schritt 9: `N` – MT-Objekt

Das Modellobjekt `N` wird als potentieller Kandidat gefunden. Das MT-Objekt `N` überprüft wiederum, ob das durch den MT-Link verbundene MT-Objekt `p` bereits gebunden ist. Da das der Fall ist, wird zwischen den beiden zugehörigen Modellobjekten eine Entsprechung für den MT-Link gesucht. Ein solcher Link existiert jedoch nicht. Das MT-Objekt `N` erzeugt daher eine negative Rückmeldung.

ZYKLUS 1

`Zustand vor Verlassen von Schritt 9`



Schritt 10: `v1` – MT-Verzweigung

Die Kontrolle wird von `N` an `pr` und von da weiter an `v1` zurückgegeben. Die negative Rückmeldung von `N` wird dabei mit nach oben gereicht. Die MT-Verzweigung aktiviert auf Grund der negativen Rückmeldung die zweite Alternative.

Schritt 11: `v1` – MT-Verzweigung

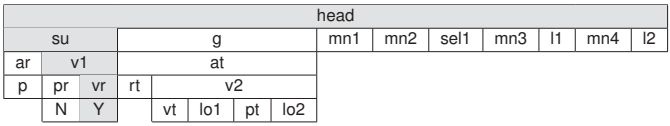
Da die Auswertung der Rückmeldungen im `PostTasking` der MT-Verzweigung durchgeführt wird, kann die zweite Alternative nicht im gleichen Zyklus durchlaufen werden. Stattdessen wird die Kontrolle zunächst zurück an `su` und von da an `head` gegeben. Im **zweiten Zyklus** liegt weiterhin das Kommando `CMD_RUN` am MT-Objekt `su` an und wird an die Auftragnehmer weitergeleitet. Während `ar` und `p` das Kommando `CMD_NONE` erhalten und in den Schritt `ST_HOLD` wechseln, wird in der MT-Verzweigung nun die zweite Alternative angestoßen.

Schritt 12: `Y` – MT-Objekt

Die MT-Objekte `vr` und `Y` finden potentielle Kandidaten in den Modellobjekten `V24` und `Y`. Der Link zwischen `ActuatorInputPoint` und `ActuatorProcessInterface` ist in diesem Fall vorhanden. Es wird eine positive Rückmeldung generiert.

ZYKLUS 2

`Zustand vor Verlassen von Schritt 12`

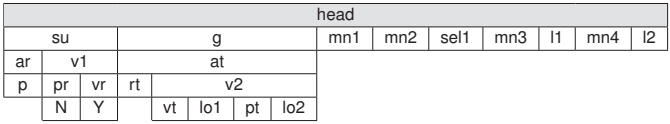


Schritt 13: `head` – Kopfelement

Die positive Rückmeldung wird bis ins Kopfelement hochgereicht. Dieses aktiviert daraufhin die rechte Teilproduktion. Da die Rückmeldung wiederum im `PostTasking` ausgewertet wurde, wird die Kontrolle zunächst jedoch an die globale Taskliste zurückgegeben und erst im **dritten Zyklus** wird das rechte Teilpattern aktiviert. Eine Aktivierung des linken Teilpattern erfolgt in diesem Zyklus nicht.

ZYKLUS 3

`Zustand vor Verlassen von Schritt 13`



Schritt 14: `g` – MT-Objekt

Das MT-Objekt `g` beschafft sich über den Korrespondenzlink den Objektnamen des durch `su` gebundenen Modellobjektes und sucht anschließend einen potentiellen Kandidaten im ACPLT/csHMI, der ebenfalls diesen Objektnamen trägt und auch die anderen Anforderungen erfüllt. Das Modellobjekt `TU1` erfüllt diese Anforderungen. Eine Abfrage am Datenbankobjekt stellt sicher, dass die beiden Modellobjekte `TU1` in der Prozessführung und `TU1` im ACPLT/csHMI über ein Kontextelement miteinander verbunden sind. Wäre dies nicht der Fall, müsste die Suche nach einem potentiellen Kandidaten für `g` fortgeführt werden. In diesem konkreten Beispiel gibt es jedoch eine positive Rückmeldung vom Datenbankelement, so dass `g` seinen Auftragnehmer aktivieren kann.

Schritt 15: `at` – MT-Objekt

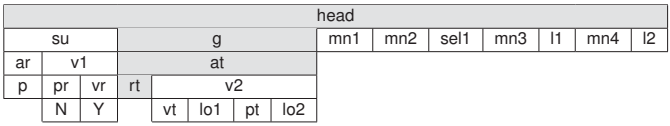
Das MT-Objekt `at` findet nach demselben Muster das Modellelement `Y24` im csHMI als potentiellen, bereits übersetzten Kandidaten.

Schritt 16: `rt` – MT-Objekt

Das MT-Objekt `rt` wird im Modellobjekt `Y` fündig. Da es selbst keine Auftragnehmer besitzt, liefert es eine positive Rückmeldung.

ZYKLUS 3

Zustand vor Verlassen von Schritt 16



Schritt 17: `v2` – MT-Verzweigung

Das MT-Objekt `at` bearbeitet weiter seine interne Taskliste und aktiviert `v2`. Diese MT-Verzweigung erkennt anhand der entsprechenden Metavariable seine Zugehörigkeit zum Teilpattern der Zieldomäne und holt sich deshalb über den Korrespondenzlink die aktive Alternative der zugehörigen MT-Verzweigung in der Teilproduktion der Quelldomäne. Da in der Quelldomäne die zweite Alternative aktiv ist, aktiviert `v2` ebenfalls die Auftragnehmer in der zweiten Alternative.

Schritt 18: `vt` – MT-Objekt

Das MT-Objekt `vt` informiert sich mit Hilfe des Korrespondenzlinks über den Objektnamen des gebundenen Modellobjekts von `vr` und durchsucht im ACPLT/csHMI die Linkliste von `Y24` nach einer Instanz der Klasse `ValveTemplate` mit gleichem Namen. Im konkreten Beispiel sucht es nach einem Objekt `V24`, wird dabei aber nicht fündig. Es wird daher eine Instanz der Klasse `ValveTemplate` angelegt und mit `Y24` verlinkt. Anschließend wird eine positive Rückmeldung generiert.

Schritt 19: `lo1` – MT-Objekt

Die MT-Verzweigung `v2` aktiviert auch den zweiten Auftragnehmer in seiner Taskliste `lo1`. Bei diesem wiederholt sich sinngemäß Schritt 18, so dass am Ende das Modellobjekt `Y24` des ACPLT/csHMI zwei untergeordnete Modellobjekte `V24` und `Y` besitzt. Anschließend wird eine positive Rückmeldung generiert.

Schritt 20: `head` – Kopfelement

Die positive Rückmeldung wird entlang der Rückmeldestruktur bis zum Kopfelement weitergereicht. Da dieses nun sowohl vom der linken als auch vom rechten Teilpattern eine positive Rückmeldung erhalten hat, werden nacheinander die Objekte der Korrespondenzdomäne angestoßen. Dies erfolgt im **vierten Zyklus**.

Schritt 21: `mn2` – V2V

Die Korrespondenzobjekte im Kontext der Regel melden dem Kopfelement, dass sie als Kontext zum Einsatz kamen. Diese Information kann für spätere Modelländerungen relevant sein, bei denen Teile des Kontextes verändert werden.

Schritt 22: `I2` – V2V

Die Korrespondenzobjekte, die nicht Teil des Kontextes der Regel sind, melden an das Kopfelement die durch sie verknüpften Modellelemente anhand des Pfades.

Schritt 23: `head` – Kopfelement

Das Kopfelement meldet alle gesammelten Daten über den erfolgreichen Transformationsschritt an das Datenbankobjekt, das diese Daten dauerhaft vorhält.

Schritt 24: Suche nach weiteren passenden Modellobjekten

Im **fünften Zyklus** sendet das MT-Objekt `su` zunächst das Kommando `CMD_NONE` an seinen ersten Auftragnehmer `ar`, der dies an seinen eigenen Auftragnehmer weiterleitet. Der zweite Auftragnehmer von `su` erhält erneut das Kommando `CMD_RUN` und wird somit beauftragt nach weiteren potentiellen Kandidaten zu suchen. In diesem Fall liefern jedoch beide Alternativen eine negative Rückmeldung, die bis zum MT-Objekt `su` weitergereicht wird. Dieses wechselt in den Schritt `ST_HOLD`, um dem Kopfelement zu melden, dass die Suche im aktuellen Zyklus nicht erfolgreich war. Das Kopfelement verzichtet daher auf die Aktivierung der rechten Teilproduktion und der Korrespondenzobjekte. Im nächsten Zyklus erhält die Verzweigung das Kommando zum Zurücksetzen. Alle anderen MT-Elemente der linken Domäne bleiben im Zustand `ST_HOLD`. Das rechte Teilpattern sowie die Korrespondenzobjekte werden wiederum nicht angestoßen. Anschließend wird das MT-Objekt `ar` beauftragt, einen weiteren potentiellen Kandidaten zu suchen. Dieses gibt diesen Auftrag zunächst weiter an `p`. Es wird daher nach einer weiteren Instanz der Klasse `ActuatorProcessInterface` unterhalb des Modellobjektes `Y24` gesucht. Da ein solches nicht existiert, wird eine negative Rückmeldung von `p` erzeugt und `ar` wechselt in den Schritt `ST_HOLD`.

Auf die gleiche Weise wird auch ein `PumpTemplate` samt `Line` im ACPLT/csHMI angelegt passend zum `PumpRequest` und `N` der Prozessführung.

Funktionsweise einer MT-Modelltransformation

Während in den letzten Abschnitten die Funktionsweise einzelner MT-Elemente und deren Zusammenspiel im Vordergrund standen, liegt nun das Hauptaugenmerk auf dem MT-Scheduler und dem Gesamtverlauf einer Modelltransformation. Abbildung C.4 zeigt nochmals den SFC des Schedulers aus Abbildung 6.13. Es wird ersichtlich, dass nach dem Starten des MT-Schedulers zunächst das `TriggerMemory` initialisiert wird. Dazu werden die Trigger aller Regeln in die Taskliste `TriggerMemory` gehangen und durch das Kommando `CMD_RUN` aktiviert.

Trigger sind einzelne MT-Elemente, deren Suchpfad nicht auf die beiden korrelierenden Modelle beschränkt ist. So überwacht zum Beispiel der Trigger für die Regel aus Abbildung C.1a die Rückmeldung der Regel, die für die Übersetzung des `ActuatorRequest` und des `ActuatorProcessInterface` zuständig ist. Gibt diese Regel die Rückmeldung `Ready`, so schlägt der Trigger an. Die Zustandsüberwachung anderer Regeln als Trigger ist nur dann geeignet, wenn die Transformation nur als reine Batch-Transformation zum Einsatz kommt. Soll ein späteres inkrementelles Update durchgeführt werden, muss direkt auf Modelländerungen reagiert werden. Eine mögliche Realisierung ist, die Trigger so zu gestalten, dass sie dauerhaft eine positive Rückmeldung geben. Dieser Ansatz würde die Performance jedoch deutlich schwächen. Besser ist eine Realisierung über Dienste, die die korrelierenden Modelle überwachen und mittels Variablen über Modelländerungen informieren. Die Trigger können dann so gestaltet sein, dass sie auf Änderungen der Variablen reagieren.

Schlägt einer der Trigger im `TriggerMemory` an, so wird die zugehörige Regel in die Taskliste `WorkingMemory` eingehängt und das Kommando `CMD_RUN` an das Kopfelement gesendet.

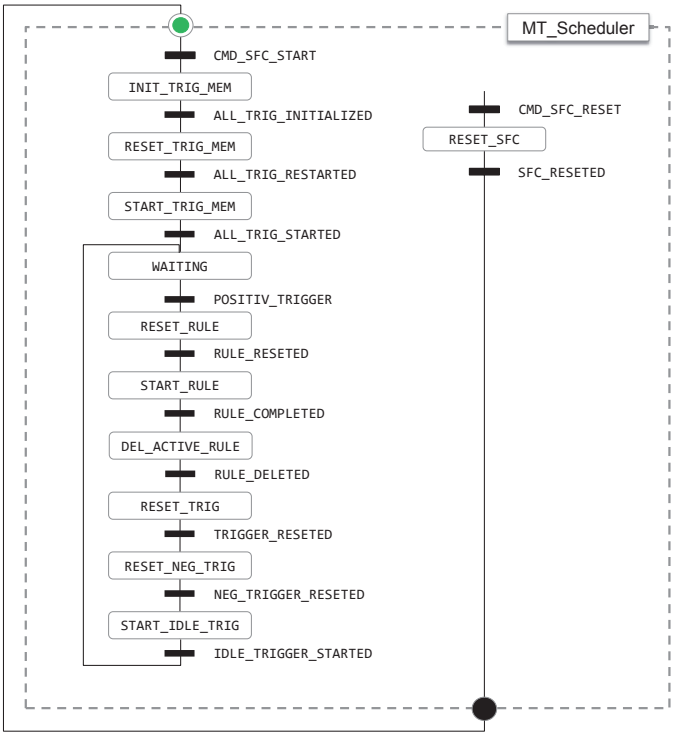


Abbildung C.4: ACPLT/MT-Scheduling

Schlagen mehrere Trigger gleichzeitig an, so wird anhand der Position des Triggers in der Taskliste entschieden, welche Regel bearbeitet wird. Diese wiederum ist abhängig von der Reihenfolge der Regeln in der Regelbasis. Trigger, die einmal eine positive Rückmeldung erzeugt haben, verbleiben zunächst im Schritt `ST_POS` bis die zugehörige Regel bearbeitet wurde.

Die aktivierte Regel im `WorkingMemory` wird zunächst zurückgesetzt, um ggf. zurückgebliebene Artefakte aus vorangegangenen Durchläufen zu beseitigen. Anschließend wird die Regel vollständig abgearbeitet, d.h. alle Übersetzungsschritte, die durch sie im aktuellen Modellzustand durchgeführt werden können, werden auch durchgeführt. Anschließend gibt die Regel die Rückmeldung `Ready`. Parallel zur Bearbeitung der Regel werden die Trigger des `TriggerMemory` weiter bearbeitet. Schlägt in der Zwischenzeit ein weiterer Trigger an, so verbleibt er im Zustand `ST_POS`.

Liefert die Regel im `WorkingMemory` die Rückmeldung `Ready`, so wird sie vom MT-Scheduler wieder aus der Taskliste entfernt und der zugehörige Trigger wird zurückgesetzt und dadurch veranlasst, erneut auf das entsprechende Ereignis zu reagieren.

Literaturverzeichnis

[Alb96] Harald Albrecht. *Technologiepapier Nr. 1: Übersicht über ACPLT/KS*. Techn. Ber. ACPLT, RWTH Aachen University, 1996 (Referenziert auf Seite 32).

[Alb97a] Harald Albrecht. *Technologiepapier Nr. 2: Der Nachrichtentransport*. Techn. Ber. ACPLT, RWTH Aachen University, 1997 (Referenziert auf Seite 32).

[Alb97b] Harald Albrecht. *Technologiepapier Nr. 3: Manager, Server und Klienten*. Techn. Ber. ACPLT, RWTH Aachen University, 1997 (Referenziert auf Seite 32).

[Anj+15] Anthony Anjorin, Erhan Leblebici, Roland Kluge, Andy Schürr und Perdita Stevens. "A Systematic Approach and Guidelines to Developing a Triple Graph Grammar". In: *Bidirectional Transformations. 4th International Workshop, Bx 2015. L'Aquila, Italy. July 24, 2015. Proceedings*. 2015, S. 81–95 (Referenziert auf Seite 50).

[Anj14] Anthony Anjorin. "Synchronization of models on different abstraction levels using triple graph grammars". Diss. TU Darmstadt, 2014 (Referenziert auf den Seiten 50, 105).

[Aßm+14] Uwe Aßmann, Sebastian Götz, Jean-Marc Jézéquel, Brice Morin und Mario Trapp. "A Reference Architecture and Roadmap for Models@run.time Systems". In: *Models@run.time*. Hrsg. von Nelly Bencomo, Robert France, Betty H.C. Cheng und Uwe Aßmann. Bd. 8378. Lecture Notes in Computer Science. Springer International Publishing, 2014, S. 1–18 (Referenziert auf Seite 54).

[Ber+15] Gábor Bergmann, István Dávid, Ábel Hegedüs, Ákos Horváth, István Ráth, Zoltán Ujhelyi und Dániel Varró. "Viatra 3: A Reactive Model Transformation Platform". In: *Theory and Practice of Model Transformations . Theory and Practice of Model Transformations: 8th International Conference, ICMT 2015, Held as Part of STAF 2015, L'Aquila, Italy, July 20-21, 2015.* 2015, S. 101–110 (Referenziert auf Seite 54).

[BS09] Stefan Biffl und Alexander Schatten. "A Platform for Service-Oriented Integration of Software Engineering Environments". In: *Proceedings of the 2009 Conference on New Trends in Software Methodologies, Tools and Techniques: Proceedings of the Eighth SoMeT_09*. 2009, S. 75–92 (Referenziert auf Seite 40).

[CH06] Krzysztof Czarnecki und Simon Helsen. "Feature-based survey of model transformation approaches". In: *IBM Systems Journal* 45.3 (2006), S. 621–645 (Referenziert auf Seite 44).

[Cho65] Noam Chomsky. *Aspects of the Theory of Syntax*. The MIT Press Paperback Series. M.I.T. Press, 1965 (Referenziert auf Seite 17).

[Div08] Industry Sector / Industry Automation Division. *Siemens verstärkt sich durch Übernahme von innotec mit Software für die Prozessindustrie*. [Online; Stand 14.06.2015]. 2008. URL: http://www.siemens.com/press/de/pressemitteilungen/2008/industry_automation/iia2008081645.htm (Referenziert auf Seite 41).

[Ens01] Udo Enste. *Generische Entwurfsmuster in der Funktionsbausteintechnik und deren Anwendung in der operativen Prozeßführung*. Bd. 884. Fortschritt-Berichte VDI Reihe 8. VDI Verlag, 2001. (zugleich Dissertation. RWTH Aachen University. 2001) (Referenziert auf den Seiten 26, 33).

[EPL13] EPLAN Software & Service. *Neue Schwester für EPLAN: Kuttig Computeranwendungen GmbH*. [Online; Stand 14.06.2015]. 2013. URL: http://www.eplan.de/de/unternehmen/presse/pressemeldungen/view/article/neue-schwester-fuer-eplan-kuttig-computeranwendungen-gmbh/ (Referenziert auf Seite 41).

[ERD11] Ulrich Epple, Markus Remmel und Oliver Drumm. “Modellbasiertes Format für RI-Informationen”. In: *Automatisierungstechnische Praxis : atp edition* 53 (2011), S. 62–71 (Referenziert auf Seite 24).

[Fre92] Gottlob Frege. “Sinn und Bedeutung”. In: *Zeitschrift für Philosophie und philosische Kritik* NF 100 (1892). S. 25–50. nachgedruckt In *Funktion, Begriff, Bedeutung: fünf logische Studien. Kleine Vandenhoeck-Reihe* (1962), S. 38–63 (Referenziert auf den Seiten 14, 15).

[GE14] Sten Grüner und Urlich Epple. “Regelbasiertes Engineering mit Graphabfragen”. In: *Tagungsband. Dagstuhl-Workshop MBEES: Modellbasierte Entwicklung eingebetteter Systeme X. Model-Based Development of Embedded Systems. 05.03.2014-07.03.2014*. 2014, S. 105–111 (Referenziert auf Seite 42).

[GK07] Joel Greenyer und Ekkart Kindler. “Reconciling TGGs with QVT”. In: *Model Driven Engineering Languages and Systems: 10th International Conference, MODELS 2007, Nashville, USA, September 30 - October 5, 2007. Proceedings*. 2007, S. 16–30 (Referenziert auf Seite 52).

[GKE12] Sten Grüner, David Kampert und Ulrich Epple. “A Model-Based Implementation of Function Block Diagram”. In: *Tagungsband. Dagstuhl - Workshop MBEES: Modellbasierte Entwicklung eingebetteter Systeme VI. Model-Based Development of Embedded Systems. 06.02.2012-08.02.2012*. 2012, S. 81–90 (Referenziert auf Seite 32).

[GPR11] Joel Greenyer, Sebastian Pook und Jan Rieke. “Preventing information loss in incremental model synchronization by reusing elements”. In: *Modelling – Foundation and Applications 7th European Conference, ECMFA 2011, Birmingham, UK, June 6-9, 2011, Proceedings*. 2011, S. 144–159 (Referenziert auf Seite 55).

[Güt09] Knut Güttel. “Konzept zur Generierung von Steuerungscode unter Verwendung wissensbasierter Methoden in der Fertigungsautomatisierung”. In: *AUTOMATION 2009 . Der Automationskongress in Deutschland*. Bd. 2067. VDI-Berichte. VDI-Verlag, 2009, S. 309–312 (Referenziert auf den Seiten 41, 42).

[GWE14] Sten Grüner, Peter Weber und Ulrich Epple. “A model for discrete product flows in manufacturing plants”. In: *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation, ETFA 2014, Barcelona, Spain, September 16-19, 2014*. 2014, S. 1–8 (Referenziert auf den Seiten 4, 42, 105).

[I4.0] Plattform Industrie 4.0. *Umsetzungsstrategie Industrie 4.0*. [Online; Stand 03.08.2015]. 2015. URL: http://www.plattform-i40.de/sites/default/files/150410_Umsetzungsstrategie_0.pdf (Referenziert auf Seite 1).

[JE12] Holger Jeromin und Ulrich Epple. “Anwendungs- und herstellerneutrales Modell zur Darstellung und Interaktion mit leittechnischen Funktionen”. In: *AUTOMATION 2012. 13. Branchentreff der Mess- und Automatisierungstechnik*. Bd. 2171. VDI-Berichte. VDI-Verlag, 2012, S. 219–222 (Referenziert auf den Seiten 25, 33).

[JE13] Holger Jeromin und Ulrich Epple. “Modellbasiertes und technologieneutrales HMI für eingebettete Komponenten”. In: *Dagstuhl-Workshop MBEES: Modellbasierte Entwicklung eingebetteter Systeme IX, Schloss Dagstuhl, Germany, April 24-26, 2013*. 2013, S. 80–89 (Referenziert auf Seite 32).

[Kla12] Felix Klar. “Efficient and Compatible Bidirectional Formal Language Translators based on Extended Triple Graph Grammars”. Diss. TU Darmstadt, 2012 (Referenziert auf den Seiten 44, 98, 99).

[KME12] Tina Krausser, Henning Mersch und Ulrich Epple. “Rule-based Adaption of Distributed Automation Systems at Operation Time”. In: Bd. 45. IFAC Proceedings Volumes 6. IFAC Online, 2012, S. 793–798 (Referenziert auf den Seiten 105, 106).

[Kön08] Alexander Königs. “Model Integration and Transformation - A Triple Graph Grammar-based QVT Implementation”. Diss. TU Darmstadt, 2008 (Referenziert auf den Seiten 44, 48, 49, 66, 70, 99).

[KQ11] Tina Krausser und Gustavo Quirós. “An IEC-61131-based rule system for Integrated Automation Engineering: Concept and Case Study”. In: *IEEE Xplore. Digital Library. Proceedings of the 9th IEEE International Conference on Industrial Informatics. 26-29 July 2011. Caparica, Lisbon, Portugal*. 2011 (Referenziert auf den Seiten 54, 92, 96).

[Kra+12] Tina Krausser, Marius Lauder, Michael Schlereth, Ulrich Epple und Andy Schürr. “Integrated Graph Transformations in Automation Systems”. In: Bd. 45. IFAC Proceedings Volumes 2. IFAC Online, 2012, S. 872–877 (Referenziert auf den Seiten 54, 92, 96).

[KSY10] Tina Krausser, Stefan Schmitz und Liyong Yu. “Regelbasierte Vollständigkeitsüberprüfung von Automatisierungslösungen”. In: *AUTOMATION 2010. Der 11. Branchentreff der Mess- und Automatisierungstechnik*. Bd. 2092. VDI-Berichte. VDI-Verlag, 2010, S. 55–58 (Referenziert auf Seite 97).

[KW07] Ekkart Kindler und Robert Wagner. *Triple graph grammars: Concepts, extensions, implementations, and application scenarios*. Techn. Ber. tr-ri-07-284. Universität Paderborn, 2007 (Referenziert auf den Seiten 52, 55).

[KWB03] Anneke G. Kleppe, Jos Warmer und Wim Bast. *MDA Explained: The Model Driven Architecture: Practice and Promise*. Addison-Wesley Longman Publishing Co., Inc., 2003 (Referenziert auf Seite 37).

[Lau12] Marius Lauder. “Incremental Model Synchronization with Precedence-Driven Triple Graph Grammars”. Diss. TU Darmstadt, 2012 (Referenziert auf den Seiten 44, 51, 99, 104).

[Leb+15] Erhan Leblebici, Anthony Anjorin, Andy Schürr und Gabriele Taentzer. “Multi-amalgamated Triple Graph Grammars”. In: *Graph Transformation: 8th International Conference, ICGT 2015, Held as Part of STAF 2015, L'Aquila, Italy, July 21-23, 2015. Proceedings*. 2015, S. 87–103 (Referenziert auf den Seiten 44, 51, 105).

[Leb+16] Erhan Leblebici, Anthony Anjorin, Andy Schürr und Gabriele Taentzer. “Multi-amalgamated triple graph grammars: Formal foundation and application to visual language translation”. In: *Journal of Visual Languages & Computing* (2016) (Referenziert auf den Seiten 44, 51, 105).

[M4P] Lars Evertz und Ulrich Epple. “M4P.AC - Auf der Schnittlinie zwischen Informationswelt und realer Welt”. In: *12. Fachtagung EKA - Entwurf komplexer Automatisierungssysteme, 09.-10.05.2012, Magdeburg, Otto-von-Guericke-Universität Magdeburg/ifak*. 2012 (Referenziert auf Seite 94).

[ME11] Henning Mersch und Ulrich Epple. “Requirements on Distribution Management for Service-Oriented Automation Systems”. In: *IEEE Xplore. Proceedings of 2012 IEEE 17th International Conference on Emerging Technologies & Factory Automation ETFA 2012. September 17-21, 2012. Krakow, Poland*. 2011 (Referenziert auf Seite 106).

[Mer+11] Henning Mersch, Daniel Behnen, Dominik Schmitz, Ulrich Epple, Christian Brecher und Matthias Jarke. “Gemeinsamkeiten und Unterschiede der Prozess- und Fertigungstechnik”. In: *at - Automatisierungstechnik* 59.1 (2011), S. 7–17 (Referenziert auf Seite 2).

[Mer16] Henning Mersch. *Deterministische, dynamische Systemstrukturen in der Automatisierungstechnik*. Bd. 1245. Fortschritt-Berichte VDI Reihe 8. VDI Verlag, 2016. (zugleich Dissertation. RWTH Aachen University. 2016) (Referenziert auf den Seiten 31, 42).

[Mey02] Dirk Meyer. *Objektverwaltungskonzept für die operative Prozessleittechnik*. Bd. 940. Fortschritt-Berichte VDI Reihe 8. VDI Verlag, 2002. (zugleich Dissertation. RWTH Aachen University. 2002) (Referenziert auf den Seiten 24, 32).

[New12] VDI News. *Ein Dank an die Aktiven - Erfolgreiche Richtlinienarbeit in der GMA*. [Online; Stand 30.03.2012]. 2012. URL: http://www.vdi.de/6930.0.html?tx_ttnews[tt_news]=55812 (Referenziert auf Seite 24).

[NN04] Joachim Nagelmann und Alexander Neugebauer. *iFBSpro*. [Online; Stand 29.05.2017]. 2004. URL: http://www.ltsoft.de/uploads/media/iFBSpro164.pdf (Referenziert auf den Seiten 32, 74).

[Pol94] Martin Polke. *Prozessleittechnik*. Oldenbourg Verlag, 1994 (Referenziert auf Seite 39).

[Pra71] Terrence W. Pratt. “Pair grammars, graph languages and string-to-graph translations”. In: *Journal of Computer and System Sciences* 5.6 (1971), S. 560–595 (Referenziert auf Seite 44).

[Pre13] Cideo Pressestelle. *Übernahme der CIDEON AG ist Meilenstein in der Unternehmensgeschichte*. [Online; Stand 14.06.2015]. 2013. URL: http://www.cideon.de/site/d/de/holding/presse-news/archiv/uebernahme-flg.php (Referenziert auf Seite 41).

[Qui11] Gustavo Quirós. *Model-based Decentralised Automatic Management of Product Flow Paths in Processing Plants*. Bd. 1183. Fortschritt-Berichte VDI Reihe 8. VDI Verlag, 2011. (zugleich Dissertation. RWTH Aachen University. 2011) (Referenziert auf den Seiten 4, 24, 25, 32, 42, 93, 95).

[RL07] OptXware Research und Development LLC. *The Viatra-I Model Transformation Framework Users Guide*. Techn. Ber. OptXware Research und Development LLC., 2007 (Referenziert auf Seite 54).

[Sch+15] Andreas Schüller, André Scholz, Rainer Drath, Thomas Tauchnitz und Thomas Scherwietes. “Speed-Standardisierung am Beispiel der PLT-Stelle : Datenaustausch mit dem Namur-Datencontainer”. In: *Atp-Edition* 57.01-02 (2015), S. 36–46 (Referenziert auf Seite 31).

[Sch10] Stefan Schmitz. *Grafik- und Interaktionsmodell für die Vereinheitlichung grafischer Benutzungsschnittstellen der Prozessleittechnik*. Bd. 1176. Fortschritt-Berichte VDI Reihe 8. VDI Verlag, 2010. (zugleich Dissertation. RWTH Aachen University. 2010) (Referenziert auf den Seiten 2, 24, 25, 32).

[Sch11] Georg Schnitger. *Skript zur Vorlesung „Formale Sprachen und Berechenbarkeit“*. [Online; Stand 14.06.2015]. 2011. URL: http://www.tks.cs.uni-frankfurt.de/data/teaching/ss12/th-inf-2/skript-schnitger-ss11.pdf (Referenziert auf den Seiten 18, 19).

[Sch12] Anne Schneller. “Parametrieren statt programmieren”. In: *VDI nachrichten* 6 (2012) (Referenziert auf den Seiten VII, VIII, 90).

[Sch14] Michael Schlereth. “Platform Independent Specification of Engineering Model Transformations”. Diss. TU Darmstadt, 2014 (Referenziert auf Seite 43).

[Sch95] Andy Schürr. “Specification of Graph Translators with Triple Graph Grammars”. In: *Graph-Theoretic Concepts in Computer Science 20th International Workshop. WG ’94, Herrsching, Germany, June 16 - 18, 1994. Proceedings*. 1995, S. 151–163 (Referenziert auf den Seiten 44, 51).

[SE06] Stefan Schmitz und Ulrich Epple. “On rule based automation of automation”. In: *Proceedings of the 5th MATHMOD : 5th Vienna Symposium on Mathematical Modelling. February 8-10, 2006. Vienna University of Technology, Austria*. 2006 (Referenziert auf den Seiten 42, 43).

[SE08] Stefan Schmitz und Ulrich Epple. “Automatisiertes Engineering leittechnischer Funktionen durch integrierte Regeln”. In: *Entwurf komplexer Automatisierungssysteme, EKA 2008: Beschreibungsmittel, Methoden, Werkzeuge und Anwendungen ; 10. Fachtagung, 15. bis 16. April 2008 Tutorium, 16. bis 17. April 2008 Fachtagung, Magdeburg, Denkfabrik im Wissenschaftshafen / Ifak, Institut für Automation und Kommunikation e.V., Magdeburg*. 2008, S. 241–252 (Referenziert auf den Seiten 42, 43).

[SE12] Andreas Schüller und Ulrich Epple. “PandIX - Exchanging P&I diagram model data”. In: *IEEE Xplore. Proceedings of 2012 IEEE 17th International Conference on Emerging Technologies & Factory Automation ETFA 2012. September 17-21, 2012. Krakow, Poland*. 2012 (Referenziert auf Seite 13).

[SE13] Andreas Schüller und Ulrich Epple. “Ein Modellserver zur Nutzung von R&I-Fließbild-Informationen”. In: *AUTOMATION 2013. 14. Branchentreff der Mess- und Automatisierungstechnik*. Bd. 2209. VDI-Berichte. VDI-Verlag, 2013, S. 223–226 (Referenziert auf den Seiten 25, 33).

[SK12] Michael Schlereth und Tina Krausser. “Platform-Independent Specification of Model Transformations at Runtime Using Higher-Order Transformations”. In: *Modellierung 2012. 14.-16. März 2012. Bamberg Proceedings*. 2012, S. 123–138 (Referenziert auf den Seiten 34, 54).

[SME05] Stefan Schmitz, Ansgar Münnemann und Ulrich Epple. “Komponentenmodell für den systematischen Entwurf von Prozessführungsfunktionen”. In: *GMA Kongress 2005. Automation als interdisziplinäre Herausforderung*. Bd. 1883. VDI-Berichte. VDI-Verlag, 2005, S. 817–824 (Referenziert auf Seite 2).

[SSE08] Stefan Schmitz, Markus Schlütter und Ulrich Epple. “R&I - Grundlage durchgängigen Engineerings”. In: *AUTOMATION 2008. Lösungen für die Zukunft*. Bd. 2032. VDI-Berichte. VDI-Verlag, 2008, S. 55–59 (Referenziert auf Seite 28).

[SSE09] Stefan Schmitz, Markus Schluetter und Ulrich Epple. “Automation of Automation - Definition, components and challenges”. In: *IEEE Xplore. Digital Library. Proceedings of the 2009 IEEE Conference on Emerging Technologies & Factory Automation. 22-25 Sept. 2009*. 2009, S. 1–7 (Referenziert auf Seite 41).

[ST10] Industry Automation Siemens AG Industry Sector und Drive Technologies. *How can I manage all automation software tasks in one engineering environment?* [Online; Stand 14.06.2015]. 2010. URL: https://w5.siemens.com/belux/web/nl/industrie/industrie/tia-portal/Documents/e20001-a340-p230-x-7600.pdf (Referenziert auf Seite 2).

[Sta73] Herbert Stachowiak. *Allgemeine Modelltheorie*. Springer-Verlag, 1973 (Referenziert auf den Seiten 8, 10, 11, 14, 15).

[TC3] Beckhoff Automation GmbH & Co. KG. *TwinCAT 3 | eXtended Automation (XA)*. [Online; Stand 30.05.2017]. URL: https://www.beckhoff.com/german.asp?twincat/twincat-3.htm (Referenziert auf Seite 88).

[The+14] Stefan Theurich, Martin Wollschlaeger, Andreas Schüller und Ulrich Epple. “Dienstbasierte Prüfung von CAEX-Exporten mit standardisierten Bibliotheken”. In: *AUTOMATION 2014. Smart X - Powered by Automation*. Bd. 2231. VDI-Berichte. VDI/VDE GMA, 2014, S. 157–168 (Referenziert auf Seite 29).

[UOS12] Leon Urbas, Michael Obst und Markus Stöss. “Formal Models for High Performance HMI Engineering”. In: Bd. 45. IFAC Proceedings Volumes 2. IFAC Online, 2012, S. 854–859 (Referenziert auf den Seiten 5, 41).

[VG13] Thomas Vogel und Holger Giese. *Model-Driven Engineering of Adaptation Engines for Self-Adaptive Software: Executable Runtime Megamodels*. Techn. Ber. 66. Hasso Plattner Institute for Software Systems Engineering, University of Potsdam, Germany, 2013 (Referenziert auf den Seiten 44, 53).

[Vis+98a] Shankar Viswanathan, Charlotta Johnsson, Rajagopalan Srinivasan, Venkat Venkatasubramanian und Karl Erik Ärzen. “Automating operating procedure synthesis for batch processes: Part I. Knowledge representation and planning framework”. In: *Computers & Chemical Engineering* 22.11 (1998), S. 1673–1685 (Referenziert auf Seite 41).

[Vis+98b] Shankar Viswanathan, Charlotta Johnsson, Rajagopalan Srinivasan, Venkat Venkatasubramanian und Karl Erik Ärzen. “Automating operating procedure synthesis for batch processes: Part II. Implementation and application”. In: *Computers & Chemical Engineering* 22.11 (1998), S. 1687–1698 (Referenziert auf Seite 41).

[Vog+09a] Thomas Vogel, Stefan Neumann, Stephan Hildebrandt, Holger Giese und Basil Becker. “Incremental Model Synchronization for Efficient Run-time Monitoring”. In: *Models in Software Engineering Workshops and Symposia at MODELS 2009, Denver, CO, USA, October 4-9, 2009, Reports and Revised Selected Papers*. 2009, S. 124–139 (Referenziert auf den Seiten 44, 53).

[Vog+09b] Thomas Vogel, Stefan Neumann, Stephan Hildebrandt, Holger Giese und Basil Becker. “Model-driven Architectural Monitoring and Adaptation for Autonomic Systems”. In: *Proceedings of the 6th IEEE International Conference on Autonomic Computing Barcelona, Spain June 15-19, 2009*. 2009, S. 67–68 (Referenziert auf den Seiten 44, 53).

[Wag08] Thomas Wagner. “Agentenunterstütztes Engineering von Automatisierungsanlagen”. Diss. Universität Stuttgart, 2008 (Referenziert auf Seite 42).

[WE15] Constantin Wagner und Ulrich Epple. “Sprechende Kommandos als Grundlage moderner Prozessführungsschnittstellen”. In: *AUTOMATION 2015. Benefits of Change - the Future of Automation*. Bd. 2258. VDI-Berichte. VDI/VDE GMA, 2015, S. 157–168 (Referenziert auf den Seiten 26, 33).

[Yu+12] Liyong Yu, Gustavo Quirós, Tina Krausser und Ulrich Epple. “ACPLT + IEC 61131-3 = Dynamic Reconfigurable Models@runtime”. In: *Softwaretechnik-Trends*. Bd. 32. 2. GI - Gesellschaft für Informatik, 2012, S. 90–91 (Referenziert auf den Seiten 24, 32).

Normen und Richtlinien

[AML1] AutomationML consortium. *AutomationML Whitepaper Part 1 - Architecture and general requirements*. 2014 (Referenziert auf den Seiten 25, 31, 40).

[AML2] AutomationML consortium. *AutomationML Whitepaper Part 2 - Role class libraries*. 2013 (Referenziert auf den Seiten 25, 31).

[AML3] AutomationML consortium. *AutomationML Whitepaper Part 3 - Geometry and kinematics*. 2013 (Referenziert auf den Seiten 25, 31).

[AML4] AutomationML consortium. *AutomationML Whitepaper Part 4 - Logic*. 2010 (Referenziert auf den Seiten 25, 31).

[DIN10628] Deutsches Institut für Normung. *DIN EN 10628: Fließschemata für verfahrenstechnische Anlagen - Allgemeine Regel*. 2000 (Referenziert auf den Seiten 5, 13, 25, 27).

[DIN19227] Deutsches Institut für Normung. *DIN 19227: Graphische Symbole und Kennbuchstaben für die Prozeßleittechnik*. 1993 (Referenziert auf Seite 25).

[DIN62424] Deutsches Institut für Normung. *DIN EN 62424: Festlegung für die Darstellung von Aufgaben der Prozessleittechnik in Fließbildern und für den Datenaustausch zwischen EDV-Werkzeugen zur Fließbilderstellung und CAE-Systemen*. 2009 (Referenziert auf den Seiten 28, 29).

[IEC61131] International Electrotechnical Commission. *IEC 61131: Programmable controllers, Part 3: Programming languages*. 2003 (Referenziert auf den Seiten 2, 26, 30, 56).

[IEC62424] International Electrotechnical Commission. *IEC PAS 62424: Specification for Representation of process control engineering requests in P&I Diagrams and for data exchange between P&ID tools and PCE-CAE tools*. 2008 (Referenziert auf den Seiten 2, 25, 40).

[MOF] Object Management Group. *OMG Meta Object Facility (MOF) Core Specification*. Version 2.4.1. 2011 (Referenziert auf den Seiten 9, 17).

[MOF-QVC] Object Management Group. *Meta Object Facility (MOF) Query/View/Transformation Specification*. Version 1.1. 2011 (Referenziert auf Seite 55).

[NA35] NAMUR. *NA35: Abwicklung von PLT-Projekten*. 2003 (Referenziert auf Seite 1).

[PandIX] Ulrich Epple, Markus Remmel und Oliver Drumm. *PandIX Modellbeschreibung*. Version 5.01. 2010 (Referenziert auf den Seiten 2, 5, 12, 13, 20, 25, 30, 40, 89).

[UML] Object Management Group. *OMG Unified Modeling LanguageTM (OMG UML), Infrastructure*. Version 2.4.1. 2011 (Referenziert auf den Seiten 9, 15, 17).

[UML2] Object Management Group. *OMG Unified Modeling Language (OMG UML), Superstructure*. 2011 (Referenziert auf Seite 17).

[VDI3681] Verein Deutscher Ingenieure. *VDI/VDE 3681: Einordnung und Bewertung von Beschreibungsmitteln aus der Automatisierungstechnik*. 2005 (Referenziert auf den Seiten 15, 24).

[VDI3682] VDI/VDE. *VDI-Richtlinie 3682: Formalisierte Prozessbeschreibung.* 2015 (Referenziert auf Seite 41).

Lebenslauf

Persönliche Daten

Name:	Tina Mersch (geb. Kraußer)
Geburtsdatum:	30. November 1980
Geburtsort:	Sömmerda

Ausbildungsdaten

Allgemeine Hochschulreife:	08/1995 – 06/1999 Albert-Schweizer Gymnasium, Erfurt
Hochschulausbildung:	10/1999 – 09/2001 Studium der Naturwissenschaftlichen Informatik Universität Bielefeld
	07/2000 – 04/2001 Studium der Informatik (Datorteknik) Technische Hochschule - KTH Stockholm
	10/2001 – 07/2005 Studium der Informatik Universität Bremen Abschluss: Diplom-Informatik

Wissenschaftliche Tätigkeit

Wissenschaftliche Mitarbeiterin:	06/2005 – 10/2006 Security Engineering Group RWTH Aachen University
Wissenschaftliche Mitarbeiterin:	11/2006 – 09/2012 Lehrstuhl für Prozessleittechnik RWTH Aachen University

Berufliche Tätigkeit

Produktentwicklung:	seit 11/2012 Beckhoff Automation GmbH & Co. KG, Verl

Verl den 8. Oktober 2018

Online-Buchshop für Ingenieure

VDI nachrichten **BUCHSHOP**

Online-Shops

Fachliteratur und mehr - jetzt bequem online recherchieren & bestellen unter: www.vdi-nachrichten.com/Der-Shop-im-Ueberblick

Täglich aktualisiert: Neuerscheinungen VDI-Schriftenreihen

Im Buchshop von vdi-nachrichten.com finden Ingenieure und Techniker ein speziell auf sie zugeschnittenes, umfassendes Literaturangebot.

Mit der komfortablen Schnellsuche werden Sie in den VDI-Schriftenreihen und im Verzeichnis lieferbarer Bücher unter 1.000.000 Titeln garantiert fündig.

Im Buchshop stehen für Sie bereit:

VDI-Berichte und die Reihe **Kunststofftechnik:**

Berichte nationaler und internationaler technischer Fachtagungen der VDI-Fachgliederungen

Fortschritt-Berichte VDI:

Dissertationen, Habilitationen und Forschungsberichte aus sämtlichen ingenieurwissenschaftlichen Fachrichtungen

Newsletter „Neuerscheinungen":

Kostenfreie Infos zu aktuellen Titeln der VDI-Schriftenreihen bequem per E-Mail

Autoren-Service:

Umfassende Betreuung bei der Veröffentlichung Ihrer Arbeit in der Reihe Fortschritt-Berichte VDI

Buch- und Medien-Service:

Beschaffung aller am Markt verfügbaren Zeitschriften, Zeitungen, Fortsetzungsreihen, Handbücher, Technische Regelwerke, elektronische Medien und vieles mehr – einzeln oder im Abo und mit weltweitem Lieferservice

Die Reihen der Fortschritt-Berichte VDI:

1 Konstruktionstechnik/Maschinenelemente
2 Fertigungstechnik
3 Verfahrenstechnik
4 Bauingenieurwesen
5 Grund- und Werkstoffe/Kunststoffe
6 Energietechnik
7 Strömungstechnik
8 Mess-, Steuerungs- und Regelungstechnik
9 Elektronik/Mikro- und Nanotechnik
10 Informatik/Kommunikation
11 Schwingungstechnik
12 Verkehrstechnik/Fahrzeugtechnik
13 Fördertechnik/Logistik
14 Landtechnik/Lebensmitteltechnik
15 Umwelttechnik
16 Technik und Wirtschaft
17 Biotechnik/Medizintechnik
18 Mechanik/Bruchmechanik
19 Wärmetechnik/Kältetechnik
20 Rechnerunterstützte Verfahren (CAD, CAM, CAE CAQ, CIM ...)
21 Elektrotechnik
22 Mensch-Maschine-Systeme
23 Technische Gebäudeausrüstung

ISBN 978-3-18-526108-1