Methoden zur adaptiven Benutzerinteraktion bei der semi-autonomen Aufgabenbearbeitung in Rehabilitationsszenarien

Marco Cyriacks

Universität Bremen 2012

Methoden zur adaptiven Benutzerinteraktion bei der semi-autonomen Aufgabenbearbeitung in Rehabilitationsszenarien

Vom Fachbereich für Physik und Elektrotechnik
der Universität Bremen

zur Erlangung des akademischen Grades

Doktor-Ingenieur (Dr.-Ing.)

genehmigte Dissertation

von

Dipl.-Ing. Marco Cyriacks
wohnhaft in München

Referent:	Prof. Dr.-Ing. Axel Gräser
Korreferent:	Prof. Dr.-Ing. Udo Frese
Eingereicht am:	7. März 2012
Tag des Promotionskolloquiums:	5. September 2012

Kurzfassung

Durch die zunehmende Leistungsfähigkeit moderner Computersysteme werden immer anspruchsvollere Funktionalitäten in Software- oder Mechatroniksystemen realisiert, eine Tendenz, die unmittelbar zur Vergrößerung der Systemkomplexität führt und die Bedienung durch die Benutzer erschwert. Aus diesem Grund rückt die Nutzbarkeit von technischen Systemen immer weiter in den Fokus der aktuellen Entwicklungen, insbesondere bei Systemen, die nicht ausschließlich über die Benutzerschnittstelle mit Benutzern kommunizieren, sondern darüber hinaus physikalisch mit ihnen interagieren. Systeme, die in diesen Einsatzbereich fallen, sind unter anderem Systeme zur gesellschaftlichen Wiedereingliederung von Menschen mit Behinderung, so genannte Rehabilitations- oder Unterstüzungsroboter.

Die vorliegende Arbeit widmet sich der Entwicklung von Methoden zur adaptiven Benutzerinteraktion im Kontext einer Softwarearchitektur für Rehabilitationsroboter. Das Ziel besteht in der Schaffung eines Rahmenwerks, das die Basis für die Anpassungsfähigkeit der Benutzerschnittstelle bildet.

Die vorgestellten Methoden zur Realisation der Adaptivität basieren auf einer durchgängigen Modularisierung der Benutzerschnittstelle durch Überführung der Funktionalitäten in Module, die zur Laufzeit und in Abhängigkeit von den zur Verfügung stehenden Systemressourcen aktiviert und deaktiviert werden können. Des Weiteren erfolgt die Entwicklung eines bidirektionalen Kommunikationskanals zwischen der Benutzerschnittstelle und den aktivierten Modulen sowie zwischen den Modulen untereinander. So können häufig wiederkehrende Funktionalitäten in Module ausgelagert und von anderen Modulen verwendet werden. Die Kommunikation wird auf Basis einer eigens dafür entwickelten Spezifikationssprache vollständig zur Laufzeit validiert um so ein robustes Laufzeitverhalten zu erreichen.

Durch die ausführliche Evaluation der dem Zielsystem zugrunde liegenden Softwarearchitektur können darüber hinaus offene Probleme aufgezeigt werden, die mit der Realisation von Adaptivität in der Benutzerschnittstelle in Konflikt stehen. Auf Basis einer weiteren Spezifikationssprache ist es möglich, Lösungen für die offenen Probleme zu finden, um das gesetzte Ziel zu erreichen. Als Basis der Entwicklungen wird eine Abstraktionsschicht zwischen der Benutzerschnittstelle und den übrigen Schichten der Architektur entworfen, wodurch die vollständige Entkopplung der Benutzerschnittstelle von systemspezifischen Komponenten erreicht werden kann.

Zum Nachweis der entwickelten Funktionalität und der damit verbundenen Möglichkeiten zur Adaptivität erfolgt die exemplarische Integration eines Moduls, das über einen Mustererkennungsalgorithmus in der Lage ist, zukünftige Benutzeraktionen abzuschätzen.

Abstract

The ever increasing performance of modern computing systems enables the realization of more challenging functionalities in software and mechatronic systems. This tendency results in an increase in system complexity and also makes the operation by users more difficult. Therefore, recent developments are focusing more strongly on the usability of technical systems, especially in case of systems that do not only communicate with users via a user interface but also interact with them physically. Systems that support social reintegration of persons with disabilities, so-called rehabilitation or support robots, fall into this area.

This thesis focuses on the development of methods for adaptive user interactions within a software architecture for rehabilitation robots. The objective is the development of a software framework that acts as a basis for the adaptability of the graphical user interface.

The methods presented to realize adaptivity are based on a user interface modularization by encapsulating all functionalities into modules. These modules can be activated or deactivated during run-time depending on the availability of resources. Furthermore, a bi-directional communication channel between the user interface and active modules as well as among modules is established. Thus it becomes possible to source common functionality out into modules and to have it reused by other modules. The communication is based on a specification language that has been developed to enable validation and to reach robust run-time behavior.

An extensive review of the software architecture used for the target system identified open problems that previously prevented the realization of adaptivity within the user interface. By using another specification language, finding solutions for those open problems becomes possible as well as achieving the set objective. The development is based on an abstraction layer between the user interface and the remaining layers of the software architecture. This realizes full decoupling of the user interface from system specific functionality.

To proof the concept for adaptivity within the user interface, the implementation of a module integrating an algorithm for pattern recognition is exemplarily shown with the aim to predict future actions of the user by evaluating previous actions.

Inhaltsverzeichnis

Glossar

a-Priori Der Begriff „a-Priori“ wird in der Informatik häufig in Verbindung mit Wissen verwendet. a-Priori-Wissen bezeichnet dabei Informationen, die unabhängig von Erfahrungen vorliegen und so als Ausgangspunkt für Algorithmen und Rückschlüsse dienen können. 15

ACE Das „**A**daptive **C**ommunication **E**nvironment“ ist ein Softwarerahmenwerk zur Implementation von plattformunabhängigen Netzwerkanwendungen. ACE bildet darüber hinaus die Grundlage für die freie CORBA Implementation TAO. 13

CORBA Die „**C**ommon **O**bject **R**equest **B**roker **A**rchitecture“ ist die Spezifikation eines Softwarerahmenwerkes das plattformübergreifende Protokolle und Dienste definiert. Es existieren Implementationen in vielen unterschiedlichen Programmiersprachen (zum Beispiel C++, PHP, Java) auf deren Basis sich interoperable Anwendungen entwickeln lassen. 13

GNU Octave GNU Octave ist eine freie Alternative zum proprietären MATLAB und ist mit dessen Skriptsprache in großem Umfang kompatibel. Es dient der Lösung mathematischer Probleme und wird häufig in wissenschaftlichen/universitären Einrichtungen eingesetzt (siehe [Wik10, Stichwort: „GNU Octave“]). 11

Metadaten Metadaten sind Informationen über andere Daten, also Informationen, die andere Daten näher spezifizieren. Bei Daten die in Tabellenform vorliegen können die Metadaten zum Beispiel die Anzahl und Datentypen der Spalten sowie die Anzahl der Datensätze beschreiben. 50

MVR Die MVR dient in der MASSiVE-Architektur der Berechnung von Abständen zwischen Umgebungsobjekten und wird bei der Kollisionsvermeidung innerhalb der Bewegungsplanung eingesetzt. Die MVR wird als Serverkomponente betrieben und kann in Verbindung mit einer grafischen Anwendung zusätzlich das aktuelle Modell der Systemumgebung zur Laufzeit darstellen. 189

Objektreferenz	Eine Objektreferenz (häufig auch als IOR beziehungsweise „**I**nteroperable **O**bject **R**eference“ bezeichnet) dient der Herstellung von Verbindungen zwischen CORBA-Objekten. Dazu enthält die Objektreferenz alle notwendigen Informationen, die für die Akquise eines Zielobjektes und das Öffnen eines Kommunikationskanals benötigt werden. 19
OSS	„**O**pen **S**ource **S**oftware“ oder auch FLOSS („Free/Libre/Open Source Software“) Anwendungen unterliegen einer speziellen Softwarelizenz, die die Offenlegung sämtlicher zur Erstellung benötigten Quellen vorschreibt. Anwendungen die einer OSS Lizenz unterliegen werden häufig von großen Gruppen verschiedener Programmierer entwickelt. 9
Petri-Netz	Über Petri-Netze ist es möglich einfache verteilte Systeme zu modellieren. Sie wurden nach [Wik10, Stichwort: „Petri-Netz“] in den 1960er Jahren entwickelt und werden heute neben der Informatik auch in vielen weiteren technischen Bereichen eingesetzt. 17, 29f.
Plug-In	Plug-Ins oder auch Erweiterungen dienen bei der Softwareentwicklung der Erweiterung von Anwendungen über eine vom Entwickler festgelegte Schnittstelle. Die Integration eines Plug-Ins kann zur Laufzeit erfolgen, da dieses in der Regel als dynamisch nachladbare Bibliothek vorliegt. 12, 52
Qt	Qt ist eine plattformunabhängige Software-Bibliothek zur Anwendungsentwicklung in der Programmiersprache C++. Die Bibliothek ist vollständig objektorientiert und wird unter einer freien Lizenz FLOSS verteilt. 39
Sequenzer	Der Sequenzer implementiert die Planungsebene der MASSiVE-Architektur. Er wird verwendet um Szenarien zu starten, Szenarien zu stoppen und Benutzereingriffe zu initiieren. Darüber hinaus liefert er die Stati aller aktuell ausgeführten Szenarien. 189
Szenarien	Szenarien oder auch Handlungsszenarien bezeichnen im Kontext der MASSiVE-Architektur Teilhandlungen die vom Robotersystem autonom ausgeführt werden können. 15, 18, 40
TAO	„**T**he **A**CE **O**RB“ ist eine freie CORBA Implementation, die auf ACE aufsetzt. TAO erbt dadurch die plattformunabhängigkeit von ACE und ist somit auf vielen unterschiedlichen Plattformen einsetzbar. 13

Weltmodell	Über das Weltmodell erfolgt der gesamte Datenaustausch zwischen den Skill-Server Komponenten der MASSiVE-Architektur. Intern verwaltet das Weltmodell sämtliche Daten in einer Tabelle in der jedes Datum über eine ID eindeutig referenziert wird. Die Darstellung der ID erfolgt als 5-Tupel und beinhaltet Metadaten über jedes einzelne Datum und wird bei der Kommunikation der Planungsebene mit der reaktiven Ebene und innerhalb der reaktiven Ebene verwendet. Weiterführende Informationen zu diesem Thema sind [Mar03] zu entnehmen. 189

Akronyme

ACE	The **A**DAPTIVE **C**ommunication **E**nvironment. 13, 34, 42, 51
ADL	**A**ctivities of **d**aily **L**iving. 23, 172
ALSA	**A**dvanced **L**inux **S**ound **A**rchitecture. 204, 219
API	**A**pplication **P**rogramming **I**nterface. 219
APK	**A**-**P**riori **K**nowledge. 142
ASR	**A**utomatic **S**peech **R**ecognition. 26, 204, 219
BCI	**B**rain **C**omputer **I**nterface. 23, 108, 204ff., 220
BSD	**B**erkeley **S**oftware **D**istribution. 11
CAN	**C**ontroller-**A**rea **N**etwork. 198, 203
CASE	**C**omputer **A**ided **S**oftware **E**ngineering. 49
CBR	**C**ase-**B**ased **R**easoning. 30f.
COP	**C**omposed **Op**erator. 16f.
CORBA	**C**ommon **O**bject **R**equest **B**roker **A**rchitecture. 13f., 34–37, 41ff., 46, 51, 81–85, 91, 102, 112, 122, 142, 192
CSV	**C**omma-**S**eparated **V**alues. 157
DDS	**D**evice-**D**etection-**S**erver. 47, 90f.
DNA	**D**eoxyribo**n**ucleic **a**cid. 26
DTD	**D**ocument **T**ype **D**efinition. 56
EEOP	**E**lementary **E**xecutable **Op**erator. 17f.
EMMA	**E**pisodes **M**ining using **M**emory **A**nchor. 28
EOG	**E**lectro**o**culo**g**ram. 23
ERM	**E**ntity **R**elationship **M**odell. 144
FRIEND	**F**unctional **R**obot arm with user-fr**IEN**dly interface for **D**isabled people. 2, 8, 13, 119, 169, 180, 192, 194, 196f., 214, 217
GUI	**G**raphical **U**ser **I**nterface. 19, 21ff.
IAT	**I**nstitut für **A**utomatisierungs**t**echnik. 2f., 8f.
IOR	**I**nteroperable **O**bject **R**eference. 42, 46
IP	**I**nternet **P**rotocol. 12, 200
KAIST	**K**orean **A**dvanced **I**nstitute of **S**cience and **T**echnology. 2
KARES	**KA**IST **R**ehabilitation **E**ngineering **S**ystem. 2
KDD	**K**nowledge **D**iscovery on **D**atabases. 3f., 25, 27f., 68f., 141ff., 159, 180
KI	**K**ünstliche **I**ntelligenz. 2–5, 22, 24, 179

LGPL	**L**esser **G**NU **P**ublic **L**icence. 12
MASSiVE	**M**ulti-Layer **A**rchitecture for **S**emi-Autonomous **S**ervice Robots w**i**th **V**erified Task **E**xecution. 3f., 9f., 13f., 17ff., 29f., 33f., 37, 40–44, 47, 49f., 53, 81–84, 86, 89f., 92f., 100f., 104, 111f., 115, 117ff., 121, 125, 138, 141ff., 168, 180, 188ff., 192f., 200, 209, 212, 216f., 221
MinEpi	**Min**imal occurence based **Epi**sode discovery. 28, 74, 79f.
MMS	**M**ensch-**M**aschine-**S**chnittstelle. 13ff., 18, 33–41, 43f., 46f., 50, 81f., 87, 90–93, 95–112, 115–127, 130–133, 135–138, 141–146, 148f., 167, 169f., 172ff., 177–180, 183f., 186, 188ff., 204–215, 219
MOC	**M**eta-**O**bject **C**ompiler. 53
MVC	**M**odel **V**iew **C**ontroller. 14, 34, 178f.
MVR	**M**odeled **V**irtual **R**eality. 191, 209
OpenRAVE	**Open** **R**obotics **A**utomation **V**irtual **E**nvironment. 9–13
ORB	**O**bject **R**equest **B**roker. 42, 83ff.
OSS	**O**pen-**S**ource-**S**oftware. 9f., 22, 28
PDF	**P**ortable **D**ocument **F**ormat. 65
PR2	**P**ersonal **R**obot **2**. 10
Prolog	**Pro**grammation en **Log**ique. 25
PS_A	**P**rocess **S**tructure - **A**bstract. 15, 17
PS_E	**P**rocess **S**tructure - **E**lementary. 17f.
PTH	**P**an-**T**ilt **H**ead. 194, 203
QSDB	**Q**uasi **S**tatic **D**ata**b**ase. 142f., 146, 157, 167
RBI	**R**eality-**B**ased **I**nteraction. 20, 117
ROS	**R**obot **O**perating **S**ystem. 9ff., 13
ROVIS	**RO**bust machine **VI**sion for **S**ervice robotics. 193, 216
RTK	**R**un-**T**ime **K**nowledge. 142f., 146, 157, 167
SC	**S**urface-**C**omputing. 20
SDB	**S**tatic **D**ata**b**ase. 142
SDK	**S**oftware **D**evelopment **K**it. 204, 219, 221
SDL	**S**erver **D**escription **L**anguage. 87
SQL	**S**tructured **Q**uery **L**anguage. 27, 64, 130, 157, 167
SSVEP	**S**teady **S**tate **V**isually **E**voked **P**otential. 204
STAIR	**ST**anford **AI** **R**obot. 10
SVG	**S**calable **V**ector **G**raphics. 99

SVRL	**S**chematron **V**alidation **R**eport **L**anguage. 66ff., 130
TAO	**T**he **A**CE **O**RB. 13, 34, 42, 51
TCP	**T**ransmission **C**ontrol **P**rotocol. 12, 200, 205f., 220
TDIDT	**T**op-**D**own **I**nduction of **D**ecision **T**rees. 30
TRE	**T**ask **r**epresenting **e**lements. 142
TTS	**T**ext **T**o **S**peech. 207
UDP	**U**ser **D**atagram **P**rotocol. 37, 206, 220
UML	**U**nified **M**odeling **L**anguage. 49f., 53, 131, 134
URI	**U**niform **R**esource **I**dentifier. 56
URL	**U**niform **R**esource **L**ocator. 223
W3C	**W**orld **W**ide **W**eb **C**onsortium. 54, 57f., 62f., 66, 124
WIMP	**W**indows, **I**cons, **M**enüs and **P**ointing device. 19f., 33, 116f.
WinEpi	**Win**dow based **Epi**sode discovery. 28, 74f., 79f., 141, 146, 149–152, 156, 158–162, 165, 168ff., 172, 174f., 180f., 223
WYSIWYG	**W**hat **Y**ou **S**ee **I**s **W**hat **Y**ou **G**et. 19
XHTML	**E**xtensible **H**yper**T**ext **M**arkup **L**anguage. 54
XML	E**x**tensible **M**arkup **L**anguage. 50, 54–59, 62–68, 83f., 87ff., 92–97, 100, 103, 120f., 123f., 126–131, 177ff., 184
XML-RPC	E**x**tensible **M**arkup **L**anguage **R**emote **P**rocedure **C**all. 11
XSD	**X**ML **S**chema **D**efinition. 57–61
XSL	**E**xtensible **S**tylesheet **L**anguage. 65
XSLT	**E**xtensible **S**tylesheet **L**anguage **T**ransformations. 65ff., 130
YALE	**Y**et **A**nother **L**earning **E**nvironment. 28

Zu viele Menschen machen sich nicht klar, dass wirkliche Kommunikation eine wechselseitige Sache ist.

Lee Iacocca - US-amerikanischer Manager und Publizist

1 Einleitung

Zu Beginn der vierziger Jahre des vergangenen Jahrhunderts entstanden die ersten Entwicklungen im Bereich der Computertechnik [Hei04]. Die damaligen Rechenmaschinen waren jedoch alles andere als interaktiv. Die Benutzerschnittstellen bestanden aus mechanischen Elementen und als Speichermedien kamen Lochkarten zum Einsatz. Zu dieser Zeit konnte das oben angegebene Zitat also nicht im Kontext von Computersystemen gefallen sein. Stand damals noch die Herausforderung der Entwicklung von rechnenden Maschinen im Vordergrund, so liegen heute die Schwerpunkte auf der Benutzbarkeit sowie der allgegenwärtigen Verfügbarkeit. Dabei wurde die echte wechselseitige Kommunikation erst mit steigender Rechenleistung der Computersysteme in den letzten zehn Jahren möglich. Heute ist der Computer mehr Mittel zum Zweck als das Ziel von Forschungsprojekten.

technologische Entwicklung

Etwas zeitversetzt zu diesen Entwicklungen kam im Jahr 1961 der erste industrielle Roboter im General Motors Werk in Ternstedt, New Jersey zum Einsatz [Eng89]. Ausgehend von diesem ersten Einsatz entstanden weitere Robotersysteme, die immer mehr an Leistung und später auch an Mobilität gewannen. So kamen Robotersysteme zum Einsatz, die auf einer fahrbaren Basis montiert Transportaufgaben innerhalb einer industriellen Produktionsumgebung ausführten. Solche Systeme sind genau wie stationäre Industrieroboter bis heute im Einsatz. Im Laufe der Zeit kamen viele weitere Einsatzgebiete von Robotern hinzu. Das sind zum Beispiel Robotersysteme zur Untersuchung von Rohrleitungssystemen oder Roboter zum Einsatz in der Raumfahrt. Auf Basis dieser Systeme entstand später eine weitere Art von Robotern, die als Teleroboter bezeichnet werden. Diese werden nicht durch Programme gesteuert, sondern mittels einer Fernbedienung durch Menschen. Letztere werden dabei in die Lage versetzt, die ausgeführten Bewegungen über ein Kamerasystem zu verfolgen. Die Roboter dienen den Menschen also als Hilfsmittel zur Erweiterung der eigenen motorischen Möglichkeiten.

Industrieroboter

Eine logische Konsequenz aus diesen Entwicklungen von Robotern zur Unterstützung von Menschen spiegelte sich Ende der achtziger Jahre in Form der Überlegungen zum ersten Service-Roboter „HelpMate“ wieder [Eng89]. Mit der Idee zu diesem System wurde das Ziel verfolgt, einen Roboter zu entwickeln, der die Angestellten in einer Klinik bei der Betreuung von Patienten unterstützt. Zehn Jahre später griffen weitere Forschungsgrup-

Unterstützungsroboter

pen diese Grundidee auf. Darunter auch das KAIST[1] mit dem KARES[2]-System [SLB99] und das IAT[3] der Universität Bremen mit dem FRIEND[4]-System [MRL+01]. Letzteres befindet sich heute in der dritten Entwicklungsgeneration und bildet als FRIEND III die Grundlage der Softwarearchitektur, in deren Kontext die vorliegende Arbeit entstand.

1.1 Betrachtete Forschungsgebiete

beherrschbare Systemkomplexität

Die Entwicklung von Robotersystemen zur Unterstützung von Menschen entstand aus der Kombination der einleitend beschriebenen Forschungsschwerpunkte im Bereich der Computertechnik und der Robotik. Inzwischen sind die Übergänge zwischen beiden Bereichen jedoch fließend, da die moderne Robotik ohne den Einsatz von Computern aufgrund der eingesetzten komplexen Steuerungs- und Regelalgorithmen nicht möglich wäre. Vielmehr sind umfangreiche Softwarearchitekturen notwendig, um die gesamte Systemkomplexität beherrschbar zu halten. Als Teil dieser Architekturen dienen die Benutzerschnittstellen der Kommunikation zwischen Mensch und Maschine, welche in den letzten Jahren vermehrt durch Methoden aus dem Bereich der KI[5]-Forschung ergänzt wird.

1.1.1 Softwarearchitekturen

Architekturen

Im Rahmen der vorliegenden Arbeit werden verschiedene Softwarearchitekturen betrachtet, um deren Eigenschaften zueinander in Beziehung zu setzen. Im Speziellen geht es dabei um Softwarearchitekturen, die der Steuerung von Service- oder Rehabilitationsrobotern dienen, mit dem Schwerpunkt auf frei verfügbaren Architekturen, die bereits eine nennenswerte Verbreitung erreicht haben.

1.1.2 Mensch-Maschine-Kommunikation

Kommunikation

Da nicht alle Architekturen eine Benutzerschnittstelle als Teilkomponente beinhalten, besteht in der Untersuchung von Benutzerschnittstellen für Service- oder Rehabilitationsroboter der zweite Schwerpunkt dieser Arbeit. Benutzerschnittstelle meint in diesem Kontext sowohl grafische Benutzerschnittstellen, die über einen Monitor dem Benutzer präsentiert werden, als auch entsprechende Eingabemedien, die zur Steuerung der Benutzerschnittstelle eingesetzt werden. In diesem Zusammenhang werden jedoch nur Benutzerschnittstellen für die Endbenutzer eines Systems thematisiert.

[1] **K**orean **A**dvanced **I**nstitute of **S**cience and **T**echnology
[2] **KA**IST **R**ehabilitation **E**ngineering **S**ystem
[3] **I**nstitut für **A**utomatisierungs**t**echnik
[4] **F**unctional **R**obot arm with user-fr**IEN**dly interface for **D**isabled people
[5] **K**ünstliche **I**ntelligenz

1.1.3 Erkenntnisgewinnung aus Datenbeständen

Neben den speziell auf die Robotik ausgerichteten Forschungsbereichen erfolgt die Betrachtung eines Teilgebietes der KI-Forschung: Erkenntnisgewinnung aus Datenbeständen (auch als KDD[6] bezeichnet). Dieser Forschungsbereich widmet sich der Untersuchung von Daten mit dem Ziel der Ableitung von Regelmäßigkeiten. Die in diesem Zusammenhang entwickelten Methoden lassen sich auf viele Anwendungsgebiete übertragen und auch für die Realisation von adaptiven Benutzerschnittstellen einsetzen.

KDD

1.2 Ziele der Arbeit

Die Ziele der vorliegenden Arbeit bestehen in der Entwicklung von Methoden zur adaptiven Benutzerinteraktion innerhalb einer Softwarearchitektur zur semi-autonomen Aufgabenbearbeitung in der Rehabilitationsrobotik. Dazu sind verschiedene offene Probleme zu adressieren, die die vollständige Adaptivität bisher erschweren oder gänzlich verhindern. Die der Arbeit zu Grunde liegende Softwarearchitektur dient der Steuerung von Rehabilitationsrobotern, mit Hilfe derer es Menschen mit Behinderung ermöglicht werden soll, einen Teil ihrer Autonomie zurück zu erlangen. Im Speziellen geht es daher um die Schaffung eines Rahmenwerkes, das die adaptive Interaktion zwischen einem Benutzer mit Behinderung und dem technischen System erlaubt.

Rahmenwerk

Die Basis der Untersuchungen bildet die Softwarearchitektur MASSiVE[7], die am IAT zur Steuerung von Rehabilitationsrobotern eingesetzt wird. Zur Realisation des Gesamtzieles ist es zunächst notwendig, die durch die Softwarearchitektur gegebenen Rahmenbedingungen auf folgende Punkte hin zu untersuchen:

- Welche Elemente der bestehenden Benutzerschnittstelle realisieren bereits einen adaptiven Ansatz?
- Gib es offene Probleme die mit den Zielen der vorliegenden Arbeit in Konflikt stehen?
- Welche Rahmenbedingungen müssen durch die Softwarearchitektur geboten werden, um die Adaptivität der Benutzerschnittstelle zu realisieren?
- Über welche Mechanismen wird der Zugriff auf die Systemkomponenten realisiert?

Auf Basis dieser Fragen wird im Verlauf der vorliegenden Arbeit ein Rahmenwerk entwickelt, welches die unabhängige Modellierung und Entwicklung von Elementen einer erweiterbaren Benutzerschnittstelle ermöglicht. Dazu erfolgt die Entwicklung einer Abstraktionsschicht über die die Unabhängigkeit der Benutzerschnittstelle von den übrigen

Flexibilität

[6] **K**nowledge **D**iscovery on **D**atabases
[7] **M**ulti-Layer **A**rchitecture for **S**emi-Autonomous **S**ervice Robots w**i**th **V**erified Task **E**xecution

Elementen der Softwarearchitektur erreicht werden kann. Die Benutzerschnittstelle wird so zu einer flexibel einsetzbaren Komponente der Architektur, über die im weiteren Verlauf der Entwicklungen nicht nur die Grundfunktionalitäten zur Steuerung des Gesamtsystems „Rehabilitationsroboter“ zur Verfügung gestellt werden, sondern darüber hinaus die Möglichkeit besteht, beliebige Elemente zu integrieren, die der Kommunikation und Interaktion mit dem Benutzer dienen.

Künstliche Intelligenz

Des Weiteren erfolgt die exemplarische Umsetzung von ausgewählten Methoden aus dem Bereich der KI. Konkret geht es hier um die Entwicklung einer Komponente, die mittels eines Verfahrens zur Mustererkennung in der Lage ist, auf Basis der Nutzungsdaten eine Vorhersage über das Verhalten des Benutzers zu treffen. So werden für zukünftige Entwicklungen die Möglichkeit der Nutzerprofilerstellung und weitere Anwendungen in Zusammenhang mit der Adaptivität ermöglicht.

1.3 Aufbau der Arbeit

Stand der Technik

Im **Kapitel 2** („Stand von Wissenschaft und Technik“, Seite 7) folgt die Beschreibung des aktuellen Stands der Technik und Forschung zu den in dieser Arbeit betrachteten Bereichen. Dabei werden der aktuelle Stand der Entwicklung von Benutzeroberflächen erläutert und mögliche Anwendungsfelder der KI-Forschung gegeben.

Problembeschreibung

Kapitel 3 („Problembeschreibung und Anforderungen“, Seite 33) beschreibt die Problemstellungen, die den Ausgangspunkt dieser Arbeit bilden. Das beinhaltet die detaillierte Beschreibung der Vorgänge in der MASSiVE-Architektur mit dem Ziel, die Problematiken herauszustellen, die mit der Umsetzung der verfolgten Ziele im Widerspruch stehen. Als Basis für die darauf folgenden Kapitel werden hier die zu beachtenden Anforderungen formuliert.

Theorie

Nach der Problembeschreibung folgt in **Kapitel 4** („Theoretische Grundlagen und Definitionen“, Seite 49) die Einführung in die benötigten theoretischen Grundlagen der in dieser Arbeit verwendeten Methodiken. Diese umfassen die eingesetzten Formalismen zur Modellierung der entwickelten Architekturanpassungen sowie ausgewählte Algorithmen aus dem Bereich des KDD.

Abstraktion

Der Hauptteil dieser Arbeit beginnt mit **Kapitel 5** („Abstraktion vom Gesamtsystem“, Seite 81). Es beschreibt die notwendigen Anpassungen an der MASSiVE-Softwarearchitektur unter Berücksichtigung der zuvor in Kapitel 3 formulierten Anforderungen. Eine Auflistung und Beschreibung von Systemressourcen in der in diesem Kapitel entwickelten Beschreibungssprache folgt im Anhang A.2.1.

dynamische Erweiterung

Aufbauend auf der angepassten Softwarearchitektur erfolgt in **Kapitel 6** („Dynamische Erweiterung der Benutzerschnittstelle“, Seite 115) die Entwicklung des Rahmenwerkes zur Realisation einer adaptiven Benutzerschnittstelle. Die Auflistung und Beschreibung

der in diesem Zusammenhang realisierten Elemente der Benutzerschnittstelle sind in Anhang A.2.3 zu finden.

Lernen

Die exemplarische Integration eines Algorithmus aus dem Bereich der KI schließt im **Kapitel 7** („Adaption durch KDD“, Seite 141) den Hauptteil der vorliegenden Arbeit ab.

Diskussion

Im letzten **Kapitel 8** („Zusammenfassung und Ausblick“, Seite 177) erfolgt eine zusammenfassende Diskussion der Ergebnisse sowie ein Ausblick auf mögliche oder notwendige Folgearbeiten.

Der Computer arbeitet deshalb so schnell, weil er nicht denkt.

Gabriel Laub - polnisch-deutscher Schriftsteller

2

Stand von Wissenschaft und Technik

Seit Jahrzehnten widmen sich viele Forschungsgruppen dem Gebiet der Servicerobotik. Innerhalb dieses Forschungsgebietes hat man sich das Ziel gesetzt, die modernen Entwicklungen in der Robotik und Software in intelligente Unterstützungssysteme zu überführen. Der Fokus liegt dabei auf einer möglichst intuitiven Bedienbarkeit, so dass die Systeme einer breiten Nutzergruppe zugänglich gemacht werden können.

Hilfe im Haushalt

Laut [Wik10, Stichwort: „Serviceroboter“] werden Serviceroboter derzeit überwiegend als Systeme für den Gebrauch im Haushalt definiert. Hier gibt es autonome Systeme wie zum Beispiel Staubsauger (siehe [Alf09] oder [Kle09]), Rasenmäher (siehe [Fri09], [EUR09] oder [BEL09]) oder Systeme zur Unterhaltung (siehe [Son09]). Sie besitzen nach [Wik10, Stichwort: „Serviceroboter“] die folgenden Merkmale:

- Verrichtung einer Dienstleistung
- Kein Einsatz in industrieller Produktion
- Kein speziell geschulter Bediener
- Oft gering oder unstrukturierte Umgebung

Rehabilitation

Ein in den letzten zehn Jahren besonders viel Aufmerksamkeit erhaltenes Teilgebiet der Servicerobotik ist die Rehabilitationsrobotik. Darin ist die Entwicklung von Unterstützungssysteme für Menschen mit Behinderung das Ziel. Diese Systeme dienen entweder der medizinischen Rehabilitation von Menschen oder als Hilfsmittel um durch Krankheit oder Unfall verlorene Autonomie zurück zu erlangen. Zu letzteren zählen zum Beispiel Systeme, die den Benutzer bei der Ausführung von alltäglichen Handlungen wie Essen, Trinken oder ähnlichen Aufgaben unterstützen (siehe [Top02], „Handy 1“-System). Alle Systeme dieser Art sind speziell für eine Aufgabe entwickelt worden und auch nur für diese einsetzbar. Sie besitzen in der Regel keine grafische Benutzerschnittstelle, sondern werden über Tasten oder Joysticks bedient.

Eine weitere Gruppe dieser Unterstützungssysteme sind Rollstühle die in Kombination mit einem Roboterarm eingesetzt werden. Die verwendeten Roboterarme bieten

Unterstützungssysteme

in der Regel sechs Freiheitsgrade und werden ebenfalls über Joysticks oder Tastaturen beziehungsweise Schalter bedient. In diese Kategorie fällt der in Abbildung 2.1a dargestellte iARM der Firma Exact Dynamics aus den Niederlanden [Exa09]. Dieser Roboterarm kann an einem beliebigen Rollstuhl montiert werden, trägt circa $1,5kg$ Nutzlast und hat eine Reichweite von $80cm$. Die Steuerung erfolgt über fünf unterschiedliche Menüs, die vom Benutzers durch Joystick oder Tastatur bedient werden. Da der Einsatz des Roboterarms unabhängig von einem bestimmten Rollstuhl ist, wurde er bereits in mehreren Forschungsprojekten eingesetzt. So bildete er auch die Basis für den ersten am IAT entwickelten Rehabilitationsroboter FRIEND.

Einen ähnlichen Aufbau besitzt der in Abbildung 2.1b dargestellte Roboterarm BRIDGET der Firma Focal Meditech, ebenfalls aus den Niederlanden. Sowohl BRIDGET als auch iArm besitzen sechs Freiheitsgrade und müssen direkt über den Benutzer gesteuert werden.

(a) *iARM [Exa09, Ass09]*

(b) *Personal Robot BRIDGET [FOC09]*

(c) *LWA3 [SCH11]*

Abbildung 2.1: *Häufig bei der Entwicklung von Unterstützungssystemen eingesetzte Roboterarme*

FRIEND

Im Rahmen der Entwicklung des Nachfolgemodells vom Rehabilitationsroboter FRIEND wurde am IAT erstmals ein Roboterarm mit sieben Freiheitsgraden eingesetzt. Es handelt sich dabei um den LWA3 Roboterarm der Firma Schunk mit Sitz in Deutschland,

der in Abbildung 2.1c dargestellt ist[1]. Im Gegensatz zu den Roboterarmen mit sechs Freiheitsgraden ergeben sich deutliche Verbesserungen bei den Möglichkeiten zur Manipulation von Gegenständen, insbesondere in unstrukturierten Umgebungen. Der durch diese größere Flexibilität entstehende Nachteil der komplexeren Ansteuerung wurde am IAT durch die Einführung einer intelligenten Bewegungsplanung überwunden (siehe [Ojd09]).

zunehmende Komplexität

Da die Komplexität der entwickelten Rehabilitationsroboter mit steigender Funktionalität in vielen Projekten zunahm, entstanden parallel zu deren Entwicklung Projekte in denen die Entwicklung von Softwarearchitekturen oder Softwarerahmenwerken zur Steuerung dieser komplexen Systeme im Mittelpunkt stand. Zwei dieser Projekte sowie die am IAT entwickelte Softwarearchitektur MASSiVE werden im folgenden vorgestellt.

2.1 Rahmenwerke und Architekturen

Bausteine für komplexe Systeme

Rahmenwerke enthalten im allgemeinen Bausteine für die Implementation von Hardware und sie können als Baukasten bei der Entwicklung von komplexen Systemen verstanden werden. Eine Architektur beschreibt im Gegensatz dazu den Aufbau eines Gesamtsystems und das Zusammenspiel seiner unterschiedlichen Komponenten mit einem gemeinsamen Ziel. Ähnliche Definitionen werden auch bei der Softwareentwicklung eingeführt: Softwarerahmenwerke stellen die Bausteine und Werkzeuge zur Implementation einer Softwarearchitektur zur Verfügung. Im Kontext der Servicerobotik besteht das gemeinsame Ziel aller Komponenten einer Softwarearchitektur überwiegend darin, ein komplexes Robotersystem, welches aus vielen heterogenen Sensoren und Aktuatoren unterschiedlicher Hersteller besteht, beherrschbar zu machen.

Komplexitätsreduktion

Nach [Wik10, Stichwort: „Architektur (Informatik)“] legen Softwarearchitekturen darüber hinaus fest, wie ihre unterschiedlichen Komponenten untereinander und mit externen Komponenten interagieren. Das Ziel des Einsatzes einer Architektur ist die Reduzierung der Systemkomplexität und die Steigerung der Möglichkeiten von Wiederverwendbarkeit, so dass Entwicklung, Wartung und Dokumentation vereinfacht werden. Bei Architekturen, die als OSS[2] (siehe Glossar zu OSS) veröffentlicht werden, stehen dabei häufig noch weitere Ziele im Vordergrund: die weite und schnelle Verbreitung sowie die Vereinfachung des Austauschs von Forschungsergebnissen auf einer gemeinsamen Basis. Diese Vorteile führten zu der Entwicklung von zahlreichen Architekturen für Robotersysteme, die meisten davon sind frei einsetzbar und erweiterbar (siehe [KS07]).

Vergleich von Forschungsergebnissen

In den folgenden Abschnitten werden drei Architekturen beschrieben, die nicht in [KS07] aufgeführt sind. Es handelt sich dabei um die Architekturen ROS[3] ([QGC+09]), Open-

[1] Der dargestellte Greifer kommt beim FRIEND-System nicht zum Einsatz.
[2] **O**pen-**S**ource-**S**oftware
[3] **R**obot **O**perating **S**ystem

RAVE[4] ([DK08]) sowie MASSiVE. Der Fokus liegt dabei auf der MASSiVE-Architektur, da sie die Grundlage für die Entwicklungen in der vorliegenden Arbeit darstellt.

2.1.1 ROS-Rahmenwerk

ROS ist kein Betriebssystem im eigentlichen Sinn, sondern ein Rahmenwerk welches Funktionalitäten zur Kommunikation der an der Steuerung eines Robotersystems beteiligten Prozesse und darüber hinaus Werkzeuge zur einfachen Entwicklung von eigenen Erweiterungen bietet. Das Gesamtsystem kann problemlos auf heterogenen Systemclustern betrieben werden da beim Entwurf konsequent ein Mikrokernel-Design (viele interagierende Einzelkomponenten) verfolgt wurde. Das ROS-Rahmenwerk wurde ursprünglich für die Robotersysteme STAIR[5] und PR2[6] aus den Bereichen Service-Robotik sowie Mobile-Manipulation entwickelt, lässt sich aber nach [QGC+09] sehr viel genereller einsetzen. Die wichtigsten Kerncharakteristika von Softwaresystemen, die mittels ROS entwickelt werden, sind:

Mikrokernel, Peer-to-Peer Rahmenwerk

- die Kommunikation als Peer-to-Peer Netz
- die werkzeugbasierte Entwicklung
- die Sprachunabhängigkeit
- die Verfügbarkeit als OSS

Mit ROS entwickelte Softwaresysteme ergeben in ihrer Implementation ein Peer-to-Peer Netzwerk. Die Notwendigkeit dazu wird von den Autoren damit begründet, dass bei Systemen, die auf einem zentralen Server basieren, Engpässe bei der Kommunikation entstehen, sobald das System über Rechnergrenzen hinaus wächst. Letzteres ist zum Beispiel der Fall wenn die Systemressourcen auf dem mobilen Roboter nicht mehr ausreichen, um komplexe Berechnungen durchzuführen. Die mit entsprechend vielen Ressourcen ausgestatteten externen Rechner müssen dann über langsame und unzuverlässige drahtlose Netzwerke angebunden werden. Diese Problematik entsteht nicht bei dem Aufbau als Peer-to-Peer System. Abbildung 2.2 zeigt eine mögliche ROS-Anwendung.

verteiltes System

Das ROS besteht in erster Linie aus Knoten (engl. „Nodes“). Diese Knoten sind Prozesse, die über Nachrichten (engl. „Messages“) zu einer bestimmten Thematik (engl. „Topics“) miteinander kommunizieren. Darüber hinaus existieren Dienste (engl. „Services“), die zur Realisierung von Broadcast-Mitteilungen eingeführt wurden. ROS realisiert konsequent das „Publish-Subscribe“-Modell (siehe [Wik10, Stichwort: „Alert-Dienst“]). Knoten registrieren sich untereinander um Nachrichten zu einer bestimmten Thematik zu erhalten. Es sind „one to one“, „one to many“ und „many to many“ Verbindungen möglich.

[4] **Open R**obotics **A**utomation **V**irtual **E**nvironment
[5] **ST**anford **AI R**obot
[6] **P**ersonal **R**obot **2**

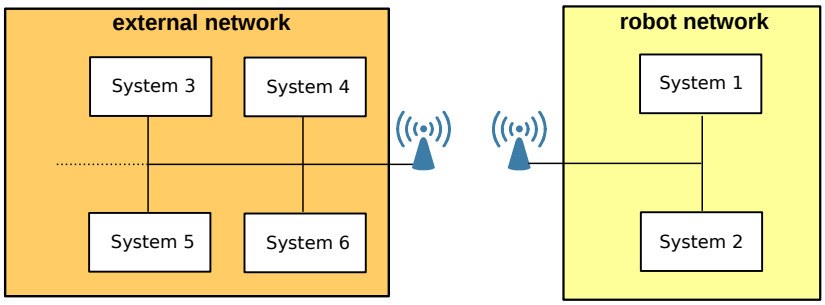


Abbildung 2.2: *Der Aufbau eines mittels ROS entwickelten Softwaresystems*

Durch die Mehrsprachigkeit können Entwickler die ROS-Knoten in ihrer bevorzugten Sprache implementieren. Nach [QGC+09] besteht die Möglichkeit des Einsatzes von C++, Python, GNU Octave und LISP, wobei bereits weitere Portierungen in der Entwicklung sind. Um in diesem Zusammenhang die Implementierung der Interprozesskommunikation zu erreichen, wird auf XML-RPC[7] zurück gegriffen. Dieser offene Standard bringt unter anderem den Vorteil mit, dass er bereits für viele Programmiersprachen zur Verfügung steht.

Mehrsprachigkeit

Das Rahmenwerk bringt viele Werkzeuge zur Entwicklung inklusive Quelltextgenerierung, Fehlersuche und zum Betrieb von ROS-Clustern mit, bietet aber keine grafische Benutzerschnittstelle für Endbenutzer. Die Veröffentlichung des gesamten Rahmenwerks erfolgt unter einer BSD[8]-Lizenz. Es kann daher sowohl für kommerzielle als auch für nicht kommerzielle Zwecke frei eingesetzt werden.

2.1.2 OpenRAVE-Architektur

Bei OpenRAVE handelt es sich um eine Softwarearchitektur zur Steuerung von mobilen Robotersystemen. Die Architektur bietet Möglichkeiten zur Entwicklung und zum Test von Algorithmen zur Manipulations- und Trajektorienplanung für mobile Roboter-Plattformen sowie humanoide Roboter. Die Alleinstellungsmerkmale der Architektur liegen nach [DK08] in folgenden Punkten:

Kontrollarchitektur für Roboter

- optimiertes Design für Echtzeitsteuerung und Überwachung
- integrierte Kernfunktionalitäten für kinematische Operationen
- Netzwerkschnittstelle zur Interpretation von Skriptsprachen wie Octave oder Matlab

[7] **Ex**tensible **M**arkup **L**anguage **R**emote **P**rocedure **C**all
[8] **B**erkeley **S**oftware **D**istribution

- integrierte Funktionalitäten zur Manipulations- und Greifplanung (erweiterbar über Plug-Ins)

- mitgelieferte Plug-Ins zum Testen von unterschiedlichen Planungsalgorithmen und der Einbindung von Steuerungsmodulen und Sensoren

Die Plug-Ins werden von OpenRAVE-Core Instanzen verwaltet, die ihrerseits die Schnittstellen für externe Skriptsprachen und grafische Anwendungen bieten. Die Kommunikation erfolgt über ein proprietäres TCP[9]/IP[10] Protokoll, welches in Zukunft durch die Möglichkeit der Kommunikation über gemeinsame Speicherbereiche ergänzt werden soll, um bei zeitkritischen Vorgängen die Kommunikationslatenz und die begrenzte Bandbreite einer Netzwerkverbindung zu vermeiden. Abgesehen von der Möglichkeit die OpenRAVE-Core Instanzen auf unterschiedlichen Systemen zu verteilen, verfolgt das Projekt das Design eines monolithischen Kernels, der zwar über Plug-Ins erweiterbar ist, aber diese alle im gleichen Prozesskontext ausführt. In Verbindung mit den externen Skript-Instanzen und grafischen Schnittstellen ergibt sich so eine Client-/Server-basierte Architektur, die in Abbildung 2.3 dargestellt ist.

monolithisch, Client/Server

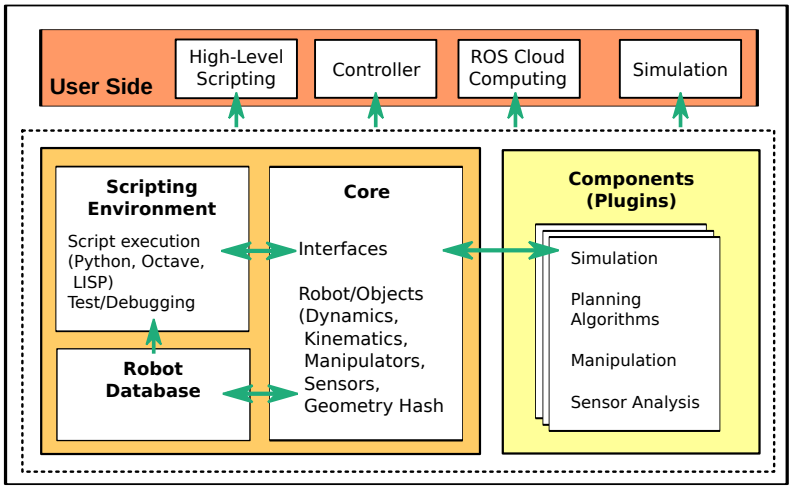


Abbildung 2.3: *Der Aufbau der OpenRAVE-Architektur*

Die in [DK08] und den Projektseiten im Internet beschriebenen grafischen Schnittstellen dienen allesamt dem Test, der Fehlersuche oder der Visualisierung von internen Vorgängen der Architektur. Auf grafische Schnittstellen, die für Endbenutzer geeignet sind, wird im Rahmen des Architekturdesigns verzichtet.

GUIs für Testzwecke

Die Architektur ist im Quelltext frei zugänglich um Forschern im Bereich der Robotik den Vergleich von Algorithmen zu erleichtern. Sie steht unter der LGPL[11]-Lizenz und kann daher auch für kommerzielle Projekte eingesetzt werden. Darüber hinaus ist die Architektur betriebssystemunabhängig.

[9] **T**ransmission **C**ontrol **P**rotocol
[10] **I**nternet **P**rotocol
[11] **L**esser **G**NU **P**ublic **L**icence

2.1.3 MASSiVE-Architektur

Die MASSiVE-Architektur wurde in [Mar03] entworfen und dann in den Folgejahren unter Einsatz der betriebssystemunabhängigen Middleware CORBA[12] (ACE[13]/TAO[14], siehe Glossar zu CORBA, ACE und TAO) für das Robotersystem FRIEND II implementiert (siehe [MPFG05, PFG05]). Als Rahmenwerk für die Entwicklung vieler Testanwendungen und der MMS[15] kam Qt zum Einsatz. Qt ist ein offenes, plattformunabhängiges Rahmenwerk für Anwendungen und grafische Benutzerschnittstellen.

Ähnlich wie bei Systemen, die mit dem ROS-Rahmenwerk realisiert wurden, erinnert auch die Struktur der MASSiVE-Architektur an ein Mikrokernel-Design, da hier viele Einzelkomponenten in der reaktiven Ebene miteinander interagieren. Gleichzeitig besitzt sie jedoch eine Client-/Server-Architektur wie OpenRAVE. Dabei bilden die Komponenten MMS, Sequenzer und Weltmodell das Rückgrat der Architektur und sind notwendig zum fehlerfreien Betrieb. Der Aufbau der MMS stellt verglichen mit den anderen Softwarearchitekturen in diesem Zusammenhang das Alleinstellungsmerkmal von MASSiVE dar: spezielle Benutzerinteraktionen sind integraler Bestandteil der Architektur (siehe [PMC+07]). Der Endanwender wird dabei in die Aufgabenbearbeitung miteinbezogen. Die so erreichte Teilautonomie wurde bereits in [KPI95] formuliert und in [Mar03] bei der Realisierung von MASSiVE als Kernelement aufgegriffen. Der Hintergrund dieser Teilautonomie liegt in der Feststellung begründet, dass derzeit keine Autonomie in unstrukturierten oder teil-strukturierten Umgebungen mit vertretbarem Aufwand erreicht werden kann. Durch den Benutzer wird diese Lücke geschlossen.

Mikrokernel, Client/Server

Wie Abbildung 2.4 zeigt, handelt es sich bei MASSiVE um eine Mehrschichtenarchitektur mit drei Schichten. Die obere Schicht ist die bereits angesprochene MMS. Ausgehend von Ihrer Grundstruktur erfolgen die im Rahmen der vorliegenden Arbeit durchgeführten Untersuchungen. Darunter liegt die Planungsschicht in Form des Sequenzers, der planenden Instanz der Gesamtarchitektur. Die auszuführenden Systemaufgaben werden von der MMS an den Sequenzer übergeben, dort geplant und über die reaktive Ebene, die dritte Schicht der Architektur, zur Ausführung gebracht. Zusätzlich zu den drei Schichten existiert das Weltmodell, welches sich über die Planungsschicht und die reaktive Ebene erstreckt. Das Weltmodell verwaltet das Wissen über die Systemumgebung in Form von symbolischen und subsymbolischen Daten. Die symbolischen Daten beschreiben die Umgebungsobjekte als abstrakte Objektklassen, welche vom Sequenzer während der Aufgabenplanung verwendet werden. Die Hauptaufgabe des Weltmodells besteht darin, den symbolischen Daten in der Planungsebene die subsymbolischen Daten zuzuordnen. Letztere werden für die Ausführung in der reaktiven Ebene benötigt. Im Gegensatz zu den symbolischen Daten umfassen die subsymbolischen Daten konkrete Informationen über die Umgebungsobjekte denen sie zugeordnet sind. Das sind unter anderem die Abmessungen, die aktuell gültigen Positionsdaten und weitere Eigenschaften. Erzeugt werden diese Daten zu Beginn der Ausführung über Monitoring-Operationen, die den aktuellen Sys-

Sequenzer und Weltmodell

Initial monitoring

[12] **C**ommon **O**bject **R**equest **B**roker **A**rchitecture
[13] The **A**DAPTIVE **C**ommunication **E**nvironment
[14] **T**he **A**CE **O**RB
[15] **M**ensch-**M**aschine-**S**chnittstelle

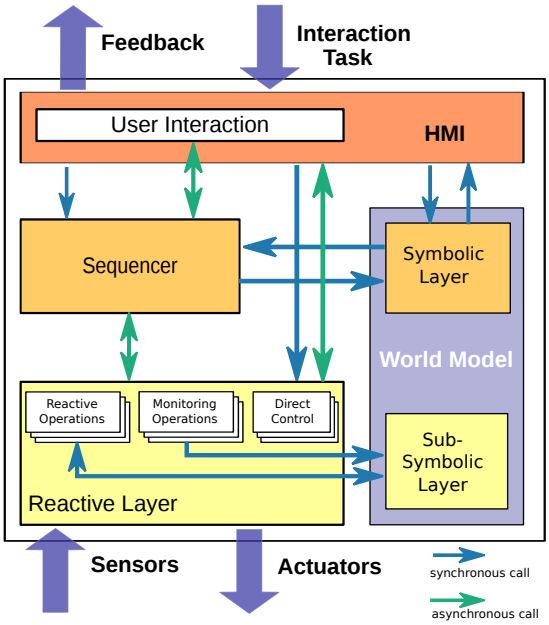


Abbildung 2.4: *Die drei Schichten der MASSiVE-Architektur („HMI“, „Sequencer“ und „Reactive Layer“)*

temzustand ermitteln (siehe [Pre03]). Dieser Vorgang wird auch als Objektverankerung bezeichnet.

In den folgenden Abschnitten erfolgt die Beschreibung der Abläufe während der Handlungsplanung und Ausführung, sowie die Integration der Benutzerinteraktionen. Für detailliertere Informationen zu diesen Vorgängen sei auf [Mar03] verwiesen.

Mensch-Maschine-Schnittstelle

Model-View-Controller

Die MMS der MASSiVE-Architektur besteht aus unterschiedlichen Komponenten, die das MVC[16]-Entwurfsmuster implementieren (siehe [GHJV95]). Dabei repräsentiert „HMI Model“ das Modell sowie den Controller und die Komponenten „GUIs“, „User Interaction“ sowie „DoF Mapping“ die Ansichten (weitere Informationen zu den Komponenten sind in [Cyr05] zu finden). Die Kommunikation der Komponenten untereinander erfolgt über CORBA. Abbildung 2.5 zeigt den detaillierten Aufbau der MMS.

Bei der Entwicklung in [Cyr05] wurde die MMS so ausgelegt, dass sie mehrere grafische Oberflächen unterstützt und die Einbindung von beliebig vielen Eingabegeräten erlaubt. Die Freiheitsgrade der Eingabegeräte werden dabei über die „DoF Mapping“ Komponente auf Systemkommandos abgebildet.

[16] **M**odel **V**iew **C**ontroller

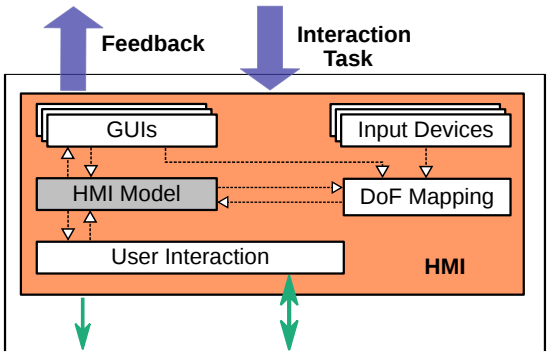


Abbildung 2.5: *Detaillierter Aufbau der MMS*

Beim Start der MMS muss auf die richtige Startreihenfolge der beteiligen Ansichten geachtet werden. Erfolgt der Start des „HMI Model“ nicht vor dem Start der Ansichten, so sind diese nicht funktionsfähig. Des Weiteren muss der Start der „DoF Mapping“ Komponente vor dem Start der Komponenten für die Eingabegeräte erfolgen. Damit trotz dieser gegenseitigen Abhängigkeiten ein robuster Start der MMS gewährleistet werden kann, erfolgt der Start der einzelnen Komponenten indirekt über eine weitere Komponente, die als „HMIStart“ bezeichnet wird und darüber hinaus keine weitere Funktionalität besitzt.

gegenseitige Abhängigkeiten

Nach dem Start einer „GUI“ ermittelt diese über das „HMI Model“ alle zur Anzeige benötigten Daten. Diese umfassen auch die Daten, die den zur Verfügung stehenden Szenarien entsprechen. Diese werden grafisch angezeigt und so dem Benutzer präsentiert. Wenn dieser eines der Szenarien durch Auswahl zur Ausführung bringt, dann setzt die Ansicht die „HMI Model“ Komponente in den Ausführungszustand und leitet die Anfrage an den Sequenzer weiter.

Handlungsplanung und Ausführung

Die Handlungsplanung im Sequenzer erfolgt in zwei Abstraktionsebenen. Ausgangspunkt ist zunächst die Aufforderung zur Ausführung eines Handlungsszenarios, welche durch den Benutzer initiiert und von der MMS an den Sequenzer delegiert wird. Der Sequenzer lädt daraufhin das zuvor von Systemprogrammierern in Form von abstrakten Ablaufstrukturen (PS_A[17]s) festgelegte a-Priori Wissen. Die Modellierung dieser Netze, welche die oberste Abstraktionsebene darstellen, kann dabei grafisch erfolgen, wie Abbildung 2.6 anhand eines Beispiels zeigt.

Ablaufstrukturen

Das Beispiel beschreibt das Handlungsszenario für den Vorgang des Herausnehmens eines Objektes aus einem Container. Dabei existieren vier Umgebungsobjekte: ein Roboter ($Ro.1$), eine Flasche ($Bo.1$), ein Kühlschrank ($Fr.1$) und ein Tablett ($Tr.1$). Die einzelnen Piktogramme beschreiben unterschiedliche Konstellationen dieser Umgebungsobjekte.

[17] **P**rocess **S**tructure - **A**bstract

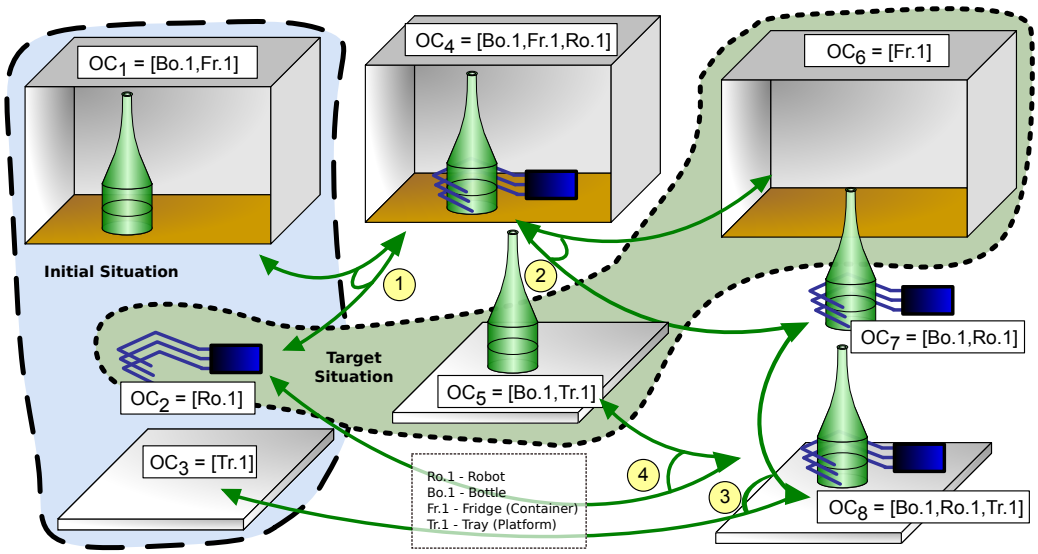


Abbildung 2.6: *Szenarienbeschreibung in Form einer abstrakten Ablaufstruktur (PS_A) zum Greifen einer Flasche*

Objektkonstellationen

Die Objektkonstellationen sind durchnummeriert und mit *OCn* bezeichnet, wobei *n* für die Nummer steht. Zwischen einigen Objektkonstellationen besteht eine Verbindung über bidirektional gerichtete Kanten. Diese Verbindungen beschreiben eine reversible Operation um unterschiedliche Objektkonstellationen ineinander zu überführen. Diese Operationen werden als COP[18]s bezeichnet und lauten für das oben angegebene Beispiel wie folgt:

1. **GraspObjectInContainer**
   Der erste ausgeführte COP ermittelt die Position und Lage des zu greifenden Objektes im Container. Wenn der Greifer geschlossen sein sollte wird er geöffnet. Im Anschluss daran wird der Greifer zum Objekt in Greifposition gebracht. Sollte dieser Schritt fehlschlagen, so wird die Kontrolle an den Benutzer übergeben. Wenn kein Fehler auftritt und der Benutzer die Ausführung nicht abbricht wird der Greifer am Ende der Ausführung geschlossen und das Objekt ist gegriffen.

2. **DepartFromContainer**
   Im zweiten COP wird der Greifer mit dem gegriffenen Objekt aus dem Container heraus bewegt und an eine freie Position im Raum überführt. Die freie Position wird, wenn sie nicht über die Systemdatenbank fest vorgegeben ist, zuvor über das Kamerasystem ermittelt.

3. **PlaceOnPlatform**
   Der dritte COP ermittelt über das Kamerasystem eine freie Position auf der Plattform und bewegt den Greifer mit dem gegriffenen Objekt über diese Position. Im Anschluss daran wird der Greifer senkrecht nach unten bewegt bis eine Kontak-

[18] **C**omposed **Op**erator

tkraft über den Kraft-Momenten-Sensor registriert wird. In diesem Moment wird die Bewegung unterbrochen.

4. **Depart**
   Der letzte COP öffnet den Greifer und bewegt ihn an eine freie Position im Raum. Die Position wird entweder fest vorgegeben oder zuvor über das Kamerasystem ermittelt.

Bevor mit der Planung begonnen wird, erfolgt die Transformation der PS_A in ein Petri-Netz nach [Mar03]. Die Initialmarkierung des Petri-Netzes wird im Rahmen der Umgebungserfassung nach [Pre03] gesetzt. Das Resultat der Planung auf dieser Abstraktionsebene ist eine Sequenz von COPs, die das System in den Zielzustand überführt.

Petri-Netze

Der Übergang in die zweite Abstraktionsebene erfolgt durch Substitution der COPs mit Funktionsblocknetzwerken (PS_E[19]s). Auch diese basieren auf Petri-Netzen. Sie legen den konkreten Ablauf der Ausführung fest indem sie die Ablaufreihenfolge der reaktiven Operationen in Abhängigkeit von ihren Ausführungsergebnissen bestimmen. Die reaktiven Operationen werden als EEOP[20]s bezeichnet. Sie stellen die elementaren Operationen des Systems, die so-genannten System-Skills dar. Abbildung 2.7 zeigt das Beispielnetz, durch welches der COP `GraspObjectInContainer` substituiert wird.

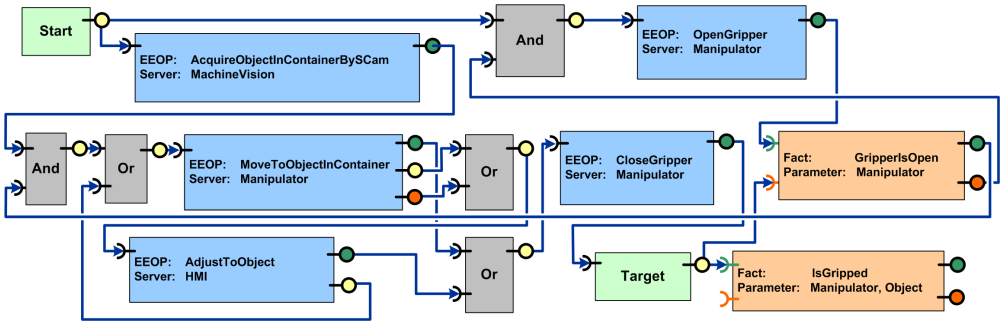


Abbildung 2.7: *Funktionsblocknetzwerk (PS_E) zum Greifen von Objekten in Containern mit integrierter Benutzerinteraktion aus [Pre09]*

Die Ausführungsergebnisse der einzelnen Skills (also der EEOPs) werden durch die Ausgänge der Funktionsblöcke repräsentiert. Durch logische Verknüpfung mit boolsch'en Operatoren oder direkter Verknüpfung mit anderen Blöcken erfolgt die Festlegung der Ausführungsreihenfolge.

logische Verknüpfungen

Die Ausführung der EEOPs erfolgt über die reaktive Ebene der MASSiVE-Architektur. Einzige Ausnahme bilden hier die Benutzerinteraktionen, welche, wie Eingangs erwähnt, ein integraler Bestandteil der Aufgabenplanung und Ausführung in MASSiVE sind. Damit sie möglichst transparent in die Handlungsplanung eingebettet werden können, ex-

[19] **P**rocess **S**tructure - **E**lementary
[20] **E**lementary **E**xecutable **Op**erator

Benutzerinteraktionen

istiert ein Skill-Server, der alle möglichen Benutzerinteraktionen implementiert. Dadurch ist es möglich, Benutzerinteraktionen wie alle anderen Skills der reaktiven Ebene aufzurufen. Aus Sicht des Sequenzers ergibt sich kein Unterschied, wie in Abbildung 2.7 unten links zu erkennen ist: die Benutzerinteraktion **AdjustToObject** ist wie jeder andere EEOP auch in das PS_E-Netz eingebettet. Lediglich Anhand des festgelegten Servers „HMI“ ist zu erkennen das die Operation durch den Benutzer ausgeführt wird.

Der zuvor angesprochene Skill-Server wird von der Komponente „User Interaction“ repräsentiert. Sobald eine Benutzerinteraktion während der Ausführung durch den Sequenzer initiiert wird, setzt die Komponente das „HMI Model“ in den Benutzerinteraktionsmodus und präsentiert dem Benutzer einen Dialog. Der Aufbau des Dialoges ist dabei von der Art der Benutzerinteraktion abhängig und wird vom Systemprogrammierer bei der Entwicklung festgelegt. Detailiertere Beispiele und mögliche Dialoge sind [PMC+07] zu entnehmen.

Hardware- und Ressourcen-Verwaltung

Wie im vorherigen Abschnitt ausführlich beschrieben wurde, erfolgt die Ausführung der Szenarien in Form von System-Skills. Diese elementaren Funktionalitäten des Systems werden innerhalb der MASSiVE-Architektur von den Skill-Servern der reaktiven Ebene und dem Skill-Server in der MMS zur Verfügung gestellt. Damit die Architektur unabhängig von den anzusteuernden Hardwarekomponenten ist, wurde innerhalb der reaktiven Ebene eine weitere Abstraktionsschicht eingeführt. Die von den Skill-Servern implementierten Skills haben keinen direkten Zugriff auf die Hardwarekomponenten, sondern greifen über die Hardware-Server auf diese zu. Dabei erfolgt eine feste Zuordnung von Skill-Servern zu Hardware-Servern. Diese Zuordnung sorgt dafür, dass ein Hardware-Server immer genau von einem Skill-Server verwaltet wird. Dabei bilden Hardware-Server und Skill-Server funktionale Gruppen, wie Abbildung 2.8 zeigt.

die reaktive Ebene

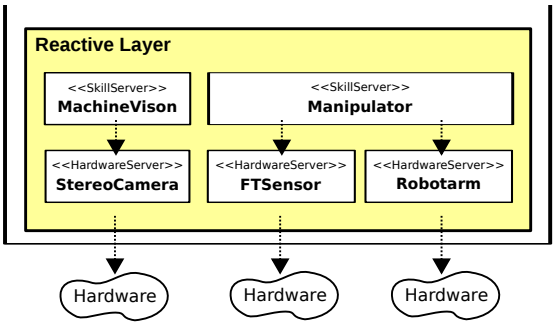


Abbildung 2.8: *Der Aufbau der reaktiven Ebene mit Skill- und Hardware-Servern*

Sollte später eine der Hardwarekomponenten ausgetauscht werden müssen, so muss lediglich der entsprechende Hardwareserver angepasst werden. Dabei wird die Schnittstelle zur Skill-Ebene nicht geändert.

Verwaltung der CORBA-Verbindungen

Da die gesamte Architektur aus einem Netzwerk von interagierenden Einzelkomponenten besteht, ist es notwendig während der Initialisierungsphase dafür zu sorgen, dass die Verbindungen der Komponenten untereinander hergestellt werden können. Um dabei keine unnötigen Abhängigkeiten zu erzeugen, werden diese Verbindungen nicht bereits beim Start der Komponenten, sondern erst später zur Laufzeit durchgeführt. Dazu kommt in der MASSiVE-Architektur ein Verzeichnisdienst zum Einsatz. Dieser Dienst verwaltet über ein Register die Objektreferenz zu jeder registrierten Komponente. Der Aufbau des Registers ähnelt dabei einem Telefonbuch: die Einträge werden über eindeutige Namen referenziert. Im Kontext von MASSiVE werden diese Namen auch als Ressourcennamen und die Komponenten als Ressourcen bezeichnet.

Verzeichnisdienst

2.2 Konzepte und Formen von Interaktion

Dieser Abschnitt soll eine Übersicht über die heute für Benutzerschnittstellen eingesetzten Interaktionskonzepte geben. Dabei werden bewusst nur Benutzerschnittstellen betrachtet die in Form einer GUI[21] realisiert sind. Nach [Dah06] sind heute die indirekte Manipulation, die direkte Manipulation, das WIMP[22] Konzept und das WYSIWYG[23] Konzept weit verbreitet.

Der Terminus *indirekte Manipulation* beschreibt die Interaktion, bei der die Kommandoeingabe durch den Nutzer indirekt zu einer Manipulation von Objekten der Benutzeroberfläche führt. Das bedeutet, dass die Aktion, die der Benutzer ausführt um die Eingabe vorzunehmen, nichts mit der auszuführenden Aktion selbst zu tun hat. Dieses Interaktionskonzept liegt in der Regel bei Tastatureingaben vor. Ein einfaches Beispiel ist hier das Paar aus Tastaturkommando `STRG + A` und der Aktion „alles auswählen“. Ersteres besitzt keinen logischen Zusammenhang zu letzterem. Diese Art der Interaktion wurde in vielen Bereichen von der direkten Interaktion abgelöst.

indirekte Manipulation

Aus der Beschreibung der indirekten Interaktion kann einfach die direkte Interaktion abgeleitet werden. Bei diesem Interaktionskonzept besteht ein direkter Zusammenhang zwischen der Aktion des Benutzers mit seinem Eingabegerät und der auf dem Computer durchgeführten Aktion. Diese Form der Interaktion liegt in der Regel bei der Bedienung mittels eines Zeigegerätes vor. So werden zum Beispiel mittels Maus Objekte wie Daten - in der Regel durch Piktogramme dargestellt - auf der Oberfläche

direkte Manipulation

[21] **G**raphical **U**ser **I**nterface
[22] **W**indows, **I**cons, **M**enüs and **P**ointing device
[23] **W**hat **Y**ou **S**ee **I**s **W**hat **Y**ou **G**et

zwischen unterschiedlichen Fenstern verschoben. Diese Aktion des Verschiebens auf der Oberfläche korrespondiert dabei mit der Aktion des Verschiebens von Daten auf dem Datenträger. Ein weiteres Beispiel wäre das Verschieben einer Datei in den Papierkorb als Korrespondenz für das Löschen derselben. In [JGH+08] beschreibt der Autor die aktuellen Tendenzen bei der Benutzerinteraktion als Versuch die Diskrepanz zwischen mit Eingabegeräten erzeugten Aktionen und auszuführenden Befehlen immer kleiner zu machen. Dabei sieht er als Zukunft der Benutzerschnittstellen das RBI[24]-Konzept, worin die Interaktion mit einem System über schnelle und hochauflösende berührungsempfindliche Displays erfolgt. Die Elemente der Benutzeroberflächen werden dabei auf Basis von physikalischen Gesetzmäßigkeiten manipuliert, so dass diese direkt auf den Nutzer reagieren. Erste Realisierungen dieser neuartigen Form von Interaktion wurden bereits auf dem iPhone™ und iPad™ der Fa. Apple ([App10]), dem Open Source Betriebssystem für Smartphones Maemo ([Mae10]) und in aktuellen Forschungsprojekten im Bereich der SC[25]-Steuerungen ([PCH10, SKS+10]) realisiert.

RBI-Konzept

Eine weitere Form der Benutzerinteraktion wird im WIMP-Konzept beschrieben, welches als Hauptmerkmale einer grafischen Benutzerschnittstelle Fenster, Piktogramme, Menüs und Zeigegerät beschreibt. Dieses Konzept ist auch heute noch das am weitesten verbreitete Konzept für grafische Benutzeroberflächen. Als direkten Nachfolger und Kern der aktuellen Forschungsanstrengungen sieht [JGH+08] eine Verschmelzung des WIMP-Konzeptes mit dem aktuellen RBI-Trend. Die Benutzeroberflächen werden immer mehr ein Abbild der realen Umgebung des Benutzer sein und sich direkt manipulieren lassen. Die Interaktion wird dabei häufig durch taktile Bildschirme erreicht, die die parallele Erkennung von mehreren Berührungspunkten (engl. Multi-Touch) erlauben. Diese Art der Eingabe scheint dabei besonders für ältere Menschen geeignet zu sein, wie [PCH10] zeigt.

WIMP

Nicht nur in der Forschung, sondern auch bei aktuellen Produktivsystemen erfolgt derzeit eine Umgestaltung und Refokussierung auf die Benutzbarkeit und die damit verbundenen Anforderungen an Benutzerschnittstellen. In der Einleitung zu einer Artikelserie beschreibt [Kle10] die Wichtigkeit, dass Benutzer auch ohne intensives Studieren von Handbüchern in der Lage sind ein Computersystem zu nutzen. In seinem Artikel der gleichen Serie beschreibt [MB10] die Notwendigkeit, die Ergonomie bei der Entwicklung in den Vordergrund zu stellen, um nicht bei der Integration und Realisierung von grafischen Effekten verloren zu gehen. Den von [JGH+08] beschriebenen Wandel weg vom ursprünglichen WIMP-Konzept sieht auch [MB10], konzentriert sich dabei jedoch auf neue Geräteklassen wie Netbooks und Smartphones, die aufgrund ihrer geringeren Bildschirmauflösung über speziell daran angepasste Betriebssysteme betrieben werden, welche im Vollbildmodus laufen und auf jegliche Fenster verzichten.

Ergonomie als Schlüsselkonzept

Zusammenfassend lässt sich sagen, dass derzeit ein Wandel bei der Gestaltung von Benutzerschnittstellen im Gange ist. Viele Entwickler konzentrieren sich auf neue Bereiche und versuchen ihre Benutzerschnittstellen schöner, intuitiver oder modularer zu gestalten.

[24] **R**eality-**B**ased **I**nteraction
[25] **S**urface-**C**omputing

2.3 Benutzerschnittstellen

Zu Beginn der Entwicklung von Computern galt noch eine Synonymbeziehung zwischen den Begriffen „benutzen“ und „programmieren“. So waren die Entwickler eines Computersystems auch gleichzeitig die Benutzer, was darin begründet lag, dass die Geräte von Experten für Experten entwickelt wurden. Den heute als solchen bezeichneten Endbenutzer gab es zu der Zeit noch nicht. Dieser Zusammenhang hat sich in den letzten zwei Jahrzehnten verändert und gilt generell nur noch im Bereich der Forschung und Entwicklung, da es hier häufig um die Entwicklung von Prototypen geht und die Bewältigung der Systemkomplexität im Vordergrund steht. In diesem Fall werden GUIs überwiegend als Testanwendungen für Entwickler verstanden. Lediglich bei Projekten mit dem konkreten Ziel der Erforschung von Ergonomieaspekten bei grafischen Benutzerschnittstellen ist das anders, wobei diese Art der Forschung noch ein relativ junges Gebiet ist.

Ergonomieaspekte

Anfang der neunziger Jahre sprach die „Gesellschaft für Informatik e.V.“ eine Empfehlung aus, um spezielle Studiengänge über Softwareergonomie in der Lehre zu etablieren (siehe [Hei04]). Nicht zuletzt durch diese Entwicklungen bestehen heute Standards wie [ISO06c], die sicherstellen, dass ein Mindestmaß an Ergonomie bei der Oberflächengestaltung eingehalten wird. In Bezug auf Benutzerschnittstellen für Menschen mit Behinderung sind in diesem Zusammenhang noch weitere Anforderungen an die Ergonomie einer Benutzerschnittstelle gestellt. Die Oberflächen müssen nicht nur intuitiv, sondern darüber hinaus auch mit speziellen Eingabegeräten bedienbar sein. Das kann konkret den Einsatz eines Monitors mit taktiler Oberfläche und großen Schaltflächen oder ganz anderer auf Personen oder spezifische Behinderungen abgestimmte Eingabemedien bedeuten. Einen Überblick über derzeit für den Einsatz durch Endbenutzer erhältliche Eingabe- beziehungsweise Ausgabemedien geben [Dah06, Hei04]. Beschrieben werden dabei nicht nur Geräte die speziell für Benutzer mit eingeschränkten kognitiven oder motorischen Fähigkeiten entwickelt wurden, sondern auch Standardgeräte.

2.3.1 Aktuelle Forschungsgebiete

[JGH+07] beschreibt in seinem Artikel unterschiedliche Forschungsgebiete im Bereich der Mensch-Computer Interaktion. Die meisten Projekte haben dabei das gemeinsame Ziel kontextbewusste Schnittstellen (engl. „context-aware“) zu entwickeln, die in der Lage sind sich an den Benutzer, seine geistige und körperliche Verfassung sowie seine Umgebung anzupassen [VPA+00]. Die dadurch erreichte Adaptivität wird in vielen Fällen durch einen multimodalen Systemansatz unterstützt, welcher den Einsatz von vielen unterschiedlichen Eingabe- und Ausgabemedien vorsieht. Die dadurch entstehenden Vorteile werden insbesondere bei Systemen, die mit vielen unterschiedlichen Benutzern in Verbindung stehen, genutzt [BWK+03].

Adaptivität und Multimodalität

Ein weiteres großes Forschungsgebiet ist das Nutzer-Monitoring. Dieses kommt insbesondere im Bereich der Rehabilitationsrobotik und somit überwiegend in Zusammenhang mit älteren Menschen oder Menschen mit körperlicher oder geistiger Behinderung zum

Nutzer-Monitoring

Einsatz. Dabei sollen Systeme den Gesundheitszustand des Benutzers beziehungsweise Patienten überwachen und in Abhängigkeit von Sensorwerten zum Beispiel das Auslösen von Notfallaktionen erlauben. Auch die Protokollierung des Gesundheitszustandes könnte behandelnden Ärzten wertvolle Informationen liefern, um die Behandlung anzupassen [RSI98]. Des Weiteren gibt es auch im Zusammenhang mit dem Nutzer-Monitoring Untersuchungen dazu, die eingangs beschriebene Erfassung der gefühlsmäßigen Verfassung des Benutzers in diesem eher medizinischen Bereich zu nutzen [Hol03, Hud03].

KI

Ein Forschungsgebiet, das mit beiden zuvor beschriebenen Bestrebungen teils eng verknüpft ist, ist der Versuch Methoden aus dem Bereich der KI bei der Entwicklung von GUIs einzusetzen. Die Durchführung von Aktionen durch die GUI ohne die direkte Kommandierung durch den Benutzer ist dabei eines der Ziele. Ein solches System soll in diesem Zusammenhang die Absicht des Benutzers erkennen und ihm entweder ungefragt zuvor kommen oder die Aktionen mit der größten Ausführungswahrscheinlichkeit im aktuellen Kontext dem Benutzer gegenüber besonders hervorheben und einfach auswählbar präsentieren. Um diese Funktionalität zu realisieren, erfolgt in der Regel eine Auswertung des Nutzerverhaltens und die Modellierung desselben, basierend auf den ermittelten Daten [ACC06].

MeeGo

Insbesondere in den letzten Jahren gewann die Untersuchung von Benutzerschnittstellen für tragbare Computer wie Smartphones, Netbooks oder Tablett-PCs vermehrt an Bedeutung. Die besondere Herausforderung in diesem Bereich besteht in der Gestaltung einer ansprechenden grafischen Oberfläche, die auch für kleine bis sehr kleine Displays geeignet ist. Im Fall der Smartphones und Tablett-PCs hat sich dabei der Einsatz von taktilen Bildschirmen durchgesetzt, was dem in [JGH+08, MB10] bereits beschriebenen Wandel bestätigt (siehe Abschnitt 2.2). Ein Ansatz, der das Ziel verfolgt für alle Geräte dieser Klasse zum Einsatz zu kommen, besteht im MeeGo-Projekt welches federführend von den Firmen Intel und Nokia ins Leben gerufen wurde. Die Entwicklung des Projektes erfolgt als OSS (siehe [Sch10]). Die zum Einsatz kommende Benutzerschnittstelle ist modular aufgebaut und verwendet eine Reiterstruktur zur Anzeige von unterschiedlichen Anwendungen. Die Reiter befinden sich am oberen Bildschirmrand und werden automatisch ausgeblendet wenn sie nicht benötigt werden, so dass die Anwendungen dann im Vollbildmodus ausgeführt werden. Die Funktionalität der Reiter ist vom Benutzer nahezu frei konfigurierbar.

2.3.2 Benutzerschnittstellen von Service-Robotern

Multimodale GUI

Bei der Entwicklung von speziellen Benutzerschnittstellen für Service-Roboter ist nach [RBSM10] ein Kriterium besonders wichtig: Es besteht die Notwendigkeit zur Integration von unterschiedlichen Eingabemedien. Die Autoren verfolgen diese Anforderung bei der Entwicklung einer grafischen, multimodalen Benutzerschnittstelle für ein Rollstuhl-System. Die beschriebene Benutzerschnittstelle ist in der Lage die Eingaben über vier unterschiedliche Medien zu verarbeiten: Sprache, Gesichtsausdruckserkennung, taktile Oberfläche und Gestikerkennung. Diese Medien sind dabei in Form von Software-Agenten in separaten Modulen implementiert, so dass die GUI in diesem Zusammenhang erweit-

erbar bleibt. Die Kommunikation dieser Agenten mit der GUI sowie die Kommunikation der GUI mit der Steuerung des Rollstuhls erfolgt durch Kommandos über eine Netzwerkverbindung. Die GUI ist in unterschiedliche Bereiche aufgeteilt, und sie bietet Informationen zu den möglichen Aktionen, über die verbundenen Eingabemedien, die vom Nutzer erzeugten Eingabesequenzen und einige Daten zur Nutzungsanalyse. Insgesamt hinterlässt die GUI den Eindruck, dass diese derzeit hauptsächlich für Entwickler und weniger für Endbenutzer gedacht ist.

Die in [LHB10] vorgestellte GUI legt den Fokus auf mögliche Endbenutzer. Sie wurde für ein elektrisches Rollstuhl-System entwickelt, welches den Benutzer beim Fahren unterstützen soll. Das Kernkonzept besteht dabei in der Steuerung über deiktische Ausdrücke (siehe [Wik10, Stichwort: „Deixis“]). Dabei werden situationsabhängig bestimmte Funktionalitäten zur Verfügung gestellt und in einer per Joystick gesteuerten grafischen Oberfläche inklusive eines Kamerabildes präsentiert. Das System bietet einen manuellen Steuerungsmodus, der der Steuerung des Systems durch den Joystick entspricht, und zwei autonome Modi: das Fahren entlang einer Wand und das Passieren von engen Passagen. Die Umschaltung zwischen dem manuellen Modus und den autonomen Modi, die weitere Interaktion auf der grafischen Oberfläche erfordern, erfolgt über einen Taster am Joystick. In den autonomen Modi kann der Benutzer über den Joystick den Mauscursor der grafischen Oberfläche bewegen, um so die nötigen Eingaben zu tätigen.

Joysticksteuerung

Ein anderes Ziel verfolgen die Entwickler der GUI für die EOG[26]-basierte Steuerung eines Roboter-Systems in [UG10]. Der Fokus der Forschung liegt hierbei in der Realisierung der Steuerung durch die Augenbewegung des Benutzers. Die entwickelte GUI erlaubt neben der Eingabe von Texten über eine in BCI[27]-Systemen häufig zum Einsatz kommende Bildschirmtastatur noch die Ausführung von verschiedenen ADL[28]-Aktionen. Letztere werden durch Piktogramme präsentiert, sind jedoch nicht mit realen Funktionalitäten belegt, sondern dienen lediglich als Testbeispiele. Darüber hinaus erfolgt im oberen Teil der GUI die Anzeige der aktuellen Messdaten des EOG-Systems, welche hauptsächlich für Entwickler von Interesse sein dürften. Die Steuerung erfolgt direkt über Augenbewegungen, die als Kommandos interpretiert werden. Die Bewegung der Augen in eine der vier Richtungen links, rechts, oben oder unten wird binär ausgewertet und bewirkt eine Änderung der selektierten Schaltfläche in die entsprechende Richtung. Doppeltes blinzeln wählt die aktuell selektierte Schaltfläche aus. Abgesehen von den angezeigten EOG-Daten besitzt diese GUI bereits an Endbenutzer angepasste Elemente.

Piktogramme

Selektion durch Cursorbewegung

Im Gegensatz zu den bereits beschriebenen GUIs erfolgt die Steuerung des in [MDBM10] vorgestellten Rollstuhl-Systems über einen Monitor mit taktiler Oberfläche. Dargestellt wird dort ein vereinfachtes 3D-Abbild der Umgebung, sowie vier Schaltflächen zum Drehen nach links, zum Drehen nach rechts, zum Stoppen und zum Fahren. Bevor die Fahrt gestartet werden kann, muss zunächst auf dem Monitor die Zielposition ausgewählt werden. Dazu ist das 3D-Abbild der Umgebung ausgehend vom Rollstuhlzentrum in

taktile Oberfläche

[26] **E**lectro**o**culo**g**ram
[27] **B**rain **C**omputer **I**nterface
[28] **A**ctivities of **d**aily **L**iving

radialsymmetrische Sektoren aufgeteilt, welche direkt vom Nutzer ausgewählt werden können, um die Zielposition vorzugeben.

2.4 Maschinelles Lernen

Im Gebiet der KI-Forschung ist es das Ziel technischen Systemen Funktionalitäten zu ermöglichen, die bisher nur von Menschen erfolgreich durchgeführt werden können. Konkret heißt das, einem System das menschliche Handeln und Denken zu ermöglichen. Nach [Wik10, Stichwort: „KI“] wird dabei zwischen der starken und schwachen künstlichen Intelligenz unterschieden, wobei erstere bis heute als nicht erreicht und visionär gilt. Im Bereich der schwachen künstlichen Intelligenz gibt es dagegen bereits erste Erfolge.

Nach [Dil06] beinhaltet KI die Grundfertigkeiten der Verarbeitung von natürlicher Sprache, der Wissensrepräsentation, des automatischen Schlussfolgerns und des Lernens. Diese Fähigkeiten sind die von Alan Touring bereits 1950 vorgeschlagenen Inhalte des „Touring-Tests“. Nach Touring sollte ein technisches System diese Grundfertigkeiten beherrschen um künstliche Intelligenz auf menschenähnlichem Niveau zu belegen. Als Erweiterung seines Tests, dem „totalen Touring-Test“, fügte er die Fertigkeiten Computersehen und Robotik hinzu. Letzteres verdeutlicht warum viele der Forschungsprojekte rund um die KI sich mit der Entwicklung von Roboter-Systemen oder humanioden Roboter-Systemen beschäftigen: diese Systeme decken alle Forschungsbereiche des „Touring-Tests“ ab.

Touring-Test

Heute bietet die KI-Forschung Raum für die unterschiedlichsten Wissenschaftsformen von der Mathematik, der Informatik über die Neurobiologie, die Psychologie bis hin zur Philosophie. Auf der Seite der Informatik, die in jedem Fall benötigt wird, um entwickelte Algorithmen auf den Zielsystemen zu implementieren, gilt es dabei in dem meisten Fällen die folgenden Bereiche zu bearbeiten, welche man auch direkt aus den Inhalten des „Touring-Tests“ ableiten kann.

- **Datenbasis**
  Die Grundvoraussetzung ist eine Datenbasis. Für Menschen besteht diese Datenbasis in der Aufnahme von Informationen über audiovisuelle Eindrücke. Für ein technisches System kommen dabei in erster Linie bereits in digitaler Form vorliegende Informationen zum Einsatz, wie sie von Sensoren, Sprach- oder Bildverarbeitenden Methoden erzeugt werden. Bevor diese Informationen in die Datenbasis übergehen, werden sie noch gefiltert um redundante oder unwichtige Informationen zu verwerfen.

Informationsverarbeitung

- **Wissensschöpfung und Repräsentaion**
  Die aufgenommenen Eindrücke in der Datenbasis müssen verarbeitet und in Form von Erinnerungen oder Wissen abgelegt werden. Dazu erfolgt eine Auswertung der Informationen der Datenbasis mit Bezug auf ein konkretes Problem. Das Ergebnis dieses Vorgangs ist strukturiertes Wissen, das in einer konsistenten Form abgelegt wird. Dieser Vorgang des „Schürfens“ von Wissen wird in der Literatur auch als

KDD und im deutschsprachigen Raum häufig als „Erkenntnisgewinnung aus Datenbeständen“ oder „Data-Mining“ bezeichnet [Wik10, Stichwort: „Data-Mining“]. Bei technischen Systemen erfolgt dabei in der Regel der Versuch der Wiedererkennung von Mustern auf Basis von statistisch-mathematischen Methoden. Data-Mining

- **Inferenz - logisches Schließen**
  Inferenz oder logische Rückschlüsse erfolgen auf der Basis von Fakten oder statistischen Annahmen, welche über Regeln miteinander in Beziehung gesetzt werden. In Abhängigkeit von den Regeln ergeben sich Konsequenzen und Schlussfolgerungen, also logische Schlüsse. Die Fakten oder die statistischen Annahmen entstammen dabei dem Systemwissen, wobei die Form der Wissensrepräsentation ausschlaggebend für die Möglichkeiten zur Inferenz ist. In der Regel muss daher die Wissensrepräsentation auf die Methodik bei der Inferenz angepasst werden. In der Literatur kommt in diesem Zusammenhang häufig Prolog[29] zum Einsatz. Das Wissen wird dabei in Form von Fakten und die Bedingungen in Form von Implikationen nach einer standardisierten Prolog-Syntax deklarativ beschrieben. Über einen Prolog-Interpreter können dann Anfragen gestellt werden, die über logische Rückschlüsse auf Basis des zuvor definierten Wissens beantwortet werden. Prolog

- **Lernen**
  Nach [BKI08] ist der Vorgang des Lernens die Änderung der Wissensbasis eines Systems aufgrund von Erfahrungen. Die Änderung besteht darin, das bestehende Wissen des Systems in Abhängigkeit von den Eingangswerten so zu verbessern, dass Aufgaben künftig erfolgreicher bearbeitet werden können. Die Bewertung der Lernfähigkeit muss hier in Abhängigkeit von der Art der Aufnahme der Eingangswerte, also den Erfahrungen gesehen werden. Der Übergang von einem nicht lernfähigen System zu einem stark lernfähigem System ist dabei fließend. Sind die Erfahrungen des Systems direkt vom Programmierer vorgegeben, so benötigt das System keine eigene Intelligenz zur Aufnahme dieses Wissens und besitzt auch keine Lernfähigkeit. Erfolgt die Aufnahme der Erfahrungen jedoch allein durch das System, so muss es eine hohe Lernfähigkeit besitzen, um darüber die Wissensbasis zu optimieren. Der konkrete Optimierungsvorgang besteht in jedem Fall in der Anpassung der Wissensbasis aufgrund von selbstständig vom System durchgeführten Rückschlüssen, wodurch das Lernen auch als adaptive Form der Inferenz verstanden werden kann. Adaptive Inferenz

Diese vier Bereiche sollen im folgenden in Verbindung mit Anwendungen aus der Literatur nochmals genauer untersucht und beschrieben werden. Darauf aufbauend werden in den folgenden Kapiteln die Lösungsansätze formuliert.

[29] **Pro**grammation en **Log**ique

2.4.1 Datenbasis

Die zum Einsatz kommende Datenbasis ist in jedem Fall direkt von der Anwendungsdomäne des zu untersuchenden Prozesses abhängig. Der Ursprung vieler Methoden liegt dabei im Bereich der Analyse von Konsumdaten in (Online-)Märkten. Hier kann die Analyse der Einkaufgewohnheiten in genereller als auch in personenbezogener Form erfolgen. In [AS95] versuchen die Autoren herauszufinden, welche Artikel von bestimmten Kunden häufig und in welcher Konstellation gekauft werden. Dazu werten sie Datenbanken aus, die alle Transaktionsdaten im betrachteten Zeitraum mit Relation zu den Kundendaten enthalten.

Transaktionsdaten

Ein ähnliches Ziel verfolgen die Autoren in [KLRMS02]. Sie analysieren als Datenbasis die DNA[30] Zusammensetzung von unterschiedlichen Getreidearten, um gemeinsame Eigenschaften zu identifizieren. Die Ergebnisse werden dazu verwendet, um im Bereich der Genetik die Forschung weiter voranzutreiben.

DNA Daten

Weitere Forschungsbestrebungen laufen im Umfeld der Untersuchung von Multimediadaten und in [ZHL+98] werden dazu unterschiedliche Ansätze und Ziele beschrieben. Im Bereich der Untersuchung von Audiodaten ist das Ziel die Sprachverarbeitung, die über die Erkennung von Mustern eine Zuordnung zu gesprochenem Text ermöglichen und so ein ASR[31]-System realisiert. [ZHL+98] widmet sich im Gegensatz dazu der Verarbeitung von Videodaten. Das Ziel besteht in der Untersuchung von Bild- und Videodaten zur Charakterisierung, Gegenüberstellung und Klassifikation.

Multimediadaten

Ein weiterer, besonders in den letzten Jahren vermehrt in den Fokus der Forschung getretener, Bereich der Forschung liegt in der Untersuchung von sequentiellen Daten. Bei diesen Daten handelt es sich in der Regel um Datensequenzen, die aus mit Zeitstempel versehenen Ereignissen bestehen. Die Quelle der Daten kann dabei eine komplexe relationale Datenbank oder auch eine einfache textuelle Ereignisliste sein. In [MM02] untersuchen die Autoren Algorithmen zur Verarbeitung von kontinuierlichen Ereignissequenzen mit dem Ziel der Erkennung von häufig vorkommenden Ereignissen. Ihre Implementationen testen sie mit zwei Arten von Datenbasen: Transaktionsdaten und Textdokumente. Einige Jahre später veröffentlichten die Autoren in [LXM08] Versuche der Verarbeitung von Meldungslisten des Linux-Kernels und die Autoren in [CCC09] die Untersuchung von Alarmmeldungen in Telekommunikationssystemen.

Ereignisdaten

Viele der in diesem Zusammenhang entwickelten Algorithmen sind bereichsübergreifend einsetzbar und basieren auf Ansätzen, deren Entwicklung bereits lange zurück liegt.

[30] **D**eoxyribo**n**ucleic **a**cid
[31] **A**utomatic **S**peech **R**ecognition

2.4.2 Wissensschöpfung und Repräsentation

Die Auswertung der gewählten Datenbasis im Rahmen des KDD Prozesses erfolgt über Algorithmen mittels derer versucht wird, Muster in den Datenbeständen zu identifizieren und zu klassifizieren. Dabei wird zum Beispiel nach Mustern gesucht, die eine zuvor festgelegte Häufigkeit erreichen. Einige dieser Algorithmen und deren Zusammenhänge untereinander stellt Abbildung 2.9 dar. Das gewonnene Wissen ist in geeigneter Form abzulegen, so dass aus der Datenbasis eine Wissensbasis, welche zum Beispiel in Form von Regeln, Entscheidungsbäumen oder Bayes'schen Netzen (siehe [Wik10, Stichwort: „Bayes'sche Netze"]) repräsentiert werden kann, wird.

Mustererkennung

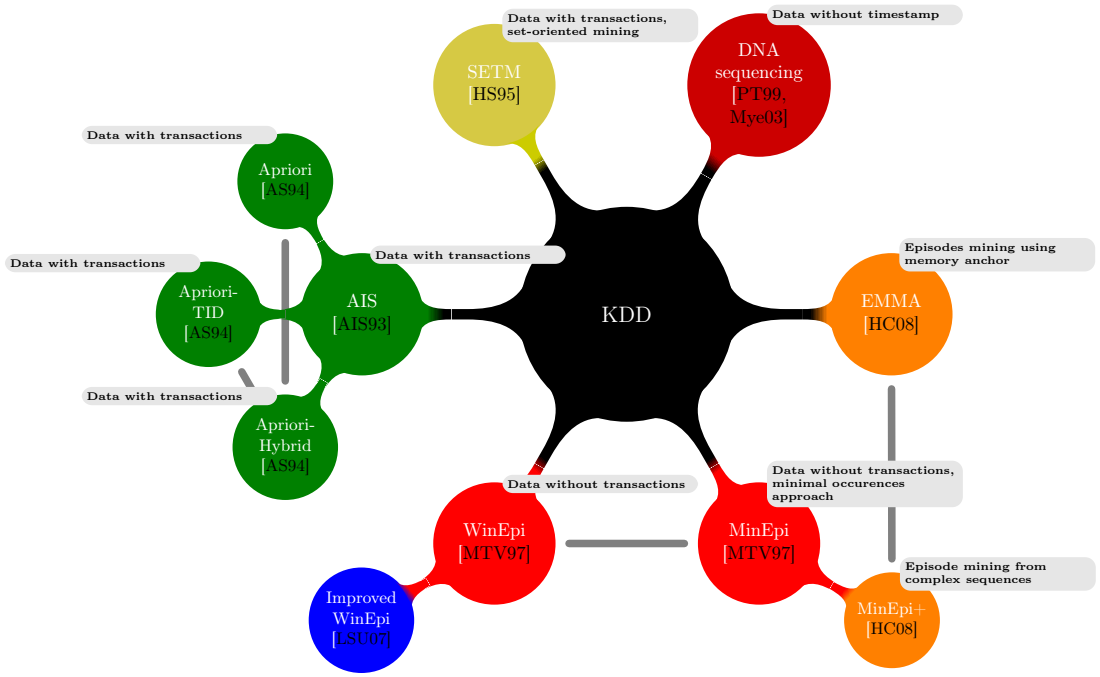


Abbildung 2.9: *Übersicht über in der Literatur vorgeschlagene Algorithmen*

Die Ursprünge des KDD liegen in der bereits einleitend beschriebenen Transaktionsanalyse. Darin werden Assoziationsregeln erstellt, über die Rückschlüsse auf das Einkaufsverhalten von Konsumenten möglich wird. Der erste und wohl bekannteste Algorithmus auf diesem Gebiet ist der Apriori-Algorithmus, welcher aus dem in [AIS93] vorgestellten AIS-Alogorithmus entwickelt und in [AS94] veröffentlicht wurde. In dieser Veröffentlichung werden auch die Algorithmen Apriori-TID und Apriori-Hybrid vorgestellt.

Apriori

Die Entwickler der Apriori-Algorithmen entwarfen diese rein mathematisch, so dass sie bis heute in den unterschiedlichsten Sprachen implementiert werden konnten. Einen anderen Ansatz verfolgten in diesem Zusammenhang die Entwickler in [HS95]: der dort vorgestellte SETM-Algorithmus wurde in SQL[32] implementiert, um transparent und di-

[32] **S**tructured **Q**uery **L**anguage

rekt auf Datenbanken arbeiten zu können. Genau wie die Apriori-Algorithmen arbeitet auch der SETM-Algorithmus auf Transaktionsdaten.

Eine weitere Domäne der KDD-Methoden ist die DNA-Sequenzanalyse, wie sie unter anderem in [PT99, Mye03] beschrieben wird. Da diese Verfahren keinen weiteren Bezug zu der vorliegenden Arbeit haben, seien sie nur der Vollständigkeit halber erwähnt.

Zur Auswertung von sequentiellen Daten wird in der Literatur häufig auf die in [MTV97] entwickelten Algorithmen verwiesen. Der als WinEpi[33] bezeichnete Algorithmus analysiert
WinEpi die Daten indem er ein Fenster zuvor definierter Breite über die gesamte Sequenz bewegt und dabei über Automaten die Häufigkeit von Untersequenzen zählt. Eine ausführliche Beschreibung zu diesem Algorithmus ist in Abschnitt 4.2.4 zu finden. Der ebenfalls in [MTV97] veröffentlichte Algorithmus MinEpi[34] verwendet kein bewegtes Fenster, benötigt sehr viel mehr Speicherplatz, besitzt jedoch nicht den Nachteil das die ermittelte Häufigkeit der Untersequenzen von ihrer Länge in Bezug zur Fenstergröße, wie es beim WinEpi-Algorithmus der Fall ist, abhängig ist.

Basierend auf den Entwicklungen in [MTV97] stellen die Autoren in [LSU07] einen weiteren Ansatz zur Ermittlung von häufigen Untersequenzen in Daten-Sequenzen vor, der
MinEpi sowohl beim Speicherbedarf als auch in der Laufzeit ein verbessertes Verhalten zeigen soll. In [HC08] beschreiben die Autoren mit MinEpi+ eine Weiterentwicklung des MinEpi-Algorithmus aus [MTV97] sowie den EMMA[35] Algorithmus, der einige Nachteile des MinEpi-Algorithmus adressiert.

Mit zunehmender Erweiterung der Einsatzgebiete von KDD-Methoden, beziehungsweise dem Data-Mining im Allgemeinen, entstanden auch Werkzeuge, die die Entwickler bei der Auswertung von Datenbasen und Modellierung von KDD- oder Data-Mining Prozessen unterstützen und die Wiederverwendbarkeit von Algorithmen erhöhen sollen. Eines dieser Werkzeuge ist das zunächst als YALE[36] in [MWK+06] vorgestellte und später in „Rapid-Miner“ umbenannte Werkzeug (siehe [Rap10]), welches sowohl umfangreiche Möglichkeiten zur Datenanalyse und Dokumentation mittels Diagrammen als auch Funktionalitäten zur Modellierung von komplexen Prozessen bietet. Die Anwendung wird unter einer OSS-Lizenz verteilt und ist somit frei verwendbar.

2.4.3 Inferenz - Logisches Schließen

Die Möglichkeiten der Inferenz sind direkt von der Art der Repräsentation abhängig oder anders ausgedrückt: Die Wissensrepräsentation muss so gewählt werden, dass die angestrebte Inferenzmethodik anwendbar wird. Aus diesem Grund ist es schwierig die Inferenz getrennt von der Wissensrepräsentation zu betrachten (siehe [BKI08]). Um

[33] **Win**dow based **Epi**sode discovery
[34] **Min**imal occurence based **Epi**sode discovery
[35] **E**pisodes **M**ining using **M**emory **A**nchor
[36] **Y**et **A**nother **L**earning **E**nvironment

dennoch Beispiele für die Anwendung geben zu können, sei wieder auf die MASSiVE-Architektur verwiesen.

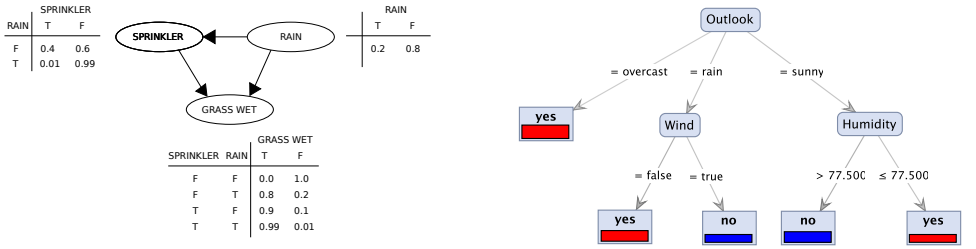
In MASSiVE kommt das regelbasierte Schließen zur Anwendung. Die Entscheidungen des Systems erfolgen auf unterschiedlichen Ebenen und mittels unterschiedlicher Methodiken. Ein Großteil der Handlungsplanung basiert auf zuvor vom Systemprogrammierer spezifiziertem Wissen, das in Form von Petri-Netzen modelliert wird. Die Wissensrepräsentation ist also in diesem Fall das Petri-Netz, welches ein direktes Abbild der möglichen Ausführungsabläufe darstellt. Wie bereits im Abschnitt 2.1.3 beschrieben wurde, erfolgt die Festlegung der Initialmarkierung im Rahmen der Umgebungserfassung. Letztere wird dabei durch den Einsatz von Prolog unterstützt. So wird es dem System ermöglicht, durch einfache Inferenzen auf Basis der Zustände von System- und Umgebungsobjekten die richtige Initialmarkierung zu erstellen und gleichzeitig den Aufwand der Umgebungserfassung zu minimieren. Dazu ein Beispiel: Ein Umgebungsobjekt, das in einem Container in der Umgebung steht, kann nicht gleichzeitig auf dem Tablett des Rollstuhlsystems stehen. Sobald eine dieser Eigenschaften detektiert wurde, muss die jeweils andere nicht überprüft werden.

regelbasiertes Schließen

Einfache Regelwerke können häufig auch mit den semantischen Mitteln einer Programmiersprache abgebildet werden. Die einfachste dieser Möglichkeiten besteht im Einsatz von bedingten Programmverzweigungen: „Wenn Bedingung A wahr ist, führe Anweisung B aus“. Bei etwas komplexeren Regelwerken kann hier darüber hinaus der Einsatz von Flussdiagrammen in Verbindung mit Quelltextgeneratoren hilfreich sein. Nachteilig an dieser Vorgehensweise ist der Fakt, dass Regelwerke, die in Form von Programmverzweigungen realisiert wurden, nicht zur Laufzeit einer Anwendung modifiziert werden können: Lernen ist somit ausgeschlossen.

Neben der semantischen Beschreibung von Regelwerken wird der Bereich des probabilistischen Schließens heute immer wichtiger. Die Methoden in diesem Bereich basieren auf der mathematischen Statistik und die Rückschlüsse eines Systems basieren auf Erfahrungen, die es zuvor bereits gemacht und in die Wissensbasis aufgenommen hat. Diese Verfahren eignen sich daher besonders gut für lernfähige Ansätze. Eine Möglichkeit der Wissensrepräsentation ist in diesem Zusammenhang die Modellierung über Bayes'sche Netze (Abbildung 2.10a) oder Entscheidungsbäume (Abbildung 2.10b).

wissensbasiertes Schließen



(a) *Beispiel eines Bayes'schen Netzes (Quelle: [Wik10])*

(b) *Beispiel eines Entscheidungsbaumes (Quelle: [Rap10])*

Abbildung 2.10: *Beispiele zur Modellierung von Wissen*

Abgesehen von diesen regelbasierten Formen des Schließens gibt es eine weitere Form, die insbesondere dann zum Einsatz kommt, wenn das Systemwissen nicht oder nur sehr aufwendig in Form von Regeln abgebildet werden kann: das fallbasierte Schließen oder CBR[37] [BKI08]. Dabei wird das Systemwissen nicht in Form von Regeln abgebildet, sondern als eine Kollektion von Fällen. Steht das System dann vor der Lösung eines neuen Falles, so vergleicht es diesen Fall mit den bereits in seiner Wissensbasis hinterlegten Fällen um solche zu identifizieren, die dem neuen Fall am ähnlichsten sind. Ist ein solcher Fall gefunden, so kann mit einer gewissen Erfolgswahrscheinlichkeit die Aktion des Falles auch für den neuen eingesetzt werden um das gewünschte Ergebnis zu erreichen. Eine der Hauptschwierigkeiten liegt in diesem Zusammenhang darin, ein anwendbares Maß für die Ähnlichkeit zweier Fälle zu definieren.

fallbasiertes Schließen

Genauso wie die Inferenz in enger Verbindung mit der Wissensrepräsentation steht, so steht sie auch in enger Verbindung mit dem Vorgang des Lernens. Leitet ein System auf Basis seines Wissens mittels Inferenz neues Wissen ab und erweitert es daraufhin seine Wissensbasis um das neue Wissen, so lernt es.

2.4.4 Lernen

Der Vorgang des Lernens besteht in der automatischen Anpassung von Systemwissen auf Basis von Erfahrungen. Um das im Zusammenhang mit den zuvor beschriebenen Methodiken der Inferenz zu realisieren, bedarf es unterschiedlicher Ansätze. Im Fall der regelbasierten Inferenz auf der Grundlage von Petri-Netzen und Prolog in MASSiVE wurde keine Lernfähigkeit realisiert. Die Regeln sind von Systemprogrammierern festgelegt worden und beschreiben im Fall der Petri-Netze mögliche Ausführungsabläufe und im Fall der Prologregeln logische Zusammenhänge zwischen Klassen von Umgebungs- und Systemobjekten. Diese Wissensbasen werden nicht automatisch erweitert.

Anders ist das bei der in [BKI08] vorgestellten Inferenz auf Basis von Entscheidungsbäumen. Hier besteht durch den Einsatz von TDIDT[38]-Verfahren die Möglichkeit, dass ein System einen Entscheidungsbaum auf Basis von Trainingsdaten erstellt und so autonom die eigene Wissensbasis erzeugt. Ein solcher Algorithmus ist der C4.5 Algorithmus, der einfach zu realisieren ist und somit ein weit verbreitetes Verfahren des maschinellen Lernens darstellt. Auch im industriellen Umfeld hat dieser Algorithmus bereits eine starke Verbreitung erfahren, wie sowohl [BKI08] als auch [KQ99] bestätigen. Da der Algorithmus nicht inkrementell arbeitet, ist es nicht möglich einen einmal erzeugten Entscheidungsbaum zu erweitern: wenn sich die Trainingsdaten ändern, muss der Baum komplett neu erstellt werden. Der Aufbau des Baumes erfolgt rekursiv, wobei der Algorithmus für jeden Knoten ein Attribut der Trainingsdaten ermittelt, welches die Daten am eindeutigsten in Gruppen klassifiziert. Über Rekursion wird dieser Vorgang dann für die Untergruppen fortgeführt.

TDIDT-Verfahren

[37] **C**ase-**B**ased **R**easoning
[38] **T**op-**D**own **I**nduction of **D**ecision **T**rees

In Bayes'schen Netzen werden alle Knoten mit der Wahrscheinlichkeitsverteilung der ihnen zugeordneten Zufallsvariable in Abhängigkeit von ihren Vorgängerknoten belegt, wie Abbildung 2.10a zeigt. Der Vorgang des Lernens besteht beim so genannten Parameter-Lernen darin, die den Knoten zugeordnete Wahrscheinlichkeitsverteilung durch neue Testdaten zu aktualisieren. Mit zunehmender Laufzeit des Systems bildet das Bayes'sche Netz die modellierte Umgebung oder den modellierten Prozess dadurch immer besser ab. Damit das funktioniert, muss die Struktur des Netzes von einem Systemprogrammierer vorgegeben worden sein. Anders ist das beim Struktur-Lernen von Bayes'schen Netzen. Hier wird das gesamte Netz automatisch auf Basis der Attribute des zu modellierenden Prozesses aus den Trainingsdaten heraus erstellt [LKMY02]. Eine sehr ausführliche Einführung in die Methoden des maschinellen Lernens Bayes'scher Netze ist in [Wit02] zu finden.

Parameter-Lernen

Struktur-Lernen

Besonders eng mit dem maschinellen Lernen verknüpft ist die zuvor in Abschnitt 2.4.3 beschriebene Methode des fallbasierten Schließens. Bei CBR ist das Lernen eine Eigenschaft, die direkt durch die angewandte Methodik bei der Inferenz als Nebenprodukt abfällt [AP94]. Wurde bei der Ähnlichkeitssuche in der Wissensbasis ein Fall gefunden, über den ein neuer Fall erfolgreich gelöst werden konnte, so muss letzterer lediglich zur Wissensbasis hinzugefügt werden und das System ist danach in der Lage, dieses Wissen später erneut anzuwenden: das System hat gelernt mit einem neuen Fall erfolgreich umzugehen.

Talente finden Lösungen, Genies entdecken Probleme.

Hans Krailsheimer - Schriftsteller

3

Problembeschreibung und Anforderungen

Das vorliegende Kapitel widmet sich dem Aufbau der am Institut für Automatisierungstechnik entwickelten Softwarearchitektur MASSiVE. Der Schwerpunkt dieser Betrachtung liegt in der Erörterung von offenen Problemen und der Herausstellung von Entwurfsdetails, die mit den in Abschnitt 1.2 beschriebenen Zielen der vorliegenden Arbeit in Konflikt stehen. Darauf aufbauend erfolgt die Festlegung von konkreten Anforderungen an die MMS und die anderen Schichten der Architektur.

Der erste Abschnitt beschreibt den Aufbau der MMS, die den Ausgangspunkt der Entwicklungen der vorliegenden Arbeit darstellt und im Hinblick auf notwendige Erweiterungen untersucht wird. Im Anschluss daran folgt die Betrachtung der Komplexität der MMS in Bezug auf die Kommunikation zwischen den Einzelkomponenten. Diese führt zu einer genauen Analyse der Möglichkeiten zur Einbindung von Eingabegeräten und der Erweiterbarkeit im Allgemeinen. Des Weiteren werden die notwendigen Anpassungen zur Realisation eines dynamischen Aufbaus der grafischen Oberfläche diskutiert. Durch diese soll eine hohe Adaptivität erreicht und ein statischer Aufbau wie in [RBSM10] vermieden werden. Die Untersuchungen der Möglichkeiten zum kontextabhängigen Aufbau sowie die Anforderungen für die Implementation und Integration von Algorithmen der KI in die MMS schließen den ersten Abschnitt ab.

Analyse

Im zweiten Abschnitt dieses Kapitels werden die anderen Schichten der MASSiVE-Architektur analysiert und in Hinblick auf die durch die Anpassung der MMS vorgegebenen Anforderungen untersucht. Die Kernthematik ist dabei die Ressourcenakquise in den unteren Schichten der Architektur sowie das Skill-Netzwerk.

3.1 Adaptivität der Benutzerschnittstelle

Einige der in Abschnitt 2.3.2 vorgestellten Entwicklungen im Bereich von Benutzerschnittstellen zeigen einen Fokus auf der Adaptivität. Ein Schlüsselwort in diesem Zusammenhang ist die Multimodalität, welche den Einsatz von vielen unterschiedlichen Eingabemedien beschreibt. Der weit verbreitete WIMP-Ansatz wird dabei zunehmend verlassen

[RBSM10]. Nicht nur bei Benutzerschnittstellen für Menschen mit Behinderung, sondern auch generell bietet der multimodale Ansatz einige Vorteile gegenüber der ausschließlichen Steuerung mittels Tastatur und Maus. Aus diesem Grund basiert die Adaptivität der bestehenden Benutzerschnittstelle von MASSiVE aus [Cyr05] allein auf dem Konzept der Multimodalität, realisiert über das MVC-Entwurfsmuster aus [GHJV95]. Die gesamte Kommunikation der Einzelkomponenten der MMS erfolgt hier über CORBA und ist dadurch sowohl betriebssystemunabhängig als auch über Rechnergrenzen hinweg skalierbar. Neben diesen Vorteilen birgt der Einsatz von CORBA in diesem Zusammenhang jedoch auch einige Nachteile, wie die folgenden Betrachtungen zeigen.

Multimodalität

3.1.1 Komplexität

Durch den Einsatz der fünf Einzelkomponenten der MMS bedarf es vieler Kommunikationskanäle zur Laufzeit. Dadurch bestehen aufgrund der zumeist bi-direktionalen Kommunikation viele Abhängigkeiten zwischen den Einzelkomponenten, ein Zusammenhang der bereits in Abschnitt 2.1.3 mit Bezug auf die einzuhaltende Startreihenfolge beschrieben wurde. Neben dieser auf den ersten Blick nachvollziehbaren Problematik ergibt sich aus den intensiv eingesetzten CORBA-Verbindungen ein weiterer Nachteil, der die Implementierung und Wartung von MMS-Komponenten erschwert: die Serialisierung (engl. Marshalling [Wik10, Stichwort: „Marshalling“]). Bei diesem Vorgang ist es notwendig, alle zu transportierenden Datentypen auf Standard CORBA-Datentypen abzubilden. Dazu müssen alle Daten vor dem Versand in ein Containerformat umgewandelt werden. Nach dem Versand der Daten erfolgt dann der umgekehrte Prozess. Dieses Vorgehen erweist sich während der laufenden Entwicklungsarbeiten zunehmend als fehleranfällig und komplex, so dass der vollständige Verzicht auf Serialisierung notwendig erscheint, um die Implementation von MMS-Komponenten zu vereinfachen.

hohe Komplexität

Anforderung 3.1. *Die in der MMS eingesetzten Kommunikationsmechanismen sollten vollständig auf Serialisierung verzichten.*

Der einfachste Weg das zu erreichen bestünde darin, auf die CORBA-Verbindungen zwischen den Einzelkomponenten der MMS weitestgehend zu verzichten. Es bleibt also zu untersuchen, inwiefern Einzelkomponenten zusammengefasst werden können um unnötige Kommunikationsverbindungen zu vermeiden und so diesem Ziel möglichst nahe zu kommen.

Anforderung 3.2. *Reduktion der Einzelkomponenten zur Vermeidung von unnötigen Kommunikationskanälen in der MMS unter Beibehaltung der Funktionalität und Skalierbarkeit.*

3.1.2 Eingabegeräte

Die Kommunikation der Eingabegeräte mit der Komponente „DoF Mapping“ erfolgt über einen Ereigniskanal. Bei dem Ereigniskanal handelt es sich um die ACE/TAO-

Implementation des im CORBA-Rahmenwerk spezifizierten „EventChannel“. Neben dem Ziel der parallelen Weiterleitung von Ereignissen der Eingaberäte wird darüber hinaus durch Einsatz des „canonical push“-Modells (siehe [HV99]) die vollständige Entkopplung der Eingabegeräte von der „DoF Mapping“-Komponente erreicht. Letztere spielt dabei die Rolle des „push consumer“ (Ereignisverbraucher) und die Eingabegeräte die Rolle der „push supplier“ (Ereigniserzeuger).

„canonical push model“

Die genaue Betrachtung dieser Vorgehensweise zeigt in Bezug auf den Einsatz des Ereigniskanals Nachteile. Der Ereigniskanal verwaltet eine Liste mit den registrierten Ereignisverbrauchern, um die Ereignisse sequentiell an diese weiterzuleiten. Wird dabei zur Laufzeit ein registrierter Ereignisverbraucher beendet, ohne dass sich dieser vom Ereigniskanal abmeldet, so verbleibt eine ungültige Referenz in der Liste des Ereigniskanals. Beim Eintreffen eines neuen Ereignisses versucht der Ereigniskanal dieses an den nicht mehr erreichbaren Ereignisverbraucher weiterzuleiten und läuft dabei in ein Time-out, das die Weiterleitung des Ereignisses an alle übrigen am Ereigniskanal registrierten Ereignisverbraucher verzögert. Durch Implementation von Routinen, die die Abmeldung der Ereignisverbraucher sicherstellen, konnte diese Problematik entschärft, aber nicht gänzlich vermieden werden.

Aufgrund der zuvor beschriebenen Problematik mit dem Einsatz des Nachrichtenkanals bedarf es einer Überprüfung, inwiefern ein alternativer Lösungsansatz zur Verfügung steht, der die Problematik nicht aufweist aber trotzdem die Verarbeitung der eingehenden Ereignisse erlaubt und darüber hinaus zu einer Entkopplung der Eingabegeräte von der MMS führt. In Bezug auf die bereits formulierte Anforderung 3.2 sollte untersucht werden, in wie fern es von Vorteil wäre, auf Einsatz des Ereigniskanals zu verzichten.

Ersatz des Nachrichtenkanals

Anforderung 3.3. *Die MMS muss eine robuste und entkoppelnde Kommunikation mit den Eingabegeräten ohne die bisherigen Einschränkungen durch den Ereigniskanal erlauben.*

Alle für die MMS entwickelten Eingabegerätekomponenten besitzen eine aktive Ereignisschleife, in der die Auswertung gegebenenfalls zum Einsatz kommender Hardwarekomponenten erfolgt. Es handelt sich dabei um eine Endlosschleife, die erst beim Stoppen der Eingabegerätkomponente verlassen wird. Innerhalb dieser Schleife kann eine einfache Auswertung mittels Polling erfolgen oder es können komplexe Regelkreise implementiert werden. Bei Versuchen zeigte sich, dass diese Vorgehensweise nicht bei allen Typen von Eingabegeräten notwendig ist. Insbesondere Eingabegeräte von externen Entwicklern werden häufig zusammen mit Bibliotheken geliefert, die die gesamte Auswertung übernehmen und mit „Call-Back“-Funktionen arbeiten. Aus diesem Grund sollte der Einsatz der Ereignisschleife in Eingabegerätkomponenten optional sein.

Anforderung 3.4. *Die Softwarekomponenten für Eingabegeräte müssen sowohl mit eigener, aktiver Ereignisverarbeitung als auch ohne implementiert werden können.*

Alle Ereignisse von Eingabegeräten werden vom Nachrichtenkanal an die Komponente „DoF Mapping“ weitergeleitet und dort ausgewertet. Zunächst erfolgt dabei eine Fal-

lunterscheidung in Abhängigkeit vom Ereignistyp. Es werden folgende Typen unterschieden:

- `COMMAND`: Kommandoereignisse bestehen aus Zeichenketten, die von einem Eingabegerät an die MMS geschickt werden. Sie kommen zum Beispiel bei Verwendung einer Spracherkennungssoftware zum Einsatz.

- `CURSOR_UPDATE`: Ereignisse, die die aktuelle Position des Mauszeigers der grafischen Oberfläche aktualisieren. Dazu wird das Ereignis an die MMS weitergeleitet und die Zeigerposition entsprechend angepasst.

- `CURSOR_CLICK`: Ereignisse, die einem Mausklick entsprechen. Sobald ein Ereignis dieser Art den Ereigniskanal erreicht, wird es weitergeleitet um in der MMS einen Mausklick zu initiieren.

- `CONFIGURE_REQUEST`: Ereignisse, die von einem Eingabegerät beim Start erzeugt werden und dazu führen, dass das entsprechende Eingabegerät von der MMS konfiguriert wird. Die Konfiguration erfolgt dabei über einen CORBA-Aufruf von der MMS auf die Schnittstelle der Komponente des Eingabegerätes ohne Einsatz des Ereigniskanals.

Bei genauerer Betrachtung wird deutlich, dass diese Fallunterscheidung innerhalb der MMS die direkte Schnittstelle der Eingabegeräte darstellt, da der Ereigniskanal für diese völlig transparent ist. Die Schnittstelle wird also direkt durch die von der MMS definierten Ereignistypen festgelegt. Die Einführung neuer Ereignisse ist ohne Änderung der MMS nicht möglich.

stark eingeschränkte Erweiterbarkeit

Zur Realisierung der zweiten Kernaufgabe der Komponente „DoF Mapping“, erhält diese Informationen über System- und Nutzerkontext von der MMS mit dem Ziel der kontextabhängigen Auswertung der eingehenden Ereignisse. Letztere kann in der Komponente „DoF Mapping“ jedoch nur durchgeführt werden, wenn dort auch die dazu notwendige Logik implementiert wird. Dieses widerstrebt jedoch einem erweiterbaren Entwurf: bei jedem neuen Eingabegerät muss auch die Logik in der „DoF Mapping“-Komponente erweitert werden, um der kontextabhängigen Abbildung gerecht zu werden. Auf diese Problematik wird im Abschnitt 3.1.3 noch genauer eingegangen.

Aus diesem Grund sollte nach einer Lösung gesucht werden, die die kontextabhängige Ereignisabbildung erlaubt, ohne bei Realisation einer neuen Erweiterung die Veränderung von Kernkomponenten der MMS zu forcieren. Im Gesamtzusammenhang betrachtet, erweist sich daher der Einsatz der Komponente „DoF Mapping“ als überflüssig, sofern weiterhin sichergestellt ist, dass die Eingabegeräte auf einer verteilten Systemarchitektur eingesetzt werden können. Die logische Konsequenz daraus besteht im Verzicht auf die Komponente „DoF Mapping“, was sich darüber hinaus mit der Forderung nach Klomplexitätsreduktion deckt (siehe Anforderung 3.2).

Anforderung 3.5. *Die Schnittstelle zur Steuerung der MMS über Eingabegeräte ist direkt von der MMS zur Verfügung zu stellen. Der Einsatz der Komponente „DoF Mapping“ wird dadurch überflüssig.*

3.1.3 Erweiterbarkeit

Die bereits angesprochene Erweiterbarkeit wird in der MMS von MASSiVE sowie der in [RBSM10] vorgestellten MMS durch den Einsatz von separaten Komponenten realisiert, die mittels Netzwerkverbindung (im Fall von MASSiVE per CORBA und Ereigniskanal und in [RBSM10] per UDP[1]-Verbindung) in das Gesamtsystem integriert werden. Die Eingabegeräte werden dadurch austauschbar und kommunizieren über eine feste Schnittstelle mit der MMS. Wie im vorherigen Abschnitt gezeigt wurde, erweist es sich als sinnvoll auf den Einsatz der Komponente „DoF Mapping“ in der MMS zu verzichten. Betrachtet man in diesem Zusammenhang den in Abbildung 2.5 dargestellten Aufbau der MMS auf Seite 15, so verbleiben lediglich Komponenten für die grafischen Oberflächen, die Eingabegeräte, die Benutzerinteraktionen und das „HMI Model“.

Reduktion der Komponenten

Ein Aspekt der Erweiterbarkeit kann in Form von Multimodalität realisiert werden, sollte jedoch nicht darauf beschränkt sein. Insbesondere in Projekten, in denen noch aktiv Forschungs- und Entwicklungsarbeit betrieben wird, ist ein generellerer Ansatz der Erweiterbarkeit zu verfolgen. Im Zusammenhang mit der MASSiVE-Architektur wurden im Verlauf der Entwicklung neben Hauptfunktionalitäten auch immer spezialisierte Testanwendungen realisiert, die in der Regel online zur Systemlaufzeit zum Einsatz kommen. Um die Systemtests zu vereinfachen, wäre es daher von Vorteil, wenn sich die Testanwendungen auch in die MMS integrieren ließen, ohne die Benutzeroberfläche für Endbenutzer unbrauchbar zu machen. Des Weiteren sind für die Entwickler in der Regel andere Daten von Interesse als für Endbenutzer, daher ist die Aufbereitung der Daten an den Nutzer anzupassen. Im Gegensatz dazu wird die in [RBSM10] vorgeschlagene Oberfläche kaum für alle Benutzer in Frage kommen: die vielen, zum Teil sehr kleinen Bedienelemente werden nicht mit jedem beliebigen Eingabegerät zu bedienen sein.

Die MMS nach [Cyr05] ist in der Lage mehrere „GUI“-Instanzen zu verwalten. Durch den Einsatz des „HMI Model“, welches die Eingabeereignisse über das „mvc“-Entwurfsmuster an alle „GUI“s weiterleitet, werden diese alle synchron gesteuert. Dadurch ist es nicht möglich, zwei parallel betriebene Instanzen der grafischen Oberfläche mit unterschiedlichen Eingabegeräten zu betreiben. Dieses zunächst als Vorteil entwickelte Verhalten (für den Einsatz von „GUI“s in unterschiedlichen Räumen der intelligenten Umgebung, siehe [Cyr05]) stellte sich im späteren Projektverlauf als Nachteil heraus.

nur synchrone GUIs

Die in Abschnitt 2.1.3 beschriebenen Benutzerinteraktionen der MASSiVE-Architektur werden dem Benutzer durch die MMS in Form von Dialogen präsentiert. Die Implementation dieser Dialoge erfolgt im Fall der MMS von MASSiVE als Teil der „GUI“s. Bei der Erweiterung der MMS um neue Benutzerinteraktionen sind also auch immer die zum

[1] **U**ser **D**atagram **P**rotocol

Einsatz kommenden „GUI“-Komponenten anzupassen, was die Erweiterbarkeit deutlich erschwert, beziehungsweise für zukünftige Drittanbieter unmöglich macht.

Komplexität verringern

Zusammenfassend betrachtet wäre ein modularer Aufbau der grafischen Komponenten der MMS von großem Vorteil. Dabei muss sichergestellt sein, dass jegliche Art von Erweiterung durchgeführt werden kann, ohne die „GUI“s anzupassen. Des Weiteren sollten sowohl die grafischen Elemente der „GUI“s als auch die Eingabegeräte über den gleichen Mechanismus in die MMS integriert werden können, um weiter die Komplexität zu verringern.

Anforderung 3.6. *Sowohl die Eingabegeräte und die Benutzerinteraktionen als auch alle anderen Elemente der MMS sind zu modularisieren. Die Module beinhalten die gesamte Funktionalität und die MMS stellt lediglich unterschiedliche Schnittstellen zur Verfügung, um die Kommunikation zwischen allen beteiligten Komponenten zu realisieren.*

Konfigurierbarkeit

Um die Eingabegeräte der MMS konfigurieren zu können, werden die Startargumente der entsprechenden Komponenten ausgewertet. Das wird nicht mehr möglich sein wenn die Eingabegeräte als Erweiterungsmodule realisiert werden. Diese werden nicht separat gestartet und besitzen daher auch keine Startargumente. Aus diesem Grund ist es notwendig eine Ersatzfunktionalität zu schaffen, die diese Problematik behebt ohne dabei zu Abhängigkeiten zwischen einzelnen Modulen zu führen.

Anforderung 3.7. *Alle Erweiterungsmodule müssen konfigurierbar sein. Um keine Abhängigkeiten zu erzeugen ist die Quelle der Konfigurationsdaten dabei für jede Erweiterung separat zu erstellen.*

3.1.4 Dynamischer Oberflächenaufbau

Adaptivität

Durch die zuvor in Anforderung 3.6 spezifizierte Umstrukturierung der Komponenten der MMS in einen über Module erweiterbaren Mechanismus wird die optimale Anpassungs- und Erweiterungsfähigkeit der MMS erreicht. Dieser Zusammenhang stellt die direkte Grundvoraussetzung für einen dynamischen Oberflächenaufbau der grafischen Benutzeroberfläche dar. Um diese Anforderung zu erfüllen müssen die zum Einsatz kommenden Module lediglich zur Laufzeit aktiviert und deaktiviert werden können.

Anforderung 3.8. *Die Erweiterungen der MMS müssen zur Laufzeit hinzugefügt und entfernt werden können. Die grafische Oberfläche hat neu hinzugefügte Erweiterungen entsprechend dynamisch anzuzeigen oder auszublenden.*

Betrachtet man in diesem Zusammenhang das Beispielszenario, bei dem ein Erweiterungsmodul die Interaktion mit dem Benutzer, zum Beispiel in Form einer Texteingabe benötigt, so ist die triviale Lösung, dass dieses Erweiterungsmodul die Texteingabe eigenständig durchführt. Langfristig gesehen verschlechtert dieses Vorgehen jedoch die Möglichkeit zur Wiederverwendung von Erweiterungsmodulen. Vorteilhaft wäre es spezielle Module

für die Texteingabe zu entwickeln, die von anderen Modulen verwendet werden können. Diese Vorgehensweise setzt jedoch die bidirektionale Kommunikation zwischen unterschiedlichen Erweiterungsmodulen voraus, was zur folgenden Anforderung führt.

Anforderung 3.9. *Die Erweiterungsmodule müssen in die Lage versetzt werden untereinander zu kommunizieren. Dabei sollte derselbe Kommunikationsmechanismus zum Einsatz kommen, der bereits in Anforderung 3.3 formuliert wurde.*

3.1.5 Ausgabe von Meldungen

Die Ausgabe von Meldungen wird durch jede Komponente separat durchgeführt. Alle Meldungen lassen sich dabei in zwei Gruppen aufteilen: Systemmeldungen, die zu Zwecken der Fehlersuche ausgegeben werden, und Nutzermeldungen, die an den Benutzer adressiert sind. Erstere werden in Textform mit Zeitstempel auf der Konsole der entsprechenden Komponente zur Laufzeit ausgegeben oder über die Start-Komponente der MMS gesammelt und in eine Datei geschrieben. Meldungen an den Benutzer können lediglich durch die „GUI“s der MMS initiiert werden. Eingabegeräte oder andere beteiligte Komponenten sind nicht in der Lage Meldungen an den Benutzer auszugeben. Diese Einschränkung muss im Rahmen der Anpassungen aufgehoben werden. Alle Module zur Erweiterung müssen in die Lage versetzt werden, sowohl Systemmeldungen als auch Nutzermeldungen auszugeben.

Nutzer- und Systemmeldungen

Anforderung 3.10. *Alle Module zur Erweiterung der MMS müssen in der Lage sein sowohl Nutzermeldungen als auch Systemmeldungen zu initiieren.*

3.1.6 Internationalisierbarkeit

Die Internationalisierbarkeit der MMS basiert auf einem Mechanismus, der von der eingesetzten Qt-Bibliothek zur Verfügung gestellt wird. Nachteil dieses Mechanismus' ist die Tatsache, dass dynamisch zur Laufzeit generierte Texte nicht übersetzt werden können. Der Einsatz von Variablen in Texten, die zur Laufzeit mit Textteilen ersetzt werden, ist so nur begrenzt möglich, denn die Inhalte der Variablen können nicht übersetzt werden. Des Weiteren muss zum Einsatz des Mechanismus' die entsprechende Anwendung gegen die Qt-Bibliothek gelinkt werden, was für Eingabegeräte in [Cyr05] nicht gewünscht war. So fehlt derzeit im Zusammenhang mit Eingabegeräten jegliche Form der Internationalisierbarkeit und ähnliches gilt für die Benutzerinteraktionen. Daraus ergibt sich folgende Anforderung für die Internationalisierbarkeit der MMS.

Multilingualität

Anforderung 3.11. *Alle Zeichenketten, die dem Benutzer präsentiert werden, müssen ohne Einschränkungen übersetzt werden können, so dass das Gesamtsystem aus MMS und allen Modulen zur Erweiterung internationalisierbar ist.*

3.1.7 Systemdaten und Umgebungskontext

Zugriff auf Systemdaten

Zur Realisierung der systemspezifischen Funktionalitäten ist es notwendig, den Erweiterungsmodulen das dazu notwendige Wissen zur Verfügung zu stellen (zum Beispiel die zur Verfügung stehenden Szenarien). Die MMS besitzt dazu eine Schnittstelle zur Systemdatenbank. Da nach Anforderung 3.6 alle Komponenten der MMS als Erweiterungsmodule zu realisieren sind, muss eine Möglichkeit geschaffen werden, diesen Modulen den notwendigen Datenbankzugriff zu gewähren.

Anforderung 3.12. *Die Erweiterungsmodule benötigen Zugriff auf die Systemdatenbank von MASSiVE. Die zur Handlungsausführung notwendigen Informationen sind der MMS über eine Schnittstelle zur Verfügung zu stellen.*

Ein weiteres Ziel der MMS ist ein kontextabhängiger Aufbau. Ein Beispielszenario wäre in diesem Zusammenhang der Einsatz eines Kamerasystems mittels dessen festgestellt wird, welches Umgebungsobjekt ein Nutzer gerade mit den Augen fokussiert. In Abhängigkeit von diesem Objekt, also in Abhängigkeit vom aktuellen Kontext des Benutzers, soll die MMS in die Lage versetzt werden kontextspezifische Funktionalitäten zur Verfügung zu stellen. Diese Funktionalität basiert auf der bereits in Anforderung 3.8 formulierten Notwendigkeit nach einem dynamischen Oberflächenaufbau. Die zur Realisierung dieser Funktionalität benötigten Kontextinformationen sind den Erweiterungen daher über eine weitere Schnittstelle der MMS zur Verfügung zu stellen.

Anforderung 3.13. *Die MMS muss den Modulen eine Schnittstelle zur Verfügung stellen, mittels derer Kontextinformationen akquiriert werden können.*

3.1.8 Integration von KI-Algorithmen

In Abschnitt 2.4 wurde in die Thematik des maschinellen Lernens eingeführt. Zusammenfassend ließen sich dabei sämtliche Methoden in diesem Bereich auf die vier Schritte Erstellen einer Datenbasis, Wissensschöpfung sowie Repräsentation, Inferenz und Lernen abbilden. Die im Rahmen der vorliegenden Arbeit angepasste und erweiterte MMS bildet die Grundlage für die Implementation von Algorithmen des maschinellen Lernens. Die bereits formulierten Anforderungen müssen sich daher auch in den Randbedingungen für die Integration dieser Algorithmen widerspiegeln.

Zugriff auf Wissensbasis

In Anforderung 3.6 erfolgt die Festlegung, dass sowohl die Eingabegeräte als auch sämtliche Funktionalitäten der MMS in Form von Modulen zu realisieren sind. Dieser Anforderung ist auch bei der Entwicklung von Algorithmen des maschinellen Lernens Rechnung zu tragen. Das bedeutet, jeder implementierte Algorithmus wird als Modul realisiert. Die für die Ausführung der Algorithmen benötigten Daten müssen daher von der MMS in Form einer Schnittstelle zur Verfügung gestellt werden.

Von den Algorithmen muss zunächst eine Datenbasis erstellt werden, die das „Gedächtnis“ des Moduls darstellt und nicht von der Laufzeit der MMS begrenzt werden darf.

Datenbasis nutzerbezogen

Bei den Daten wird es sich in der Regel um Nutzungsdaten oder Benutzereingaben handeln, mittels derer ein Modul zum Beispiel in die Lage versetzt wird Nutzungsprofile zu erstellen. Da diese Daten grundsätzlich benutzerabhängig sind, ist es notwendig die gespeicherten Daten auch im Nachhinein einem bestimmten Benutzer zuzuordnen.

Anforderung 3.14. *Die Module müssen durch eine Schnittstelle der MMS in die Lage versetzt werden eine Datenbasis aufzubauen, die unabhängig von der Laufzeit der MMS gespeichert wird. Zur Laufzeit muss sowohl der schreibende als auch der lesende Zugriff gewährleistet sein. Des Weiteren sind die Daten dem aktuellen Benutzer des Systems zuzuordnen.*

Zugriff auf Systemkomponenten

Neben der Möglichkeit eine Datenbasis zu erstellen und dauerhaft abzulegen wird auch ein Mechanismus benötigt, der den Modulen Zugriff auf Systemkomponenten ermöglicht. Dieser Zugriff erlaubt die Akquise von Umgebungsdaten, die durch externe Sensorik, also zum Beispiel durch Hardwarekomponenten, erfasst wurden.

Anforderung 3.15. *Alle Module benötigen einen Mechanismus zum Zugriff auf Systemkomponenten um so zur Laufzeit Systemdaten akquirieren und auswerten zu können.*

Des Weiteren ist es notwendig, das aus der Datenbasis extrahierte Wissen in einer geeigneten Form abzulegen. Auch dazu muss die MMS den Modulen eine Schnittstelle zur Verfügung stellen. Um die Schnittstellen nicht unnötig komplex werden zu lassen, sollte hier die gleiche Schnittstelle zum Einsatz kommen, die bereits für die Verwaltung der Datenbasis verwendet wird.

Anforderung 3.16. *Die Schnittstelle zur Datenbasis muss generisch aufgebaut sein, so dass die verfügbaren Daten in unterschiedlicher Form abgelegt werden können.*

Bei den verbleibenden Schritten, Inferenz und Lernen, handelt es sich um Vorgänge die auf der bereits vorhandenen Datenbasis arbeiten. Sie benötigen daher lediglich Schreib- und Lesezugriff auf die Datenbasis.

3.2 Kommunikation und Lokalisierung von Komponenten

Nach der detaillierten Betrachtung der Anforderungen an die adaptive Benutzerschnittstelle folgt in diesem Abschnitt die Untersuchung der Anforderungen an die anderen Schichten der MASSiVE-Architektur. Die bereits eingangs beschriebenen Komponenten der Architektur werden im Zusammenhang mit der dritten Schicht, der reaktiven Ebene, auch als Server bezeichnet. Um die Problematik bei der Kommunikation dieser Komponenten untereinander beschreiben zu können, folgt zunächst eine detailliertere Einführung zu den Mechanismen der eingesetzten Kommunikation mittels CORBA.

Jeder Server in MASSiVE wird durch ein aktives CORBA-Objekt implementiert. Die Kommunikation zwischen den Servern erfolgt mittels CORBA-Verbindungen, denen als

Verbindungsmedium eine Ethernet-Verbindung zu Grunde liegt. Der Auf- und Abbau dieser Netzwerkverbindungen erfolgt vom ORB[2] der eingesetzten CORBA-Implementation ACE/TAO. Damit Verbindungen zwischen unterschiedlichen Servern hergestellt werden können, ist jeder Server über eine eindeutige Adresse, der IOR[3] erreichbar [Obj10]. Dabei ist eine IOR zur Laufzeit eines Servers in erster Linie eine Objektinstanz, die über so genannte Lokalisierungsdienste einfach verwaltet und anderen Servern zur Verfügung gestellt werden kann. Der in MASSiVE zum Einsatz kommende Lokalisierungsdienst ist der Naming-Service.

3.2.1 Lokalisierung der Server

Der Aufbau des Naming-Services ähnelt einem Telefonbuch. Jede verwaltete IOR wird über einen Namen bei Naming-Service registriert und ist daraufhin über diesen abrufbar. Dieses Vorgehen hat den Vorteil, dass jedem Server ein symbolischer Name zugewiesen werden kann. Obwohl sich die zur Laufzeit als Objektinstanz auftretenden IORs auch in eine Zeichenkette konvertieren lassen, vereinfacht das deren Handhabung, da das Ergebnis der IOR-Konvertierung in der Regel eine mehrere hundert Zeichen lange Zeichenkette ist.

Durch den Einsatz eines Naming-Services reduziert sich das Problem der IOR-Verwaltung auf die Verwaltung von symbolischen Namen für jeden zum Einsatz kommenden Server. Um bei einer großen Anzahl von Servern die Übersichtlichkeit weiter zu gewährleisten, besitzt der Naming-Service die Möglichkeit der Angabe von gemeinsam genutzten Präfixen, dem sogenannten Kontext. Dadurch entsteht ein IOR-Kontextpfad (auch als Namenskontext bezeichnet), der viele Gemeinsamkeiten mit einem Dateipfad aufweist. Diese Möglichkeit der Kontextspezifizierung wird in MASSiVE für die Gruppierung der Server in Funktionsgruppen verwendet. So besitzen alle Skill-Server den Kontext „Skill-Server“ und alle Hardware-Server den Kontext „HardwareServer“ wie Beispiel 3.1 zeigt.

Namenskontext

Beispiel 3.1. *Beispiele für mögliche Namenskontexte von Komponenten der MASSiVE-Architektur:*

```
1 SkillServer/Manipulator
2 SkillServer/MachineVision
3 SkillServer/UserInteraction
4 HardwareServer/Robotarm
5 HardwareServer/Bumblebee
6 WorldModel
7 Sequencer
```

Jede Zeile von Beispiel 3.1 zeigt den Namenskontext für eine Komponente. Der linke Teil eines Namenskontextes wird mit dem Zeichen „/“ abgetrennt (sofern vorhanden) und als Kontext, der rechte Teil als Referenz bezeichnet. Obwohl der eingesetzte Naming-Service den Einsatz einer beliebigen Anzahl von Kontexten zulässt, wird von dieser Möglichkeit

Kontext und Referenz

[2] **O**bject **R**equest **B**roker
[3] **I**nteroperable **O**bject **R**eference

kein Gebrauch gemacht. Der einzige Kontext, der zum Einsatz kommt, spezifiziert die Art des Servers.

Durch die zuvor beschriebene Vorgehensweise ist es in der MASSiVE-Architektur notwendig, dass jeder Server, der Zugriff auf einen weiteren Server benötigt, dessen Namenskontext kennen muss. Für die Hauptkomponenten von MASSiVE, wie zum Beispiel die MMS, den Sequenzer und das Weltmodell, erfolgt die Festlegung des Namenskontextes daher in einer globalen Konfigurationsdatei, auf welche die Hauptkomponenten der Architektur Zugriff haben.

Abhängigkeiten durch Namenskontext

Aufwendiger ist die Zuordnung der Namen für die Hardware- und Skill-Server der Architektur. Diese bilden in der reaktiven Ebene der Architektur das so genannte Skill-Netzwerk und besitzen Abhängigkeiten untereinander. Aus diesem Grund wird jeder Server vor der Ausführung von Operationen initialisiert. Dabei erfolgt die Spezifikation von genutzten Servern direkt beim Aufruf der Initialisierungsoperation durch Übergabe einer Liste von Namenskontexten.

3.2.2 Das Skill-Netzwerk

Die implementierten Hardware- und Skill-Server sind in funktionelle Gruppen eingeteilt (siehe Abschnitt 2.1.3). Dabei ist für jeden Hardware-Server immer genau ein Skill-Server verantwortlich. Die Aufgabe des Skill-Servers besteht darin, die ihm zugeordneten Hardware-Server vor der Verwendung korrekt zu initialisieren. Darüber hinaus muss er Anfragen von anderen Skill-Servern, die ebenfalls Zugriff auf die Hardware-Komponente benötigen, an diese delegieren.

Systemressourcen

Die Initialisierung erfolgt über eine spezielle Schnittstellenmethode, die sowohl von den Hardware- als auch von den Skill-Servern zur Verfügung gestellt wird. Diese Methode (mit dem Namen `InitConnections()`) erhält als einziges Argument eine Liste mit Namenskontexten der zu verwendenden Server, die in diesem Zusammenhang auch als Systemressourcen bezeichnet werden. Die angegebenen Ressourcen werden ausgewertet, um die CORBA-Verbindung zu den entsprechenden Servern herzustellen. Der Aufruf der Initialisierungsmethode für die Skill-Server erfolgt dabei durch den Sequenzer oder die MMS und der Aufruf für die Hardware-Server durch die entsprechend zugeordneten Skill-Server oder, im Fall der direkten Systemkontrolle, ebenfalls durch die MMS.

Sämtliche dem Skill-Netzwerk zugeordneten Ressourcen kapseln System- oder Hardwarefunktionalitäten, die nach Anforderung 3.15 allen Modulen in geeigneter Form zur Verfügung gestellt werden müssen. Um das zu realisieren, bedarf es einer Schnittstelle in der MMS, die allen Modulen die Ressourcenakquise erlaubt. Dabei darf es zu keinerlei Abhängigkeiten der Schnittstelle von spezifischen Eigenschaften der Ressourcen kommen. Des Weiteren muss sichergestellt sein, dass gleichzeitige Zugriffe durch unterschiedliche Module nicht zu Fehlverhalten führen.

Anforderung 3.17. *Die Module müssen in der Lage sein, über eine Schnittstelle in der MMS auf Systemressourcen zugreifen zu können. Der möglicherweise auftretende asynchrone Zugriff muss sicher sein.*

3.2.3 Initialisierungsproblematik

Damit sich die Server korrekt initialisieren können, muss beim Aufruf der Initialisierungsmethode die zu übergebene Liste mit Ressourcen feststehen. Im Fall der Initialisierung durch den Sequenzer ist es daher notwendig, dass dieser die Liste kennt. Dazu erfolgt die Festlegung der Ressourcen für alle Server in der globalen MASSiVE-Konfiguration. Aus dieser liest der Sequenzer die Ressourcen für den zu initialisierenden Server und führt die Initialisierung durch, dabei muss er den Namen der Ressource kennen, deren Abhängigkeitsbeziehungen er benötigt. Abbildung 3.1 verdeutlicht diesen Zusammenhang anhand eines Beispiels.

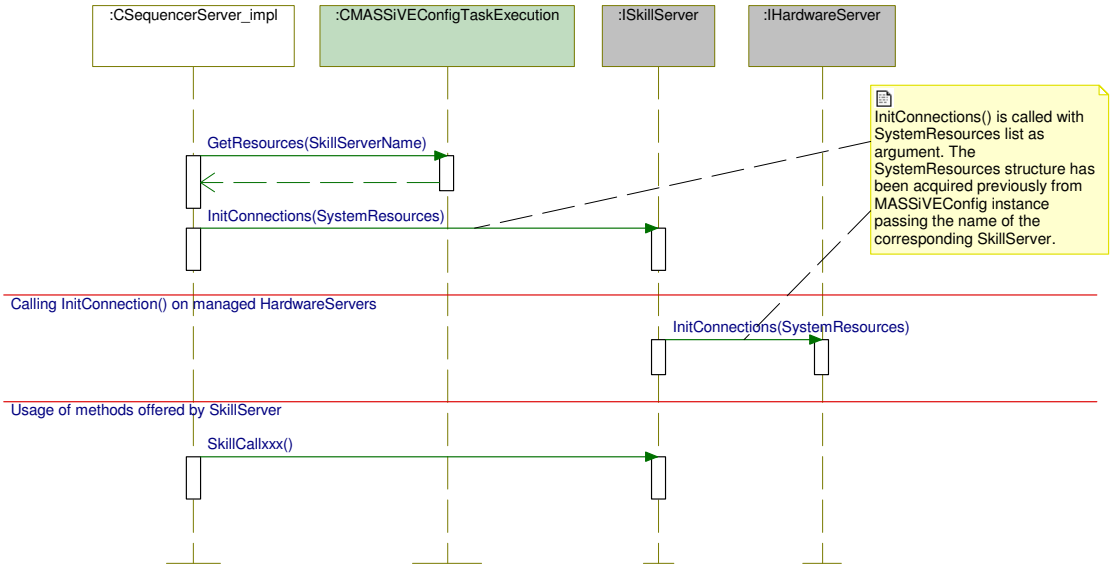


Abbildung 3.1: *Abhängigkeiten in der reaktiven Ebene*

Abhängigkeiten verhindern Adaptivität

Problematisch wird dieses Vorgehen wenn man die Forderung nach einer adaptiven MMS aus Abschnitt 3.1 mit berücksichtigt. Eine der Kernanforderungen war in diesem Zusammenhang, dass die MMS keinerlei Abhängigkeiten zu bestimmten Serverkomponenten aufweisen darf. Als Abhängigkeit gilt dabei alles, was an Daten zu einem Server zur Verfügung steht, angefangen bei dem Namenskontext bis hin zu den verwendete Ressourcen. Weil bei der Initialisierung eines Servers die Ressourcenliste angegeben werden muss, entsteht eine Abhängigkeit zu Daten, die innerhalb der MMS irrelevant sind.

Bei genauerer Betrachtung dieses Vorgehens stellt sich die Frage, ob der Sequenzer oder die MMS in der Lage sind, die Daten zu erzeugen ohne sie aus der globalen Konfiguration

zu lesen. Um diese Frage zu beantworten, bedarf es eines genaueren Blicks auf die Daten, welche Beispiel 3.2 exemplarisch zeigt.

Beispiel 3.2. *Folgende Zeilen zeigen einen Auszug aus der globalen Konfiguration:*

```
1  [SEQ_HardwareServerNamingContext]
2  Robotarm=HardwareServer, Robotarm
3  Gripper=HardwareServer, Gripper
4  FTSensor=HardwareServer, FTSensor
5  Bumblebee=HardwareServer, Bumblebee
6
7  [SEQ_SkillServerNamingContext]
8  ManipulatorSkillServer=SkillServer, Manipulator
9  CalculationSkillServer=SkillServer, Calculation
10 MachineVisionSkillServer=SkillServer, MachineVision
11 UserInteractionSkillServer=SkillServer, UserInteraction
12
13 [SEQ_SkillServerSkillServer]
14 ManipulatorSkillServer=CalculationSkillServer
15 CalculationSkillServer=ManipulatorSkillServer
16 UserInteractionSkillServer=MachineVisionSkillServer,
17                            TraySkillServer,
18                            ManipulatorSkillServer
19
20 [SEQ_SkillServerHardwareServer]
21 ManipulatorSkillServer=Robotarm, Gripper, FTSensor
22 MachineVisionSkillServer=Bumblebee
```

Die einzelnen Abschnitte der Konfigurationsdaten werden über ein Label eingeleitet, welches die Daten näher spezifiziert. Die folgenden Label kommen im Beispiel zur Anwendung:

- `SEQ_HardwareServerNamingContext`: Festlegung der Namenskontexte für alle verfügbaren Hardware-Server in Form von Schlüssel-Wert-Paaren. Die linksseitige Zeichenkette wird als Name für die Server-Komponente verwendet und ist in diesem Zusammenhang gleichzeitig ein eindeutiger Schlüssel. Rechtsseitig folgen die Kontextangaben und als letztes steht die Referenz mittels Komma getrennt.

- `SEQ_SkillServerNamingContext`: Festlegung der Namenskontexte für alle verfügbaren Skill-Server (siehe `SEQ_HardwareServerNamingContext`).

- `SEQ_SkillServerSkillServer`: Zuordnung von Skill-Servern zu anderen von ihnen verwendeten Skill-Servern. Dieser Abschnitt beschreibt das aufzubauende Skill-Netzwerk. Die Schlüssel-Wert-Paare sind für `a=b` in der Form „a verwendet b“ zu lesen, wobei für „b“ eine mittels Komma getrennte Liste angegeben werden kann. Zur Auflistung der verwendeten Server ist der Schlüssel aus dem vorherigen Konfigurationsabschnitt zu verwenden.

  Skill-Netzwerk

- `SEQ_SkillServerHardwareServer`: Zuordnung von Hardware-Servern zu Skill-Servern (siehe `SEQ_SkillServerSkillServer`).

Durch die Angaben in der globalen Konfiguration werden also die Abhängigkeiten von Server-Komponenten innerhalb der reaktiven Ebene beschrieben. Wodurch diese Abhängigkeiten entstehen wird deutlich, wenn man sich die Implementationsdetails zu den Komponenten anschaut. Dass ein Server einen anderen verwendet, also abhängig von ihm ist, heißt nichts weiter als dass er Methoden dieses Servers benötigt, um die ihm gestellte Aufgabe auszuführen. Fehlt der zu verwendende Server, so wird die Ausführung fehlschlagen. Mit diesem Hintergrundwissen wird auch klar, dass der Sequenzer oder die MMS keine Möglichkeit haben die Informationen zu generieren, da sie kein Detailwissen über die Serverimplementation besitzen. Letzteres besitzt nur der Entwickler, der den Server implementiert hat. Die Daten über die Abhängigkeiten stehen also im direkten Bezug zum Server selbst, so dass diese auch vom Server beziehungsweise dessen Entwickler geliefert werden sollten. Dieser Rückschluss führt direkt zur folgenden Anforderung:

Abhängigkeitsinformationen

Anforderung 3.18. *Jeder Server beziehungsweise dessen Entwickler hat die Abhängigkeitsinformationen zu liefern. Die* `InitConnections()`*-Methode darf zu keiner Abhängigkeit der MMS von ressourcenspezifischen Informationen führen.*

Die Ablage der Daten in die globale Konfiguration ist nicht nur fehleranfällig sondern auch schwer zu warten. Ein grafischer Editor existiert nicht und eine Validierung der Eintragungen ist auch unmöglich. Es wird dabei insbesondere dann problematisch, wenn Dritthersteller später Komponenten in Form von Servern liefern wollen: sie haben keinen Zugriff auf die globale Konfiguration.

3.2.4 Problematik bei der Ressourcenakquise über Namenskontexte

Die Verwaltung der Server erfolgt über einen Naming-Service, das heißt, die IOR eines Servers wird über einen Namenskontext erfragt, wie es bereits im vorherigen Abschnitt beschrieben wurde. Dadurch entsteht aufgrund der fehlenden Typsicherheit ein weiterer Nachteil bei diesem Verfahren, was damit zusammenhängt, dass die Namenskontexte völlig frei vergeben werden können. Ein Server, der „ManipulatorSkillServer“ heißt, muss nicht zwangsläufig ein Server sein, der auch die „ManipulatorSkillServer“ CORBA-Schnittstelle implementiert. Bei der Akquise einer Ressource über einen Namenskontext ist also in keinem Fall sichergestellt, dass die Ausführung von Servermethoden im Anschluss daran erfolgreich sein wird.

Trading-Service

Um diese Problematik zu umgehen, wurde in [Fre09] die Möglichkeit des Einsatzes eines anderen Lokalisierungsdienstes untersucht und implementiert. Es handelt sich dabei um den Trading-Service, dessen Aufbau mit dem eines Branchenbuchs vergleichbar ist. Der Dienst bietet die Möglichkeit Ressourcen anhand ihrer Eigenschaften zu suchen. Der Suchende erhält dann eine Liste mit Servern, deren Eigenschaften den angefragten entsprechen. Darüber hinaus lassen sich weitere Daten als benutzerdefinierte Eigenschaften der Ressourcen ablegen. Unter anderem kann man auch nach Servern suchen, die eine bestimmte Schnittstelle implementieren. Letztere wird in diesem Zusammenhang auch als Repository-ID bezeichnet.

Im Rahmen von [Fre09] wurde die MASSiVE-Architektur um die Möglichkeit der Verwendung des Trading-Servers erweitert. Des Weiteren wurde der DDS[4]-Server entwickelt, der andere Serverinstanzen oder Anwendungen über den Status von Servern informieren kann. Die Anwendungen müssen sich dazu registrieren und werden dann zyklisch über Statusänderungen informiert.

Im Verlauf der Entwicklung zeigte sich, dass sämtliche Daten, die zur Laufzeit relevant sind und vom Trading-Server geliefert werden können, auch über den Naming-Service zu erfragen sind. Der Naming-Service wird dazu jedoch nicht in seiner primären Anwendungsart eingesetzt: es werden alle Daten zu Verknüpfungen von Servern zu Namenskontexten vom DDS gespiegelt und lokal durchsucht. Der einzige Nachteil, den diese Vorgehensweise aufweist, ist die Tatsache, dass die zu akquirierenden Server aktiv und erreichbar sein müssen. Über den Trading-Service können die Daten auch abgefragt werden, wenn die Server nicht aktiv sind, sich aber zuvor einmalig registriert haben.

Integration des DDS

Um die MMS in die Lage zu versetzen, von den Vorteilen des DDS zu profitieren, muss dessen Schnittstelle in die MMS integriert werden. Damit durch dieses Vorgehen keine kritische Abhängigkeit zum DDS entsteht, ist die MMS in die Lage zu versetzen Ressourcen auch ohne diesen zu akquirieren.

Anforderung 3.19. *Der aus [Fre09] entstandene DDS, beziehungsweise dessen Funktionalität, ist in die MMS zu integrieren ohne eine kritische Abhängigkeit zu erzeugen.*

Durch die Neurealisierung der Ressourcenabhängigkeiten der Server-Komponenten und die Umstellung auf die Funktionalitäten des DDS, kann die MMS zukünftig in die Lage versetzt werden, sämtliche Systemressourcen zur Laufzeit zu akquirieren, ohne dabei auf die möglicherweise willkürlich gesetzten Namenskontexte angewiesen zu sein.

[4] **D**evice-**D**etection-**S**erver

Wer glaubt etwas zu sein, hat aufgehört etwas zu werden!

Sokrates - griechischer Philosoph

4

Theoretische Grundlagen und Definitionen

Das folgende Kapitel beschreibt die theoretischen Grundlagen und Definitionen, die als Hintergrund für die in den nächsten Kapiteln folgenden Ausführungen benötigt werden. Die Angaben basieren dabei in der Regel auf Notationen, die in der Literatur häufig verwendet werden; lediglich problemspezifische Anpassungen und Erweiterungen, die für den Kontext dieser Arbeit notwendig sind, wurden vorgenommen.

4.1 Formale Modellierung

In den vergangenen Jahren sind aufgrund der steigenden zur Verfügung stehenden Rechenkapazitäten von Personal-Computern die Softwarelösungen und damit die ihnen zugrunde liegenden Quelltexte immer umfangreicher geworden. Es ergaben sich neue Anforderungen an die Entwurfsverfahren und die Organisation von Projekten. Dabei musste weiterhin sichergestellt sein, dass auch sehr umfangreiche Projekte übersichtlich und umfassend dokumentiert werden können. Es hat sich gezeigt, dass es von Vorteil ist, bereits bei dem Entwurf eines Projektes auf die Unterstützung von grafischen Entwicklungswerkzeugen zu setzen. In diesem Zusammenhang hat sich bis heute die UML[1] als Modellierungsmethode für Software durchgesetzt. Es entstanden viele kommerziell und frei erhältliche Werkzeuge, die auf Basis der UML die Entwicklung unterstützen und teils vereinfachen. Dabei sind grundsätzlich zwei Arten von Werkzeugen in diesem Bereich zu unterscheiden: die reinen UML-Editoren und die so genannten CASE[2]-Werkzeuge. Bei ersteren erfolgt die Anwendung der UML ausschließlich zu Dokumentationszwecken. Eine Modellierung und Quelltextgenerierung ist nicht möglich. Letztere verlagern die Entwicklung vollständig in das CASE-Werkzeug und erlauben die Generierung des Quelltextes auf Basis des Modells [Bie93, KT08].

UML

CASE

Im Rahmen des MASSiVE Projektes erfolgt die Entwicklung der zugrunde liegenden Quelltexte mit Hilfe des CASE-Werkzeugs Rhapsody in C++ der Firma IBM. Die in

[1] **U**nified **M**odeling **L**anguage

[2] **C**omputer **A**ided **S**oftware **E**ngineering

Abschnitt 4.1.1 verwendete Darstellung der UML basiert auf diesem Werkzeug und erläutert einige der zum Einsatz kommenden Entwurfsmuster. Bei den zur Beschreibung eingesetzten Diagrammen handelt es sich um Klassendiagramme.

Modellierung in XML

Neben der Modellierung von Softwareelementen erfolgt im Rahmen der vorliegenden Arbeit die Modellierung von Metadaten zu unterschiedlichen Komponenten der MMS und der MASSiVE-Architektur. Diese Modellierung basiert auf eigens dafür definierten XML[3]-Dialekten. Die zum Verständnis der Modellierung notwendigen Hintergründe werden in Abschnitt 4.1.2 erläutert.

4.1.1 Entwurfsmuster

Entwurfsmuster sind Vorlagen für Lösungsansätze zu häufig wiederkehrenden Problemen in der objektorientierten Softwareentwicklung. Sie beschreiben auf abstrakte Weise welcher Entwurf zu einer langfristig erweiterbaren, robusten Lösung führt und so häufig auftretende Fehler vermeidet. Besonders im Bereich der Mensch-Computer-Interaktion haben Entwurfsmuster inzwischen eine große Verbreitung erreicht [Wik10, Stichwort: „Entwurfsmuster“], was nicht zuletzt in der Veröffentlichung des Buchs [GHJV95] begründet liegt.

Entwurfsmuster

Um bei der Entwicklung der in dieser Arbeit vorgestellten MMS häufige Entwurfsfehler zu vermeiden, wurde soweit möglich auf Entwurfsmuster zurückgegriffen. Die wichtigsten dieser Entwurfsmuster werden im vorliegend Abschnitt zusammenfassend beschreiben. Soweit nicht anders angegeben, basieren die angegebenen Beispiele und Notationen auf [GHJV95].

Proxy

Proxy

Das Proxy Entwurfsmuster gehört zur Gruppe der Strukturmuster und dient der Kapselung eines Objektes durch ein Stellvertreterobjekt. Sämtliche Zugriffe auf das als konkretes Subjekt bezeichnete Objekt werden über den Stellvertreter, also den Proxy, delegiert. Abbildung 4.1 zeigt das UML-Diagramm dieses Entwurfsmusters.

Über das Subjekt wird in Form einer abstrakten Klasse die Schnittstelle zum konkreten Subjekt festgelegt. Sowohl der Proxy als auch das konkrete Subjekt implementieren diese Schnittstelle. Der Klient führt seine Zugriffe über das Proxy-Objekt durch, welches jeden Aufruf umsetzt und an das konkrete Subjekt weiterleitet.

Dieses Entwurfsmuster kann unter anderem dazu eingesetzt werden, Zugriffe von mehreren Klienten auf ein komplexes Objekt zu delegieren. Dabei wird für jeden Klient ein Proxy instanziiert. Diese Proxies leiten die Aufrufe dann an das komplexe Objekt weiter. Ein

[3] **Ex**tensible **M**arkup **L**anguage

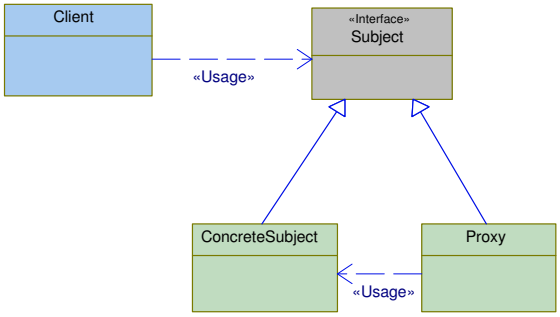


Abbildung 4.1: *Beim Proxy Entwurfsmuster delegiert ein Stellvertreterobjekt (Proxy) Methodenaufrufe an ein konkretes Objekt*

ähnlicher Anwendungsfall besteht bei der Realisierung von verteilten Systemen, wo häufig das ausführende Objekt selbst in einem anderen Systemkontext ausgeführt wird als die Klienten. Das heißt, die Klienten und das Zielobjekt existieren auf unterschiedlichen Rechnern und die einzige Verbindungsmöglichkeit zwischen beiden ist ein Netzwerk. Über das Proxy Entwurfsmuster kann die gesamte Netzwerkkommunikation zum Zielobjekt gekapselt werden, so dass die Methodenaufrufe aus Sicht der Klienten wie lokale Methodenaufrufe erfolgen können. Die Netzwerkverbindung wird somit transparent. Dieses Entwurfsmuster wird zum Beispiel in der CORBA Implementation ACE/TAO konsequent eingesetzt.

Fabrikmethode

Das Entwurfsmuster mit dem Namen Fabrikmethode gehört zur Kategorie der Erzeugungsmuster. Es dient der dynamischen Erzeugung von Objektinstanzen durch speziell dafür implementierte Methoden einer Klasse. Diese Methoden werden als Fabrikmethoden bezeichnet. Fabrik

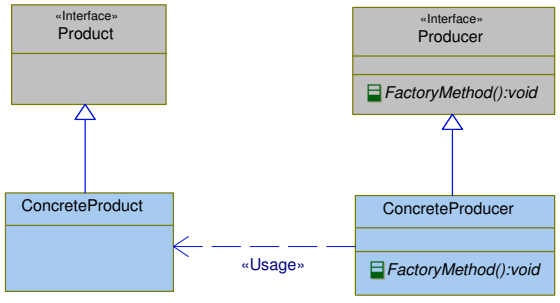


Abbildung 4.2: *Das Entwurfsmuster Fabrikmethode stellt zwei abstrakte Schnittstellen für Erzeuger und Produkt zur Verfügung, dabei werden konkrete Produkte von konkreten Erzeugern produziert*

Innerhalb der abstrakten Schnittstelle `Producer` wird die Fabrikmethode rein virtuell deklariert, so dass sie in der konkreten Erzeuger-Klasse `ConcreteProducer` implementiert werden muss. In dieser Implementation erfolgt die Instanziierung des konkreten Produkts `ConcreteProduct`, was zu einer Abhängigkeit zwischen den beteiligten Klassen `ConcreteProduct` und `ConcreteFactory` führt. Abbildung 4.2 zeigt das Modell dieses Entwurfsmusters.

Durch diese Vorgehensweise ist es möglich, die Aufrufer vollständig von den konkreten Implementierungen der Produkte zu entkoppeln. Der Zugriff erfolgt über die Schnittstellen, die durch die Basisklassen festgelegt werden.

Erweiterung durch Plug-In

Für Plug-Ins[4] wird in [GHJV95] kein Entwurfsmuster vorgestellt, was darin begründet liegt, dass es eine Vielzahl von Ansätzen gibt, um Rahmenwerke für Erweiterungen zu realisieren. Viele dieser Ansätze setzen sich aus einer Kombination unterschiedlicher Entwurfsmuster zusammen. Das Ziel ist jedoch in allen Fällen ähnlich: es geht darum,

Erweiterung

eine Anwendung erweiterbar zu gestalten. Dazu wird ein Rahmenwerk geschaffen, das die Einbindung von Erweiterungen zur Laufzeit ermöglicht und Schnittstellen für die Kommunikation zur Laufzeit festlegt. Eine sehr einfache Vorgehensweise dazu wird in [MMS03] vorgestellt. Diese basiert auf dem zuvor vorgestellten Entwurfsmuster Fabrikmethode (siehe Abschnitt 4.1.1). Einen abstrakten Überblick über im Einsatz befindliche Methodiken zur Realisierung von Erweiterungen gibt [Mar99]. Einige der dort festgelegten Definitionen werden auch im Rahmen dieser Arbeit Verwendung finden.

Eine Anwendung, die durch Erweiterungen um Funktionalität ergänzt werden kann, wird als „Framework-Providing Application“ bezeichnet. Sie ist folgendermaßen definiert.

Definition 4.1. Die „Framework-Providing Application“ bietet den Erweiterungen über Schnittstellen Zugriff auf domänenspezifische Funktionalitäten. Darüber hinaus legt sie den Kontextvertrag fest.

Der Begriff „Framework-Providing Application“ wird im Rahmen dieser Arbeit als Rahmenanwendung weiter geführt. Die Verbindung zwischen einer Erweiterung und der Rahmenanwendung wird über den als solchen bezeichneten Kontextvertrag geregelt.

Definition 4.2. Der „Plugin Contract“ (oder auch Kontextvertrag) ist der Vertrag zwischen einer Erweiterung und der Rahmenanwendung, welcher die beidseitigen Schnittstellen und wenn notwendig, die Protokolle zum Datenaustausch festlegt.

[4] In den folgenden Abschnitten wird der im deutschen Sprachraum übliche Ausdruck Erweiterung für Plug-In verwendet.

Beobachter

Der Beobachter beschreibt ein Entwurfsmuster, das den Transport von Informationen zwischen Objekten realisiert, ohne dass eine Abhängigkeit zwischen den beteiligten Objekten besteht. Das Quellobjekt (eine Instanz der Klasse `Subject`) muss lediglich die gemeinsame Schnittstelle aller Beobachter (die abstrakte Klasse `Observer`) kennen. Alle konkreten Beobachter (von `Observer` abgeleitete und instanziierte Klassen) registrieren sich beim Quellobjekt und werden von diesem über Änderungen der internen Daten informiert, so dass diese Daten gelesen und verarbeitet werden können. Abbildung 4.3 zeigt ein UML-Diagramm des Beobachter-Entwurfsmusters.

Beobachter

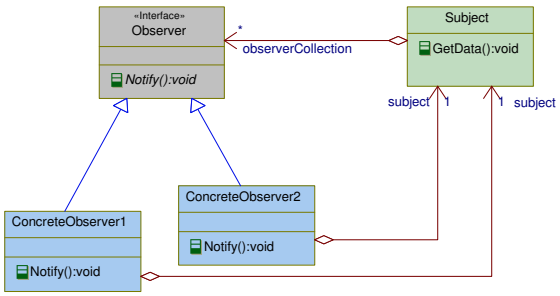


Abbildung 4.3: *Beim Beobachter-Entwurfsmuster werden Beobachter über die Änderung von Daten eines Subjektes informiert, ohne das dieses seine Beobachter kennt*

In der weit verbreiteten und auch intensiv in der MASSiVE-Architektur eingesetzten Bibliothek Qt wurde dieses Entwurfsmuster zur Realisierung der Kommunikation über „Signals“ und „Slots“ implementiert. Die Beobachter definieren dabei durch so genannte „Slots“ die Signatur von Methoden, über die sie Nachrichten erhalten können. Das Quellobjekt definiert im Gegenzug so genannte „Signals“, mittels derer der Informationsaustausch initiiert wird. Zur Laufzeit einer Anwendung werden dann die „Signals“ mit „Slots“ der gleichen Signatur verbunden, um den Nachrichtenaustausch zu ermöglichen.

Qt

Die notwendigen Implementationen der „Signals“ werden dabei in Qt durch den extern aufgerufenen MOC[5] erzeugt. Die „Slots“ werden direkt über den Programmierer als Methoden der Beobachter implementiert (eine detaillierte Beschreibung der Zusammenhänge ist [Nok10] zu entnehmen). Alternative Implementationen dieses Entwurfsmusters nutzen zur Realisation der gleichen Funktionalität das C++ Template-System.

Zuständigkeitskette

Die Zuständigkeitskette ist ein Entwurfsmuster, welches die Entkopplung des Initiators einer Anfrage von den Bearbeitern der Anfrage ermöglicht. Der Initiator besitzt keine

[5] **M**eta-**O**bject **C**ompiler

Zuständigkeitskette

Kenntnis über den Bearbeiter und umgekehrt. Die Auswahl eines konkreten Bearbeiters erfolgt erst zur Laufzeit durch die Weiterleitung einer eingehenden Anfrage an die konkreten Bearbeiter. Nur wenn ein Bearbeiter in der Lage ist eine Anfrage zu bearbeiten, nimmt er diese entgegen. Ansonsten erfolgt die Weiterleitung zum nächsten Bearbeiter, so lange bis ein konkreter Bearbeiter die Anfrage bearbeitet oder sie bereits an alle weitergeleitet wurde.

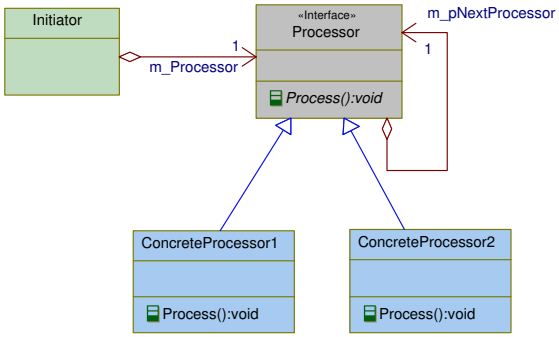


Abbildung 4.4: *Beim Entwurfsmuster Zuständigkeitskette werden Initiator und Bearbeiter einer Anfrage voneinander entkoppelt*

Beim Instanziieren der konkreten Bearbeiter ist es notwendig, die Zeiger auf den Nachfolger korrekt zu initialisieren. In der Regel erfolgt das über Zugriffsmethoden, die in Abbildung 4.4 aus Gründen der Übersichtlichkeit weggelassen wurden.

4.1.2 XML

Metasprache XML

XML ist eine Metasprache, mit deren Hilfe andere Sprachen definiert werden können. Im einfachsten Fall sind das Sprachen, um Textdaten strukturiert darzustellen. Alle von XML abgeleitete Sprachen, auch als XML-Dialekte bezeichnet, sind dabei plattformübergreifend und implementationsunabhängig einsetzbar. Der besondere Vorteil der Darstellung von Daten in einem XML-basierten Format liegt darin, dass die Daten in dieser Form sowohl menschenlesbar als auch maschinenlesbar sind und darüber hinaus umfangreiche Möglichkeiten der Validierung existieren. Eine grundlegende Einführung zu XML gibt [Min04].

Standardisierung

XML-Sprachen sind beim Austausch von Daten über Netzwerke wie das Internet (hier in Form von XHTML[6] als XML-Dialekt) sehr weit verbreitet, werden aber auch in anderen Bereichen immer häufiger eingesetzt. Seit der Standardisierung von XML durch das W3C[7] sind neben vielen als Standard etablierten XML-Dialekten auch viele Editoren entstanden, die die Arbeit mit XML erleichtern. Im Rahmen der vorliegenden

[6] **E**xtensible **H**yper**T**ext **M**arkup **L**anguage
[7] **W**orld **W**ide **W**eb **C**onsortium

Arbeit kommt in diesem Zusammenhang das Programm „oXygen“ zum Einsatz (siehe [Syn10]).

Bei der Arbeit mit Dokumenten, die in einem XML-Dialekt vorliegen, gilt es einige Begrifflichkeiten festzulegen. Die folgenden Definitionen basieren auf [Wik10, Stichwort: „XML“].

Definition 4.3. Ein Dokument, das als XML-Dialekt verfasst ist, wird als XML-Instanz bezeichnet.

Der Aufbau einer XML-Instanz setzt sich aus unterschiedlichen Strukturelementen zusammen. Diese Strukturelemente sind folgendermaßen definiert.

Definition 4.4. Die Strukturelemente einer XML-Instanz sind Elemente (engl. „tags“) und Attribute (engl. „attributes“). Alle Attribute sind Elementen zugeordnet. Ihnen können ebenso Werte zugewiesen werden wie den Elementen. Das äußerste Element der Struktur wird als Wurzelelement (engl. „root element“) bezeichnet.

```
<element attribute="value of attribute">
  value of element
</element>
```

Um zu beschreiben, ob eine XML-Instanz alle Regeln für XML-Dialekte einhält, wird der Begriff der Wohlgeformtheit eingeführt.

Definition 4.5. Eine XML-Instanz wird als wohlgeformt bezeichnet, wenn sie alle der folgenden XML-Regeln einhält:

- Eine XML-Instanz besitzt genau ein Wurzelelement.
- Alle Elemente mit Inhalt besitzen sowohl ein Start- als auch ein Endelement: `<element>contents</element>`.
- Elemente ohne Inhalt benötigen kein Endelement, müssen aber in sich geschlossen sein: `<element />`.
- Alle Elemente und deren Unterelemente sind paarig verschachtelt: bevor ein Elternelement geschlossen werden kann, müssen zuvor alle Kindelemente in umgekehrter Reihenfolge geschlossen werden.

  ```
  <rootElement>
    <childElement1>
      <childElement2>
      </childElement2>
    </childElement1>
  </rootElement>
  ```

- Ein Element darf nicht mehr als ein Attribut mit dem gleichen Namen besitzen.

Die einleitend beschriebenen Validierungsmöglichkeiten gehen weit über den Begriff der Wohlgeformtheit hinaus. Mit diesen lassen sich eigene XML-Dialekte beschreiben und automatisiert validieren.

Definition 4.6. Eine XML-Instanz ist gültig, wenn sie wohlgeformt ist, eine Referenz auf eine Grammatik enthält und erfolgreich gegen diese zuvor festgelegte Grammatik validiert.

Validierung

Der Vorgang der Validierung wird von separaten Anwendungen oder Bibliotheken, die sich in eigene Projekte integrieren lassen, durchgeführt. Dabei wird die zur Validierung benötigte Grammatik zum Beispiel in Form einer DTD[8] oder in Form eines XML-Schemas vorgegeben. Die zuletzt genannte Möglichkeit der Grammatikfestlegung wird in den folgenden Abschnitten vorgestellt. Darüber hinaus erfolgt die Beschreibung von XML-Namensräumen, XML-Schematron, Abfragesprachen und Transformationsmöglichkeiten für XML-Instanzen.

XML-Namensräume

Über XML-Namensräume (engl. „XML namespaces“) ist es möglich mehrere XML-Dialekte innerhalb einer XML-Instanz zu mischen. Sie werden als Identifikation für alle in einer XML-Instanz verwendeten Elemente eingesetzt, um diese selbst bei gleichem Elementnamen eindeutig dem ursprünglichen XML-Dialekt zuordnen zu können.

URI als Namensraum

Damit bei der Definition von Namensräumen sichergestellt ist, dass diese weltweit eindeutig sind (was besonders wichtig beim Datenaustausch über Netzwerke wie das Internet ist), werden sie als URI[9], also in der Regel als Web-Adresse dargestellt. Dabei handelt es sich nicht um eine Notwendigkeit. Grundsätzlich können beliebige Zeichenketten als Namensraum verwendet werden. Letzteres führt unweigerlich zu der Tatsache, dass eine als Namensraum verwendete Web-Adresse nicht zwangsläufig existieren muss.

Die Angabe eines Namensraumes für ein Element erfolgt direkt über das optionale Attribut `xmlns`. Beispiel 4.1 zeigt zwei Beispiele für die Angabe des Namensraumes auf diese Weise.

Beispiel 4.1. *Ein Beispiel für direkt spezifizierte Namensräume in Elementen.*

```
1 <rootElement xmlns="http://www.iat.uni-bremen.de/grammar1">
2   ... elements of grammar 1
3
4   <element xmlns="http://www.iat.uni-bremen.de/grammar2">
5     ... elements of grammar 2
6   </element>
7 </rootElement>
```

[8] **D**ocument **T**ype **D**efinition
[9] **U**niform **R**esource **I**dentifier

Da die Verwendung von direkt in den Elementen angegebenen Namensräumen die Lesbarkeit von umfangreichen XML-Instanzen deutlich erschweren würde, wird häufig lediglich mit einem lokal in der XML-Instanz definierten Präfix, der aus einer kurzen Zeichenkette ohne Sonderzeichen und Leerzeichen besteht, gearbeitet. Dieses Präfix wird jedem Elementnamen, der dem entsprechenden Namensraum angehört, vorangestellt und mittels Doppelpunkt abgetrennt. Beispiel 4.2 veranschaulicht dieses Vorgehen.

Beispiel 4.2. *Ein Beispiel für die Verwendung eines Namensraums mittels Präfix.*

```
<rootElement xmlns="http://www.iat.uni-bremen.de/grammar1"
    xmlns:g2="http://www.iat.uni-bremen.de/grammar2">
  ... elements of grammar 1

  <g2:element>
    ...
  </g2:element>
</rootElement>
```

XML-Schemasprache

Mit XML-Schema existiert eine durch das W3C standardisierte Möglichkeit die Grammatik eines XML-Dialektes festzulegen. Das XML-Schema (abgekürzt XSD[10]) einer Grammatik wird dabei als XML-Instanz erstellt. Die Überprüfung der Validität erfolgt dann durch Import der vom W3C standardisierten Schemasprache und Übergabe der Instanz an einen XML-Parser wie Xerces-C++ (siehe [The11]). XML-Schema

Im folgenden werden die für das Verständnis der in Kapitel 5 und Kapitel 6 entwickelten Grammatiken benötigten Elemente der XML-Schemasprache vorgestellt. In den dazu gegebenen Beispielen wird als Namensraumkürzel `xs`[11] verwendet. Des Weiteren erfolgt die Darstellung der grafischen Korrespondenzen zu den XSD-Elementen, wie Sie in dem für die vorliegende Arbeit verwendeten XML Editor „oXygen“ verwendet werden. In den späteren Abschnitten werden dann bei der Definition von eigenen XML-Dialekten lediglich die grafischen Abbildungen anstelle der Textform verwendet. oXygen

Import von externen Grammatiken Um die zuvor beschriebenen Möglichkeiten der Namensräume besonders in Bezug auf das Einbinden von externen Grammatiken zu realisieren, bedarf es der Möglichkeit diese in eigene Grammatiken zu importieren. Zu diesem Zweck wird von der XML-Schemasprache das `import` Element definiert.

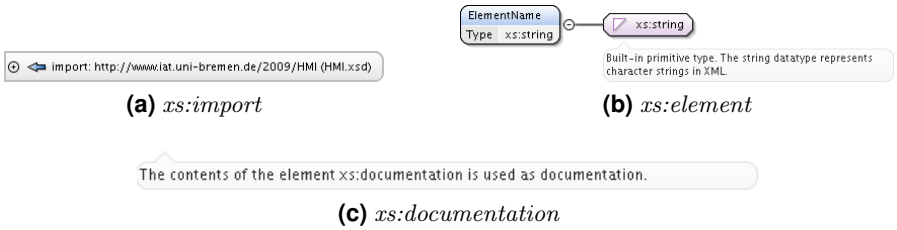
Beispiel 4.3. *Ein Beispiel für den Import eines externen XML Schemas.*

```
<xs:import namespace="http://www.iat.uni-bremen.de/2009/HMI"
           schemaLocation="HMI.xsd"/>
```

[10] **X**ML **S**chema **D**efinition
[11] XML-Namensraum `xs` entspricht „http://www.w3.org/2001/XMLSchema“

Import Namensraum

Über das Attribut `namespace` wird der Quellnamensraum des importierten XML-Schema angegeben und über das Attribut `schemaLocation` die externe XSD-Instanz. Abbildung 4.5a zeigt die grafische Korrespondenz zu diesem Element.



(a) *xs:import* **(b)** *xs:element*

(c) *xs:documentation*

Abbildung 4.5: *Grafische Repräsentation einiger XSD-Elemente*

Elemente

Spezifikation von Elementen Die Spezifikation von Elementen erfolgt in XML-Schema über das XSD-Element `xs:element`. Dabei wird über das Attribut `name` der Name des Elements in den später realisierten XML-Instanzen festgelegt. Das weitere Attribut `type` legt den Datentyp fest. Dabei kann sowohl die Angabe eines standard XSD-Datentyps als auch die Angabe eines benutzerdefinierten Datentyps erfolgen (mehr dazu in den folgenden Abschnitten).

Beispiel 4.4. *Beispiel zur Spezifikation von Elementen.*

```
1 <xs:element name="ElementName"
2             type="elementType"/>
```

Die grafische Korrespondenz zur Spezifikation von Elementen für XML-Dialekte zeigt Abbildung 4.5b.

Dokumentation

Anmerkungen und Dokumentation Um die Lesbarkeit von XML-Schemasprachen zu verbessern, sieht die XSD-Spezifikation der W3C Kommentarelemente vor. Diese können freien Text beinhalten und dienen der Beschreibung der Grammatik oder den in den folgenden Abschnitten vorgestellten Definitionen von eigenen Datentypen. Darüber hinaus haben die Elemente keine weitere Bedeutung.

Beispiel 4.5. *Ein Beispiel für Dokumentationselemente in XML Schemata.*

```
1 <xs:annotation>
2   <xs:documentation>
3     The content of the element xs:documentation is
4      used as documentation.
5   </xs:documentation>
6 </xs:annotation>
```

Anmerkungen und Dokumentation werden in ihrer grafischen Korrespondenz als Sprechblase dargestellt, wie Abbildung 4.5c veranschaulicht.

Restriktionen für Elemente oder Attribute Über Restriktionen kann die Menge der möglichen, für ein Element oder Attribut angegebenen Werte, eingeschränkt werden. Restriktionen basieren dabei immer auf XSD Standarddatentypen. Die sich aus einer Restriktion ergebende Menge an möglichen Werten für einen der Standarddatentypen wird als Facette (engl. facet) bezeichnet.

Eine Möglichkeit der Spezifikation einer Restriktion erfolgt über das Element `pattern` und einen regulären Ausdruck (siehe [Wik10, Stichwort: „Regulärer Ausdruck“]). Beispiel 4.6 zeigt eine solche Restriktion, die eine Standard-Zeichenkette derart beschränkt, dass nur noch Buchstaben (in Groß- und Kleinschreibung), die Ziffern „0“ bis „9“ sowie Unterstriche in beliebiger Anzahl (mindestens jedoch ein Zeichen) möglich sind.

reguläre Ausdrücke

Beispiel 4.6. *Beispiel einer Restriktion mittels Muster basierend auf dem Standarddatentyp* `xs:string`.

```
<xs:restriction base="xs:string">
  <xs:pattern value="([a-zA-Z0-9_])+"/>
</xs:restriction>
```

Eine weitere Möglichkeit, die ebenso eine Restriktion des Standarddatentyps `xs:string` darstellt, ist die Festlegung von gültigen Aufzählungselementen mittels `xs:enumeration`. Dazu wird die Menge der gültigen Zeichenketten exakt vorgegeben. Beispiel 4.7 veranschaulicht diese Möglichkeit.

Aufzählungen

Beispiel 4.7. *Beispiel einer Aufzählung basierend auf dem Standarddatentyp* `xs:string`.

```
<xs:restriction base="xs:string">
  <xs:enumeration value="Value1"/>
  <xs:enumeration value="Value2"/>
  <xs:enumeration value="Value3"/>
</xs:restriction>
```

Weder Restriktionen für Elemente noch Restriktionen für Attribute besitzen eine grafische Korrespondenz im verwendeten XML-Editor „oXygen“.

Spezifikation von Sequenzen Sequenzen sind XML-Elemente, die ganz bestimmte Kindelemente in einer genau festgelegten Reihenfolge enthalten. Zur Spezifikation einer solchen Sequenz wird das XSD-Element `xs:sequence` verwendet. Innerhalb dieses Elementes werden alle Kindelemente in genau der Reihenfolgen definiert, in der sie später in den XML-Instanzen erwartet werden. Besonders interessant in Verbindung mit Sequenzen ist die Möglichkeit der Festlegung von Häufigkeiten über die Attribute `minOccurs` und `maxOccurs`.

Sequenzen

Beispiel 4.8. *Beispiel einer Sequenz von Elementen.*

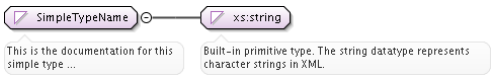
```
<xs:sequence>
  <xs:element name="ChildElement1"
              type="childElementType1"/>
```

```
4     <xs:element name="ChildElement2"
5                 type="childElementType2"
6                 minOccurs="0"
7                 maxOccurs="unbounded" />
8   </xs:sequence>
```

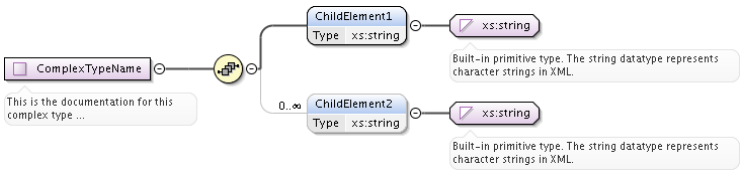
Sequenzen werden in ihrer grafischen Korrespondenz als Baum dargestellt. Dabei erfolgt die Darstellung in der Reihenfolge der Elemente von oben nach unten. Die Kanten von notwendigen Elementen werden hervorgehoben und die von optionalen werden schwach gezeichnet, sowie mit der Multiplizität versehen wie Abbildung 4.6b zeigt.

Spezifikation von Datentypen Bei benutzerdefinierten Datentypen ist grundsätzlich zwischen zwei Arten zu Unterscheiden, den einfachen Typen (engl. „simple type“, siehe Abbildung 4.6a) und den komplexen Typen (engl. „complex type“, siehe Abbildung 4.6b). Bei ersteren handelt es sich um Typen, die auf XSD-Standardtypen basieren. Sie werden als Restriktion auf die standardmäßig für XSD-definierten Typen spezifiziert und enthalten keine Kindelemente oder Attribute. Die komplexen Typen können dagegen weitere Elemente und Attribute enthalten, wie Abbildung 4.6 zeigt.

benutzerdefinierte Datentypen



(a) *simpleType*



(b) *complexType*

Abbildung 4.6: *Grafische Repräsentation einiger XSD-Elemente*

einfache Datentypen

Beispiel 4.9 demonstriert die XSD-Beschreibung eines einfachen Typs, der als Restriktion aus dem Standard Datentyp `xs:string` hervorgeht, inklusive der bereits beschriebenen Möglichkeit zur Elementdokumentation über das Element `xs:documentation`.

Beispiel 4.9. *Beispiel eines einfachen Datentyps basierend auf dem Standarddatentyp* `xs:string`.

```
1   <xs:simpleType name="SimpleTypeName">
2     <xs:annotation>
3       <xs:documentation>
4         This is the documentation for this simple type ...
5       </xs:documentation>
6     </xs:annotation>
7     <xs:restriction base="xs:string">
```

```
8       <xs:pattern value="([a-zA-Z0-9_])+"/>
9     </xs:restriction>
10  </xs:simpleType>
```

Mit einer Sequenz als Hauptinhalt zeigt Beispiel 4.10 die Spezifikation eines komplexen Datentyps. In diesem Fall ist die Häufigkeit der Kindelemente vom Typ `ChildElement2` innerhalb des neu definierten Datentyps `ComplexTypeName` unbegrenzt.

komplexe Datentypen

Beispiel 4.10. *Beispiel eines komplexen Datentyps.*

```
1   <xs:complexType name="ComplexTypeName">
2     <xs:annotation>
3       <xs:documentation>
4         This is the documentation for this complex type ...
5       </xs:documentation>
6     </xs:annotation>
7     <xs:sequence>
8       <xs:element name="ChildElement1"
9                   type="xs:string"/>
10      <xs:element name="ChildElement2"
11                  type="xs:string"
12                  minOccurs="0"
13                  maxOccurs="unbounded" />
14    </xs:sequence>
15  </xs:complexType>
```

In der grafischen Korrespondenz werden die einfachen Datentypen mit einem Dreieck gekennzeichnet (in Abbildung 4.6 an Element `SimpleTypeName` zu sehen) und die komplexen mit einem Quadrat (in Abbildung 4.6 an Element `ComplexTypeName` zu sehen).

Spezifikation von Attributen Attribute können komplexen Datentypen über das XSD-Element `xs:attribute` zugeordnet werden. Bei ihrer Spezifikation muss der Name über das Attribut `name` und der Typ über das Attribut `type` angegeben werden. Beispiel 4.11 zeigt einen komplexen Typ mit zwei Attributen.

Attribute

Beispiel 4.11. *Beispiel zur Spezifikation von Attributen innerhalb eines komplexen Datentyps.*

```
1   <xs:complexType name="ComplexTypeName">
2     <xs:attribute name="attribute1"
3                   type="xs:boolean"
4                   use="required"/>
5     <xs:attribute name="attribute2"
6                   type="xs:boolean"
7                   use="optional"/>
8   </xs:complexType>
```

Zusätzlich zu den oben beschriebenen notwendigen Attributen des Elements `xs:attribute`, kann über das optionale Attribute `use` spezifiziert werden, ob es sich um ein optionales oder notwendiges Attribut handelt. Abbildung 4.7 stellt die grafische Korrespondenz der Definition von Attributen in XML-Editor „oXygen“ dar.

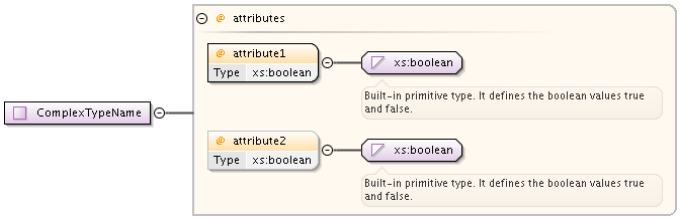


Abbildung 4.7: *Grafische Korrespondenz zu* `xs:attribute`

XPath

Datenextraktion aus XML-Instanzen

XPath ist eine (ebenfalls vom W3C standardisierte) Abfragesprache, mit deren Hilfe Knoten, also Elemente und Attribute beziehungsweise deren Werte, aus XML-Instanzen extrahiert werden können. Dazu erfolgt die Repräsentation der Elemente, Kindelemente und Attribute einer XML-Instanz innerhalb der XPath-Implementation als Baum. Über XPath-Ausdrücke kann innerhalb dieser Baumstruktur navigiert werden, um bestimmte Informationen zu extrahieren. Die Ausdrücke lassen sich dabei in die Gruppe der Pfadausdrücke, der Operanden und Funktionen aufteilen. Die zur Beschreibung angegebenen Beispiele basieren auf der folgenden XML-Instanz:

Beispiel 4.12. *Beispiel einer XML-Instanz.*

```
<RootElement>
  <Child1 attribute1="value1">
    <Child2 attribute1="value1"
            attribute2="value2">Data1</Child2>
    <Child2 attribute1="value1">Data2</Child2>
  </Child1>
</RootElement>
```

Nachfolgend nur die Erläuterungen zu einem Auszug aus der XPath-Notation. Elemente ohne Relevanz für die folgenden Betrachtungen werden nicht beschrieben.

Auswahl

Selektionen Die Navigation im Elementbaum der XML-Instanz erfolgt über Pfadausdrücke. Dabei wird die Sequenz, die den Pfad von der Dokumentwurzel (das Wurzelelement der XML-Instanz) über die Kindelemente zum Zielelement beschreibt, als Zeichenkette angegeben, in der die einzelnen Ebenen durch das Zeichen „/“ getrennt sind. Beginnt die Zeichenkette mit dem Trennzeichen „/“, so wird die nachfolgende Ebene als Wurzel interpretiert, wie in Beispiel 4.13 dargestellt ist.

Beispiel 4.13. *Selektion von Elementen.*

```
1 /RootElement/Child1
2 Child1/Child2
```

In der ersten Zeile wird ausgehend vom Wurzelelement selektiert: es werden alle Kindelemente `Child1` selektiert, die Kinder vom Wurzelelement sind. In Zeile zwei werden dagegen alle Kindelemente `Child2` selektiert, die Kindelemente von `Child1` sind. Dieser Pfadausdruck geht nicht von der Wurzel aus.

Neben der Selektion von Elementen einer XML-Instanz erfolgt die Selektion von Attributen über das Zeichen „@“, gefolgt vom Namen des Attributes mit optionaler Angabe des gesuchten Wertes, wie Beispiel 4.14 veranschaulicht.

Beispiel 4.14. *Selektion von Attributen.*

```
1 /RootElement/Child1[@attribute1]
2 //Child2[@attribute1='value1']
3 //@attribute2
```

Der Ausdruck in der ersten Zeile selektiert ausgehend vom Wurzelelement alle `Child1` Elemente, die ein Attribut `attribut1` besitzen. In der zweiten Zeile kommt eine Abwandlung des zuvor beschriebenen Pfadselektors „/“ zum Einsatz. Wird mit „//“ gearbeitet, so werden alle Elemente selektiert, die mit dem Elementnamen übereinstimmen, unabhängig davon in welcher Ebene der Baumstruktur der XML-Instanz sie liegen. Im vorliegenden Beispiel wird diese Auswahl jedoch auf die Elemente beschränkt, die ein Attribut `attribute1` mit dem Wert „value1“ besitzen. In der letzten Zeile von Beispiel 4.14 werden in Analogie dazu alle Attribute selektiert, deren Name `attribute2` ist.

Operatoren Nach der Auswahl von Elementen, Attributen oder Attributwerten können diese miteinander verknüpft werden. Dazu spezifiziert das W3C weitere Operatoren. Im Rahmen der vorliegenden Arbeit kommen zu diesem Zweck ausschließlich die boole'schen Operatoren `and` und `or` zum Einsatz. Beispiel 4.15 zeigt die Anwendung dieser Operatoren.

Verknüpfungen

Beispiel 4.15. *Boole'sche Operatoren.*

```
1 //Child2[@attribute1 and @attribute2]
2 //Child2[@attribute1 or @attribute2='value2']
```

Die Ergebnismenge des Ausdrucks in der ersten Zeile enthält alle Kindelemente `Child2`, die das Attribut `attribute1` und das Attribut `attribute2` besitzen. Der Ausdruck in der zweiten Zeile selektiert dagegen alle Elemente `Child2`, die entweder das Attribut `attribute1` oder das Attribut `attribute2` mit dem Wert „value2“ besitzen (kein exklusives Oder).

Auswertung, Manipulation

Funktionen Darüber hinaus existieren auch Funktionen, mittels derer die Ergebnismengen der Ausdrücke weiter ausgewertet oder manipuliert werden können. Aus der Menge an möglichen Funktionen kommen hier die boole'schen Funktionen `not()` und `count()` zum Einsatz, wie Beispiel 4.16 zeigt.

Beispiel 4.16. *Funktionen*

```
1 //Child2[@attribute1 and not(@attribute2)]
2 count(//Child2[@attribute1])
```

Mit der Funktion `not()` wird in der ersten Zeile die Existenz des Attributs `attribute2` negiert und die Selektion auf alle Elemente `Child2`, die das Attribut `attribute1` aber nicht das Attribut `attribute2` besitzen, beschränkt. In der zweiten Zeile wird die Größe der Ergebnismenge ermittelt, die durch die Selektion aller Elemente `Child2` mit dem Attribut `attribute1` entsteht.

XQuery

Abfragesprache für XML

Bei XQuery handelt es sich um eine Abfragesprache mit der Daten aus XML-Instanzen erfragt werden können. Dazu werden XPath-Ausdrücke mit XQuery-Funktionen zur einer Abfragesprache ähnlich der SQL für Datenbanken kombiniert.

Iterationen Seine besondere Stärke spielt XQuery aus, wenn es darum geht XML-Instanzen nicht als Ganzes zu verarbeiten, sondern gezielte Informationen aus diesen zu extrahieren. So besteht zum Beispiel die Möglichkeit, durch Schleifenkonstruktionen mittels eines XPath-Ausdrucks über selektierte Elemente zu iterieren, wie Beispiel 4.17 zeigt.

Beispiel 4.17. *Iteration mittels XQuery*

```
1 for $curChild in fn:doc("xmlInstance.xml")/RootElement/Child1
2   return $curChild
```

XQuery mit XPath

Durch die XQuery-Funktion `fn:doc()` wird eine externe XML-Instanz geladen. Über den direkt hinter der Funktion angegebenen XPath-Ausdruck erfolgt die Selektion von Elementen aus der XML-Instanz. Über die Ergebnismenge dieser Selektion wird dann über die `for` Anweisung iteriert. Dabei enthält die Variable `$curChild` (Variablen werden in XPath grundsätzlich mit dem „$“ Zeichen eingeleitet) in jeder Iteration den aktuell bearbeiteten Wert aus der Ergebnismenge. Eine dabei zu beachtende Besonderheit besteht darin, dass die Standard XQuery-Funktionen dem Namensraum `fn` zugeordnet sind. Obwohl die Angabe des Namensraumes optional ist, sollte diese allein aus Gründen der besseren Lesbarkeit immer mit angegeben werden.

Namensräume

Verwendung von Namensräumen Damit der Einsatz von Namensräumen in XQuery-Abfragen möglich wird, müssen diese zuvor mittels `declare namespace` mit einem Kürzel eingeführt werden. Im Anschluss daran kann über das Kürzel auf Elemente aus dem spezifizierten Namensraum zugegriffen werden. In Beispiel 4.18 wird für den Namensraum „http://www.iat.uni-bremen.de/HMI“ das Kürzel „hmi“ eingeführt.

Beispiel 4.18. *Einsatz von Namensräumen*

```
1 declare namespace hmi="http://www.iat.uni-bremen.de/HMI";
```

Konvertierung von Daten

Funktionen Weitere Funktionen, die neben der bereits eingeführten Funktion zum Laden einer externen XML-Instanz für weitergehende Extrahierungs- oder Konvertierungsaufgaben zur Verfügung stehen, sind zum Beispiel `fn:string()`, die die Elemente einer Ergebnismenge in Zeichenketten konvertiert und `fn:data()`, die die Daten eines XML-Elementes oder -Attributs zurückgibt. Beispiel 4.19 veranschaulich diese Funktionen.

Beispiel 4.19. *Konvertierung und Extraktion von Daten*

```
1 for $element in fn:doc("hmi.xml")//Child2
2   return fn:string(fn:data($element))
```

In der ersten Zeile iteriert die `for`-Anweisung über alle Elemente `Child2` in der XML-Instanz und gibt die Daten der in der Ergebnismenge enthaltenen Elemente konvertiert in eine Zeichenkette zurück.

XSL-Transformationen

Transformation von XML Instanzen

Die XSL[12]-Transformationen oder auch XSLT[13] sind im Gegensatz zur Abfragesprache XQuery ebenso als XML-Dialekt verfasst wie XML-Schema. Die XSLT-Instanzen enthalten jedoch Transformationsanweisungen, die eine XML-Eingabeinstanz in ein Ausgabedokument transformieren. Das Ausgabedokument muss dabei nicht zwangsläufig wieder eine XML-Instanz sein, sondern kann ein nahezu beliebiges, nicht binäres Datenformat sein. Häufig werden zum Beispiel Transformationen in das PDF[14]-Format durchgeführt.

Eine Gemeinsamkeit zu XQuery-Ausdrücken besteht bei XSLT darin, dass die Transformationen über XPath-Ausdrücke Daten aus der XML-Eingangsinstanz extrahieren, um sie über XML Vorlagen (engl. „Templates“) in das Ausgabeformat zu wandeln. Die eigentliche Transformation wird dabei von einem XSLT-Prozessor durchgeführt. Es stehen unterschiedliche freie und kommerzielle XSLT-Prozessoren zur Verfügung, wobei im Rahmen der vorliegenden Arbeit Xalan-C von der Apache Stiftung zum Einsatz kommt. XSLT wird dabei lediglich zur Validierung von XML-Instanzen mittels XML-Schematron eingesetzt, welche im folgenden Abschnitt beschrieben wird.

[12] **E**xtensible **S**tylesheet **L**anguage
[13] **E**xtensible **S**tylesheet **L**anguage **T**ransformations
[14] **P**ortable **D**ocument **F**ormat

XML-Schematron

Validierung semantischer Zusammenhänge

Die umfangreichen Spezifikationsmöglichkeiten mittels XML-Schema dienen alle der Festlegung einer Grammatik für XML-Instanzen, also der Definition des syntaktischen Aufbaus. Die Validierung erfolgt dabei durch Bibliotheken oder externe Anwendungen, die einen XML-Parser implementieren. Sollen innere Restriktionen oder Abhängigkeiten wie zum Beispiel semantische Zusammenhänge in einer XML Instanz überprüft werden, so ist das mit XML-Schema nicht möglich.

Um diese Validierungsmöglichkeiten trotzdem zu realisieren, erfolgte durch das W3C die Standardisierung von XML-Schematron. XML-Schematron verfolgt dabei einen anderen Ansatz; es verwendet XPath-Ausdrücke, um Muster zu extrahieren, und konvertiert die zu validierenden XML-Eingabeinstanzen mittels XSLT in das Ausgabeformat SVRL[15], ein XML-Dialekt zur Beschreibung des Validierungsergebnisses.

Validierung durch XSLT Transformation

Dieser Ablauf kann dabei in einen Online- und einen Offline-Bearbeitungsschritt unterteilt werden. Der Offline-Schritt beinhaltet die Erstellung der Schematron-Instanz, welche auch ein XML-Dialekt ist. Ein Auszug aus den Validierungsmöglichkeiten und dem Aufbau der Schematron Instanz folgt später in diesem Abschnitt. Die erstellte XML-Schematron-Instanz und die unter [ISO10] zur Verfügung gestellte Standardimplementation in Form von XSLT-Transformationsvorschriften stellen gemeinsam die Basis für den ersten Transformationsschritt durch den XSLT-Prozessor dar (siehe Abbildung 4.8). Das Ergebnis dieser vereinfacht dargestellten Transformation (weitere Informationen zum genauen Ablauf sind [ISO10] zu entnehmen) ist eine XSLT-Instanz, die die Transformationsvorschrift für die Validierung repräsentiert.

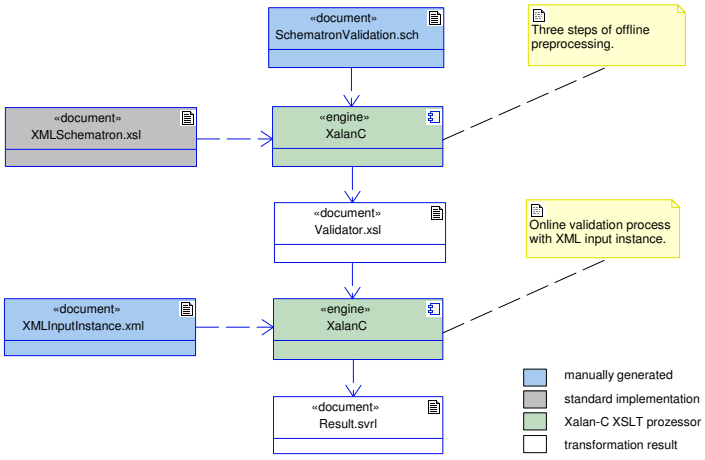


Abbildung 4.8: *Vereinfachter Ablauf der Schematron Validierung*

Innerhalb des Online-Bearbeitungsschrittes erfolgt im Anschluss daran die Transformation der zu validierenden XML-Eingabeinstanzen mit der im Offline-Schritt erzeugten

[15] **S**chematron **V**alidation **R**eport **L**anguage

XSLT-Transformationsvorschrift. Das Ergebnis dieser Transformation ist das Validierungsergebnis in Form einer SVRL-Instanz, welche dann zum Beispiel mittels XQuery auf aufgetretene Fehler und etwaige Fehlermeldungen hin untersucht werden kann.

Im folgenden werden die Grundlagen der Schematron-Spezifikation vorgestellt. Eine vollständige Übersicht ist [ISO10] zu entnehmen.

Globale Dokumentstruktur XML-Schematron-Instanzen, also die Basis für die Erstellung der validierenden XSLT-Transformationsvorschrift, basieren auf Elementen, die im Namensraum „http://purl.oclc.org/dsdl/schematron“ definiert sind. Dazu wird dieser Namensraum zunächst über das bereits bekannte Attribut `xmlns` und unter dem Kürzel „sch“ eingeführt. Es folgt die optionale Festlegung des Titels der XML-Schematron-Instanz über das Element `sch:title` und die Festlegung weiterer Quellnamensräume aus dem zu validierenden XML-Dialekt über das XML-Schematron-Element `sch:ns`.

Schematron Instanz

Beispiel 4.20. *Globale Dokumentstruktur einer XML Schematron Instanz*

```
<sch:schema xmlns:sch="http://purl.oclc.org/dsdl/schematron">
  <sch:title>Schematron title ...</sch:title>
  <sch:ns prefix="hmi"
          uri="http://www.iat.uni-bremen.de/HMI" />

  <sch:pattern name="Pattern name ...">
  </sch:pattern>
</sch:schema>
```

Im Anschluss daran erfolgt die Spezifikation der zu validierenden Muster über das Element `sch:pattern`. Von diesem Element können eines oder beliebig viele in einer XML-Schematron-Instanz angegeben werden. Dabei ist es möglich, die einzelnen Muster über die Angabe eines optionalen Namens (angegeben über das Attribut `name`) zu unterscheiden.

Regelspezifikation und Kontext Im Element `sch:pattern` der zuvor beschriebenen Musterspezifikation erfolgt die Angabe der Regeln und des Regelkontextes, die das Muster beschreiben. Der Regelkontext wird über einen XPath-Ausdruck festgelegt.

Regeln

Beispiel 4.21. *Regelspezifikation und Regelkontext.*

```
<sch:rule context="//Child1">
</sch:rule>
```

Die Regel in Beispiel 4.21 ist nur im Kontext aller in der XML-Instanz angegebenen Elemente `Child1` gültig.

Test und Fehlermeldung Die Spezifikation der durchzuführenden Tests kann auf zwei Arten erfolgen. Die erste ist die Formulierung einer Zusicherung in Form eines XPath-Ausdrucks mit dem Attribut `test` des XML-Schematron-Elements `sch:assert`. Sollte dieser Test fehlschlagen, so wird der im Element angegebene Text als Fehlermeldung ausgegeben.

Tests formulieren

Die zweite Variante dreht dieses Verhalten um. Über das Attribut `test` des XML-Schematron-Elements `sch:report` lässt sich eine Zusicherung formulieren, deren Text im SVRL-Ergebnisbericht erscheint, wenn die Zusicherung erfüllt ist, wie Beispiel 4.22 veranschaulicht.

Beispiel 4.22. *Report und Assert zur Formulierung von Zusicherungen.*

```
<sch:assert test="not(@Child1)
                  and @Child2='value2'">
  If attribute Child1 is not present
  attribute Child2 has to be equal to 'value2'!
</sch:assert>
<sch:report test="@Child2='value2'">
  Child2 is equal to 'value2'!
</sch:report>
```

Sowohl das Element `sch:assert` als auch das Element `sch:report` sind als Kindelemente einer zuvor spezifizierten Regel innerhalb des `sch:rule` Elementes einzusetzen.

4.2 Erkenntnisgewinnung aus Datenbeständen - KDD

Das Ziel von KDD besteht darin, Wissen in großen Datenbeständen zu erschließen. Diese Datenbestände können zum Beispiel Transaktionsdaten von Handelsvorgängen oder auch Meldungslisten von Alarm- oder Softwaresystemen sein (siehe Abschnitt 2.4).

Rapid-Miner

Die folgenden Abschnitte geben eine Einführung in das Thema und beschreiben zunächst einige Notationen für den Einsatz des KDD-Werkzeugs „Rapid-Miner“ ([Rap10]), welches zur Vorabanalyse der aufgenommenen Datensätze eingesetzt wird. Im Anschluss daran wird die eingesetzte mathematische Notation für die Beschreibung von Ereignissequenzen beschrieben, um abschließend den eingesetzten Basisalgorithmus vorzustellen.

4.2.1 Hintergrund

Data-Mining

KDD wird in der Literatur überwiegend mit Methoden der Analyse des Konsumverhaltens der Kunden von Handelsunternehmen in Verbindung gebracht. Daher beschreiben viele Veröffentlichungen zum Data-Mining Hintergründe aus diesem Bereich. Dabei ist es das Ziel unterschiedliche Eigenschaften von Kunden zu Kundengruppen zusammenzufassen (engl. „clustering“). Es werden Eigenschaften eines Kunden wie persönliche

Merkmale (zum Beispiel Angaben zu Name, Geburtsdatum und Wohnort) mit dem Konsumverhalten (die bisherigen Einkäufe) in Beziehung gesetzt. Daraus lassen sich abschließend Regeln wie „Kunden die Produkt *A* gekauft haben, kaufen häufig auch Produkt *B*“ erzeugen und zur individuellen Werbung einsetzen.

Ein ähnliches Ziel verfolgen die Entwickler mit KDD im Bereich von Sicherheits- und Analysesystemen für technische Anlagen. Häufig werden hier textbasierte Meldungslisten verwendet, um Fehler und Problemzustände zu protokollieren. Diese Protokolle bestehen in der Regel aus einer Meldung pro Zeile mit Angabe eines Zeitstempels, der Meldungsart und einer Beschreibung. Die Auswertung dieser Daten kann Rückschlüsse auf häufig auftretende Probleme und deren Zusammenhänge untereinander geben.

4.2.2 Grundbegriffe

Für den Einsatz des KDD Werkzeugs Rapid-Miner ist es notwendig einige Begriffe zu definieren, die später im Verlauf der vorliegenden Arbeit wieder aufgegriffen werden. Die folgenden Definitionen entsprechen der in [Rap10] festgelegten Notation:

- Die Gesamtheit aller zur Verfügung stehender Daten wird als **Beispielmenge** (engl. „Example Set“) bezeichnet und entspricht im Kontext der vorliegenden Arbeit dem in Abschnitt 2.4.1 eingeführten Begriff Datenbasis. Beispielmenge

- **Attribute** beschreiben die Merkmale oder Eigenschaften einer beobachteten Situation. In anderer Literatur oder Sprachen werden auch die folgenden Begriffe verwendet: Eigenschaft, Feature, Einflussfaktor, Indikator, Variable oder Signal. Attribute

- Ein **Zielattribut** oder auch „Label“ ist ein besonderes Attribut, das ein Merkmal beschreibt, das nicht für alle Beispiele der Beispielmenge zur Verfügung steht. Das Ziel des Data-Mining ist es zu bestimmen, mit welcher Wahrscheinlichkeit ein Zielattribut bestimmte Werte annimmt. Oft wird auch von der Zielvariablen gesprochen. Zielattribut

- Über das **Konzept** wird festgelegt, welche Attribute eine beobachtete Situation beschreiben. Das Ziel ist es dabei nicht die Situation vollständig oder wenn anwendbar physikalisch korrekt zu beschreiben, sondern die Menge der für das Data-Mining zur Verfügung stehenden Attribute und des unbekannten Zielattributes zu spezifizieren. Konzept

- Jedes Objekt der Beispielmenge beschreibt ein **Beispiel** des zuvor festgelegten Konzeptes. Beispiele werden im Kontext der vorliegenden Arbeit auch als Datensätze bezeichnet. Beispiel

- Attribute können unterschiedliche **Rollen** einnehmen. Ein Zielattribut nimmt zum Beispiel die Rolle eines Labels ein. Weitere mögliche Rollen sind **Weight** (Gewicht) Rollen

und **ID** (Identifizierung). Alle Attribute ohne spezielle Rollen werden als **reguläre** Attribute bezeichnet.

Datentypen

- Neben unterschiedlichen Rollen besitzen Attribute unterschiedliche **Datentypen**. Diese unterscheiden, ob es sich bei dem Attribut zum Beispiel um eine Zahl, einen Freitext oder einen nominellen Wert handelt.

Metadaten

- Die Eigenschaften eines Konzeptes werden als **Metadaten** bezeichnet. Sie beschreiben die Gesamtheit aller Informationen, die ein Konzept definieren. Man kann dabei auch von Daten über Daten sprechen.

4.2.3 Mathematische Beschreibung von Ereignissequenzen

Ereignisse-quenzen

Zur Beschreibung des später eingeführten Algorithmus ist es notwendig, eine eindeutige mathematische Notation festzulegen. In der Literatur wird häufig die Notation nach [MTV97] zitiert, beziehungsweise als Basis für die Beschreibung von entwickelten Algorithmen eingesetzt, welche auch im Rahmen der vorliegenden Arbeit Anwendung finden wird.

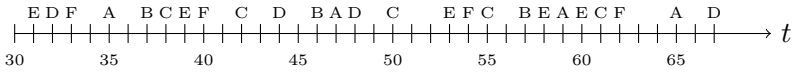


Abbildung 4.9: *Beispieldaten für die Beschreibung der Notation*

Um eine anschauliche Erklärung zu erreichen, führen die in Abbildung 4.9 dargestellten Beispieldaten in mehreren Beispielen durch die gesamte Beschreibung. Die untere Beschriftung der Achse ist der Zeitstempel, die oberen Beschriftungen stehen für Ereignisse zu einem bestimmten Zeitpunkt t.

Ereignisse

Ereignisse

Ereignisse sind das Kernelement sämtlicher Betrachtungen. Sie beschreiben zeitliche Abläufe und setzen sich aus einem Ereignistyp und einem Zeitstempel zusammen. Der Ereignistyp gehört zu einer Menge von möglichen Ereignistypen, die im Vorfeld festgelegt wird. Die Zeitstempel besitzen eine konstante zeitliche Auflösung, welche ebenso vor Beginn von Berechnungen festgelegt wird.

Definition 4.7. Ein Ereignis e ist ein Paar bestehend aus Ereignistyp $A \in \mathbb{E}$ und einem ganzzahligen Wert t, welcher den Zeitpunkt beschreibt, an dem e eingetreten ist. Dabei ist $\mathbb{E}$ die Menge von möglichen Ereignistypen.

$$e = (A, t)$$

Den Beispieldaten aus Abbildung 4.9 liegt $\mathbb{E} = \{'A', 'B', 'C', 'D', 'E', 'F'\}$ als Menge möglicher Ereignistypen zugrunde. Die zwei Beispielereignisse e_4 und e_{10} traten zu den Zeitpunkten $t = 35$ beziehungsweise $t = 44$ ein.

Beispiel 4.23.

$$e_4 = ('A', 35)$$
$$e_{10} = ('D', 44)$$

Die mathematische Beschreibung einer Menge von Ereignissen $\mathbb{S}$ wird auch kurz als Ereignismenge bezeichnet und enthält die Ereignisse in zeitlich geordneter Reihenfolge.

Ereignismenge

Definition 4.8. Eine Ereignismenge $\mathbb{S}$ ist eine Menge mit n geordneten Ereignissen $e_i = (A_i, t_i)$, wobei $t_i \leq t_{i+1}$, $i = 1, ..., n$ ist.

Beispiel 4.24 verdeutlicht diesen Zusammenhang. Alle Ereignisse werden in der Ereignissequenz anhand ihres Zeitstempels aufsteigend sortiert.

Beispiel 4.24.

$$\begin{aligned}\mathbb{S} &= \{e_1, e_2, ..., e_n\} \\ &= \{(A_1, t_1), (A_2, t_2), ..., (A_n, t_n)\} \\ &= \{('A', 35), ..., ('C', 42), ..., ('C', 50)\}\end{aligned}$$

Da jede Ereignismenge $\mathbb{S}$ eine spezifische Startzeit T_s^s und eine spezifische Endzeit T_e^s besitzt, kann sie auch als Ereignissequenz s dargestellt werden. Die Startzeit T_s^s und die Endzeit T_e^s bilden dabei ein rechtsoffenes Intervall, wie Beispiel 4.25 zeigt.

Ereignissequenz

Beispiel 4.25.

$$\begin{aligned}s &= (\mathbb{S}, T_s^s, T_e^s) \\ &\quad T_s^s \leq t_i < T_e^s, \text{ with } i = 1, ..., n \\ s &= (\{(A_1, t_1), (A_2, t_2), ..., (A_n, t_n)\}, T_s^s, T_e^s) \\ &= \big(\{('E', 31), ('D', 32), ..., ('D', 67)\}, 31, 68\big)\end{aligned}$$

Fenster

Betrachtet man nur einen Ausschnitt aus einer Ereignissequenz s, so bezeichnet man diesen als Fenster. Ein Fenster ist eine Untersequenz, die durch den Startzeitpunkt T_s, den Endzeitpunkt T_e und die daraus ermittelte Breite $T_e - T_s$ vollständig beschrieben ist.

Fenster

Definition 4.9. Ein Fenster auf Ereignissequenz $s = (\mathbb{S}, T_s^s, T_e^s)$ ist eine Untersequenz $w = (\mathbb{W}, T_s^w, T_e^w)$, mit $T_s^s \leq T_s^w < T_e^s$, $T_s^s < T_e^w \leq T_e^s$ und $T_s^w < T_e^w$.

Folgendes Beispiel zeigt die Ereignissequenz s mit dem vollständigen Satz an Beispieldaten aus Abbildung 4.9 und eine mögliche Untersequenz in Form des Fensters w mit der Breite $T_e^w - T_s^w = 2$.

Beispiel 4.26.

$$s = \big(\{('E', 31), ('D', 32), ('F', 33), ..., ('A', 65), ('D', 67)\}, 31, 68\big)$$
$$w = \big(\{('D', 32), ('F', 33)\}, 32, 34\big)$$

Anzahl möglicher Fenster

Eine weitere wichtige Größe ist die Anzahl an möglichen Fenstern der Breite N auf eine gegebene Ereignissequenz $\mathbb{S}$ unter der Voraussetzung, dass das erste Fenster $N-1$ Zeiteinheiten vor s beginnt und $N-1$ Zeiteinheiten nach s endet.

Definition 4.10. Für die gegebene Ereignissequenz $s = (\mathbb{S}, T_s^s, T_e^s)$ mit der Länge $|s| = T_e^s - T_s^s$ existieren $T_e^s - T_s^s + N - 1$ Fenster der Breite N. Die Menge aller Fenster auf s ist $\omega(s, N)$. Die Breite eines Fenster wird auch als $width(w)$ beschrieben.

$$\omega(s, N) = |s| + N - 1 \tag{4.1}$$
$$= T_e^s - T_s^s + N - 1 \tag{4.2}$$

In Beispiel 4.27 erfolgt die Berechnung der Anzahl an Fenstern für die in Abbildung 4.9 gegebene Ereignissequenz.

Beispiel 4.27.

$$s = (\mathbb{S}, 30, 68)$$
$$|\omega(s, 4)| = 68 - 30 + 4 - 1 = 41$$
$$width(w_i) = N = 4,\ i = 1, 2, ..., 41$$

Episoden

Episoden als Ereignismuster

Episoden sind als Ereignismuster zu verstehen, deren Vorkommen in einer gegebenen Ereignissequenz zu untersuchen ist. Eine Episode wird durch eine Menge von Knoten, deren Ordnung untereinander, und eine Abbildungsfunktion, die den Knoten Ereignistypen zuordnet, definiert.

Definition 4.11. Episode α ist ein Tripel $(\mathbb{V}, \leq, g)$ mit $\mathbb{V}$ als Menge von Knoten, $\leq$ als Ordnungsvorschrift der Knoten in $\mathbb{V}$ und $g : \mathbb{V} \to \mathbb{E}$ als Funktion die jeden Knoten in $\mathbb{V}$ auf einen Ereignistyp aus $\mathbb{E}$ abbildet.

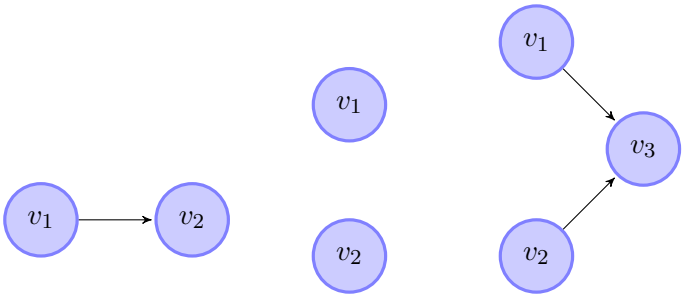
Ordnung

Die Ordnungsvorschrift legt fest, inwiefern die Knoten zueinander in Beziehung stehen. Dabei gilt es zwei Fälle zu unterscheiden: Wenn die Ordnung der Knoten unberücksichtigt bleibt, liegt die so genannte einfache Halbordnung vor. In diesem ersten Fall beschreiben die Episoden parallele Ereignismuster, bei denen die Reihenfolge der Ereignisse irrelevant

ist. Der zweite Fall ist die so genannte Totalordnung. Hier beschreiben die Episoden serielle Ereignismuster bei denen die Reihenfolge der Ereignisse relevant ist.

In beiden Fällen ist darüber hinaus zwischen injektiven und nicht injektiven Episoden zu unterscheiden. Eine injektive Episode wird beschrieben, wenn die Ordnungsvorschrift eine injektive Funktion ist und ein Element der Zielmenge $\mathbb{E}$ höchstens einmal einem Element der Definitionsmenge $\mathbb{V}$ zuweist. D.h., die Menge von Knoten $\mathbb{V}$ der Episode enthält keine Ereignistypen mehrfach.

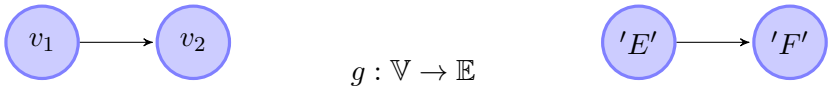
Beispiel 4.28. *Abbildung 4.10 zeigt die drei Epsioden α, β und γ. Bei der in Abbildung 4.10a dargestellten Episode handelt es sich um eine serielle Episode. Die Ereignisse treten zeitlich aufeinander folgend ein. Die in Abbildung 4.10b dargestellte Episode ist dagegen eine parallele Episode. Die Ereignisse treten gleichzeitig ein. Die Episode γ in Abbildung 4.10a ist dagegen weder eindeutig seriell noch eindeutig parallel, sondern eine Mischform von gleichzeitig und aufeinander folgend eintretenden Ereignissen.*



(a) *Episode α* **(b)** *Episode β* **(c)** *Episode γ*

Abbildung 4.10: *Beispiele für drei unterschiedliche Arten von Episoden*

Über die Abbildungsfunktion $g : \mathbb{V} \to \mathbb{E}$ erfolgt die Zuordnung von konkreten Ereignistypen in $\mathbb{E}$ auf die Menge von Knoten in $\mathbb{V}$, wie Abbildung 4.11 zeigt.



(a) *Allgemeine Episode mit zwei Ereignissen* **(b)** *Abbildungsfunktion* **(c)** *Konkrete Episode nach der Abbildung*

Abbildung 4.11: *Abbildung der Ereignisse einer seriellen Episode auf konkrete Ereignistypen*

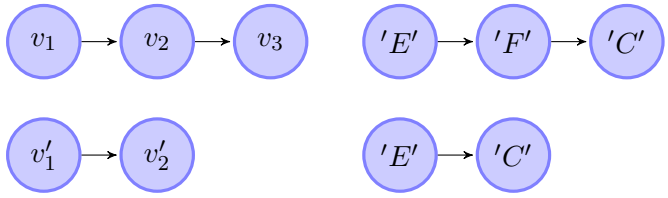
Soll eine Untermenge der Knoten einer Episode betrachtet werden, so können diese als Unterepisode repräsentiert werden. Die Unterepisode wird dabei wie eine normale Episode repräsentiert, unterliegt aber den in der folgenden Definition beschriebenen Restriktionen. Unterepisode

Definition 4.12. Unterepisode β ist das Tripel $(\mathbb{V}', \leq', g')$ und die Unterepisode von $\alpha = (\mathbb{V}, \leq, g)$, dargestellt als $\beta \preceq \alpha$, wenn die injektive Abbildung $f : \mathbb{V}' \to \mathbb{V}$ existiert, so dass $g'(v) = g(f(v))$ für alle $v \in \mathbb{V}'$, und für alle $v, w \in \mathbb{V}'$ mit $v \leq' w$ also $f(v) \leq f(w)$.

injektive Abbildungsfunktion

Um sicherzustellen, dass kein Ereignistyp der Zielmenge $\mathbb{V}$ mehrfach zugewiesen wird und die Ordnungsvorschrift gültig bleibt, ist es notwendig, eine injektive Abbildungsfunktion zu verwenden. Beispiel 4.29 veranschaulicht diese Zusammenhänge.

Beispiel 4.29. *Abbildung 4.12a zeigt die Episode $\mathbb{V}$ sowie eine mögliche Unterepisode $\mathbb{V}'$. Die Abbildung ist injektiv, das heißt, jedes Element aus der Definitionsmenge $\mathbb{V}'$ wird auf genau ein Element der Zielmenge $\mathbb{V}$ abgebildet. In diesem Fall ist die Definitionsmenge immer kleiner als die Zielmenge.*



(a) *Allgemeine Episoden*

(b) *Konkrete Episoden $\mathbb{V}$ und $\mathbb{V}'$ nach der Abbildung*

$g : \mathbb{V} \to \mathbb{E}$
$g(v_1) =' E'$
$g(v_2) =' F'$
$g(v_3) =' C'$

(c) *Abbildung g*

$f : \mathbb{V}' \to \mathbb{V}$
$f(v'_1) = v_1$
$f(v'_2) = v_3$

(d) *Abbildung f*

$g' : \mathbb{V}' \to \mathbb{E}$
$g'(v'_1) =' E'$
$g'(v'_3) =' C'$

(e) *Abbildung g'*

Abbildung 4.12: *Episode $\mathbb{V}$ und eine Unterepisode $\mathbb{V}'$ sowie notwendige Abbildungen*

Abbildung 4.12b zeigt die resultierenden konkreten Episoden und die Abbildungen 4.12c bis 4.12e die durchgeführten Abbildungen.

4.2.4 Algorithmen

Von den in Abschnitt 2.4.2 (ab Seite 27) beschriebenen Algorithmen werden im folgenden Abschnitt die Algorithmen WinEpi und MinEpi aus [MTV97] erläutert. Die Algorithmen MinEpi+ und EMMA [HC08] finden in der vorliegenden Arbeit keine Anwendung und werden daher nicht genauer betrachtet.

WinEpi

Der in [MTV97] vorgestellte und sehr häufig in der Literatur referenzierte WinEpi-Algorithmus dient der Ermittlung von Mustern in Ereignissequenzen. Das Grundprinzip des Algorithmus besteht darin, alle möglichen Fenster (einer zuvor festgelegten Breite) einer Ereignissequenz auf häufige Episoden zu untersuchen. Die Suche erfolgt dabei als Breitensuche, angefangen bei Episoden der Länge $l = 1$. Dazu werden zunächst Episoden dieser Länge generiert, die so genannten Kandidaten $\mathbb{C}_l$. Deren Häufigkeit wird im Anschluss daran in einem Durchlauf der Ereignissequenz ermittelt, um nicht häufige Kandidaten ausschließen zu können. Als Ergebnis erhält man die Menge der häufigen Episoden $\mathbb{F}_l$ der Länge l. Abschließend erfolgt die Generierung von neuen Kandidaten der Länge $l = l + 1$ und der Vorgang beginnt erneut. Die Berechnung endet, wenn keine häufigen Episoden mehr gefunden werden, so dass die Generation von Kandidaten eine leere Menge liefert ($\mathbb{C}_l = \emptyset$). Als Parameter werden die Ereignissequenz $\mathbb{S}$, die Fensterbreite $width(w)$ und der Häufigkeitsschwellwert min_f zur Ermittlung der häufigen Episoden benötigt. Nachfolgend wird dieser Haupt-Algorithmus $\mathcal{F}(s, width(w), min_f)$ beschrieben.

WinEpi

Algorithmus 1 Haupt-Algorithmus WinEpi nach [MTV97]

1: **procedure** $\mathcal{F}(s, width(w), min_f)$
2: $l \leftarrow 1$
3: $\mathbb{C}_l \leftarrow \{\alpha \in \epsilon \quad | \quad |\alpha| = 1\}$
4: **while** $\mathbb{C}_l \neq \emptyset$ **do**
5: $\mathbb{F}_l \leftarrow \{\alpha \in \mathbb{C}_l \quad | \quad f(\alpha, s, width(w)) \geq min_f\}$
6: $l \leftarrow l + 1$
7: $\mathbb{C}_l \leftarrow \{\alpha \quad | \quad |\alpha| = l$ and for all sub-episodes β of α such that $\beta \in \mathbb{F}_{|\beta|}\}$
8: **end while**
9: **for** all l **do**
10: $\mathbb{F} \leftarrow \mathbb{F} \cup \mathbb{F}_l$
11: **end for**
12: **return** $\mathbb{F}$
13: **end procedure**

Zu Beginn des Algorithmus' wird zunächst die Menge an Kandidaten $\mathbb{C}_1$ aus der Menge an Ereignistypen ϵ erstellt. Daraufhin erfolgt abwechselnd der Datenbankdurchlauf (in Zeile 5 von Algorithmus 1), das Erhöhen der Episodenlänge und die Generierung neuer Kandidaten (in Zeile 7 von Algorithmus 1).

Sowohl der Datendurchlauf als auch die Generation von Kandidaten sollen in den folgenden Absätzen ausführlicher vorgestellt werden. Die Unteralgorithmen dieser Vorgänge sind in [MTV97] ausführlich beschrieben.

Ermittlung der Häufigkeit Nach [MTV97] ergibt sich die relative Häufigkeit einer

Häufigkeit von Episoden

Episode aus der Anzahl an Fenstern, in denen eine Episode α vollständig enthalten ist, bezogen auf die Gesamtzahl an Fenstern:

Definition 4.13. Bei gegebener Ereignissequenz s, dem Fenster w auf s mit der Breite $width(w)$ ist die Häufigkeit f einer Episode α:

$$f(\alpha, s, width(w)) = \frac{|\{w \in \omega(s, width(w)) \quad | \quad \alpha \text{ occurs in } w\}|}{|\omega(s, width(w))|} \tag{4.3}$$

In Beispiel 4.30 erfolgt die Berechnung der Häufigkeit einer Episode der Länge $l = 2$. Es werden Fenster der Breite $N = 3$ betrachtet.

Beispiel 4.30. *Bei gegebener Ereignissequenz* $\mathbb{S} = (\{'A', 'E', 'A', 'F', \}, 30, 34)$ *ergibt sich folgende Häufigkeit für die Episode* $\alpha = 'EA'$ *bei Betrachtung aller Fenster der Breite* 3*:*

$$\begin{aligned} f('EA', s, 3) &= \frac{|\{w \in \omega(s, 3) \quad | \quad 'EA' \text{ occurs in } w\}|}{|\omega(s, 3)|} \\ &= \frac{2}{6} \end{aligned}$$

Zur Verdeutlichung zeigt Abbildung 4.13 die Ereignissequenz, alle möglichen Fenster der Breite 3 *(dargestellt durch die Klammern in den Zeilen* 1 *bis* 6*) sowie die Fenster, die die gesuchte Episode* α *vollständig enthalten (farbig hervorgehoben). Zwei von sechs Fenstern enthalten die Episode* α*.*

	28	29	30 A	31 E	32 A	33 F	34
1	(		)				
2		(		)			
3			(		)		
4				(		)	
5					(		)
6						(	)

Abbildung 4.13: *Darstellung der Ereignissequenz* $s = (\{'A', 'E', 'A', 'F', \}, 30, 34)$ *mit allen Fenstern der Breite* 3

Schwellwert

Bei der Untersuchung von Ereignissequenzen werden alle Episoden als häufig bezeichnet, deren Häufigkeit einen gewissen Schwellwert erreicht oder überschreitet.

Definition 4.14. Die Menge aller Episoden in $\mathbb{S}$ mit einer Häufigkeit f, die gleich oder größer als der Schwellwert min_f ist, wird wie folgt beschrieben.

$$\mathbb{F} = \mathcal{F}(s, width(w), min_f)$$

Die Variation des Schwellwerts min_f hat also eine direkte Auswirkung auf die Menge der detektierten häufigen Episoden, wie das folgende Beispiel zeigt. Ausgangsbasis ist dabei die gleiche Ereignissequenz $\mathbb{S}$ und die gleiche Fensterbreite $N = 3$ wie in Beispiel 4.30, lediglich der Schwellwert min_f wird variiert.

Beispiel 4.31. *Bei gegebener Ereignissequenz* $s = (\{'A', 'E', 'A', 'F', \}, 30, 34)$ *ergeben sich folgende Mengen häufiger Episoden für die unterschiedlichen Schwellwerte* $min_f = \frac{3}{6}$ *und* $min_f = \frac{2}{6}$*:*

$$\mathcal{F}\left(s, 3, \frac{3}{6}\right) = \{\}$$
$$\mathcal{F}\left(s, 3, \frac{2}{6}\right) = \{'EA'\}$$

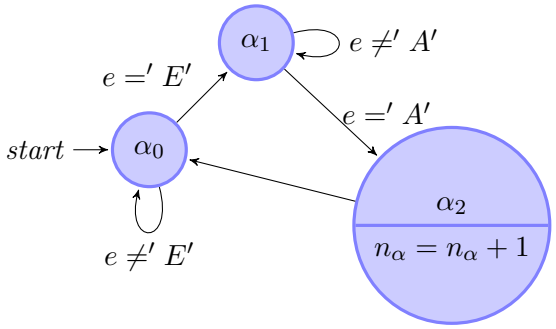
Der Vorgang des Zählens der Fenster und die Filterung der Kandidaten entsprechend ihrer Häufigkeit erfolgt in Algorithmus 1 in Zeile 5:

$$\mathbb{F}_l \leftarrow \{\alpha \in \mathbb{C}_l \quad | \quad f(\alpha, s, width(w)) \geq min_f\}$$

Automaten zählen Episoden

Für die Implementierung des Zählens von häufigen Episoden schlägt [MTV97] im Falle der Betrachtung von seriellen Episoden den Einsatz von Automaten vor. Vereinfacht ausgedrückt existieren bei dieser Vorgehensweise für jeden Kandidaten der Länge $l = |\alpha|$ maximal l Automaten. Die Zustandswechsel der Automaten erfolgen dann synchron mit dem Eintreffen eines bestimmten Ereignistyps in der Ereignissequenz $\mathbb{S}$. Dieser Zusammenhang wird in Beispiel 4.32 dargestellt.

Beispiel 4.32. *Angenommen man betrachtet als Kandidaten die Episode* $\alpha =' EA'$ *und* e *als aktuelles Ereignis in der Datensequenz, dann verwendet der Algorithmus den folgenden Automaten, um die Anzahl der Vorkommen* n_α *der Episode* α *zu zählen:*



Zu Beginn des Algorithmus wird für jeden Kandidaten ein Automat im Zustand α_0 erzeugt. In diesem verbleibt der Automat so lange bis sich das Ereignis $\alpha[0]$ (im Fall des Beispiels $'E'$) im Fenster befindet. Wenn dieser Fall eintritt wird der Automat in den nächsten Zustand überführt, so dass er auf das Ereignis $\alpha[1]$ wartet (im Fall des Beispiels $'A'$), gleichzeitig wird ein neuer Automat angelegt, der wieder auf das Ereignis $\alpha[0]$ wartet. Hat ein Automat alle Übergänge von Zustand α_0 bis α_n durchgeführt, so wird der Zähler für den Kandidaten α um eins erhöht und der Automat in Zustand $\alpha[0]$ zurückgesetzt (wenn nicht bereits ein Automat in diesem Zustand existiert).

Da nur das Vorkommen von Kandidaten gezählt werden darf, die sich vollständig im aktuell betrachteten Fenster befinden, werden die Automaten zurückgesetzt sobald das Ereignis, welches für den Übergang des Automaten in Zustand $\alpha[1]$ verantwortlich war, das Fenster wieder verlässt.

Generierung von Kandidaten Die Initialkandidaten der Länge $l = 1$ werden zu Beginn aus der Menge von Ereignistypen ϵ erzeugt (siehe Algorithmus 1) indem jeder vorkommende Ereignistyp zu $\mathbb{C}_1$ hinzugefügt wird. Der darauffolgende Datendurchlauf über die Ereignissequenz s filtert dann alle nicht häufig vorkommenden Episoden aus $\mathbb{C}_1$ heraus, was zur Menge der häufigen Episoden der Länge $\mathbb{F}_1$ führt. Die Generierung von Kandidaten einer Länge $l > 1$ erfolgt in Zeile 7 von Algorithmus 1:

Initialkandidaten

$$\mathbb{C}_l \leftarrow \{\alpha \quad | \quad |\alpha| = l \text{ and for all sub-episodes } \beta \text{ of } \alpha \text{ such that } \beta \in \mathbb{F}_{|\beta|}\}$$

Bei der Generierung von Kandidaten der Länge $l + 1$ werden die häufigen Kandidaten anhand ihrer ersten $l-1$ Ereignistypen zu Blöcken gruppiert. Alle Episoden eines Blocks werden dann durch Kombination von immer zwei Episoden (unter Beibehaltung des gemeinsamen $l - 1$ Ereignistypen) in die Kandidaten überführt. Zur Reduzierung (engl. „Pruning“) der Kandidatenmenge findet Satz 4.1 Anwendung.

Satz 4.1. *Wenn eine Episode α bezogen auf Ereignissequenz $s = (\mathbb{S}, T_s^s, T_e^s)$ häufig ist, dann sind all ihre Unterepisoden $\beta \preceq \alpha$ auch häufig.*

Reduzierung der Kandidaten

Die folgenden Beispiele veranschaulichen sowohl die Kandidatengenerierung als auch die Reduzierung von Kandidaten durch Satz 4.1. Zu beachten ist dabei, dass folgende Beispiele ausschließlich auf injektiven seriellen Episoden basieren, da nur diese im Rahmen der vorliegenden Arbeit zur Anwendung kommen.

Beispiel 4.33. *Bei gegebenen häufigen Kandidaten der Länge 1 in $\mathbb{F}_1$ erfolgt die Generierung der Kandidaten der Länge $l = 2$ ($\mathbb{C}_2$) durch Kombination aller Episoden einer Gruppe aus $\mathbb{F}_1$.*

$$\mathbb{F}_1 = \{'E', 'F', 'C', 'A'\}$$
$$\Downarrow$$
$$\mathbb{C}_2 = \{'EF', 'EC', 'EA', 'FE', 'FC', 'FA', 'CE', 'CF', 'CA', 'AE', 'AF', 'AC'\}$$

Betrachtet man die Menge von häufigen Episoden $\mathbb{F}_1$ und die daraus erzeugten Kandidaten, so wird deutlich, dass im ersten Schritt (Generierung von Kandidaten der Länge $l = 2$) der Satz 4.1 noch nicht zur Reduzierung von Episoden führt. Das liegt daran, dass in diesem Schritt noch alle häufigen Episoden der gleichen Gruppe angehören, da keine gemeinsamen Ereignistypen vorliegen. Erst ab einer Kandidatenlänge von $l = 3$ reduziert die Anwendung des Satzes die Anzahl der zuvor erzeugten Kandidaten:

Beispiel 4.34. *Angenommen der Datendurchlauf führt zur Menge von häufigen Episoden $\mathbb{F}_2$. Nach der Generierung von möglichen Kandidaten $\mathbb{C}_3$ durch Kombination aller*

Episoden einer Gruppe in $\mathbb{F}_2$ erfolgt die Überprüfung der Häufigkeit aller Unterepisoden der neuen Kandidaten. Wenn nicht alle Unterepisoden eines Kandidaten auch häufing sind, wird der Kandidat verworfen.

$$\begin{aligned}
\mathbb{F}_2 &= \{'EF', 'EC', 'FA', 'CF'\} \\
&\Downarrow \text{ Kandidaten nach Kombination innerhalb der Gruppen} \\
\mathbb{C}_3 &= \{\cancel{'EFC'}, 'ECF'\} \\
&\Downarrow \text{ Unterepisode } 'FC' \text{ von } 'EFC' \text{ ist nicht häufig in } \mathbb{F}_2 \\
&\Downarrow \text{ Kandidaten nach Reduzierung} \\
\mathbb{C}_3' &= \{'ECF'\}
\end{aligned}$$

Komplexitätsbetrachtung Die Laufzeit des Algorithmus' für die Erkennung von injektiven seriellen Episoden ist nach [MTV97] für jeden Durchlauf abhängig von der Anzahl der Kandidaten $\mathbb{C}_l$ und der Länge der Ereignissequenz n.

MinEpi

Als Ausgangspunkt für den MinEpi-Algorithmus gilt weiterhin der Satz 4.1. Daher ist es möglich, dass sowohl der Algorithmus 1 als auch das Vorgehen bei der Kandidatengenerierung weiter verwendet werden kann. Die Unterschiede zwischen dem MinEpi und dem WinEpi-Algorithmus liegen in der Ermittlung der häufigen Episoden.

MinEpi

Ermittlung der Häufigkeit Im Gegensatz zum WinEpi-Algorithmus wird bei MinEpi kein gleitendes Fenster eingesetzt, um die Vorkommen von Episoden zu zählen. MinEpi untersucht die Ereignissequenz auf minimale Vorkommnisse (engl. „minimal occurences“) von Episoden. Ein minimales Vorkommen der Episode α ist in diesem Zusammenhang ein Zeitintervall $[t_s, t_e]$ der Ereignissequenz, das folgende Bedingungen erfüllt:

„minimal ocurrences“

1. das Zeitintervall $[t_s, t_e]$ schließt ein Vorkommen von α ein
2. kein Teilintervall $[t_s', t_e']$ von $[t_s, t_e]$ mit $t_s' \geq t_s$ und $t_e \geq t_e'$ schließt α ein

Ein minimales Vorkommen von α ist demnach ein minimales Zeitfenster, das α einschließt. Der Ausdruck $mo(\alpha)$ beschreibt dann die Menge der minimalen Vorkommnisse einer Episode α. Die Häufigkeit einer Episode α wird als $|mo(\alpha)|$ angegeben. Zur Ermittlung der häufigen Episoden kommt ein Schwellwert min_{sup} zum Einsatz, so dass Zeile 5 in Algorithmus 1 zu folgendem Ausdruck angepasst werden muss.

$$\mathbb{F}_l \leftarrow \{\alpha \in \mathbb{C}_l \quad | \quad |mo(\alpha)| \geq min_{sup}\}$$

So werden alle Episoden, deren minimale Vorkommnisse den Schwellwert min_{sup} erreichen oder überschreiten, zu den häufigen Episoden gezählt, um im nächsten Schritt

daraus die Kandidaten für Durchgang $l + 1$ zu erzeugen. Die ausführliche Beschreibung der Berechnung von $mo(\alpha)$ ist [MTV97] zu entnehmen.

Komplexitätsbetrachtung Nach [MTV97] ist die Zeitkomplexität des MinEpi-Algorithmus' nur von der Länge der Ereignissequenz abhängig und damit linear. Daher ist dieser Algorithmus schneller als der WinEpi-Algorithmus. Des Weiteren ermöglicht er die Erstellung von fensterübergreifenden Regeln. Nachteilig ist am MinEpi-Algorithmus dagegen die höhere Speicherkomplexität, da zur Laufzeit alle minimalen Vorkommnisse gespeichert werden müssen.

hoher Speicherbedarf

Zur Reduzierung des hohen Speicherbedarfs des MinEpi-Algorithmus' schlägt [MTV97] zwei Ansätze vor. Der erste besteht darin, dass nur noch die minimalen Vorkommnisse des aktuellen Durchlaufs l und die des nächsten Durchlaufs $l+1$ gespeichert werden. Der zweite Ansatz sieht die Kombination des MinEpi-Algorithmus mit anderen Algorithmen zur Musteranalyse für die ersten Durchläufe vor, da gerade bei diesen der Speicherbedarf aufgrund der vielen zu speichernden minimalen Vorkommnisse sehr hoch ist. So müssen für den ersten Durchlauf n minimale Vorkommnisse gespeichert werden, wobei n der Länge der Eingangssequenz entspricht. Denkbar wäre zum Beispiel der Einsatz des WinEpi-Algorithmus für die ersten Durchläufe, um dann bei Unterschreiten einer kritischen Anzahl an Kandidaten auf den MinEpi-Algorithmus umzustellen.

Ich denke, dass es einen Weltmarkt für vielleicht fünf Computer gibt.

Thomas J. Watson - Unternehmer

5

Abstraktion vom Gesamtsystem

Bevor in den Kapiteln 6 und 7 auf die konkrete Realisierung von Erweiterungsmodulen und Beispielalgorithmen für die MMS eingegangen wird, erfolgt zunächst die Entwicklung der notwendigen Architekturanpassung, auf die die MMS aufbauen wird. Letztere wird später eine Plattform darstellen, die von den komplexen Abläufen im Gesamtsystem und den daran beteiligten Komponenten abstrahiert. Dazu beschreibt das vorliegende Kapitel die Modellierung einer flexiblen Abstraktionsschicht für die MMS der MASSiVE-Architektur sowie die Entwicklung der zur Verfügung zu stellenden Schnittstellen für Benutzer und Anwender. Der nächste Abschnitt widmet sich daher der Identifikation von Kernfunktionalitäten der MMS, die direkt aus notwendigen Anforderungen heraus abgeleitet und formuliert werden. Im Anschluss daran folgt in Abschnitt 5.4 Schrittweise der Entwurf der zuvor festgelegten Kernkomponenten.

Modellierung

5.1 Beschreibung des Ansatzes

In Anforderung 3.2 wird die Notwendigkeit der Komplexitätsreduktion formuliert, da die bestehende Anzahl an Einzelkomponenten der MMS die Handhabung und Erweiterung derselben erschwert. Darauf aufbauend beschreibt Anforderung 3.3 die Loslösung von den Einschränkungen des eingesetzten Ereigniskanals. Aus diesen Anforderungen lässt sich als erster möglicher Ansatz die Entwicklung eines Kommunikationsmechanismus, ohne Einsatz des Ereigniskanals ableiten. In diesem Fall wäre es notwendig, die Eingabegeräte auf andere Weise mit der MMS zu verbinden. Hier könnte wieder auf den Einsatz des CORBA-Ereigniskanals zurückgegriffen werden, was jedoch zum vollständigen Verzicht auf Serialisierung bei der Kommunikation in Widerspruch stehen würde (siehe Anforderung 3.1).

Komplexitätsreduktion

Aus diesen Betrachtungen und den bereits formulierten Anforderungen leitet sich folgendes Bild für einen angepassten Aufbau der MMS ab: Die „GUI“s und die Eingabegeräte müssen über einen Mechanismus zur Erweiterung in die MMS integriert werden. Die Kernkomponente „HMI Model“ wird dann neben der Komponente für die Ausführung der Benutzerinteraktionen zur einzigen Komponente der MMS (siehe Abbildung 5.1). Die

bestehenden CORBA Kommunikationskanäle werden auf Schnittstellen dieser Komponente abgebildet. Durch diese Vorgehensweise kann auch der Anforderung 3.6 Rechnung getragen werden. Bezogen auf die MASSiVE-Architektur könnte so die Trennung der MMS von ihren Funktionalitäten erreicht werden.

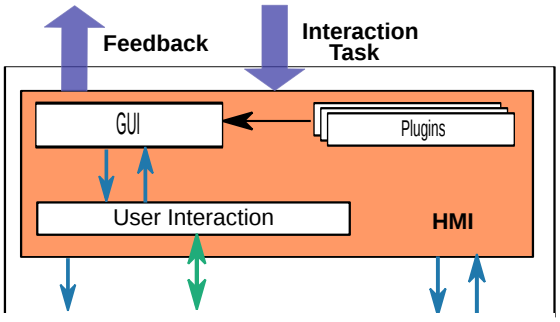


Abbildung 5.1: *Aufbau der MMS nach Realisierung der notwendigen Anpassung*

dynamisch nachladbare Bibliotheken

Als Ansatz für die Realisierung dieser Anpassungen soll der in [Mar99] vorgestellte Mechanismus als Basis dienen. Die Integration der Erweiterungen erfolgt als dynamisch zur Laufzeit nachladbare Bibliotheken, die durch die MMS bzw. dem Kontextvertrag vorgegebene Schnittstellen implementieren.

Lösungsansatz 5.1. *Um die MMS als flexibel erweiterbares Rahmenwerk zu realisieren, wird sie Schnittstellen für Erweiterungen definieren. Die MMS fungiert dabei als Abstraktionsschicht zu den übrigen Komponenten der MASSiVE Architektur, sie ist die Rahmenanwendung.*

Vor der Definition der Schnittstellen erfolgt zunächst die Betrachtung der notwendigen Anpassungen zur Realisierung der benötigten Funktionalitäten, auf die später durch die Schnittstellen zugegriffen werden soll. Im nächsten Abschnitt wird die notwendige dynamische Erfassung von Systemkomponenten beschrieben, die später die Realisation der Adaptivität erlauben wird.

5.2 Dynamische Erfassung von Systemkomponenten

„launchd“ als Referenz

Recherchiert man nach Methoden zur Erfassung von Systemkomponenten, so stößt man schnell auf Systeme, die beim Start von Betriebssystemen zum Einsatz kommen. Apple setzt seit Mac OS X in diesem Zusammenhang auf die Systemkomponente „launchd“ (siehe [Wik10, Stichwort: „launchd“]). „launchd“ übernimmt dabei nicht nur die Aufgabe des Startens von Diensten im Moment in dem sie das erste Mal benötigt werden, sondern überwacht gestartete Dienste auch zur Laufzeit. Da viele der Funktionalitäten von „launchd“ auf die Erfassung von Systemkomponenten in MASSiVE übertragen werden

können, erfolgt die Festlegung von Lösungsansätzen im vorliegenden Abschnitt auf Basis der von „launchd“ eingesetzten Methodik.

Anforderung 3.17 formuliert die Notwendigkeit einer Schnittstelle, über die der Zugriff auf die Systemressourcen gekapselt wird. Dieser Abschnitt beschreibt ausgehend vom ist-Zustand (siehe Abschnitt 2.1.3) die notwendigen Implementationsdetails zur Realisierung der dazu notwendigen Funktionalität.

Eine Grundvoraussetzung für den Entwurf einer Schnittstelle ist, dass die Schnittstelle zu keinerlei Abhängigkeiten zwischen den beteiligten Komponenten führt. Darüber hinaus darf die Verwendung einer Schnittstelle nicht von internem Komponentenwissen abhängen, eine Eigenschaft, die in MASSiVE verletzt wird (siehe Abschnitt 3.2.3). Der folgende Abschnitt beschreibt die zur Auflösung dieser Problematik notwendigen Architekturanpassungen.

Abhängigkeiten vermeiden

5.2.1 Notwendige Architekturanpassungen

Nach Anforderung 3.18 ist es notwendig, die Initialisierungsmethode der MASSiVE Komponenten aus der reaktiven Ebene so anzupassen, dass diese keine Abhängigkeit zur der Liste mit den verwendeten Ressourcen aufweist. Dazu sollen die Abhängigkeitsinformationen auf anderem Weg vom Entwickler oder der Komponente selbst geliefert werden.

Auflösung der Ressourcenabhängigkeit

Die nächstliegendste Umsetzung dieser Anforderung besteht in der Verwendung von Spezifikationsdateien für die Komponenten der MASSiVE-Architektur. Verglichen mit der Kodierung der notwendigen Informationen in den Objektcode der Anwendungen hat das einen entscheidenden Vorteil: Die Daten können ohne erneute Übersetzung der Komponente angepasst werden. Da Spezifikationsdateien aufgrund von Falscheinträgen leicht zu Fehlern führen können, sollen diese für MASSiVE Server-Komponenten auf einen eigens dafür festgelegten XML-Dialekt basieren, der im folgenden Abschnitt entwickelt wird.

Spezifikation von Ressourcen

Lösungsansatz 5.2. *Spezifikation eines XML-Dialektes für die Festlegung von komponentenspezifischen Daten.*

Neben der Festlegung der Server-Spezifikation sind darüber hinaus weitere Anpassungen erforderlich, die die Auswertung der Spezifikationen ermöglichen. Dafür ist zunächst der Aufbau der Komponenten der reaktiven Ebene zu untersuchen. Alle Komponenten bestehen grundsätzlich aus zwei Klassen. Die erste Klasse implementiert die CORBA-Schnittstelle und wird vom ORB als aktives Objekt zur Laufzeit instanziiert. Diese wird im folgenden als Implementationsklasse bezeichnet. Die zweite Klasse implementiert den Haupteinsprungpunkt des Betriebssystems und wird im folgenden als Anwendungsklasse

bezeichnet. Abbildung 5.2 zeigt diesen Aufbau am Beispiel eines Hardware-Servers in Form eines Klassendiagramms.

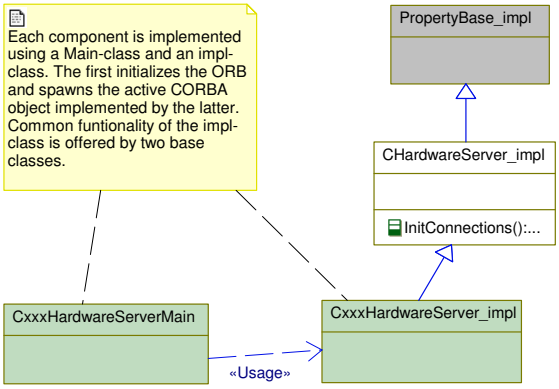


Abbildung 5.2: *Aufbau von Komponenten in der reaktiven Ebene (am Beispiel eines Hardware-Servers)*

Anwendungsklasse

Die Anwendungsklasse `CxxxHardwareServerMain` wertet Kommandozeilenargumente aus, über die das Laufzeitverhalten der Anwendung beeinflusst werden kann. Vergleicht man den Aufbau der derzeit implementierten Komponenten, so wird deutlich, dass viele der Funktionalitäten in der „Main“-Klasse für alle Komponenten identisch sind. Darunter fallen zum einen identische Kommandozeilenargumente und zum anderen Funktionalitäten zur Instanziierung des CORBA ORB sowie zur Aktivierung der CORBA Schnittstelle.

Implementationsklasse

Die Implementationsklasse `CxxxHardwareServer_impl` erbt zunächst gemeinsame Funktionalitäten von der Basisklasse `CHardwareServer_impl`. Diese Basisklasse erbt wiederum Funktionalitäten der Klasse `PropertyBase_impl`, welche im Rahmen von [Fre09] entwickelt wurde und auch dort ausführlich beschrieben ist. Ihre Kernfunktionalität besteht darin, serverspezifische Eigenschaften zur Laufzeit vom Trading-Service zu erfragen. Da das Lesen von Eigenschaften einen ähnlichen Hintergrund besitzt wie die angestrebte Realisation einer XML-basierten Serverspezifikation, ist es sinnvoll, letztere als Erweiterung der `PropertyBase_impl`-Klasse zu realisieren. Die Validierung und Auswertung der Spezifikationsdateien kann dabei über die freie Bibliothek Xerces-C++ (siehe [The11]) erfolgen.

Diese bisher auf Serverkomponenten vom Typ Hardware-Server beschränkte Betrachtung gilt auch für andere Typen, deren Aufbau generell dem in Abbildung 5.2 dargestellten entspricht. Es erfolgt lediglich eine dem Typ der Serverkomponenten entsprechende Anpassung der Klassennamen.

Lösungsansatz 5.3. *Die Auswertung der XML-basierten Komponentenspezifikation für die Serverkomponenten der MASSiVE-Architektur erfolgt mittels Xerces-C++ und wird über die bestehende Klasse* `PropertyBase_impl` *integriert.*

Um die Validierung und Auswertung der Spezifikationen durchzuführen, müssen die Komponenten den Dateinamen der entsprechenden XML-Instanzen kennen. Der ein-

fachste Weg das zu realisieren besteht in der Verwendung von festen Namen, die dem Namen der jeweiligen Komponente entsprechen. So ist jede Komponente ohne weitere Informationen in der Lage, den Namen der Spezifikation zu ermitteln und sie zu laden. Da diese Vorgehensweise als alleinige Methodik langfristig zu unflexibel ist, wird jedoch bewusst darauf verzichtet. Sie wird jedoch als optionales Vorgehen zur Dateinamensermittlung realisiert, das dann zum Einsatz kommt, wenn der Name der Spezifikation nicht als Argument beim Start übergeben wird. Wird der Name dagegen als Argument übergeben, so kann die Spezifikation direkt geladen, validiert und ausgewertet werden. Durch diese Vorgehensweise kann ein und dieselbe Komponente mehrfach mit unterschiedlichen Spezifikationen gestartet werden.

Laden der Spezifikation

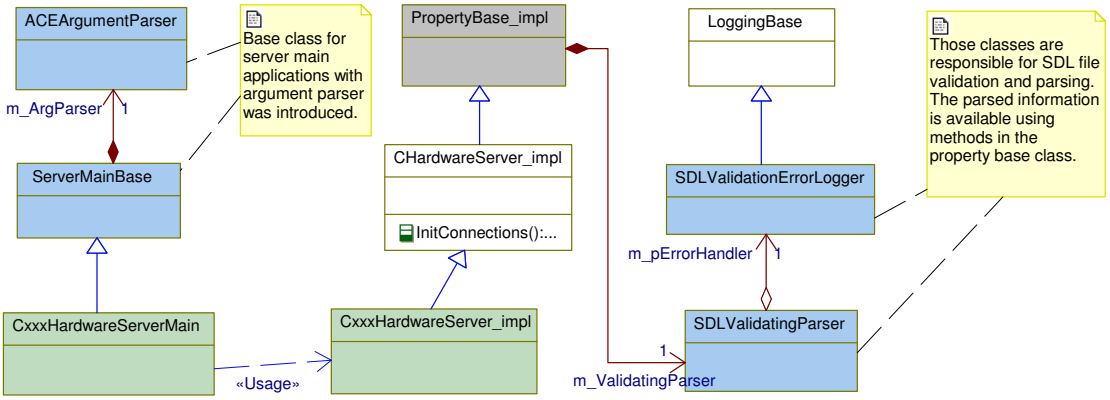


Abbildung 5.3: *Erweiterung des Aufbaus von Komponenten in der reaktiven Ebene (am Beispiel der Hardware-Server)*

Zuvor wurde bereits festgestellt, dass viele Funktionalitäten in den Anwendungsklassen der verschiedenen Komponenten identisch sind. Mit Einführung der Server-Spezifikation kommt eine identische Funktionalität dazu: die Auswertung derselben. Um langfristig die bisher praktizierte mehrfache Implementation identischer Funktionalitäten zu beenden, werden dazu zwei weitere Klassen eingeführt. Zum einen die `ServerMainBase`-Klasse und zum anderen die `ACEArgumentParser`-Klasse[1]. Die erste übernimmt die gesamte Funktionalität, die zuvor in allen Anwendungsklassen redundant implementiert wurde. Die zweite übernimmt die Auswertung der Kommandozeilenargumente. Abbildung 5.3 zeigt den Aufbau der Komponenten nach den durchgeführten Anpassungen.

Abbildung 5.4 veranschaulicht die Vorgänge beim Start einer Komponente. Zunächst erfolgt die Initialisierung des `ACEArgumentParser` und die Auswertung der Kommandozeilenargumente. Im Anschluss daran wird der CORBA ORB initialisiert, um direkt danach die Implementationsklasse zu instanziieren (die von der `PropertyBase_impl` abgeleiteten Klassen werden im Sequenz-Diagramm zur besseren Übersicht nicht dargestellt). Es folgt die Servant Initialisierung über die Methode `SetupServant()`, welche als Argument die zuvor instanziierte Implementationsklasse erhält. Während der Initialisierung wird zunächst der Name der Spezifikation ermittelt, um diese im Anschluss daran zu validieren und auszuwerten. Die Informationen aus der Spezifikation, also auch die Abhängigkeitsin-

Startsequenz

[1] Der Namenszusatz „ACE“ ist ein Hinweis auf die eingesetzte „ACE“-Bibliothek

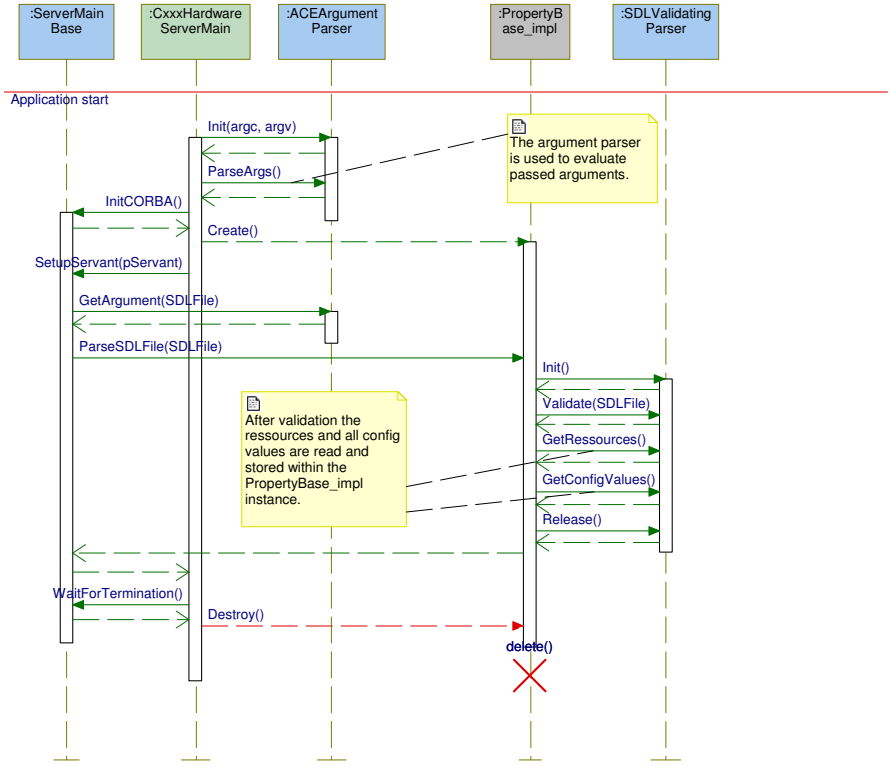


Abbildung 5.4: *Startablauf der Komponenten der reaktiven Ebene nach Umsetzung der notwendigen Anpassungen*

Start ohne Abhängigkeitsinformationen

formationen, stehen dann in der Instanz der `PropertyBase_impl`-Klasse zur Verfügung. Die Initialisierungsmethode `InitConnections()` kann aufgrund dieser Anpassungen jetzt ohne Angabe der Liste mit Abhängigkeitsinformationen eingesetzt werden, da sie diese Informationen direkt aus ihrer Basisklasse `PropertyBase_impl` beziehen kann.

Initialisierung von Ressourcen

Initialisierungsvorgänge

Neben der reinen Akquise und Verbindungsherstellung sind je nach Typ der Ressource unterschiedliche Initialisierungsvorgänge vor dem ersten Zugriff durch eine Erweiterung durchzuführen. In der MASSiVE-Architektur existieren drei Servertypen, die die folgenden Initialisierungsvorgänge erfordern:

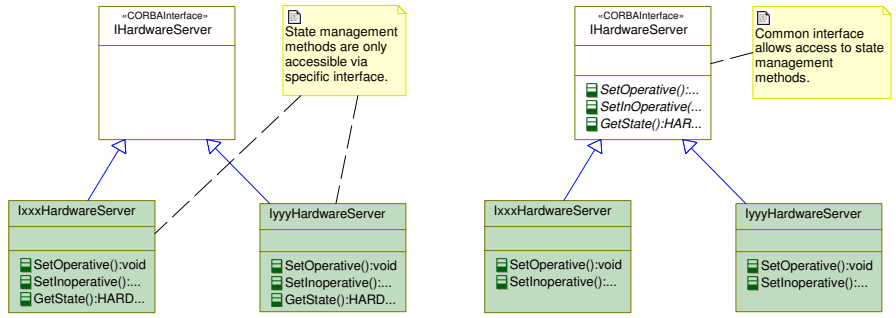
- `HardwareServer` müssen über die `InitConnections()` Methode initialisiert werden. Dabei darf lediglich eine Verbindung zur Logging-Komponente hergestellt werden. Darüber hinaus müssen sie mittels `SetOperative()` aktiviert werden.

- `SkillServer` müssen über die `InitConnections()` Methode initialisiert werden, dabei sind Verbindungen zu den verwalteten `HardwareServern` und anderen `SkillServern` erlaubt.

- `DataServer` müssen über die `InitConnections` Methode initialisiert werden. Dabei darf lediglich eine Verbindung zur Logging-Komponente hergestellt werden.

Da diese Initialisierungsvorgänge die Verwendung von Ressourcen in den Erweiterungen unnötig komplex gestalten würden, sind sie in der MMS zu kapseln. Dabei darf keinerlei Abhängigkeit der MMS von den Ressourcen erzeugt werden. Um dieses Ziel zu erreichen, muss die MMS die Vorgänge über die gemeinsame Basisschnittstelle und nicht über die spezialisierten Schnittstellen der Ressourcen durchführen. Eine Überprüfung der gemeinsamen Schnittstelle ergibt jedoch, dass die notwendigen Methoden gar nicht von dieser Schnittstelle, der Klasse `HardwareServer`, exportiert werden, wie Abbildung 5.5a zeigt. Vielmehr werden die Methoden für jeden `HardwareServer`, also jede Spezialisierung der Basisschnittstelle neu exportiert und implementiert. In einer weiteren Architekturanpassung wird diese Problematik (wie Abbildung 5.5b zeigt) behoben.

Basisschnittstelle



(a) *Methoden nicht in der Basisschnittstelle* **(b)** *Methoden in der Basisschnittstelle*

Abbildung 5.5: *Schnittstellen der Komponenten vor und nach der Anpassung (am Beispiel der Hardware-Server)*

Die Server-Beschreibungssprache (SDL)

Nach der Entwicklung der Architekturanpassungen zur Integration der Spezifikationsdateien folgt im vorliegenden Abschnitt die Formulierung des einzusetzenden XML-Dialektes. Für die Instanzen dieses als Server-Beschreibungssprache (SDL[2]) bezeichneten Dialektes wird die Dateiendung „sdl“ verwendet, so dass die Dateien bereits an der Endung erkannt werden können.

XML-Dialekt

Verwendete Ressourcen Die Motivation zur Einführung der Server-Spezifikationen bestand darin, die Problematik bei der Komponenteninitialisierung zu beheben. Um diese Entwicklung abzuschließen, müssen die Abhängigkeitsinformationen in die Serverspezifikation aufgenommen werden. Dazu sind alle Informationen zu einem Server, die zuvor

Abhängigkeitsinformationen

[2] **S**erver **D**escription **L**anguage

in der globalen Konfigurationsdatei abgelegt waren, auf das XML-Schema abzubilden. Diese Information umfassen die folgenden Daten:

- Name der Ressource
- Typ der Ressource
- Namenskontext der Ressource

Diese Daten werden mit Name und Typ als Attribute im XML-Schema Datentyp `Resource` zusammengefasst. Zusätzlich erfolgt die Einführung eines weiteren Attributs `mandatory`, über welches es möglich ist festzulegen, inwiefern die Ressource für den Betrieb notwendig ist oder nicht. Der Namenskontext wird in Form einer Sequenz von XML-Tags aufgenommen. Dazu folgt zunächst die Angabe des Kontext-Pfades soweit dieser bei der Ressource verwendet wurde und im Anschluss daran die Referenz. Es ergibt sich die in Abbildung 5.6 dargestellte XML-Spezifikation.

Namenskontext

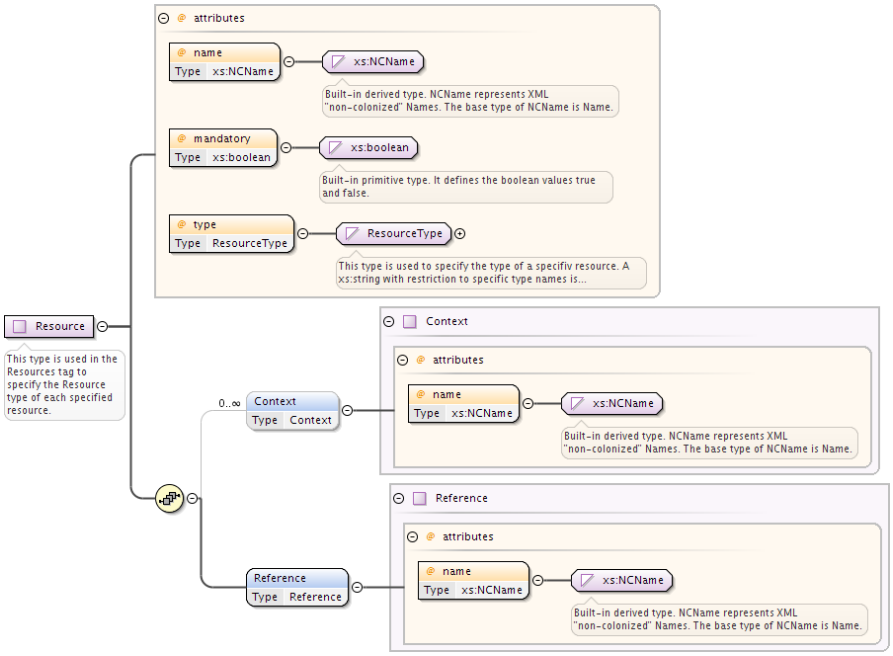


Abbildung 5.6: *XML Schema des Datentyps* `Resource`

Konfigurationswerte Neben den verwendeten Ressourcen wird die Möglichkeit geschaffen Konfigurationswerte in die Spezifikationen mit aufzunehmen. Diese sollen es ermöglichen, eine Serveranwendung mehrfach zu starten und dabei unterschiedliche Laufzeitkonfigurationen zu verwenden. Diese Funktionalität ermöglicht zum Beispiel die Verwendung von unterschiedlichen Kalibrierungsdaten für zwei Sensoren, die mit der selben Hardware-Server Komponente angesprochen werden.

Laufzeitkonfigurationen

Alle Konfigurationswerte werden als Schlüssel/Wert-Paar in der Spezifikation aufgenommen. Dabei gibt es vier gültige Datentypen: Zeichenkette, Wahrheitswert, Ganzzahl und Fließkommazahl. Die Festlegung der Werte in der Spezifikationsdatei erfolgt in Form von Attributen des XML-Tags `ConfigValue` wie Abbildung 5.7 zeigt.

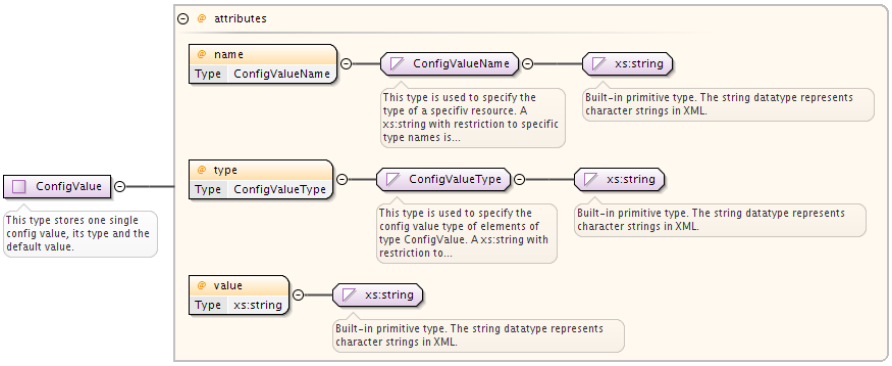


Abbildung 5.7: *XML Schema des Datentyps* `ConfigValue`

Aufbau der Spezifikation Das komplette XML-Schema der Komponentenspezifikation setzt sich als Sequenz aus den zuvor definierten Datentypen zusammen. Das Element, das die Wurzel der Spezifikationsdateien bildet, ist vom Datentyp `ServantDescription`. Es beinhaltet eine Sequenz mit beliebig vielen Ressourcen und beliebig vielen Konfigurationswerten. Zusätzlich dazu besitzt der Knoten das Attribut `name`, über das der Name der zugeordneten Komponente angegeben wird. Abbildung 5.8 zeigt das kom-

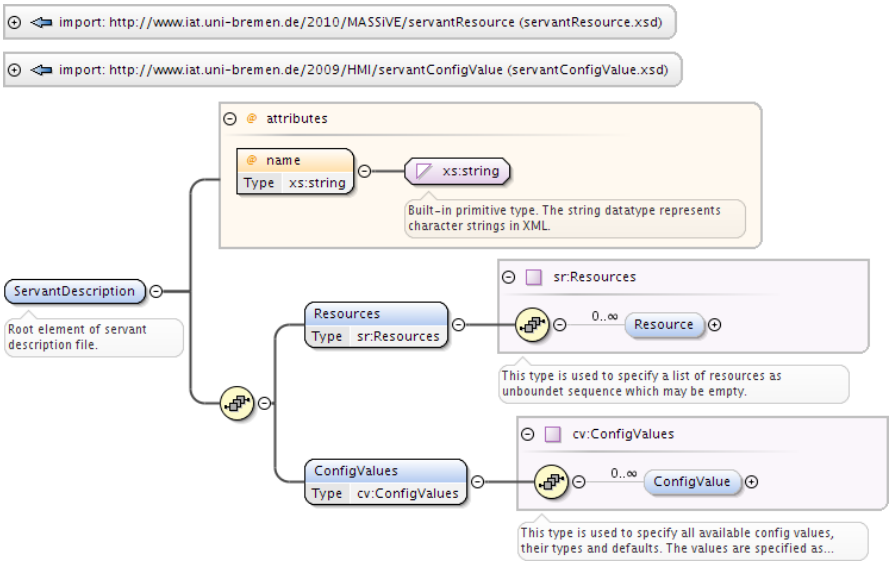


Abbildung 5.8: *XML Schema des Datentyps* `ServantDescription`

plette XML-Schema, das die Spezifikationssprache für MASSiVE Serverkomponenten

beschreibt. Alle Spezifikationen der unter Anwendung dieser Beschreibungssprache spezifizierten Ressourcen sind in Anhang A.2.1 zu finden.

5.2.2 Online Ressourcenerfassung

„launchd"-Konzepte

Zur Realisation der Erfassung von Systemkomponenten in der MASSiVE-Architektur lassen sich einige der Ideen von „launchd" auf die MMS übertragen, da die Vorgehensweise stark der in MASSiVE bereits angewandten ähnelt: Alle Komponenten werden zugleich gestartet und erst danach erfolgt der Aufbau des Kommunikationsnetzwerkes über die bereits hinlänglich beschriebene Initialisierungsmethode. Zur Laufzeit des Systems ließen sich weitere Konzepte von „launchd" nutzen, um der Anforderung 3.8 gerecht zu werden und den dynamischen Oberflächenaufbau in Bezug auf vorhandene Ressourcen zu ermöglichen. In diesem Zusammenhang wären da folgende Funktionalitäten zu nennen:

- Überwachung von Systemressourcen zur Laufzeit
- Laden von Erweiterungen sobald sie benötigt werden
- Entladen von Erweiterungen sobald sie nicht mehr benötigt werden
- Laden von Erweiterungen in Abhängigkeit vom Zustand der verwendeten Systemressourcen

Device-Detection-Server

Die ersten Untersuchungen zum Thema der Laufzeitüberwachung von Systemressourcen in MASSiVE werden in [Fre09] beschrieben. Die praktischen Ergebnisse dieser Untersuchungen stehen inzwischen in Form des entwickelten DDS zur Verfügung, welcher nach Anforderung 3.19 die Grundlage für die online Ressourcenerfassung bilden soll.

Lösungsansatz 5.4. *Die zur Akquise und Überwachung von Ressourcen in [Fre09] entwickelte Funktionalität wird in die MMS integriert.*

Statusüberwachung

Der DDS ist eine eigenständige Komponente, die unter Verwendung des Naming- oder Trading-Service (Naming-Service mit Einschränkungen, siehe Abschnitt 3.2.4) in der Lage ist, die Systemkomponenten online zu überwachen. Dazu können sich Anwendungen über die Schnittstelle des DDS mit einem Suchmuster registrieren, um zur Laufzeit über Statusänderungen informiert zu werden. Über die Suchmuster filtert der DDS alle gewünschten Komponenten aus seinen internen Statustabellen heraus und informiert die registrierten Komponenten sobald sich der Status von überwachten Ressourcen ändert. Die Statusänderungen werden dabei in die Gruppen `NEW_DEVICE`, `DELETED_DEVICE`, `GONE_ONLINE` sowie `GONE_OFFLINE` klassifiziert. Um diese Informationen für die dynamische Anzeige von Erweiterungen nutzen zu können ist es notwendig, beim Laden zu ermitteln, von welchen Ressourcen eine Erweiterung abhängig ist. Dann kann für jede Erweiterung ein Filter registriert werden, so dass die MMS vom DDS bei Statusänderungen informiert werden kann.

Lösungsansatz 5.5. *Die Schnittstelle des DDS wird in die MMS integriert.*

5.2.3 Abstraktion der Hardware-/Skill-Ebene

Nach Anforderung 3.17 ist den Erweiterungen der Zugriff auf Systemressourcen zu ermöglichen. Wie in Abschnitt 2.1.3 bereits beschrieben wurde, wird dieser Zugriff über die von den Komponenten exportierte CORBA-Schnittstelle realisiert. Der Verbindungsaufbau wird zuvor durch die Akquise der Ressource mittels Namenskontext realisiert. Ebenso wurde die Problematik bei der Verwendung von Namenskontexten in Abschnitt 3.2.4 ausführlich beschrieben. Auch diese Problematik kann durch Einsatz des DDS verhindert werden.

Ressourcenakquise

Verwendet man zur Akquise der Systemressourcen keine Namenskontexte, sondern eine eindeutige Identifikation der jeweiligen Ressource, so ist es möglich, diese Problematik gänzlich zu vermeiden. Zuvor muss lediglich die eindeutige ID identifiziert werden. In diesem Zusammenhang bietet der DDS die Funktionalität nach den so-genannten Repository-IDs zu suchen. Bei einer Repository-ID handelt es sich um eine Zeichenkette, die dem Klassennamen der exportierten CORBA-Schnittstelle, erweitert um ein paar Zusatzinformationen, entspricht. Dieser eignet sich ideal als eindeutige Kennung, da die Schnittstelle einer Ressource diese eindeutig identifiziert. Darüber hinaus ergeben sich folgende Vorteile:

Repository-ID

1. Eine Erweiterung benötigt keinerlei Informationen über den Namenskontext, mit dem eine Ressource am Lokalisierungsdienst registriert wurde, sie sucht lediglich nach einer Ressource, die eine bestimmte Schnittstelle implementiert.

2. Beim Suchen nach der Repository-ID erhält die Erweiterung eine Liste mit allen Ressourcen, die die entsprechende Schnittstelle implementieren. Dadurch ist es möglich auf unterschiedliche Ressourcen von gleichen Typ zuzugreifen, ohne deren Namenskontext zu kennen.

Um diese Funktionalität in der MMS zu integrieren, muss eine Lösung gefunden werden um die Vorgänge der Ressourcenakquise mittels Repository-ID so transparent wie möglich zu gestalten. Dazu wird eine Klasse eingesetzt, deren Funktionalität darin besteht, die Namenskontexte aller zur Verfügung stehenden Ressourcen zu einer vorgegebenen Repository-ID über den DDS zu ermitteln. Eine Instanz dieser Klasse nutzt dazu die Suchfunktionalität des DDS und speichert die Ergebnisse der Suche zwischen, so dass die Ergebnisse bei einer erneuten Anfrage bereits zur Verfügung stehen. Abbildung 5.9 zeigt den dazu entwickelten Aufbau.

Über die Methode `InterfaceIDToContext()` können Anfragen an den DDS delegiert werden. Als Argument muss der Methode die Repository-ID der gesuchten Ressource übergeben werden. Bei erfolgreicher Suche erhält der Aufrufer den Namenskontext der Ressource als Rückgabewert. War die Suche nicht erfolgreich, so wird ein ungültiger Kontext zurückgegeben. Sollten mehrere Ressourcen gefunden worden sein, so wird der Namenskontext der ersten gefundenen Ressource zurück gegeben. Optional kann der Methode `InterfaceIDToContext()` auch die als Referenzalternative bezeichnete Nummer der in diesem Fall zu liefernden Ressource übergeben werden. Dazu existiert darüber hinaus die Methode `GetNumberOfAlternatives()`, über die bereits im Vorfeld die Anzahl

Akquisevorgänge

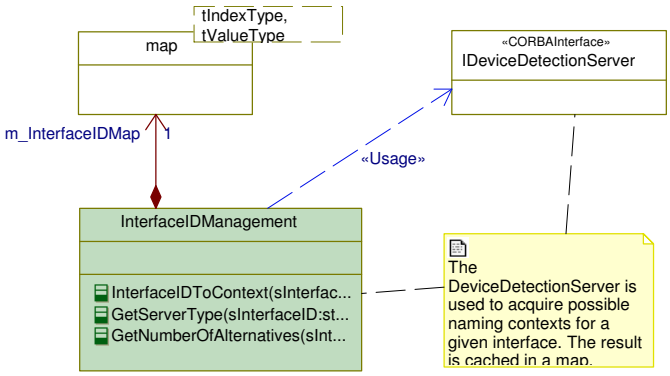


Abbildung 5.9: *Konvertierung einer Schnittstellen-ID über den DDS*

der zur Verfügung stehenden Ressourcen mit dem gesuchten Interface ermittelt werden kann. Die letzte Methode `GetServerType()` ermittelt lediglich den Typ einer Ressource.

5.3 Spezifikationssprachen der MMS

Im Zusammenhang mit der Entwicklung des XML-Dialektes zur Ressourcenspezifikation für die Komponenten der MASSiVE-Architektur wurde gezeigt, dass der Einsatz von XML eine Möglichkeit bietet, zur Laufzeit verifizierbare Spezifikationen zu erstellen. Diese Erkenntnis soll im folgenden dazu genutzt werden, den in Abschnitt 3.1 formulierten Anforderungen 3.9 und 3.11 gerecht zu werden.

5.3.1 Kommunikationssystem

Durch die Festlegung, dass alle Eingabegeräte nach Anpassung der MMS als Erweiterung zu integrieren sind, entfällt der Ereigniskanal als Kommunikationsmedium. Die einzige Verbindung von den Erweiterungen zu der MMS besteht in Form der Schnittstellen zwischen diesen Komponenten.

Ersatz des Ereigniskanals

Für die Realisation der Kommunikation der MMS mit den Erweiterungen und umgekehrt müssen daher zwei Schnittstellen zur Verfügung gestellt werden. Die erste wird von der MMS exportiert, um den Erweiterungen als Kommunikationskanal zur MMS zu dienen, und die zweite wird von den Erweiterungen exportiert, um den Kommunikationskanal von der MMS zu den Erweiterungen zu bilden. Bei nur einer Schnittstelle wäre die bidirektionale Kommunikation nur durch Polling (siehe [Wik10, Stichwort: „Polling (Informatik)“] realisierbar.

Lösungsansatz 5.6. *Um den bidirektionalen Datenaustausch zu ermöglichen, müssen die MMS und alle Erweiterungen eine Schnittstelle zur Kommunikation zur Verfügung stellen.*

Die Betrachtung des Nachrichtenaufkommens darf bei der Entwicklung dieses Kommunikationssystems nicht außer Acht gelassen werden. Die notwendigen Mechanismen in der MMS müssen darauf abgestimmt sein alle ein- und ausgehenden Nachrichten in endlicher Zeit zu verarbeiten. Insbesondere der Einsatz von Erweiterungen für Eingabegeräte, die als aktive Objekte realisiert werden, kann zu einem hohen und dazu noch asynchronem Nachrichtenaufkommen führen. Aus diesem Grund sind alle Kommunikationsanfragen in der MMS zwischenzuspeichern und dann sequentiell zu bearbeiten. Darüber hinaus ist zu überprüfen, ob die zu verarbeitenden Anfragen gültig sind, um so die robuste Verarbeitung zu gewährleisten.

Nachrichtenaufkommen

Lösungsansatz 5.7. *Es wird eine Komponente innerhalb der MMS realisiert, die über einen Zwischenspeicher verfügt und in der Lage ist, unabhängig von den übrigen Vorgängen in der MMS alle eingehenden Anfragen (engl. Requests) sequentiell zu bearbeiten*[3]*. Diese als Request-Dispatcher bezeichnete Komponente wird dazu als aktives Objekt realisiert. Zur Validierung der eingehenden Anfragen ist eine Spezifikation zu entwerfen, mittels derer alle gültigen Anfragen definiert werden können.*

Nach der Entwicklung der Lösungsansätze ist die Grammatik für die XML-basierte Anfragespezifikation zu formulieren. Dazu soll zunächst der Aufbau einer Anfrage betrachtet werden.

Eine Anfrage besteht aus einer ID, über die die Anfrage eindeutig zu identifizieren ist. Der Aufbau dieser ID wird aufgrund von Kompatibilitätsgründen an den Aufbau der bereits in der MMS zum Einsatz kommenden Deskriptoren (siehe [Cyr05]) angelehnt. Angepasst werden lediglich die fünf Elemente aus denen die ID besteht, deren Aufbau Abbildung 5.10 entnommen werden kann. Das erste Element ist eine Zeichenkette mit der die Anfrage benannt wird. Erlaubt sind alle Zeichenketten, die dem regulären Ausdruck `([a-zA-Z0-9_])*` genügen. Das zweite Element dient der Zielangabe für eine Anfrage, welche bei der gerichteten Kommunikation (siehe Abschnitt 5.4.1) zum Einsatz kommt. Auch hier sind Zeichenketten erlaubt, die dem regulären Ausdruck `([a-zA-Z0-9_])*` genügen. Das nächste Element bleibt unverändert und gibt die Ebene innerhalb der MASSiVE-Architektur an, innerhalb derer die Anfrage zum Einsatz kommt. Hier sind die gültigen Werte `HMI`, `EEL`, `TMP` oder `SEQ`. Relevant für die Anfragen der MMS ist hierbei jedoch nur die Angabe `HMI`. Das vorletzte Element beschreibt den Typ der Anfrage. Dieser ist entweder `Info` für Meldungsdaten, `Data` für beliebige Daten oder `Command` für Steuerungskommandos. Das letzte Element spezifiziert den Modus. Handelt es sich um die erste Anfrage einer Kommunikationsfolge, so ist der Wert `Request`, ansonsten `Reply`.

Kommunikation über Anfragen

Nach der Definition des XML-Schema für die Spezifikation der Anfrage-IDs folgt nun die Festlegung der Spezifikation für die transportierten Daten. Jede Anfrage kann beliebig viele Daten transportieren. Jedes mögliche Datum wird dabei in Form des XML-Tags `Date` spezifiziert, wie Abbildung 5.11 zeigt. Das XML-Tag besitzt fünf Attribute, über die zunächst ein Name, der Datentyp und die Datenrichtung festzulegen ist. Bei den übrigen Attributen handelt es sich um Wahrheitswerte. Das erste gibt an, ob das aktuelle Datum

Abbildung auf Spezifikation

[3] Im folgenden wird nur noch vom deutschen Begriff „Anfrage“ Gebrauch gemacht.

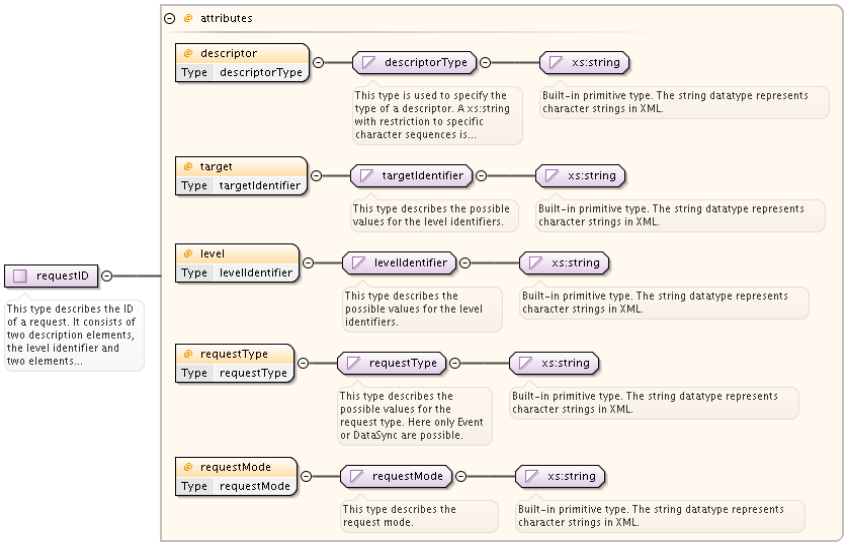


Abbildung 5.10: *XML Schema des Datentyps* `requestID`

für die Erstellung einer Datenbasis serialisiert und verwendet werden soll. Die genauen Hintergründe zu dieser Funktionalität werden im Kapitel 7 ausführlich beschrieben. Das letzte Attribut legt fest, ob das aktuelle Datum für die Anfrage optional ist oder nicht.

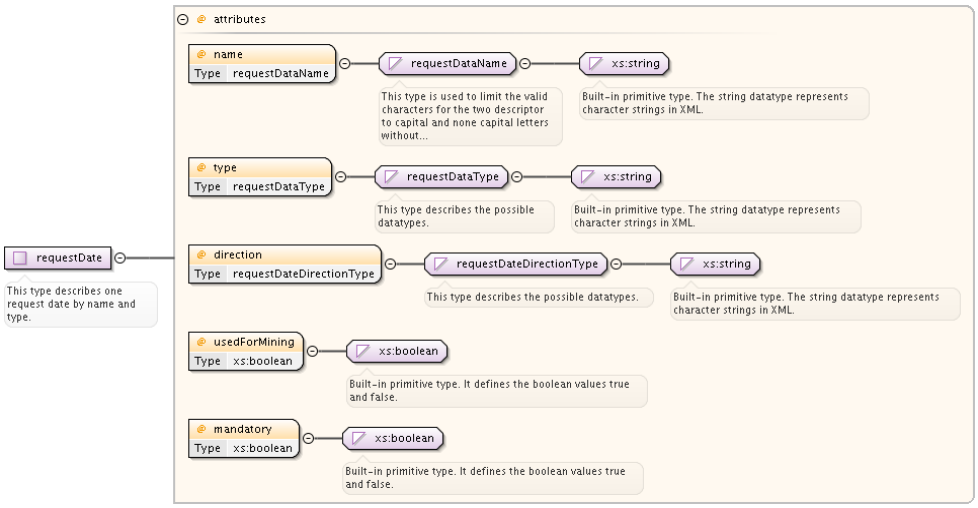


Abbildung 5.11: *XML Schema des Datentyps* `requestDate`

Anfrageidentifikation

Bevor die bereits definierten XML-Schemata im Gesamtschema für die Spezifikation der möglichen Anfragen zusammengefasst werden, erfolgt die Zusammenfassung der bereits definierten ID, einer kurzen Beschreibung und einer beliebigen Anzahl der zuvor definierten Daten zu einer Anfrage im XML-Tag `Request`, wie Abbildung 5.12 zeigt.

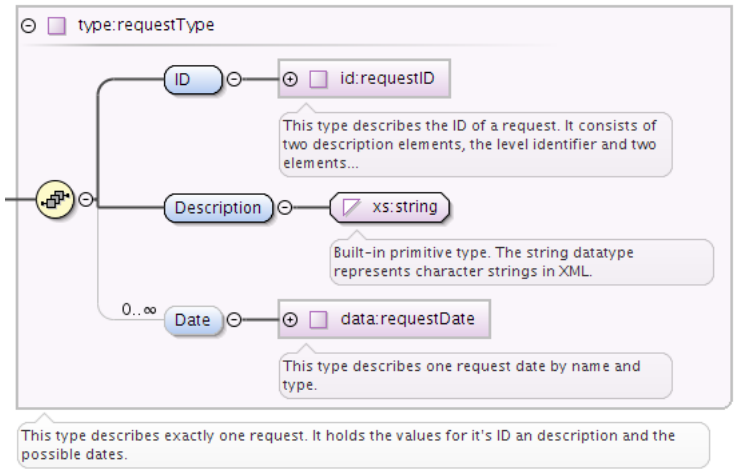


Abbildung 5.12: *XML Schema des Datentyps* `requestType`

Der letzte Schritt ist die Formulierung des Gesamtschemas für die Definition von gültigen Anfragen, welches aus einer unbegrenzten Sequenz besteht. Das Wurzelelement der Spezifikation wird durch das XML-Tag `RequestList` repräsentiert. Abbildung 5.13 zeigt die gesamte Grammatik der Spezifikation von Anfragen für die MMS.

Gesamtschema

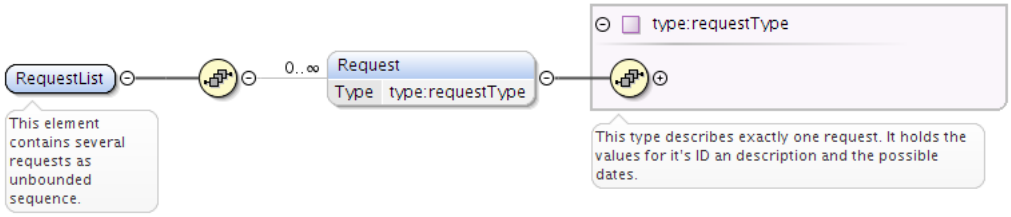


Abbildung 5.13: *XML Schema des Datentyps* `RequestList`

5.3.2 Internationalisierung

Die in Anforderung 3.11 formulierte Notwendigkeit von uneingeschränkter Internationalisierbarkeit der MMS und all ihrer Erweiterungen erfordert weitere Anpassungen. Vor der Festlegung dieser Anpassungen erfolgt zunächst ein Blick auf die derzeitige Funktionalität. Der von der Qt-Bibliothek zur Verfügung gestellte Mechanismus zur Übersetzung von Anwendungen basiert auf der Kapselung der (für den Nutzer) sichtbaren Zeichenketten über die globale Methode `Tr()`. Nach Abschluss der Entwicklungen können alle `Tr()`-Aufrufe mit Hilfe eines externen Werkzeugs aus den Quelltexten extrahiert und in eine so-genannte Übersetzungsdatei überführt werden. Diese bildet die Grundlage für den Qt-Linguist, eine weitere externe Anwendung, die die offline Übersetzung aller zuvor extrahierter Zeichenketten erlaubt. Da die gesamte Übersetzung auf der Methodenextraktion basiert wird deutlich, warum diese Vorgehensweise nicht mit Zeichenketten umgehen kann, die zur Laufzeit generiert werden (siehe Abschnitt 3.1.6).

Übersetzung der sichtbaren Zeichenketten

online Übersetzung

Aus diesen Gründen muss für das Vorgehen bei der Übersetzung der MMS eine andere Methodik eingesetzt werden. Da in Teilen der MMS die Zeichenketten bereits im Methodenaufruf `Tr()` eingefasst sind, soll die angepasste Lösung auf dieser Grundidee aufbauen. Das reduziert die notwendigen Anpassungen auf ein Minimum. Im Gegensatz zur Lösung aus der Qt-Bibliothek soll die Erstellung der Übersetzungsdatei jedoch nicht offline aus den Quellen, sondern online zur Laufzeit erzeugt werden. Das ermöglicht es auch Texte übersetzen zu lassen, die zur Laufzeit erzeugt werden. Der Ablauf sieht folgende Schritte vor:

1. Alle Zeichenketten, die für den Benutzer sichtbar sind, werden in den überschriebenen Methodenaufruf `Tr()` eingefasst. Die Methode erhält als Argument die zu übersetzende Zeichenkette und gibt als Ergebnis die übersetzte Zeichenkette zurück.

2. Bei jedem Aufruf der `Tr()`-Methode wird überprüft, ob die gewünschte Übersetzung zur Verfügung steht. Ist das nicht der Fall, so wird die zu übersetzende Zeichenkette abgespeichert.

3. Beim Beenden der MMS werden alle fehlenden Übersetzungen in eine externe Datei geschrieben, so dass die Übersetzungsdatei automatisch zur Laufzeit erstellt wird.

4. Ein Übersetzer kann auf Basis der automatisch erstellten Übersetzungsdatei fehlende Übersetzungen ergänzen.

Die Liste aller Übersetzungen, die als Quelle für die globale `Tr()`-Methode dient, wird in Form eines XML-Dialektes abgelegt. So ist es auch hier möglich, die Gültigkeit der Datei zur Laufzeit zu validieren. Die Festlegung der dafür benötigten Grammatik gestaltet sich recht einfach, wie Abbildung 5.14 zeigt.

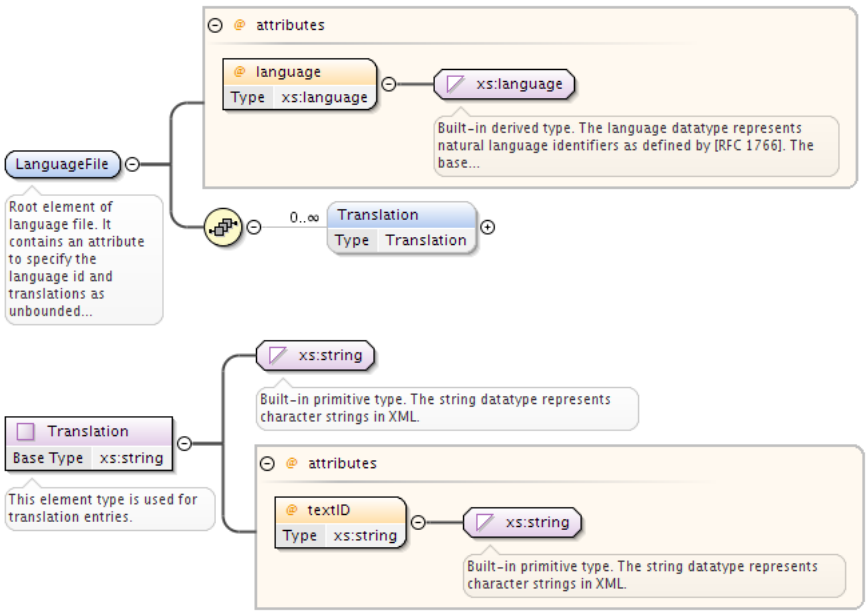


Abbildung 5.14: *XML Schema des Datentyps* `languageFile`

Damit die Übersetzungen unabhängig voneinander in unterschiedlichen Sprachen gepflegt werden können, soll später für jede Sprache genau eine XML-Instanz zum Einsatz kommen. Jede dieser Instanzen enthält als Wurzelelement das XML-Tag `LanguageFile` mit dem Attribut `language`. Letzteres gibt die Zielsprache als ISO-Code (siehe [ISO06b]) an. Die einzelnen Übersetzungen folgen dann als Kindelemente des Wurzelknotens in Form einer Sequenz von XML-Tags des Typs `Translation`. Diese XML-Tags besitzen das Attribut `TextID`, über das die Übersetzung referenziert wird. Es handelt sich dabei um die Zeichenkette in der Ursprungssprache. Mit dem Wert des XML-Tags `Translation` wird die Übersetzung festgelegt.

XML-Instanz für Übersetzungen

5.4 Modellbasierte Entwicklung des Rahmenwerks

Im Anschluss an die Definition der für die MMS zum Einsatz kommenden XML-basierten Spezifikationssprachen widmet sich der vorliegende Abschnitt dem Entwurf der Kernfunktionalitäten der MMS.

Nachfolgend werden zunächst die Funktionalitäten beschrieben, auf denen das in Kapitel 6 entwickelte Rahmenwerk für Erweiterungen aufbauen wird. Der zweite Abschnitt beschreibt die Vorgänge und Methodiken zur Aktivierung und Deaktivierung von Erweiterungen. Der letzte Abschnitt widmet sich der Realisierung von Benutzerinteraktionen.

5.4.1 Kernfunktionalitäten für Erweiterungen

Die Grundlage der nach [Mar99] beschriebenen Methodik zur Realisierung eines Erweiterungsrahmenwerkes ist die Definition des Erweiterungsvertrages. Dieser wird im Zusammenhang mit der Anpassung der MMS durch die noch zu spezifizierenden Schnittstellen gebildet. Nach Anforderung 3.6 sind sowohl die Eingabegeräte als auch alle weiteren Funktionalitäten zur Steuerung des Systems in Form von Erweiterungen zu realisieren. Diese zwei Arten von Erweiterung unterscheiden sich lediglich darin, dass die Erweiterungen für die Integration von Eingabegeräten keine grafische Komponente besitzen. Ihre Interaktion mit dem Benutzer erfolgt ausschließlich durch das externe Eingabegerät. Aus diesem Grund ist es notwendig, die Schnittstelle für diese zwei Grundarten von Erweiterungen unterschiedlich auszuführen. Dennoch besitzen beide Arten auch gemeinsame Attribute und Funktionalitäten, weshalb der in Abbildung 5.15 dargestellte Aufbau der Basisklassen gewählt wird.

notwendige Erweiterungsarten

Zur einfachen Organisation wird für alle Erweiterungen eine ID in Form einer Zeichenkette festgelegt, die als Name bezeichnet wird. Die Abfrage dieser ID erfolgt über die Methode `GetID()` der Basisschnittstelle aller Erweiterungen (die Klasse `IViewPlugin`). Des Weiteren erhält die Schnittstelle die Methode `SetExposedInterfaces()`, über die den Erweiterungen die von der MMS exportierten Schnittstellen weitergereicht werden.

ID zur Identifikation von Erweiterungen

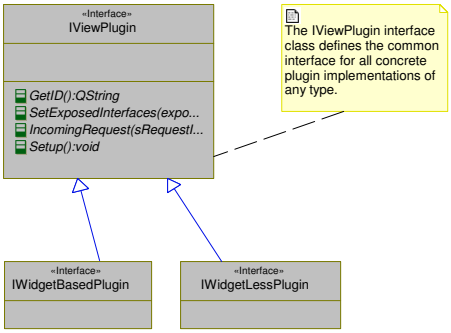


Abbildung 5.15: *Die Basisklasse der Schnittstellen definiert die Basismethoden aller Erweiterungen.*

Über die Methode `IncomingRequest()` wird die Kommunikation von der MMS mit den Erweiterungen realisiert. Alle Anfragen werden darüber an die entsprechenden Erweiterung weitergeleitet. Die letzte Methode `Setup()` dient der Initialisierung von Erweiterungen. Sie wird direkt nach dem Laden einer Erweiterung durch die MMS aufgerufen.

Interfacecontainer

Die Übertragung der Schnittstellen von der MMS an die Erweiterungen erfolgt zur Vereinfachung von späteren Anpassungen in einer als `InterfaceContainer` bezeichneten Struktur. Diese beinhaltet Zeiger auf alle verfügbaren Schnittstellen. Für nicht verfügbare Schnittstellen ist der Zeiger als ungültig markiert. Abbildung 5.16 zeigt den Entwurf.

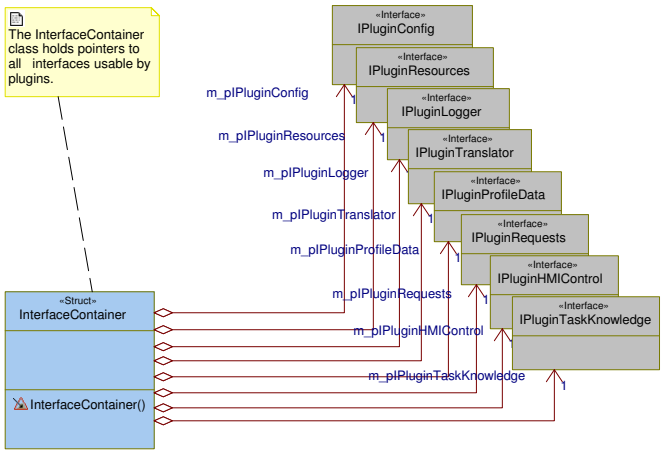


Abbildung 5.16: *Der Schnittstellen-Container verwaltet die Zeiger auf durch Erweiterungen verwendete Schnittstellen.*

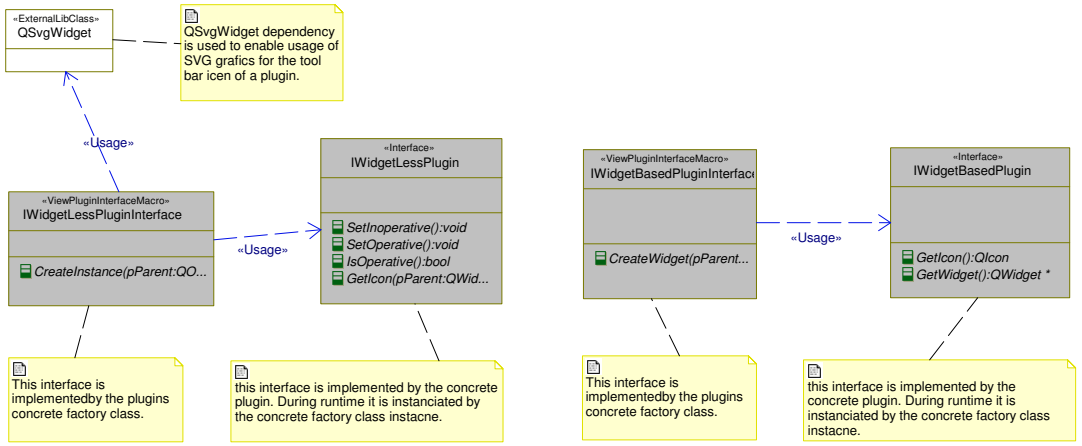
Basisschnittstellen

Die zwei Basisschnittstellen der Erweiterungsgrundtypen exportieren spezifische Funktionalitäten an die MMS. Bezogen auf die Erweiterungen für Eingabegeräte sind das vier Methoden. Die ersten zwei Methoden `SetOperative()` und `SetInoperative()` wurden aus den bestehenden Komponenten für Eingabegeräte übernommen. Sie dienen der Aktivierung und Deaktivierung des entsprechenden Eingabegerätes. Bei Eingabegeräten, die als aktive Objekte implementiert werden, sind sie darüber hinaus für das Starten

und Stoppen des entsprechenden Threads verantwortlich. Die Methode `IsOperative()` gibt Auskunft über den aktuellen Status des Eingabegerätes. Die letzte Methode dient der Rückgabe eines Piktogramms, welches durch die MMS angezeigt werden kann. Alle Piktogramme sind im SVG[4]-Format zu erstellen und fest zu den entsprechenden Erweiterungen zu linken. Der Entwurf ist in Abbildung 5.17a dargestellt.

Die Erweiterungen mit grafischer Komponente besitzen in ihrer Schnittstelle nur zwei exportierte Methoden. Die erste Methode dient wiederum der Rückgabe eines Piktogramms, welches von der MMS zur Anzeige gebracht werden kann, um die Navigation in der Benutzeroberfläche zu vereinfachen. Bei der letzten Methode handelt es sich um eine Fabrikmethode nach dem gleichnamigen, in Abschnitt 4.1.1 vorgestellten Entwurfsmuster. Diese erzeugt zur Laufzeit dynamisch die grafische Komponente der Erweiterung, um diese in der MMS darzustellen. Abbildung 5.17b zeigt diesen Entwurf.

Entwurfsmuster: Fabrik



(a) *Erweiterungen ohne grafische Komponente* **(b)** *Erweiterungen mit grafischer Komponente*

Abbildung 5.17: *Schnittellen die als „Plugin-Contract“ von Erweiterungen implementiert werden müssen*

Die zuvor beschriebenen Schnittstellen repräsentieren die Basisschnittstellen der konkreten Erweiterungen. Damit diese auch zur Laufzeit in die MMS integriert werden können, werden darüber hinaus noch für jeden Grundtyp der Erweiterungen konkrete Fabrikklassen benötigt. Die Instanzen dieser Klassen sind in der Lage, die konkreten Erweiterungen über eine Fabrikmethode zu erzeugen. Dazu zeigt Abbildung 5.17 neben den Basisschnittstellen der konkreten Erweiterungen auch noch die Basisschnittstellen der konkreten Fabriken für die Erweiterungstypen. Diese sind von jeder Erweiterung zu implementieren.

Nach der einleitenden Festlegung der Basisstruktur für das Rahmenwerk zur Integration von Erweiterungen folgt in den nächsten Abschnitten die Festlegung der Schnittstellen, die von der MMS an die Erweiterungen exportiert werden, sowie die dahinter liegende

[4] **S**calable **V**ector **G**raphics

Funktionalität in der MMS. Viele der in den folgenden Abschnitten beschriebenen Lösungsansätze basieren auf der Festlegung einer Spezifikationssprache für Erweiterungen, mittels derer Metadaten zu jeder Erweiterung festgelegt werden können. Diese Spezifikationssprache wird in Kapitel 6 entwickelt.

Zugriff auf Konfigurationsdaten

Zur Ablage von Konfigurationsdaten wird in der MASSiVE-Architektur ein Mechanismus eingesetzt, der von der Qt-Bibliothek zur Verfügung gestellt wird. Dieses System basiert auf einer Konfigurationsdatei in Textform. Auf diese wurde in den vergangenen Abschnitten bereits als globale Konfiguration verwiesen. In dieser Konfiguration können Daten in Form von Schlüssel/Wert-Paaren abgelegt werden. Nach Anforderung 3.7 muss eine solche Funktionalität auch den Erweiterungen zur Verfügung gestellt werden.

Konfigurationsschnittstelle

Da die Erweiterungen keinen direkten Zugriff auf die globale Konfiguration haben, ist es notwendig den Zugriff über die MMS zu delegieren. Dazu hat die MMS eine Schnittstelle zu exportieren, mittels derer die Erweiterungen Konfigurationswerte lesen können. Damit die MMS in der Lage ist die Konfigurationswerte der unterschiedlichen Erweiterungen zu liefern, müssen ihr diese vorliegen. Dazu werden alle möglichen Konfigurationswerte einer Erweiterung über die später ausführlich eingeführte XML-basierte Spezifikation angegeben. Da diese Spezifikation für die MMS nur lesbar ist, muss weiterhin eine Möglichkeit der Änderung von Konfigurationswerten realisiert werden.

Lösungsansatz 5.8. *Über eine Spezifikation liefert jede Erweiterung die von ihr verwendeten Konfigurationswerte und deren Standardwerte. Diese werden durch die MMS in die globale Konfiguration überführt, um dort verwaltet werden zu können. Über eine Schnittstelle bietet die MMS den Erweiterungen lesenden Zugriff auf ihre und globale Einstellungen.*

Der Lösungsansatz besteht also in einer hybriden Lösung aus statischer Spezifikation durch die Erweiterungen und dynamischer Verwaltung in der globalen Konfiguration. Abbildung 5.18 verdeutlicht diesen Lösungsansatz.

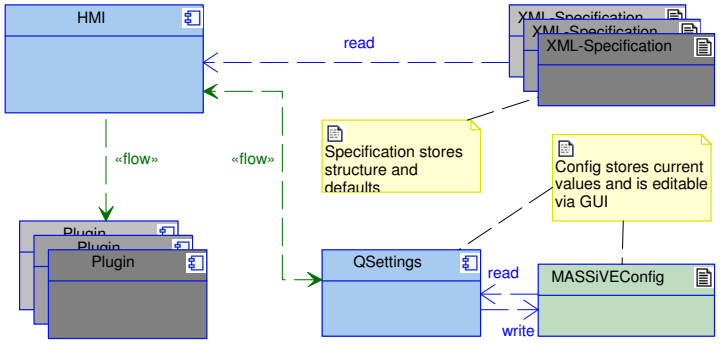


Abbildung 5.18: *Zugriff auf die Konfigurationsdaten*

Bei dieser Vorgehensweise erfolgt beim Start der MMS die Auswertung der Erweiterungsspezifikationen. Alle Konfigurationswerte von Erweiterungen, die noch nicht erfasst wurden, werden in die globale Konfiguration übernommen. Im Anschluss daran können die Erweiterungen auf Basis der Werte in der globalen Konfiguration arbeiten. Diese Vorgehensweise besitzt den weiteren Vorteil, dass so die in der MASSiVE-Architektur zum Einsatz kommende grafische Konfigurationsanwendung auch für die Einstellungen der Erweiterungen zum Einsatz kommen kann.

Spezifikation und globale Konfiguration

Für den Zugriff auf die globale Konfiguration kommt eine Instanz der von der Qt-Bibliothek zur Verfügung gestellten Klasse `QSettings` zum Einsatz. Diese erlaubt das Lesen und Schreiben der Konfiguration. Um den Erweiterungen den Zugriff darauf zu geben, erfolgt die Definition der in Abbildung 5.19 dargestellten Schnittstelle.

Lesen und Schreiben durch MMS

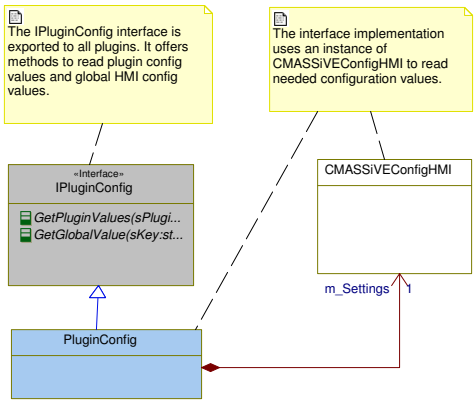


Abbildung 5.19: *Die Implementation der Schnittstelle zum Lesen und Schreiben von Konfigurationsdaten durch Erweiterungen.*

Die Schnittstelle `IPluginConfig` besitzt lediglich zwei Methoden. Die erste Methode `GetPluginValues()` dient dem Lesen aller Konfigurationswerte für die Erweiterung. Die zweite Methode `GetGlobalValue()` ermöglicht den Erweiterungen den lesenden Zugriff auf globale Werte. Insgesamt wurde bei beiden Methoden auf den schreibenden Zugriff verzichtet, um in jedem Fall die Konsistenz der Konfiguration sicherstellen zu können. Implementiert wird die Schnittstelle in einer Instanz der Klasse `PluginConfig`, die dazu eine Instanz der Klasse `CMASSiVEConfigHMI` einsetzt. Letztere kapselt den Zugriff auf eine Instanz der Klasse `QSettings`.

lesender Zugriff für Erweiterungen

Resourcenverwaltung

Im Zusammenhang mit der Ressourcenverwaltung der MMS wurden in Abschnitt 5.2 einige Anpassungen an der bestehenden Vorgehensweise durchgeführt. Das Resultat der dynamischen Ressourcenerfassung ist jetzt auf eine Schnittstelle für die Erweiterungen abzubilden. Die in Abschnitt 5.2.3 entwickelte Klasse `InterfaceIDManagement` soll nun eingesetzt werden, um den Erweiterungen die Ressourcenakquise zu ermöglichen.

Schnittstelle zur Ressourcenerfassung

Dazu benötigt die MMS zunächst Informationen darüber, von welchen Ressourcen eine konkrete Erweiterung abhängig ist. Genau wie die Informationen zu Konfigurationswerten werden auch diese Metadaten in die Spezifikation der Erweiterungen aufgenommen. Dazu können die bereits in Abschnitt 5.3.1 entwickelten Datentypen eingesetzt werden, wie später in Abschnitt 6.3.1 beschrieben wird.

Durch die Aufnahme der benötigten Ressourcen in die Spezifikation einer Erweiterung stehen diese Daten der MMS zur Verfügung. Benötigt eine Erweiterung Zugriff auf eine Ressource, so akquiriert sie diese durch Angabe des Namens, unter dem die entsprechende Ressource in ihrer Spezifikation abgelegt ist. Die MMS kann daraufhin in der Spezifikation nachschlagen, um die benötigte Schnittstelle zu erfahren. Über diese Information erfragt sie den Namenskontext von der Instanz der Klasse `InterfaceIDManagement` um daraus die Objektreferenz zu erzeugen, welche an die Erweiterung weitergeleitet wird. Der Entwurf der Implementation dieser Funktionalität ist in Abbildung 5.20 dargestellt.

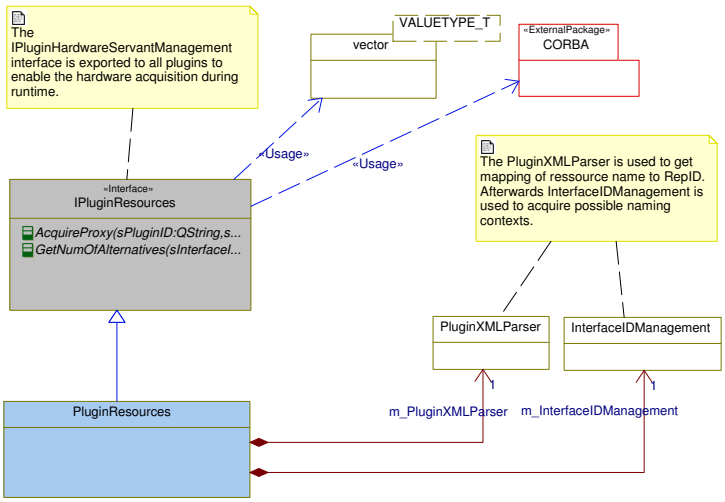


Abbildung 5.20: *Die Implementation der Schnittstelle zur Akquise von Systemressourcen durch Erweiterungen.*

Akquise und Zugriff ohne Abhängigkeit in der MMS Schnittstelle

Da die MMS keinerlei Abhängigkeit zu bestimmten Ressourcen aufweisen darf, ist es notwendig in der Schnittstelle darauf zu achten, dass nur Datentypen eingesetzt werden, die eine gemeinsame Basis aller Ressourcen darstellen. Aus diesem Grund kommt als Parameter für die Rückgabe der Objektreferenz von akquirierten Ressourcen an die Erweiterung eine Instanz der Klasse `CORBA::Object` zum Einsatz. Bei dieser handelt es sich um die Basisklasse aller CORBA-Schnittstellen. Die zur Akquise von den Erweiterungen einzusetzende Methode ist `AcquireProxy()`. Als Argumente müssen die ID der Erweiterung, der Name der gewünschten Ressource, der Ausgabeparameter für die Objektreferenz und die Nummer der Referenzalternative übergeben werden. Die zweite Methode ermittelt die Anzahl der zur Verfügung stehenden Ressourcen des gewünschten Typs.

Ausgabe von Meldungen und Logging

Die Möglichkeit zur Ausgabe von Nutzermeldungen sowie das Mitschreiben von Systemmeldungen ist nach Anforderung 3.10 (siehe Seite 39) auch den Erweiterungen zu ermöglichen. Dazu werden für die zwei Arten von Meldungen unterschiedliche Methodiken zum Einsatz kommen. Die Ausgabe von Nutzermeldungen ist von den Erweiterungen über die in Abschnitt 5.4.1 beschrieben Schnittstelle zur Steuerung der MMS zu initieren. Diese erzeugt dann zur Laufzeit einen Informationsdialog und präsentiert diesen dem Benutzer.

Die Systemmeldungen werden über den Logging-Mechanismus der MMS ausgegeben. Dieser wird durch Vererbung der Schnittstellenimplementation von der `LoggingBase` Klasse integriert. Diese Schnittstelle besteht aus drei Methoden über die Meldungen ausgegeben werden können. Zwei Methoden (`LogEvent()` mit zwei unterschiedlichen Signaturen) dienen diesem Zweck. Der Unterschied der beiden Methoden besteht in der Funktionsweise bei der Ausgabe: die erste führt zur direkten Ausgabe der Meldung und die zweite schreibt die Meldung in einen Zwischenspeicher. Mit letzterer können so längere Meldungen über mehre Methodenaufrufe zusammengesetzt werden. Interessant ist das vor allem bei der Ausgabe einer Meldung über Schleifenkonstrukte. Mit der dritten Methode `LogStream()` wird der Inhalt des Zwischenspeichers ausgegeben und im Anschluss daran gelöscht. Abbildung 5.21 zeigt den Entwurf der Schnittstelle.

Schnittstelle für System- und Nutzermeldungen

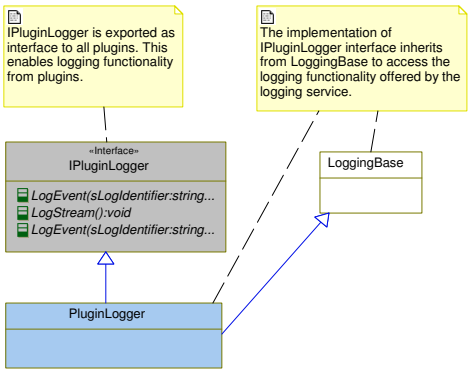


Abbildung 5.21: *Die Implementation der Schnittstelle zur Ausgabe von Meldungen durch Erweiterungen.*

Internationalisierbarkeit

Die im Abschnitt 3.1.6 beschriebene Anpassung der Methodiken zur Internationalisierbarkeit definierte zunächst nur die Vorgehensweise bei der Übersetzung von Zeichenketten in der MMS. Dazu wurde eine XML-basierte Spezifikation erstellt, mittels derer die Übersetzungen definiert werden können. Diese Vorgehensweise wird auch bei der Übersetzung der Erweiterungen zum Einsatz kommen. Dazu wird die Spezifikation für die

Übersetzungen zu den Metadaten der Erweiterungen hinzugefügt. Die Festlegung der in der Spezifikation eingesetzten Grammatik wird später in Abschnitt 6.3.1 erfolgen.

Tr()-Methode

Die Schnittstelle, über die von der MMS die Funktionalität zur Übersetzung von Zeichenketten an die Erweiterungen exportiert wird, besteht aus zwei Methoden. Die erste ist die `Tr()`-Methode, die die gleiche Funktionalität wie das Pendant in der MMS besitzt und auch in den Erweiterungen zu verwenden ist. Zusätzlich besteht hier noch die Möglichkeit sämtliche zur Verfügung stehenden Übersetzungen einer Erweiterung mit der Methode `TrPlugin()` zu lesen. Abbildung 5.22 zeigt den Entwurf der Schnittstelle.

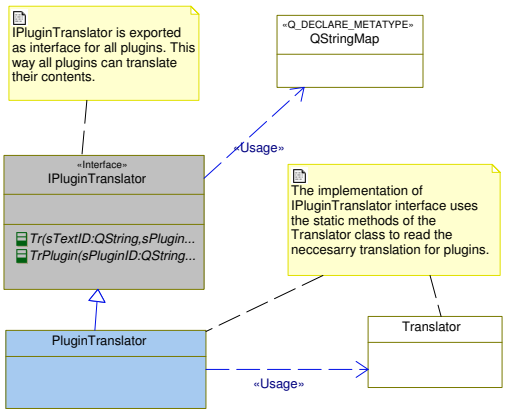


Abbildung 5.22: *Die Implementation der Schnittstelle zur Akquise von Übersetzungsdaten durch Erweiterungen.*

Die Implementierung der notwendigen Funktionalitäten erfolgt in der Implementationsklasse der Schnittstelle durch die Verwendung einer Instanz der Klasse `Translator`, die auch in der MMS für die Übersetzung eingesetzt wird.

Lesen und Schreiben generischer Daten

Erstellung von Datenbasen

Eine Grundvoraussetzung für die in Abschnitt 3.1.8 beschriebene Möglichkeit zur Realisation von Adaptivität in der MMS besteht in der Erstellung von Datenbasen. Diese bilden die Grundlage für die Integration von Methoden aus dem Bereich des in Abschnitt 2.4.1 beschriebenen maschinellen Lernens. Um den Entwicklern von Erweiterungen die Möglichkeit des Anlegens von Datenbasen zu ermöglichen, ist nach Anforderung 3.14 und 3.16 eine Schnittstelle in der MMS zu realisieren, die diese Funktionalität exportiert. Abbildung 5.23 zeigt den Entwurf dieser Schnittstelle.

Da die Daten über die Laufzeitgrenzen der MMS hinweg gespeichert werden müssen, kommt als Speicherort innerhalb der MASSiVE-Architektur nur die Systemdatenbank

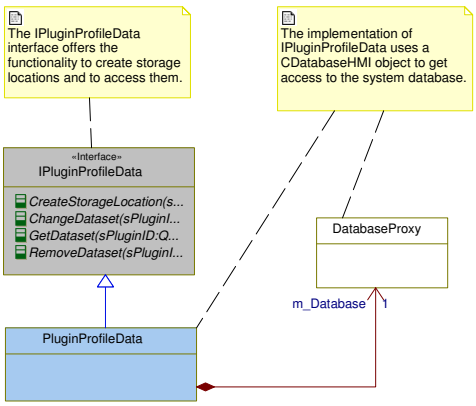


Abbildung 5.23: *Die Implementation der Schnittstelle zum Lesen und Schreiben von Profildaten und Nutzungsstatistiken durch Erweiterungen.*

in Frage. Dazu setzt die Implementationsklasse der Schnittstelle eine Instanz der Klasse `DatabaseProxy` zur Realisation des Datenbankzugriffs ein.

Die Schnittstelle besitzt vier Methoden. Die erste Methode `CreateStorageLocation()` dient dem Anlegen einer Datenbasis. Dazu ist neben der ID der Erweiterung eine ID für die Datenbasis und die Anzahl der Spalten festzulegen. Die ID der Erweiterung wird benötigt, damit die Datenbasis der korrekten Erweiterung zugeordnet werden kann. Die ID für die Datenbasis selbst dient dem späteren Zugriff und der Realisierung von mehreren Datenbasen für eine einzelne Erweiterung. Um die Organisation der Datenbasen zu vereinfachen, wird dabei auf eine Tabellenstruktur zurück gegriffen. Diese lässt sich direkt auf die Struktur der eingesetzten relationalen Datenbank abbilden, um so komplexe Konvertierungen zu vermeiden (siehe Abschnitt 7.1.3).

relationale Datenbank

Die Methode `ChangeDataset()` dient dem Anlegen eines neuen Datensatzes oder dem Ändern eines bestehenden Datensatzes und einer bestimmten Datenbasis. Über die Methode `GetDataset()` kann ein Datensatz gelesen und über die Methode `RemoveDataset()` gelöscht werden. Allen Methoden ist dabei die ID der Erweiterung, die ID der Datenbasis sowie die Nummer des Datensatzes zu übergeben. Darüber hinaus erhält die erste Methode die Daten in Form einer Liste der Werte aller Spalten und die zweite Methode einen Wahrheitswert, über den festgelegt wird, ob nur die Daten des aktuellen Nutzers relevant sind oder alle Daten, die in der Datenbasis zur Verfügung stehen.

Kommunikation zwischen Erweiterungen

Abschnitt 5.3.1 beschäftigte sich ausführlich mit dem in Anforderung 3.9 beschriebenen Kommunikationssystem der MMS. Bei diesen Betrachtungen wurde die Kommunikation zwischen den Erweiterungen jedoch außer Acht gelassen. Zur Realisation dieser Art der Kommunikation ist es nicht möglich entsprechende Schnittstellen auf beiden Seiten zu definieren, da mehr als zwei Kommunikationspartner miteinander verbunden werden

bidirektionale Kommunikation

müssen. Die einzige Möglichkeit bestünde in der Auslegung als Bussystem bei dem die Kommunikationsdaten von einer Erweiterung zur nächsten weitergeleitet werden bis eine Erweiterung die Daten verarbeitet. In dieses Bussystem könnte auch die MMS mit integriert werden, so dass keine weiteren Schnittstellen für die Kommunikation zwischen der MMS und den Erweiterungen notwendig wären. Diese Vorgehensweise hat jedoch einen gravierenden Nachteil: es entsteht eine häufig als „single point of failure“ bezeichnete Schwachstelle. Sollte eine der Erweiterungen fehlerhaft sein und ihrer Aufgabe der Weiterleitung der Kommunikationsdaten nicht nachkommen, so ist das Kommunikationssystem blockiert.

Entwurfsmuster: Zuständigkeitskette

Zur Realisierung der Kommunikation wird daher auf das in [GHJV95] vorgestellte und in Abschnitt 4.1.1 beschriebene Entwurfsmuster der Zuständigkeitskette zurückgegriffen. Will eine Erweiterung mit einer anderen Erweiterung kommunizieren, so leitet sie ihre Anfrage zunächst an die MMS weiter. Diese ist dann in der Lage, die Anfrage auf die korrekte Priorisierung zu überprüfen, und agiert im Anschluss daran als Klient, der die Anfrage an die übrigen Erweiterungen weiterreicht. Durch dieses Vorgehen ist es möglich, dass eine Erweiterung mit allen anderen Erweiterungen in Kontakt tritt.

Um auch die direkte Kommunikation zwischen zwei Erweiterungen zu realisieren, wird das Entwurfsmuster angepasst. Die MMS leitet die eingehenden Anfragen nur an die Erweiterungen weiter, die in der Lage sind diese zu verarbeiten. Dazu ist es notwendig, dass die MMS darüber informiert ist, welche Anforderungen eine Erweiterung verarbeiten kann. Dazu wird eine als Request-Dispatcher bezeichnete Komponente in der MMS eingeführt. Diese erhält Zugriff auf die Spezifikationen der Erweiterungen und kann so anhand der Anfrage-ID der eingehenden Anfragen entscheiden, an welche Erweiterung(en) diese weiterzuleiten sind.

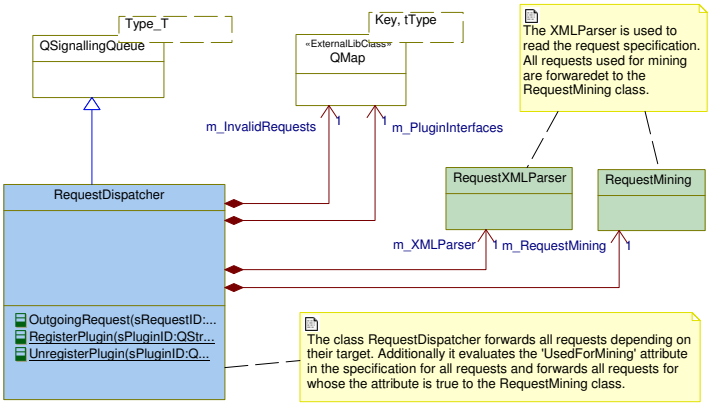


Abbildung 5.24: *Modell der Klasse* `RequestDispatcher` *über die die Kommunikation realisiert wird*

Request-Dispatcher

Abbildung 5.24 zeigt das Modell des Request-Dispatchers, realisiert durch die Klasse `RequestDispatcher`. Die Hauptfunktionalität erbt der Request-Dispatcher von der Klasse `QSignallingQueue`, die ein aktives Objekt mit integrierter Warteschleife realisiert. Der Request-Dispatcher nutzt eine Instanz der Klasse `RequestXMLParser` um die Spezifikationen der Erweiterungen auf die eingehenden Anfrage-IDs zu überprüfen. Darüber hinaus

setzt der Request-Dispatcher eine Instanz der Klasse `RequestMining` ein. Diese dient der Serialisierung der Daten von eingehenden Anfragen, die für den Aufbau einer Datenbasis verwendet werden sollen (siehe Abschnitt 5.3.1). Der Zugriff auf die Erweiterungen zwecks Weiterleitung der Anfragen erfolgt über eine Abbildungstabelle vom Typ `QMap`, in der alle Erweiterungen beim Laden durch die Methode `RegisterPlugin()` registriert werden. Sollte eine Erweiterung wieder deaktiviert werden müssen, so wird die Registrierung über die Methode `UnregisterPlugin()` aufgehoben.

Die weiterzuleitenden Anfragen erreichen den Request-Dispatcher der MMS über die Methode `OutgoingRequest()`, welche direkt durch die Schnittstelle an die Erweiterungen exportiert wird, wie Abbildung 5.25 zeigt.

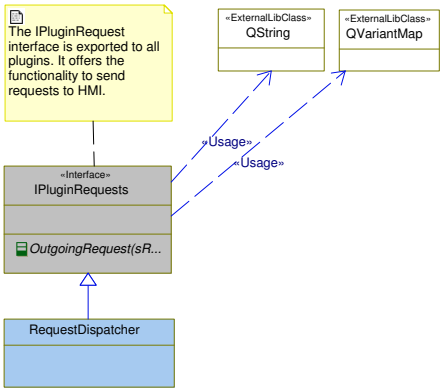


Abbildung 5.25: *Die Implementation der Schnittstelle zur Initiierung von Anfragen durch Erweiterungen.*

Die Methode `OutgoingRequest()` kann von Erweiterungen aufgerufen werden, um die Kommunikation mit der MMS oder anderen Erweiterungen aufzunehmen. Der Request-Dispatcher speichert alle Anfragen in der Warteschleife und bearbeitet sie dann sequentiell. Dabei versucht er die Anfragen nach dem folgenden Muster an die Erweiterungen zu übergeben:

1. Anhand der Zielbezeichnung in der Anfrage-ID wird versucht die Anfrage an die entsprechende Erweiterung zu übergeben. Ist diese Erweiterung nicht aktiviert, so wird mit Schritt 2 fortgefahren.

2. Anhand der zur Verfügung stehenden Erweiterungsspezifikationen ermittelt der Request-Dispatcher in wie fern die benötigte Erweiterung auf dem System zur Verfügung steht. Ist das der Fall, so wird versucht diese Erweiterung zu aktivieren. Im Anschluss daran erfolgt die Weiterleitung der Anfrage an die Erweiterung.

3. Wenn keiner der Versuche zur erfolgreichen Weiterleitung einer Anfrage führt, so wird versucht diese der Reihe nach an jede geladene Erweiterung auszuliefern.

Steuerung der MMS

Durch die Abwendung vom Einsatz des Ereigniskanals für die Integration von Eingabegeräten wird es notwendig, die Steuerungsfunktionalität der MMS anzupassen. In diesem Zusammenhang ist nach Anforderung 3.5 eine Schnittstelle zu schaffen, die die Steuerung der MMS ermöglicht. Dazu werden zunächst vier unterschiedliche Steuerungsarten definiert (siehe auch [Pan10]), die von der MMS zur Verfügung gestellt werden.

Cursorsteuerung

Direkte Cursorsteuerung Bei der direkten Cursorsteuerung erfolgt die Auswertung von fünf Freiheitsgraden, d.h. es werden fünf unterschiedliche Kommandos von der Benutzerschnittstelle interpretiert. Die fünf möglichen Kommandos werden den Funktionen „aufwärts“, „abwärts“, „links“, „rechts“ und „Auswahl“ zugeordnet. Diese Kommandos sind von dem Eingabegerät zyklisch abzusetzen, um eine Bewegung in die entsprechende Richtung aufrecht zu erhalten. Diese Art der Steuerung dient dem Zweck der Integration von Eingabegräten, die in der Lage sind zyklisch und in kurzen Zeitabständen Kommandos zu erzeugen. So wurde die Integration eines Joystick über diese Methodik erfolgreich getestet. Des Weiteren konnte ein BCI-System erfolgreich integriert werden. Bei letzterem zeigte sich jedoch ein Nachteil: die genaue Positionierung des Mauszeigers ist insbesondere bei Eingabegräten mit einer Totzeit im Bereich von wenigen Sekunden schwierig, da der Mauszeiger in diesem Fall nachläuft.

Tabulatorsteuerung

Tabulator-basierte Steuerung Das zweite Verfahren basiert auf einer bereits in vielen Betriebssystemen realisierten Möglichkeit der Steuerung mittels Tabulatortaste. Dabei wird generell zwischen aktiven Schaltflächen mittels Tabulatortaste gewechselt. Als Rückmeldung an den Benutzer wird die aktuelle aktive Schaltfläche grafisch hervorgehoben. Das Verfahren ähnelt daher dem in [UG10] eingesetzten Steuerungsverfahren. Die Reihenfolge, in der die Schaltflächen ausgewählt werden, ist statisch, sie kann zur Laufzeit nur komplett umgekehrt (durch zusätzliches Drücken der Umschalttaste), aber nicht umsortiert werden. Benötigt werden dabei nur drei Freiheitsgrade, die den Kommandos „vor“, „zurück“ und „Auswahl“ zugeordnet werden, so dass auch diese Art der Steuerung für den Einsatz von Eingabegeräten mit wenig Freiheitsgraden geeignet ist. Diese Art der Steuerung unterliegt nicht dem Nachteil des Nachlaufens, da hierbei die Kommandos nicht zyklisch vom Eingabegerät erzeugt werden müssen. Jedes Kommando wird nur einmal an die Benutzerschnittstelle weitergereicht und dort ausgewertet. Nachteilig an diesem Vorgehen ist, dass der Benutzer unter Umständen sehr viele Eingaben vornehmen muss, um zum Ziel (der erfolgreichen Auswahl einer Schaltfläche) zu kommen. Die Wahrscheinlichkeit dafür steigt mit wachsender Anzahl an sichtbaren Schaltflächen.

automatische Cursorbewegung

Intelligente Cursorsteuerung Bei dieser Steuerung handelt es sich um eine Mischform der bereits vorgestellten Steuerungsmöglichkeiten. Wie bereits bei der direkten Cursorsteuerung werden wieder fünf Freiheitsgrade für die Funktionen „aufwärts“, „abwärts“, „links“, „rechts“ und „Auswahl“ benötigt. Bei dieser Form müssen die Kommandos jedoch nicht zyklisch von der Erweiterung erzeugt werden, sondern wie bei der Tabula-

torsteuerung sequentiell. Der Mauszeiger wird dabei automatisch bewegt und automatisch gestoppt. Dazu ermittelt die Benutzerschnittstelle die Positionen der Schaltflächen in unmittelbarer Nähe des Mauscursors und erstellt ein unsichtbares Raster durch die Mittelpunkte der Schaltflächen. In einem zweiten Schritt werden alle Kanten des Rasters, die gemeinsam durch die gleichen Schaltflächen laufen, zu einer zusammengefasst, wie die Skizze in Abbildung 5.26 für die horizontale Bewegungsrichtung verdeutlicht. Das Stoppen des Mauszeigers erfolgt dann nach jedem Kommando, wenn die horizontale oder vertikale Position des Mauscursors eine der Kanten kreuzt. Durch dieses Vorgehen lässt sich die Anzahl der nötigen Kommandos deutlich reduzieren.

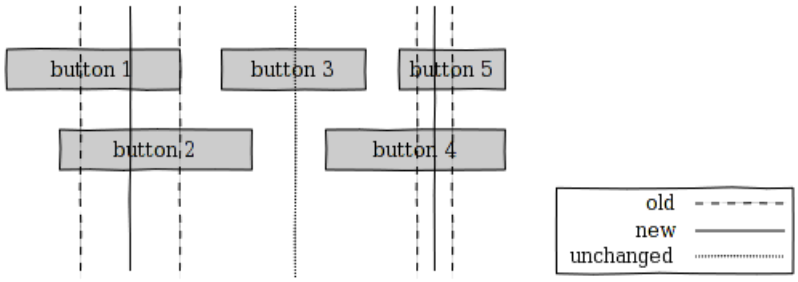


Abbildung 5.26: *Intelligente Cursorsteuerung*

Hierarchische Steuerung Bei der hierarchischen Steuerung werden die Elemente der grafischen Oberfläche logisch in Menüebenen angeordnet. Es ergibt sich dadurch ein Menübaum, in dem über die vier Kommandos „hoch“, „runter“, „Auswahl“ und „zurück“ navigiert werden kann. Es werden also vier Freiheitsgrade benötigt. Das jeweils aktive Element wird ähnlich wie bei der Tabulatorsteuerung grafisch hervorgehoben. Auch diese Art der Steuerung reduziert die Anzahl der nötigen Kommandos, besitzt aber den Nachteil, dass zur Anordnung der Schaltflächen in die Baumstruktur ein Eingriff in die Anwendung notwendig ist.

Menüebenen

Die Abbildung dieser Steuerungsformen wird in der MMS über unterschiedliche Kommandotypen erreicht. Diese sind von den Erweiterungen über die von der MMS zur Verfügung gestellte Schnittstelle an die MMS weiterzuleiten. Den Entwurf der Schnittstelle zeigt Abbildung 5.27.

Zusätzlich zu der Methode für das Absetzen von Steuerungskommandos enthält die Schnittstelle noch eine Methode, mittels derer die Erweiterungen in die Lage versetzt werden die in Anforderung 3.10 beschriebenen Nutzermeldungen abzusetzen.

Informationen zu Szenarien

Die in Anforderung 3.12 formulierte Notwendigkeit des Zugriffs auf die zur Verfügung stehenden autonom ausführbaren Handlungen wird in einer weiteren Schnittstelle der

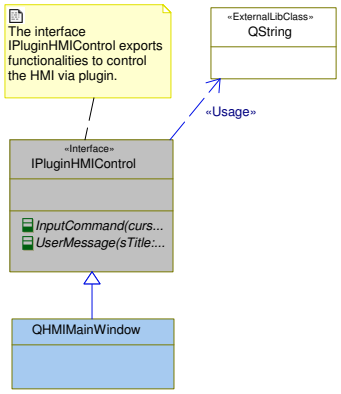


Abbildung 5.27: *Die Implementation der Schnittstelle zur Steuerung der MMS durch Erweiterungen.*

Zugriff auf Systemdaten

MMS realisiert. Diese erlaubt über mehrere Methoden die Abfrage von Szenarien, der funktionalen Gruppen, in die die Szenarien eingeteilt sind, sowie der menschenlesbaren Titel und Beschreibungen. Abbildung 5.28 zeigt den Entwurf dieser Schnittstelle.

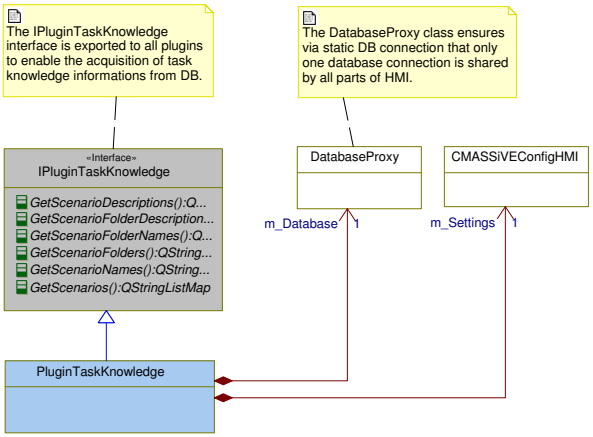


Abbildung 5.28: *Die Implementation der Schnittstelle zur Akquise von Szenarienwissen durch Erweiterungen.*

Szenarien

Die Schnittstellenimplementationsklasse `PluginTaskKnowledge` nutzt Instanzen der Klassen `DatabaseProxy` und `CMASSiVEConfigHMI` zur Akquise der nötigen Daten. Über Letztere ermittelt sie die aktuell eingestellte Systemsprache, um im Anschluss daran die Informationen aus der Datenbank in der korrekten Sprache zu beziehen. Dazu stehen sechs Methoden zur Verfügung. Die Methode `GetScenarioFolders()` ermittelt alle in der Datenbank spezifizierten funktionalen Gruppen, um die lesbaren Namen dieser Gruppen im Anschluss daran mit der Methode `GetScenarioFolderNames()` zu erfragen. Über die Methode `GetScenarioFolderDescription()` ermitteln die Erweiterungen Beschreibungen zu den funktionalen Gruppen. Analog dazu sind die Methoden zur Akquise der Szenarien aufgebaut. Die Methode `GetScenarios()` ermittelt alle Szenarien. Mittels

`GetScenarioNames()` erfolgt die Abfrage der lesbaren Namen der Szenarien und über `GetScenarioDescriptions()` die Abfrage der Beschreibungen.

5.4.2 Aktivierung und Deaktivierung von Erweiterungen

Die Aktivierung von Erweiterungen erfolgt zunächst auf Basis der Angaben in der globalen Konfiguration der MASSiVE-Architektur. Wie in Abschnitt 5.4.1 bereits festgelegt wurde, enthält diese nach der ersten Initialisierung die Angaben aus den Spezifikationen der Erweiterungen und die aktuellen Konfigurationswerte. Darunter befindet sich auch jeweils ein Konfigurationswert, der für alle Erweiterungen[5] festlegt, ob diese automatisch beim Start der MMS aktiviert werden soll. Dazu untersucht der Display-Manager der MMS beim Start alle Spezifikationen auf diesen Konfigurationswert und lädt alle Erweiterungen, die dementsprechend konfiguriert sind, automatisch. Die Grundfunktionalität für das Laden der Erweiterungen wird dabei von der Klasse `QPluginLoader` aus der Qt-Bibliothek zur Verfügung gestellt. Weitere Informationen dazu sind der Referenzdokumentation [Nok10] zu entnehmen. Abbildung 5.29 zeigt den vereinfachten Entwurf des Display-Managers.

Display-Manager

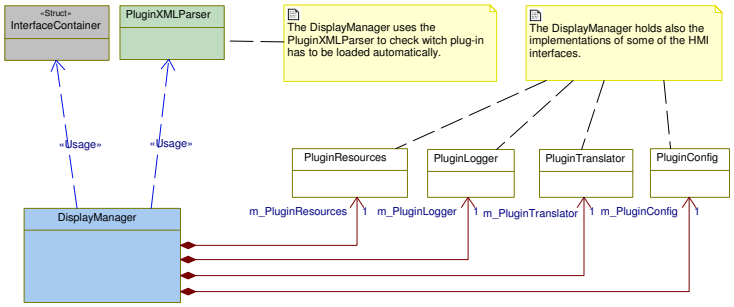


Abbildung 5.29: *Vereinfachter Entwurf des Display-Managers der MMS*

In dieser Abbildung sind auch die Schnittstellenimplementationen, die direkt vom Display-Manager verwaltet werden, sowie die Klasse `PluginXMLParser` zu sehen. Letztere übernimmt die Auswertung der Spezifikationen der Erweiterungen. Der Ablauf beim Laden von Erweiterungen erfolgt in mehreren Schritten. Die grobe Struktur wird dabei von dem in Abbildung 5.30 dargestellten Flussdiagramm vorgegeben. Die Art einer Erweiterung entscheidet darüber, wo sie in der MMS zur Anzeige gebracht wird. Die ausführliche Erläuterung der möglichen Anzeigepositionen folgt in Abschnitt 6.1. Bevor die grafische Komponente einer Erweiterung oder deren Piktogramm angezeigt wird, erfolgt jedoch zunächst die Initialisierung der Erweiterung über den Aufruf der Methoden `SetExposedInterfaces()` und `Setup()`. Erstere übergibt den Container mit den Schnittstellen zur MMS und letztere führt erweiterungsspezifische Initialisierungen durch.

verschiedene Anzeigepositionen

[5] Eine Ausnahme bilden hier die Erweiterungen für Benutzerinteraktionen.

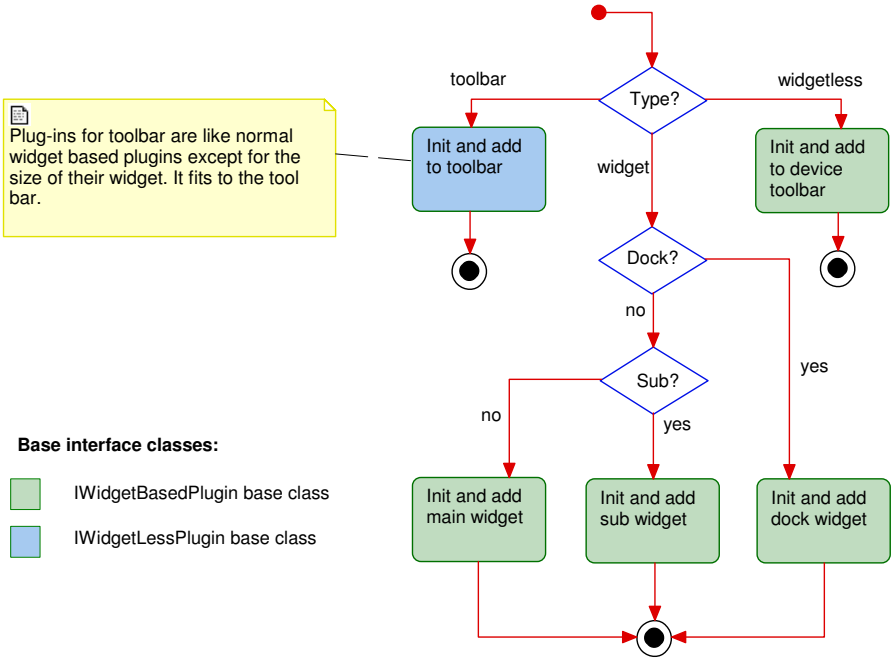


Abbildung 5.30: *Vorgänge beim Laden von Erweiterungen*

dynamische Aktivierung

Neben der Aktivierung von Erweiterungen beim Start der MMS besteht eine weitere Möglichkeit, die in Abschnitt 5.4.1 vorgestellt wurde. Die Aktivierung erfolgt hier dynamisch zur Laufzeit sobald eine Anfrage den Request-Dispatcher der MMS erreicht, die nicht direkt an ein Erweiterung weitergeleitet werden kann. Diese Vorgehensweise basiert auf der grundsätzlichen Vorgehensweise, wie sie auch von „launchd“ verfolgt wird. Insbesondere bei der Aktivierung von Erweiterungen zur Benutzerinteraktion wird diese noch weitere Vorteile bringen, wie Abschnitt 5.4.3 zeigen wird.

5.4.3 Initiierung von Benutzerinteraktionen

Bearbeitung von Benutzerinteraktionen

Das Grundprinzip der MASSiVE Architektur basiert auf der Möglichkeit den Benutzer in die Aufgabenbearbeitung mit einzubeziehen. Die als Benutzerinteraktion bezeichneten Eingriffe werden dabei vom System initiiert und vom Benutzer ausgeführt. Aus Sicht des Systems ist der Benutzer dabei eine Ressource wie jede andere. Benutzerinteraktionen werden durch den Sequenzer in der Planungsebene der MASSiVE Architektur initiiert und verhalten sich genauso wie andere von der Planungsebene ausgeführte Operationen, so dass eine Teilkomponente der MMS der Planungsebene eine Schnittstelle bieten muss. Diese Teilkomponente wird in der MASSiVE-Architektur durch den User-Interaction-Skill-Server realisiert, dessen Aufbau dem der anderen Skill-Server entspricht. Die einzige Besonderheit ist die CORBA-Schnittstelle zur MMS, über die vom Sequenzer initiierte Benutzerinteraktionen an diese weiterleitet werden. In der MMS führt das zu einer Anfrage, die vom Request-Dispatcher bearbeitet wird und dann in der Aktivierung der entsprechenden Erweiterung resultiert, die die Dialoginteraktion mit dem Benutzer durchführt (siehe Abschnitt 5.4.1).

Durch die Realisation der Komponente zur Initiierung von Benutzerinteraktionen im User-Interaction-Skill-Server ist es notwendig darauf zu achten, dass die Erweiterungen, über die die Benutzerinteraktionen dem Benutzer präsentiert werden, keinen Zugriff auf weitere Systemressourcen erhalten. Das liegt darin begründet, dass bei der Ausführung von Skills alle eingesetzten Ressourcen der Ressourcenverwaltung des Sequenzers unterliegen. Die Zugriffe haben in diesem Moment ausschließlich über das Skill-Netzwerk zu erfolgen.

Wer einen Fehler gemacht hat und ihn nicht korrigiert, begeht einen zweiten.

Konfuzius - chinesischer Philosoph

6

Dynamische Erweiterung der Benutzerschnittstelle

Die Grundvoraussetzungen für eine adaptive MMS für Rehabilitationsroboter wurden im Kapitel 2 ausführlich dargestellt. Ausgehend von diesen Betrachtungen wird jetzt als Kernelement das Rahmenwerk zur Realisierung der adaptiven Erweiterungen für die MMS entwickelt. Einleitend erfolgt zunächst der Entwurf der angepassten grafischen Oberfläche der MMS, so dass diese der in der Anforderung 3.6 formulierten Notwendigkeit zum modularen Aufbau gerecht wird. Im Anschluss daran folgt die Beschreibung der Lösungsansätze für das Rahmenwerk zur Integration der Erweiterungen. Daraus werden die in Abschnitt 5 eingeführten Spezifikationen abgeleitet. Abschließend folgt die Betrachtung der Implementationsdetails. Die Beschreibung der bisher entwickelten Erweiterungen sowie deren Funktionalitäten sind in Anhang A.2.3 zu finden.

modulare MMS

6.1 Grafische Benutzerschnittstelle

Bei der Entwicklung der Benutzerschnittstelle für die MASSiVE-Architektur lagen die Entwicklungsschwerpunkte im Bereich der Nutzbarkeit (siehe [Cyr05]), was in erster Linie zu einer Benutzerschnittstelle mit großen Schaltflächen und einem strukturierten Aufbau führte. Die Möglichkeit des adaptiven Oberflächenaufbaus stand bei diesen Entwicklungen nicht im Fokus.

Im folgenden soll daher aufbauend auf der bestehenden Struktur der Benutzerschnittstelle die Modularisierung vorgenommen werden. Zunächst wird die Oberfläche dazu in unterschiedliche Bereiche eingeteilt. Der Hauptbereich nimmt ungefähr zweidrittel der Gesamtfläche ein und dient der Darstellung der zur Verfügung stehenden Systemszenarien in Form einer Verzeichnisstruktur. Seitlich daneben erstreckt sich horizontal der Steuerungsbereich für das Gesamtsystem. Darüber hinaus befinden sich am Rand der Oberfläche weitere Bereiche in denen Aufgabenstatus und Meldungslisten angezeigt werden können.

Einteilung der Oberfläche

WIMP Paradigma

Die Benutzeroberfläche basiert auf der Klasse `QMainWindow` aus der Qt-Bibliothek. Diese bietet grundlegende Funktionalitäten für die Anwendungsentwicklung nach dem WIMP Paradigma. Dazu gehören unter anderem auch an-dockbare Fenster, mittels derer die Elemente der MMS am Rand der Oberfläche realisiert wurden. Die Inhalte dieser Fenster sind jedoch statisch gelinkt, also ein fester Bestandteil der MMS. Die Implementation der benötigten Logik ist ebenfalls Bestandteil der MMS. Die Anzeige der Benutzerinteraktionen erfolgt in Form von separaten Fenstern, die zentriert auf der Oberfläche angezeigt werden und so einen Teil davon verdecken.

Aufgrund der Modularisierung sämtlicher Funktionalitäten ist es notwendig, diese statische Verbindung von MMS und Funktionalität auch in der grafischen Oberfläche aufzulösen. Dazu werden sämtliche grafischen Elemente aus der MMS entfernt. Diese sollen durch grafische Elemente ersetzt werden, die von den nachladbaren Erweiterungen erzeugt werden. Die Positionierung der Erweiterung wird über deren Spezifikation festgelegt und orientiert sich dabei an dem in Abbildung 6.1 dargestellten Aufbau.

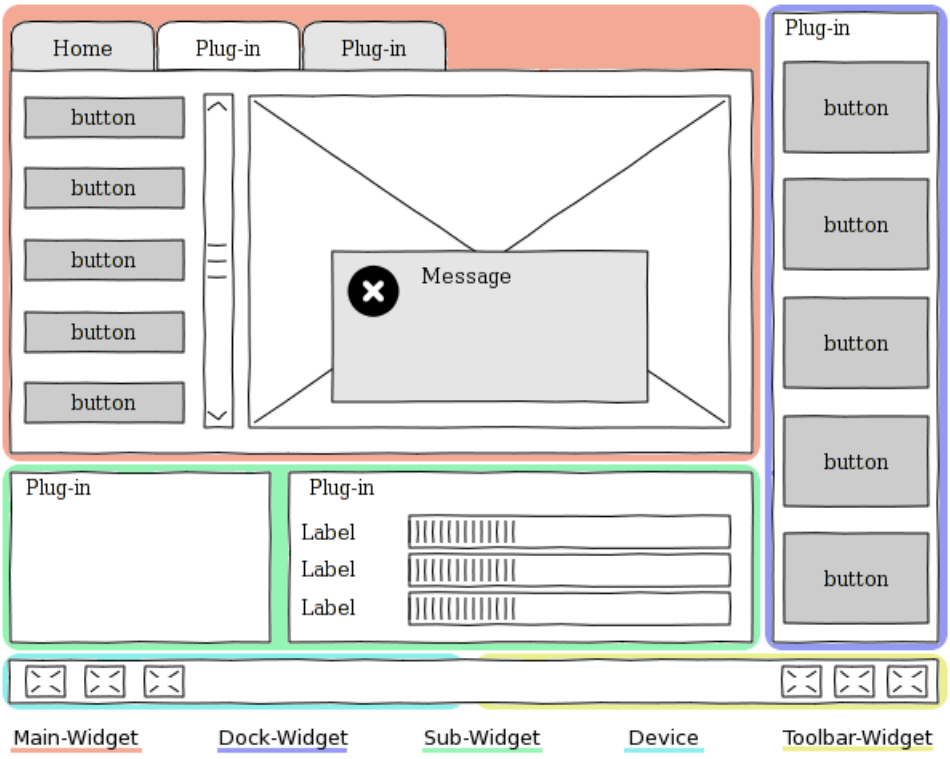


Abbildung 6.1: *Entwurf und Aufteilung der grafischen Oberfläche in unterschiedliche grafische Bereiche*

MeeGo

Verglichen mit dem Aufbau der bestehenden Oberfläche der MMS besteht somit die einzige Änderung im Aufbau des Hauptbereiches. Dieser beinhaltet jetzt einen Bereich, in dem grafische Elemente in einer Register-Struktur angeordnet werden, und ist angelehnt and den in Abschnitt 2.3.1 vorgestellten Aufbau des Linux-basierten Betriebssystems „MeeGo“. Durch die Anzeige in Form von Reitern ist es möglich auf den Einsatz

von Fenstern vollständig zu verzichten, was insbesondere den Einsatz von Eingabegeräten mit wenig Freiheitsgraden stark vereinfacht. Des Weiteren ist es durch den Einsatz von Reitern möglich mehrere Erweiterung zeitgleich darzustellen. Der Benutzer muss lediglich zwischen den Reitern wechseln. Bei einer Benutzerinteraktion kann ein zusätzlicher Reiter eingeblendet werden, der sofort den Fokus erhält. Nachdem die Benutzerinteraktion durch den Benutzer abgeschlossen wurde, wird der Reiter wieder entfernt.

Farbig hervorgehoben sind in Abbildung 6.1 die bereits in Abschnitt 5.4.2 angesprochenen unterschiedlichen Anzeigebereiche der MMS. Der als „Main-Widget“ bezeichnete Bereich ist der Bereich mit der Reiterstruktur. Er entspricht dem Hauptbereich der ursprünglichen MMS-Oberfläche. Der „Dock-Widget“-Bereich dient der Integration von Elementen am Rand der Oberfläche. Darüber hinaus existiert mit dem „Sub-Widget“-Bereich eine Möglichkeit Erweiterungen mit kleiner grafischer Komponente unterhalb des Hauptbereiches einzublenden. Eine weitere Neuerung in Bezug auf die bestehende MMS-Oberfläche ist ein am unteren Rand eingeführte Statuszeile. Dort lassen sich die Piktogramme der Erweiterungen für Eingabegeräte „Device“ und die Erweiterungen mit sehr kleiner grafischer Komponente einblenden „Toolbar-Widget“.

„Main“, „Dock“, „Sub“ und „Device“

6.2 Ein Rahmenwerk für Erweiterungen

Im Kapitel 5 erfolgte die ausführliche Entwicklung der MMS als Rahmenwerk, ihrer Subsysteme und deren Funktionsweise. Auf die Implementierung von Funktionalitäten, die eine Abhängigkeit zum realen System, also zum eingesetzten Rehabilitationsroboter bedeutet hätte, wurde bewusst verzichtet. Durch diese Maßnahme war es möglich, die MMS komplett hardware- beziehungsweise systemunabhängig zu gestalten (was die Abhängigkeit zur MASSiVE-Architektur nicht mit einschließt). Ein weiteres Resultat dieser Vorgehensweise ist, dass die MMS ohne Erweiterung keinerlei Funktionalität für die Systemsteuerung zur Verfügung stellt. Die gesamte Funktionalität der MMS (sowohl systemunabhängige als auch auch systemabhängige Funktionen) müssen daher mittels Erweiterungen realisiert werden. Das setzt unmittelbar die Möglichkeit voraus, sowohl alle Interaktionsformen (vgl. Abschnitt 2.2 auf Seite 19) als auch alle möglichen Ein- und Ausgabegeräte auf Erweiterungen abbilden zu können. Aus diesem Zusammenhang lassen sich direkt zwei unterschiedliche Gruppen von Erweiterungen ableiten. Zum einen Erweiterungen, die mittels grafischer Elemente in Interaktion mit den Nutzer stehen – und somit auch zur Realisierung der im Abschnitt 2.2 vorgestellten Interaktionsformen indirekte Manipulation, direkte Manipulation, RBI und WIMP eingesetzt werden können – und zum anderen Erweiterungen, die Ein- und Ausgabegeräte realisieren. Desweiteren müssen die speziellen Anforderungen der MASSiVE-Architektur im Zusammenhang mit dem Konzept der geführten Benutzerinteraktionen Beachtung finden, was zu der dritten Gruppe von Erweiterungen führt. Demzufolge werden im Weiteren folgende Erweiterungen betrachtet:

hardware- und systemunabhängig

- Erweiterung für Ein- Ausgabegerät (referenziert als Typ 1)

- Erweiterung mit grafischer Komponente (referenziert als Typ 2)
- Erweiterung für geführte Benutzerinteraktion (referenziert als Typ 3)

Die Steuerung eines Rehabilitations-Robotersystems erfordert eine Nutzerschnittstelle, die einfach an die spezifische Behinderung des Nutzers angepasst werden kann. Diese Anpassung erfolgt in der Regel durch spezielle Eingabegeräte, die über die dafür vorgesehene Gruppe von Erweiterungen in die MMS zu integrieren sind. Die Notwendigkeit dabei die spezifischen Bedürfnisse des jeweiligen Nutzers und die Anforderungen der Robotersteuerung in Einklang zu bringen, kann durch eine möglichst hohe Flexibilität bei der Entwicklung der Erweiterungen erreicht werden. Insbesondere hier macht sich der Einsatz von Erweiterungen positiv bemerkbar, da ein bestehendes System durch einfaches Austauschen der Erweiterungen an sich ändernde Anforderungen durch den Benutzer angepasst werden kann.

flexible Erweiterungen

Die zuvor festgelegten Gruppen von unterschiedlichen Erweiterungen werden im Folgenden genauer betrachtet, um die konkreten Lösungsansätze zu formulieren und die notwendigen Spezifikationen festlegen zu können.

6.2.1 Typ 1: Erweiterung für Ein- Ausgabegerät

Erweiterungen vom Typ 1 müssen in der Lage sein, möglichst flexibel beliebige reale Eingabesysteme in die MMS zu integrieren. Dabei gilt es sowohl Eingabegeräte zu integrieren, die mittels HardwareServer über die MASSiVE-Architektur angesteuert werden, als auch solche, deren Hardware direkt über eine Hersteller-API angesprochen wird, oder die gänzlich ohne Hardware (zum Beispiel Sprachverarbeitung) auskommen.

Eingabegeräte

Eingabeereignisse in der grafischen Oberfläche der MMS können nur über Signale erzeugt werden, die von Schaltflächen oder anderen Oberflächenelementen emittiert werden. Da Eingabegeräte vom Typ 1, wie zuvor festgelegt wurde, jedoch keine grafische Komponente besitzen, müssen diese in der Lage sein, über einen anderen Mechanismus Eingabeereignisse zu generieren. Damit dieses Verhalten ermöglicht werden kann, müssen diese Erweiterungen gegebenenfalls als aktive Objekte realisiert werden, was bereits in Anforderung 3.4 formuliert wurde und so direkt zum ersten Lösungsansatz führt:

Lösungsansatz 6.1. *Die Erweiterungen für Ein- und Ausgabegeräte werden die Möglichkeit bieten, optional als aktive Objekte realisiert zu werden, um die Möglichkeit zu bieten externe Hardware zyklisch abzufragen. Dazu erfolgt die Integration eines eigenen Threads (siehe [Wik10, Stichwort: „Thread (Informatik)“]) in diesen Erweiterungen, der bei Bedarf dazu verwendet werden kann.*

Da diese Erweiterungen später ebenso dynamisch angepasst werden können müssen, wie die MMS selbst, besteht darüber hinaus die Notwendigkeit von Konfigurationsmöglichkeiten. Das beginnt mit der Festlegung, ob eine Erweiterung bei startender MMS automatisch aktiviert werden soll oder nicht, und endet bei der Festlegung von

Schnittstellen, an denen die erweiterungsspezifische Hardware angeschlossen ist. Auch Anpassungen durch den Benutzer zur Laufzeit sind denkbar und sollten daher von der Konfigurationsschnittstelle abgedeckt werden. Dazu wurde in Abschnitt 5.4.1 bereits eine Schnittstelle zur MMS entwickelt.

6.2.2 Typ 2: Erweiterung mit grafischer Komponente

Erweiterungen mit grafischer Komponente dienen in erster Linie der Anpassung der MMS an die system- oder umgebungsspezifischen Gegebenheiten. So müssen im Falle des FRIEND-Systems zum Beispiel Rollstuhl- und Umgebungssteuerfunktionen als grafische Erweiterung realisiert werden. Darüber hinaus sind aber auch MASSiVE-spezifische Funktionalitäten, wie die Szenarienauswahl und das Steuerungspanel, als Erweiterung zu realisieren.

grafische Erweiterung

Lösungsansatz 6.2. *Die Funktionalitäten der bestehenden MMS-Oberfläche werden auf Erweiterungen abgebildet. Die Platzierung der grafischen Elemente in der modularisierten Oberfläche der MMS erfolgt analog zu der Platzierung in der ursprünglichen Oberfläche.*

6.2.3 Typ 3: Erweiterung für geführte Benutzerinteraktion

Die Erweiterungen vom Typ 3 sind vergleichbar mit den Erweiterungen vom Typ 2. Der Unterschied liegt lediglich darin, dass diese Erweiterungen immer als Reiter im „Main-Widget“ Bereich anzuzeigen sind. Des Weiteren steht ihnen nicht der freie Zugriff auf beliebige Systemressourcen zu, da ihre Aktivierung und Deaktivierung komplett über die Planungsebene gesteuert wird, ebenso der Zugriff auf die Ressourcen.

Benutzerinteraktion

Lösungsansatz 6.3. *Die Benutzerinteraktionen werden wie Erweiterungen vom Typ 2 realisiert jedoch beim Laden zwangsläufig als Reiter im „Main-Widget“-Bereich angezeigt. Dazu wird in der Spezifikation als Typ der Erweiterung Benutzerinteraktion beziehungsweise Skill angegeben.*

6.3 Spezifikationssprache für Erweiterungen

Teil des nach [Mar99] im Abschnitt 4.1.1 vorgestellten Rahmenwerks zur Integration von Erweiterungen ist der Vertrag zwischen Anbieter der Erweiterungsschnittstelle, in diesem Fall die MMS, und der Erweiterung selbst. Er enthält Informationen über die Erweiterung und wie sie von der MMS zu aktivieren ist. Diese Informationen können direkt nach der Instanziierung von der Erweiterung erfragt werden oder müssen in separater Stelle zur Verfügung stehen. Gegen die Akquise dieser Informationen über die Erweiterung spricht die Tatsache, dass es so notwendig wäre jede Erweiterung beim

Start der MMS zu instanziieren, um die Informationen zu erfragen. Aus diesem Grund ist es sinnvoll von dieser Variante Abstand zu nehmen.

Spezifikation für Erweiterungen

Die Alternative der Verwendung einer separaten Datenquelle für diese Informationen setzt voraus, dass die Datenquelle nur syntaktisch und semantisch korrekte Daten liefert. Nur so kann zur Laufzeit die robuste Aktivierung und Deaktivierung von Erweiterungen erreicht werden. Einen sehr guten Ansatz bietet hier die im Abschnitt 4.1.2 vorgestellte Daten-Modellierung in Form von XML. Damit dieser Ansatz zugleich der gerade beschriebenen Anforderung nach syntaktischer und semantischer Korrektheit gerecht wird, ist dazu ein XML-Dialekt inklusive der nötigen Mechanismen zur Validierung zu entwickeln. Das Ergebnis ist die Möglichkeit der Spezifikation aller notwendigen Informationen über Erweiterungen, wie sie bereits in Abschnitt 5.4.1 beschrieben wurde.

6.3.1 Festlegung der Grammatik

Damit die Erweiterungen völlig unabhängig voneinander und von der MMS sind, muss jede der Erweiterungen über eine eigene Spezifikation verfügen. Das heißt, nach Festlegung der Grammatik für diese Spezifikation ist jeweils eine XML-Instanz für jede Erweiterung neben der Erweiterung selbst zu erstellen.

Ausgehend von Abschnitt 3.1 müssen zunächst solche Anforderungen betrachtet werden, die direkt über die Spezifikation realisiert oder zumindest vorbereitet werden können. So beschreibt Anforderung 3.4 ein Design- beziehungsweise Implementationsdetail welches später zu betrachten ist. Die Anforderungen 3.7 und 3.17 beschreiben dagegen die Forderung nach Konfigurierbarkeit und einem Ressourcen-Management, welche einleitend betrachtet werden müssen.

Konfigurierbarkeit

Initialisierung der Konfiguration

Zur Realisation der Konfigurierbarkeit der MMS wurde bereits in Abschnitt 5.4.1 die Verwaltung von Einstellungen beschrieben und eine Schnittstelle für die Erweiterungen entworfen. Über die Schnittstelle erhalten die Erweiterungen Zugriff auf die globale Konfiguration, um dort von Ihnen genutzte Einstellungen nachzuschlagen. Um die Einstellungen der Erweiterungen zu initialisieren ist nach Abschnitt 5.4.1 ein Teil der Spezifikation zu nutzen, der alle für eine Erweiterung zur Verfügung stehenden Konfigurationswerte mit Typ und Standardbelegung enthält. Der dazugehörige Entwurf eines XML Schemas ist in Abbildung 6.2 dargestellt. Jeder einzelne Konfigurationswert wird als ein XML-Element vom Typ `ConfigValue` repräsentiert, welches zwei Attribute besitzt, über die der Name des Konfigurationswertes und der Typ festgelegt werden.

Der Name ist dabei eine Zeichenkette vom Typ `xs:string`[1] mit Beschränkung auf Großbuchstaben, Ziffern und Unterstriche (regulärer Ausdruck `([A-Z0-9_])*`) und der Typ ist eine

[1] XML Namensraum `xs` entspricht „http://www.w3.org/2001/XMLSchema“

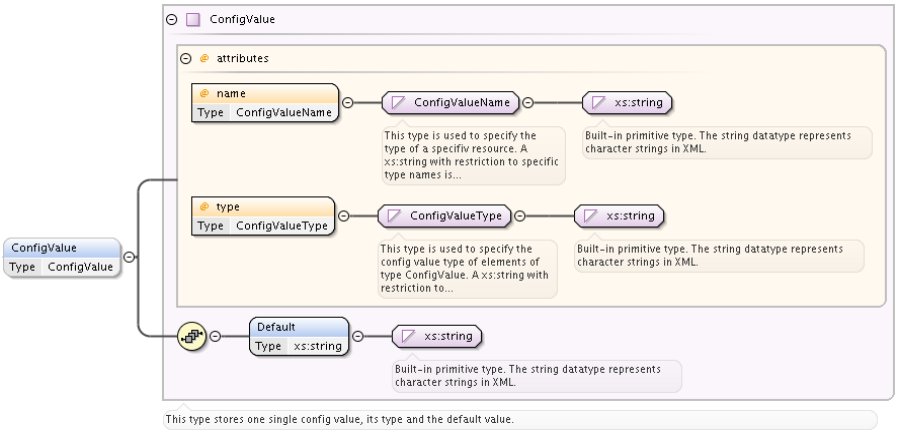


Abbildung 6.2: *XML Schema des Datentyps* `ConfigValue`

auf die in Tabelle 6.1 dargestellten Elemente beschränkte Zeichenkette, ebenfalls vom Typ `xs:string`.

Zeichenkette	Beschreibung
`QString`	Zeichenkette
`bool`	Wahrheitswert
`int`	Vorzeichenbehaftete Ganzzahl
`uint`	Vorzeichenlose Ganzzahl
`double`	Gleitkommazahl

Tabelle 6.1: *Mögliche Typen für Konfigurationswerte*

Zur Spezifikation der Konfigurationswerte einer Erweiterung können beliebig viele dieser Elemente in einer Sequenz zusammengefasst werden. Die angegebenen Standardwerte werden verwendet, um der MMS die Initialisierung der Einstellungen bei erstmaliger Aktivierung neuer Erweiterungen zu ermöglichen. Änderungen an den Werten erfolgen dann nur noch über die Konfigurationskomponente der MASSiVE-Architektur, welche, wie im Abschnitt 5.4.1 beschrieben, auch für die MMS eingesetzt wird.

einheitliche Konfigurationsanwendung

Festlegung der Ressourcen

Nach der Festlegung der Grammatik für die Konfigurationswerte von Erweiterungen sind die von einer Erweiterung benötigten Ressourcen zu betrachten. Alle Ressourcen werden den Erweiterungen über eine von der MMS implementierte Schnittstelle zur Verfügung gestellt. Um die MMS in die Lage zu versetzen zur Laufzeit Erweiterungen in Abhängigkeit von den zur Verfügung stehenden Ressourcen dynamisch zu aktivieren, benötigt sie Informationen über alle für den Betrieb einer Erweiterung notwendigen Ressourcen. Zu diesem Zweck ist ein weiteres Element in die Spezifikation aufzunehmen. Abbildung 6.3 zeigt den dazugehörigen XML Schema Entwurf.

benötigte Ressourcen

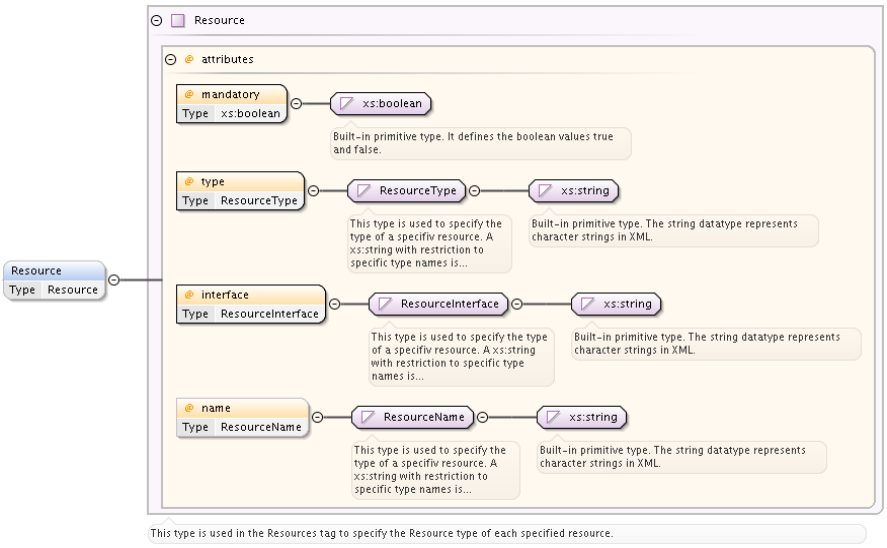


Abbildung 6.3: *XML Schema des Datentyps* `Resources`

Das eingeführte Element ist vom Typ `Ressource` und besitzt drei notwendige und ein optionales Attribut. Über das notwendige Attribut `mandatory` wird festgelegt, ob die beschriebene Ressource optional oder für den Betrieb der Erweiterung notwendig ist. Das Attribut ist vom Typ `boolean` und kann die Werte `true` und `false` annehmen. Über die Angabe von `type` wird die Art der Ressource festgelegt. Dieses Attribut ist vom Typ `xs:string` mit Beschränkung auf die in Tabelle 6.2 dargestellten Zeichenketten.

Zeichenkette	Beschreibung
`HardwareServer`	HardwareServer der reaktiven Ebene
`SkillServer`	SkillServer der reaktiven Ebene
`DataServer`	DataServer der reaktiven Ebene
`Server`	beliebige Serverkomponente

Tabelle 6.2: *Mögliche Typen für Systemressourcen*

Akquise durch Repository-ID

Über das Attribut `interface` wird das von der Ressource zur Verfügung gestellte CORBA-Interface beschrieben. Dieses ist ein Teil der Repository-ID und wird von der MMS dazu verwendet, die Akquise der Ressource durchzuführen, so dass der Zugriff durch die Erweiterung erfolgen kann (siehe Abschnitt 5.2.3). Als Datentyp kommt für dieses Attribut `xs:string` mit einer Restriktion auf Buchstaben (klein, groß), Ziffern oder Unterstriche zum Einsatz (regulärer Ausdruck `([a-zA-Z0-9_])*`). Das optionale Attribut `name` wird aus Gründen der Abwärtskompatibilität eingeführt. Es wird verwendet, wenn der im Abschnitt 5.2 entwickelte Mechanismus zur dynamischen Ressourcendetektion fehlschlägt. In diesem Fall wird der Wert des Attributs als Namenskontext verwendet, um die Verbindung zur Ressource direkt herzustellen. Das Attribut ist vom Typ `xs:string` mit Restriktion auf Buchstaben (klein, groß), Ziffern, Unterstriche und Schrägstriche zur Festlegung des Kontextpfades (regulärer Ausdruck `([a-zA-Z0-9_/])*`).

Kommunikation über Anfragen

Kommunikation zwischen Erweiterungen

Nach Anforderung 3.9 besteht die Notwendigkeit zu einer Kommunikationsmöglichkeit zwischen den Erweiterungen und der MMS. Diese wird direkt über den in Abschnitt 5.4.1 beschriebenen Mechanismus realisiert. Die Quelle der dazu von der MMS benötigten Informationen über mögliche Anfragen wurde bei der Entwicklung dieses Mechanismus bereits erwähnt, jedoch bisher nicht betrachtet. Da auch diese Informationen zur vollständigen Spezifikation einer Erweiterung gehören, sind sie in die Grammatik mit aufzunehmen.

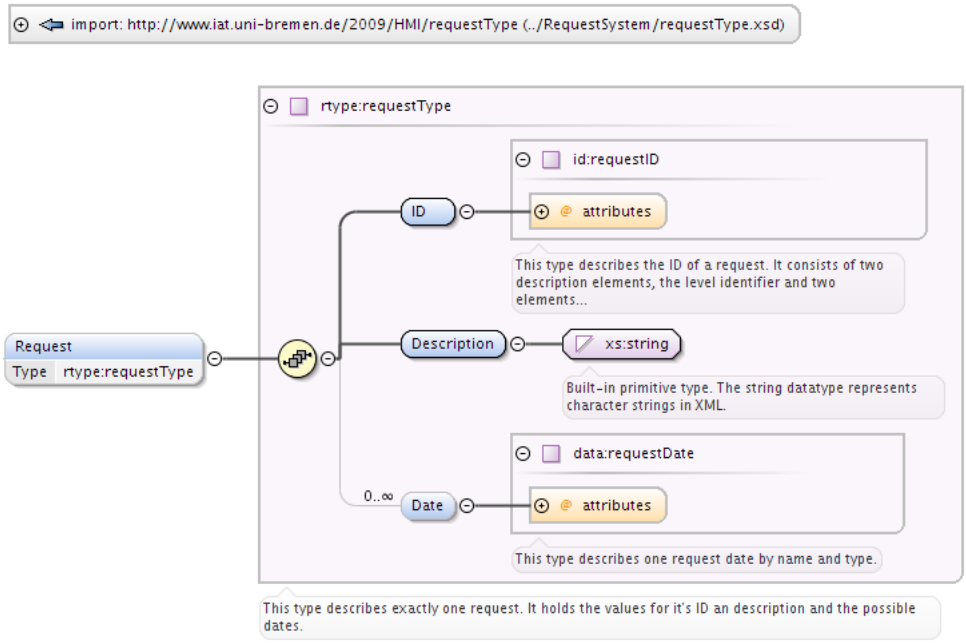


Abbildung 6.4: *XML Schema der integrierten* `Request`*-Liste*

Das dazu definierte XML Schema wurde bereits im Abschnitt 5.3.1 vorgestellt und soll hier daher nicht näher beschrieben werden. Zur Verwendung der Datentypen in den XML Schemata der Spezifikationen für die Erweiterungen werden lediglich die entsprechenden XML Schemata importiert, wie Abbildung 6.4 zeigt.

Internationalisierbarkeit

multilinguale Erweiterungen

Es folgt die Festlegung des XML Schemas für die in Anforderung 3.11 formulierte Notwendigkeit der Internationalisierbarkeit von Erweiterungen. Dazu werden die dem Benutzer präsentierten Zeichenketten in die Spezifikation mit aufgenommen. Dabei kommt die Ablage als „Schlüssel-Wert“-Paar zum Einsatz. Die Originalzeichenkette, wie sie vom Entwickler bei der Implementation angegeben wurde, wird als Schlüssel verwendet (siehe Abschnitt 5.3.2). Das hat den Vorteil, dass bei fehlender Übersetzung in der Spezifika-

tion der Schlüssel direkt als Text ausgegeben werden kann. Aus diesem Grund darf der Schlüssel `textID` auch Leerzeichen und sonstige Sonderzeichen enthalten, genau wie die Übersetzung. Abbildung 6.5 zeigt das entsprechende XML Schema.

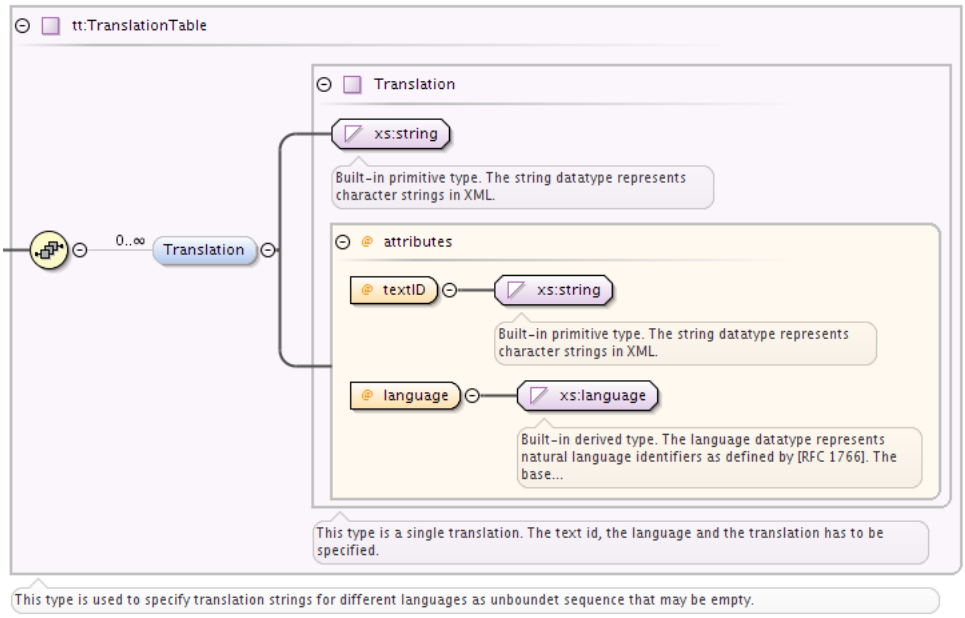


Abbildung 6.5: *XML Schema des Datentyps* `TranslationTable`

Einsatz von Sprachverarbeitungslösungen

Sprachsteuerung

Neben den bisher beschriebenen Elementen werden über einen entsprechenden XML-Schema Import weitere Elemente aus der Grammatik-Spezifikation des W3C für Sprachverarbeitungssysteme mit aufgenommen. Diese sollen die Steuerbarkeit aller Erweiterungen durch Sprachverarbeitung vorbereiten und die zur Verfügung stehenden Sprachkommandos enthalten. Die vom W3C standardisierte Grammatik kommt zum Einsatz, da sie direkt von vielen Sprachverarbeitungslösungen interpretiert werden kann. Detaillierte Informationen zu der Grammatik sind [HM04] zu entnehmen.

Zusammenfassung aller Teilspezifikationen

Gesamtspezifikation

Nach der Zusammenstellung der XML-Schemata für die unterschiedlichen Elemente der Spezifikation der Erweiterungen müssen diese in ein XML-Gesamtschema zusammengeführt werden. Neben den bereits beschriebenen Elementen sind weitere Angaben aufzunehmen, die der MMS zur Laufzeit benötigte Informationen über die entsprechende Erweiterung liefern. Dazu gehören Informationen über Name, Dateiname und Typ sowie

drei weitere Attribute, die das Aktivierungsverhalten, die Aktivierung als Benutzerinteraktion und die Positionierung in der MMS festlegen (siehe Abschnitt 6.1).

Das Attribut `name` ist vom Datentyp Zeichenkette mit Restriktion auf Buchstaben (groß und klein), Ziffern und Unterstriche (regulärer Ausdruck `([a-zA-Z0-9_])+`). Beim Attribut `fileName` handelt es sich ebenfalls um eine Zeichenkette mit Restriktion, die hier jedoch auch Punkte zulässt (als Suffixtrenner). Name

Der Typ einer Erweiterung wird über das Attribut `type` festgelegt, wobei zwischen den in Tabelle 6.3 dargestellten Typ-Bezeichnern unterschieden wird.

Zeichenkette	Beschreibung
`WidgetLess`	Erweiterung für Ein- oder Ausgabegerät
`WidgetBased`	Erweiterung mit grafischer Komponente
`Toolbar`	Erweiterung mit grafischer Komponente (dynamisches Toolbar-Piktogramm)

Tabelle 6.3: *Mögliche Typen für Erweiterungen*

Alle Erweiterungen, die ein Ein- oder Ausgabegerät integrieren und somit keine grafische Komponente besitzen, werden als `WidgetLess` spezifiziert und die Erweiterungen mit grafischer Komponente als `WidgetBased`. Eine Sonderrolle spielen die Erweiterungen, die als `Toolbar` spezifiziert werden. Auf der Implementationsseite verhalten sich diese wie die Erweiterungen mit grafischer Komponente, ihre Anzeigeposition wird allerdings auf die Toolbar der MMS beschränkt, wodurch nur wenig Platz für das grafische Element zur Verfügung steht. Eingesetzt wird diese Art von Erweiterung zum Beispiel für die Anzeige von Statusinformationen durch kleine Piktogramme. Typ

Das Attribut `dynamicLoad` legt das Aktivierungsverhalten der Erweiterung fest. Ist der Wert `true`, so wird die Erweiterung automatisch beim Start der MMS aktiviert (Ausnahme: Benutzerinteraktionen, vgl. Abschnitt 6.3.4). Das Attribut `Skill` markiert eine Erweiterung als Benutzerinteraktion, was dazu führt, dass diese ausschließlich durch die Planungsebene von MASSiVE aktiviert werden kann. automatische Aktivierung

Die übrigen Attribute `subWidget` und `dock` definieren die Platzierung der Erweiterung in der MMS. Dabei werden alle als `subWidget` markierten Erweiterungen im unteren Bereich der MMS dargestellt und die als `dockType` markierten im veränderbaren Dock-Bereich des MMS Hauptfensters. Dazu stehen für das Attribut `dock` die in Tabelle 6.4 angegebenen Werte zur Verfügung. Platzierung

Zeichenkette	Beschreibung
`Left`	linker Dock-Bereich
`Top`	oberer Dock-Bereich
`Right`	rechter Dock-Bereich
`Bottom`	unterer Dock-Bereich

Tabelle 6.4: *Mögliche Dock-Positionen für Erweiterungen*

Wichtig ist im Hinblick auf diese zwei Attribute, dass sie sich gegenseitig ausschließen und nur angegeben werden dürfen, wenn es sich um eine Erweiterung mit grafischer Komponente handelt. Dazu mehr im Abschnitt 6.3.4. Mit Angabe der zuvor beschriebenen Elemente ergibt sich somit das in Abbildung 6.6 dargestellte XML Schema.

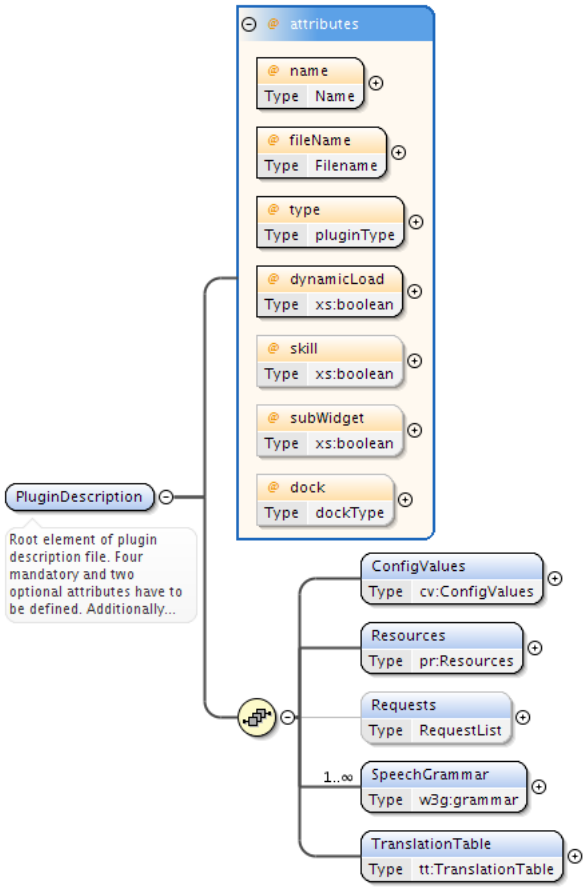


Abbildung 6.6: *XML Schema des Datentyps* `PluginDescription`

6.3.2 Validierung der Grammatik

Validierung beim Start der MMS

Die Validierung der zuvor in Form des XML Schemas festgelegten Grammatik der Erweiterungsspezifikationen kann direkt durch einen XML Schema fähigen XML-Parser ausgeführt werden. Dazu muss die MMS lediglich alle Spezifikationsdateien beim Start öffnen, dem Parser übergeben und im Anschluss daran auswerten. Sollten bei der Validierung Fehler auftreten, so ist eine Fehlerbeschreibung auszugeben und der Startvorgang der MMS abzubrechen.

Lösungsansatz 6.4. *Die MMS validiert die Spezifikationen aller Erweiterungen während des Startvorgangs. Bei Fehlern erfolgt die Ausgabe einer ausführlichen Fehlermeldung und der Abbruch des Startvorganges.*

6.3.3 Weitergehende Restriktionen

Über das zuvor beschriebene XML-Schema lassen sich die XML-Instanzen der Erweiterungsspezifikationen problemlos auf syntaktische und grammatikalische Korrektheit prüfen. Die Überprüfung von inneren Beziehungen zwischen Elementen des XML-Baumes ist nicht möglich. Genau diese sind jedoch unerlässlich, um die Interpretation der Spezifikationen zur Laufzeit robust zu machen, wie folgende Betrachtungen zeigen werden.

Für alle Erweiterungen zur Realisation von Benutzerinteraktionen erfolgt die Entscheidung über das Aktivieren und Deaktivieren zur Laufzeit ausschließlich auf Basis des aktuellen Status der Szenarienbearbeitung. Beides wird von der Planungsebene initiiert. Daraus folgt, dass bei diesen Erweiterungen das Attribut zur Festlegung der automatischen Aktivierung, unwahr sein muss. Aus diesem Zusammenhang lässt sich direkt die folgende Anforderung ableiten.

Sequenzer aktiviert Benutzerinteraktionen

Anforderung 6.1. *Besitzt das Element* `PlugDescription` *das optionale Attribut* `skill` *und ist dieses wahr, so muss das Attribut* `dynamicLoad` *unwahr sein.*

Eine weitere Anforderung ergibt sich bei Betrachtung der Attribute, die die Positionierung der Erweiterungen bestimmen. Es handelt sich dabei um die Attribute `type`, `dock` und `subWidget`. Indirekt könnte auch das Attribut `skill` über die Positionierung entscheiden, da Erweiterungen, die eine Benutzerinteraktion repräsentieren, immer als Reiter im Haupt-Widget der MMS dargestellt werden. Bei der Betrachtung von Abbildung 5.30 auf Seite 112 wird jedoch deutlich, dass dieses Kriterium bei der Entscheidung für die Darstellung nicht zur Anwendung kommt. Aus diesem Grund muss sichergestellt sein, dass bei allen Erweiterungen zur Realisation einer Benutzerinteraktion das Attribut `type` gleich `widgetBased` sein muss, was in Anforderung 6.2 formuliert ist.

Positionsüberprüfung

Anforderung 6.2. *Besitzt das Element* `PlugDescription` *das optionale Attribut* `skill` *und ist dieses wahr, dann muss das Attribut* `type` *gleich* `widgetBased` *sein.*

Darüber hinaus dürfen bei Erweiterungen für Benutzerinteraktionen die übrigen zwei Attribute zur Positionsfestlegung nicht vorhanden sein, woraus man eine weitere Anforderung formulieren kann.

Anforderung 6.3. *Besitzt das Element* `PlugDescription` *das optionale Attribut* `skill` *und ist dieses wahr, dann müssen die Attribute* `dock` *und* `subWidget` *fehlen.*

Als Schlussfolgerung lässt sich aus Abbildung 5.30 ableiten, dass sich die Attribute `dock`, `subWidget` und `skill` gegenseitig ausschließen. Dieses ist in Anforderung 6.4 formuliert.

Anforderung 6.4. *Von den optionalen Attributen* `dock`, `subWidget` *und* `skill` *des Elements* `PlugDescription` *darf immer nur eines angegeben sein. Darüber hinaus dürfen die Attribute* `skill` *und* `subWidget`*, wenn angegeben, nicht unwahr sein.*

Eine weitere Anforderung ergibt sich aus der Festlegung in Abschnitt 5.4.3. Hier wurde beschrieben, dass alle Erweiterungen, die eine Benutzerinteraktion realisieren, ausschließlich Zugriff auf die Ressource des UserInteraction-SkillServer erhalten dürfen.

Ressourcenbesch für Benutzer-interaktionen

Anforderung 6.5. *Wenn das Element* `PluginDescription` *das optionale Attribut* `skill` *enthält und dieses Attribut wahr ist, dann muss das Kindelement* `Resources` *genau ein Kindelement* `Resource` *mit dem Wert* `"IUserInteractionSkillServer"` *für das Attribut* `interface` *aufweisen. Wenn das Element* `PluginDescription` *das optionale Attribut* `skill` *nicht enthält oder dieses Attribut unwahr ist, dann ist der Inhalt und die Anzahl der Kind-Elemente von* `Resources` *beliebig.*

6.3.4 Validierung mittels Schematron

Abbildung auf Schematron

Damit die Spezifikationen gegen diese Restriktionen validiert werden können, soll hier auf die Validierung mittels Schematron, wie sie im Abschnitt 4.1.2 beschrieben wurde, zurückgegriffen werden. Dazu werden im Folgenden die beschriebenen Anforderungen auf Regeln des Schematrons abgebildet.

Betrachtet man die Anforderungen 6.1 und 6.2 so fällt auf, dass ihre Struktur sehr ähnlich ist. Beide haben als Ausgangspunkt, dass es sich bei der betrachteten Erweiterung um eine Benutzerinteraktion handelt. Die Restriktionen können in diesen Fällen daher in einer Regel zusammengefasst werden:

Definition 6.1. XML Schematron Regel, die die Erfüllung von Anforderung 6.1 und 6.2 validiert.

```
<sch:pattern>
  <sch:rule context="pd:PluginDescription">
    <sch:assert test="not(@skill)
                      or @skill='false'
                      or (@skill='true'
                          and @dynamicLoad='false'
                          and @type='WidgetBased')">
      If skill is 'true' dynamicLoad has to be 'false',
        type has to be 'WidgetBased' and dock as well
        as subWidget have to be ommited!
    </sch:assert>
  </sch:rule>
</sch:pattern>
```

Die in Anforderung 6.4 beschriebenen Restriktionen sorgen dafür, dass immer nur eines der Attribute skill, dock und subWidget angegeben werden kann. Zusammen mit Anforderung 6.3 kann man daraus folgende Schematron-Regel ableiten:

Definition 6.2. XML Schematron Regel, die die Erfüllung von Anforderung 6.3 und 6.4 validiert.

```
<sch:pattern>
  <sch:rule context="pd:PluginDescription">
    <sch:assert test=" (@skill
                        and not(@dock)
                        and not(@subWidget))
                       or
                        (not(@skill)
                        and @dock
                        and not(@subWidget))
                       or
                        (not(@skill)
                        and not(@dock)
                        and @subWidget)">
      There may exist only one (or none) of the following
        attributes: skill, dock or subWidget!
    </sch:assert>
    <sch:assert test="not(@subWidget)
                  or @subWidget='true'">
      Attribute subWidget has to be
        used with a value of true.
    </sch:assert>
    <sch:assert test="not(@skill)
                  or @skill='true'">
      Attribute skill has to be
        used with a value of true.
    </sch:assert>
  </sch:rule>
</sch:pattern>
```

Um eine Schematron Regel zu definieren, die gegen Anforderung 6.5 validiert, muss zunächst der Kontext der Regel so angepasst werden, dass sie nur auf die Ressourcenfestlegung der Spezifikationsdateien greift, deren `skill`-Attribut wahr ist. Die weitere Validierung erfolgt dann mittels zweier `assert`-Zusicherungen. Die erste überprüft die Anzahl der angegebenen Ressourcen und die zweite den Wert des Attributs `interface`.

Definition 6.3. XML Schematron Regel, die die Erfüllung von Anforderung 6.5 validiert.

```
<sch:pattern>
  <sch:rule context="pd:PluginDescription[@skill='true']">
    <sch:assert test="count(//pr:Resource)='1'">
      Only one resource is allowed!
    </sch:assert>
    <sch:assert test="//pr:Resource[@interface
                      ='IUserInteractionSkillServer']">
      The only resource that is allowed for skill
        plugins is the resource offering
        IUserInteractionSkillServer interface!
    </sch:assert>
  </sch:rule>
</sch:pattern>
```

6.3.5 Auswertung der Schematron-Validierung

Wie im Abschnitt 4.1.2 beschrieben wurde, handelt es sich bei der Validierung mittels XML-Schematron um die Validierung durch direkte Überführung der zu validierenden XML-Instanz in das Ausgabe- beziehungsweise Ergebnisdokument durch XSLT-Transformation. Dabei wird im ersten Schritt aus dem im Abschnitt 6.3.4 spezifizierten XML-Schematron über die drei im Abschnitt 4.1.2 beschriebenen vorbereitenden XSLT-Transformationen die XSLT-Instanz erzeugt, mit der die XML-Eingabeinstanzen (also die Spezifikationsdateien) validiert werden können.

offline Erstellung der XSLT-Instanz

Die nötigen Schritte der XML-Schematron Validierung lassen sich nach Abschnitt 4.1.2 in vier Schritte aufteilen. Um das Laufzeitverhalten der MMS nicht zu beeinträchtigen, werden die ersten drei Schritte einmalig offline durchgeführt. Dabei wird aus dem XML-Schematron das zur Validierung benötigte XSLT-Dokument erzeugt. Dieses steht danach für die online Validierung der Eingabedokumente, die dem unter 6.3.2 spezifiziertem XML-Schema genügen, zur Verfügung. Das Ergebnis ist das Ausgabedokument, eine XML-Instanz im SVRL-Format mit den Validierungsresultaten (weitere Informationen dazu in [ISO06a]).

Auswertung mittels XQuery

Eine einfache und elegante Möglichkeit die erzeugte SVRL-Ausgabe auszuwerten bietet die im Abschnitt 4.1.2 beschriebene Datenabfrage mittels XQuery. Über diese SQL-ähnliche Abfragesprache können sehr einfach Datenknoten aus XML-Instanzen extrahiert werden. In diesem konkreten Beispiel geht es nun darum, die bei der XML-Schematron Validierung aufgetretenen Fehler aus den Ausgabedateien zu extrahieren. Einen Auszug aus einer solchen Ausgabedatei zeigt das folgende Beispiel.

Beispiel 6.1. *Auszug aus einer XML Ausgabedatei im SVRL-Format*[2].

```
<svrl:failed-assert test="[...]" location="[...]">
  <svrl:text>
    If skill is 'true' dynamicLoad has to be 'false',
     type has to be 'WidgetBased' and
     dock and subWidget have to be ommited!
  </svrl:text>
</svrl:failed-assert>
```

Das Beispiel zeigt wie die Fehlermeldungen als Elemente vom Typ `<svrl:failed-assert>` `</svrl:failed-assert>` vorliegen. Die in den Schematron-Regeln festgelegten Informationstexte, sind nach der Validierung in den Kindelementen `<svrl:text></svrl:text>` enthalten. Um diese zu extrahieren bedient man sich der folgenden XQuery Datenabfrage:

```
declare namespace svrl="http://purl.oclc.org/dsdl/svrl";
for $error in doc(FILENAME)//svrl:failed-assert/svrl:text
  return fn:data($error)
```

[2]Die Attribute der Elemente wurden aus Gründen der Übersichtlichkeit durch „[...]“ ersetzt.

Zunächst wird der Namensraum der im XQuery Ausdruck verwendeten Elemente mittels `declare namespace` festgelegt. Im Anschluss daran erfolgt die Iteration über alle im Dokument vorkommenden Elemente vom Typ `svrl:failed-assert`. Als Ergebnis wird die Menge aller aufgetretenen Fehlermeldungen zurückgegeben. Sollten keine Fehler bei der Validierung aufgetreten sein, so ist die Menge leer. Details zur Realisierung dieser Auswertung folgen im nächsten Abschnitt 6.4.

leere Menge indiziert Fehlerfreiheit

6.4 Modellbasierte Entwicklung des Rahmenwerks

Nachdem die Grundlagen der Spezifikationen ausformuliert sind, bedarf es einer ausführlichen Betrachtung des gewählten Entwurfes für das Rahmenwerk der Erweiterungen. Es ist jetzt davon auszugehen, dass für jede zu entwickelnde Erweiterung zunächst eine Spezifikation erstellt wird, die gegen die in den vorherigen Abschnitten festgelegte Grammatik als auch gegen die durch ein XML-Schematron beschriebene Semantik fehlerfrei validiert. Durch diese Grundvoraussetzungen wird die Verarbeitung der Informationen einfacher und robuster.

Als Basis des Rahmenwerks für die Erweiterungen sollen Merkmale der im Abschnitt 4.1.1 beschriebenen Methodik wieder aufgegriffen werden. Dabei spielt die MMS die Rolle der Rahmenanwendung, welche hier zunächst unbeachtet bleiben wird. Der Kontextvertrag zwischen den Erweiterungen und der MMS wurde im Abschnitt 5.4.1 in Form der dort festgelegten Schnittstelle beschrieben und wird durch die Spezifikation vervollständigt.

Rahmenanwendung und Kontextvertrag

Zunächst erfolgt die Betrachtung des zuvor formulierten Lösungsansatzes 6.1, der die Notwendigkeit der Implementierung von einigen Erweiterungen als aktive Objekte beschreibt, um reaktives Verhalten, auch ohne Aktion des Nutzers, zu erreichen. Bei allen Erweiterungen, die über eine grafische Komponente verfügen, ist dieses nicht notwendig, da hier der Benutzer über Schaltflächen mit der Erweiterung interagiert. Generell kann man daher sagen, dass diese Erweiterungen nicht als aktive Objekte zu realisieren sind, jedoch stattdessen über Funktionalitäten verfügen müssen, die eine grafische Darstellung in der MMS erlauben. Das für die MMS eingesetzte Qt-Rahmenwerk besitzt für diese Zwecke die Klassen `QThread` beziehungsweise `QWidget`. Dementsprechend lässt sich die Aufteilung der Erweiterungen in die zwei Grundformen Typ 1 und Typ 2 (Typ 3 lässt sich in diesem Zusammenhang zum Typ 2 zwei hinzuzählen, siehe Abschnitt 6.2.3) direkt auf das UML-Modell übertragen. Dazu werden alle Erweiterungen für Ein- und Ausgabegeräte als Spezialisierung der Klasse `QThread` realisiert. Die übrigen Erweiterungen werden dagegen als Spezialisierung der Klasse `QWidget` implementiert. Abbildung 6.7 stellt diesen Zusammenhang dar.

zwei Implementationmodelle für alle Erweiterungen

Neben den spezifischen Funktionalitäten, die jede Erweiterung zur Verfügung stellt, muss sie zusätzlich noch die im Vertrag festgelegte Schnittstelle implementieren. Diese ist von jeder Erweiterung zu exportieren und ermöglicht so den Zugriff durch die MMS. Dabei handelt es sich um die Schnittstelle `IWidgetLessPlugin` für Ein- und Ausgabegeräte und `IWidgetBasedPlugin` für die übrigen Erweiterungen (Typ 2 und Typ 3). Diese zusät-

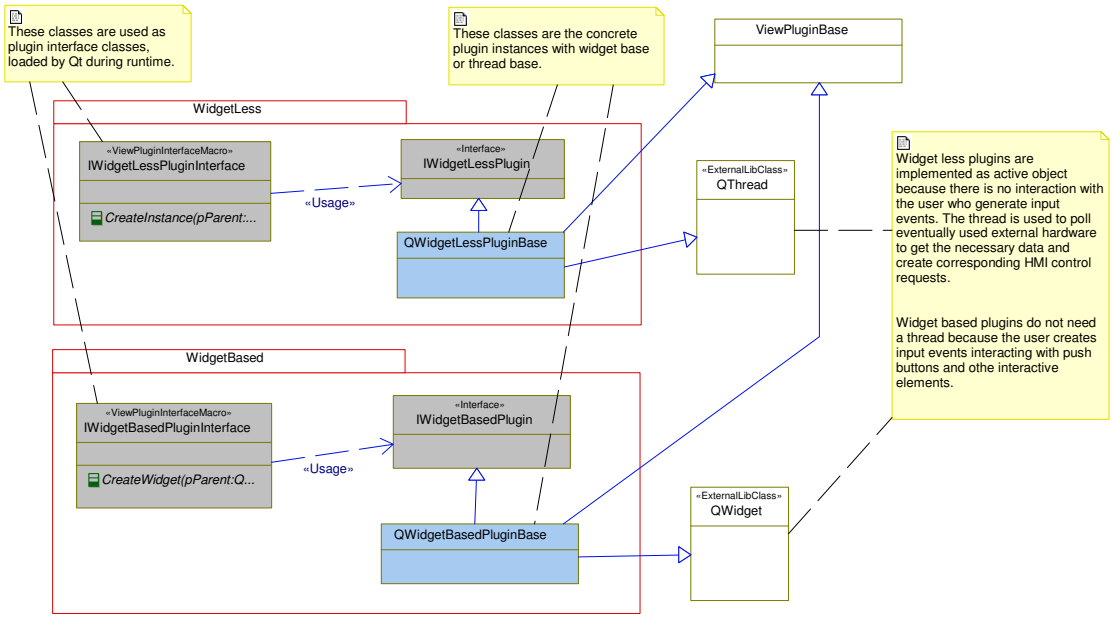


Abbildung 6.7: *UML-Entwurf der Basisklassen (blau dargestellt) für die Realisation der Erweiterungen.*

zliche Spezialisierung entspricht dem in [CEM03] beschriebenem Entwurfsmuster für Erweiterungen. Um den sich wiederholenden Implementationsaufwand für Erweiterungen zu reduzieren, werden für die unterschiedlichen Erweiterungstypen (Typ 1 und Typ 2 beziehungsweise 3) bereits einige Funktionalitäten direkt in den zuvor eingeführten Basisklassen implementiert. Das sind im Fall der Erweiterungen für Ein- und Ausgabegeräte die Methoden `SetOperative()` und `SetInoperative()`, die die Instanz der Klasse `QThread` initialisieren und den Thread starten beziehungsweise stoppen. Für die Erweiterungen mit grafischer Komponente wird dagegen die Methode `GetWidget()` implementiert. Über diese erhält die MMS Zugriff auf die `QWidget` Instanz, welche als grafisches Element angezeigt wird.

separater Thread für Ein- und Ausgabegeräte

Abbildung 6.7 zeigt den Entwurf des Rahmenwerkes für die Erweiterungen. Dabei wird auf Details in der Darstellung bewusst verzichtet. Betrachtet man den Entwurf genau, so fallen zwei Klassen auf, die bisher noch nicht diskutiert wurden. Bei diesen Klassen handelt es sich um die Schnittstelle, die jede Erweiterung an die MMS exportiert und dieser so die Möglichkeit bietet, die Erweiterungen zur Laufzeit zu erzeugen. Dazu besitzen beide jeweils eine Fabrikmethode (siehe Abschnitt 4.1.1 beziehungsweise [GHJV95] für Details), die die konkrete Erweiterung zur Laufzeit instanziiert und einen Zeiger darauf zurück gibt. Die erste Klasse `IWidgetLessPluginInterface` bildet dabei die Schnittstelle für das Instanziieren von Typ 1-Erweiterungen und die zweite Klasse `IWidgetBasedPluginInterface` die Schnittstelle für das Erzeugen der Erweiterungen vom Typ 2 und 3.

Instanziierung über Fabrikmethode

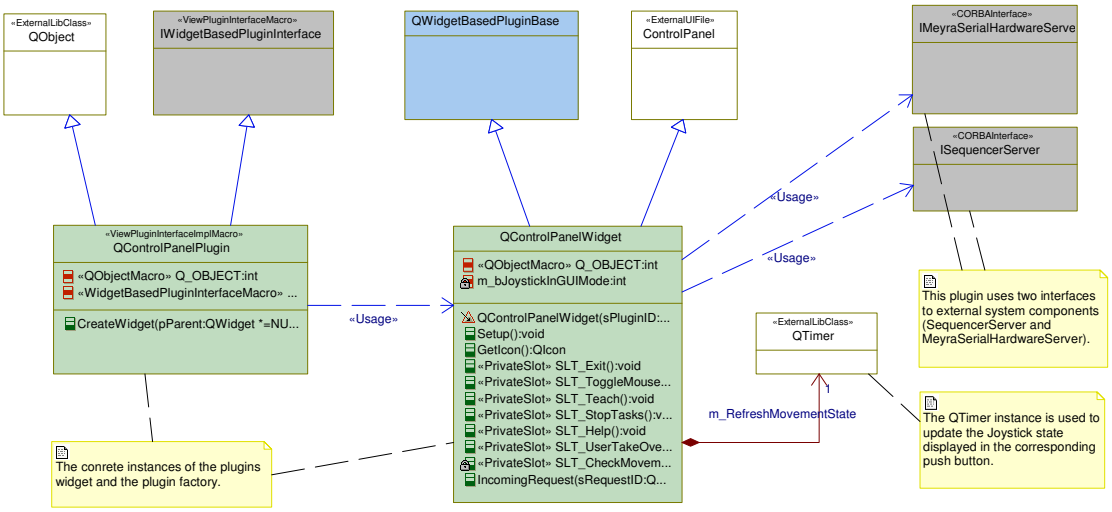


Abbildung 6.8: *Der konkrete Entwurf einer Erweiterung vom Typ 2 am Beispiel der Erweiterung „ControlPanel“*

Der konkrete Modellentwurf für die Erweiterungen soll am Beispiel der Typ 2-Erweiterung „ControlPanel“ in Abbildung 6.8 dargestellt werden (nähere Informationen zu dieser Erweiterung sind Anhang A.2.3 zu entnehmen).

Die konkrete Fabrik `QControlPanelPlugin` spezialisiert das Interface der abstrakten Fabrik indem die Methode `CreateWidget()` implementiert wird. Sie enthält die Funktionalität zum Erzeugen der Instanz der Erweiterung, welche mittels Zeiger zurück gegeben wird. Da es sich bei der Methode um eine `virtuelle` Methode handelt, ist zur Laufzeit in der MMS garantiert, dass die korrekte Implementation in der Spezialisierung aufgerufen wird, um die Instanz vom Typ `QControlPanelWidget` zu erzeugen.

Laden der Bibliotheken durch Qt

Das Kompilat aus der Implementierung der Erweiterung und den verwendeten Qt-Bibliotheken resultiert in einer dynamischen Bibliothek, die zur Laufzeit geladen werden kann. Diese Funktionalität wird dabei in der MMS durch Funktionalitäten des verwendeten Qt-Rahmenwerks durchgeführt, wie bereits ausführlich im Abschnitt 5.4.2 beschrieben wurde. Die detaillierte Beschreibung der Realisierung von Erweiterungen inklusive der einzuhaltenden Grundvoraussetzungen und der einzusetzenden Basisklassen ist der Qt Referenzdokumentation [Nok10] zu entnehmen.

Das Kernelement der Erweiterung ist die konkrete Erweiterung in Form der Spezialisierung der abstrakten Schnittstelle `QWidgetBasedPluginBase`. Zusätzlich zur Vererbung von der Schnittstellenklasse zeigt Abbildung 6.8 eine weitere Vererbung von einer Klasse des Stereotyps `<<ExternalUiFile>>`. Diese Klasse beinhaltet das vom Qt-Designer erstellte grafische Element der Erweiterung (siehe Qt Referenzdokumentation [Nok10, Stichwort: „multiple inheritance approach“]). Diese Vererbung ist daher auch nur bei Erweiterungen vom Typ 2 oder 3 notwendig, wie das Modell der Erweiterung vom Typ 1 in Abbildung 6.9 zeigt.

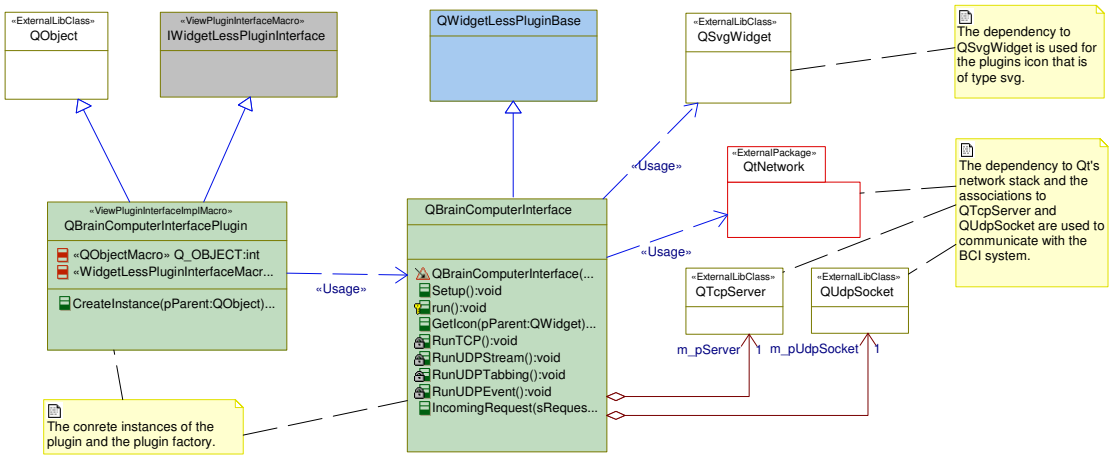
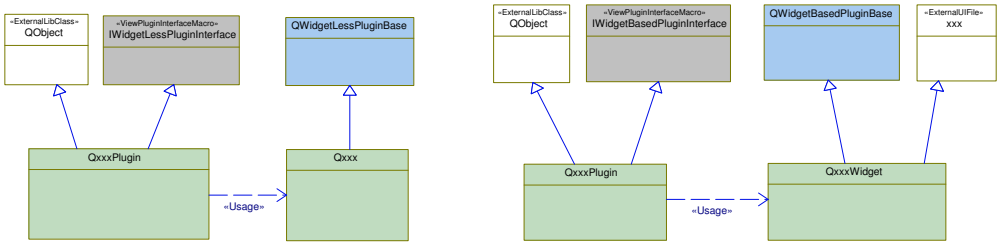


Abbildung 6.9: *Der konkrete Entwurf einer Erweiterung vom Typ 1 am Beispiel der Erweiterung „BrainComputerInterface“*

Bei den weiteren in Abbildung 6.8 und 6.9 dargestellten und bisher nicht beschriebenen Elementen handelt es sich um Elemente, die der Realisierung von erweiterungsspezifischen Funktionalitäten dienen und nicht generell zum Modell einer Erweiterung gehören.

allgemeine UML-Modelle

Zusammenfassend zeigt Abbildung 6.10 die allgemeinen UML-Modelle für Erweiterungen vom Typ 1 (Abbildung 6.10a) oder Typ 2 und 3 (Abbildung 6.10b). In den dabei verwendeten Klassennamen ist das „xxx“ durch den konkreten Namen der Erweiterung zu ersetzen, wie er zuvor in der Spezifikation festgelegt wurde (siehe Abschnitt 6.3).



(a) *Modell für Erweiterungen vom Typ 1* **(b)** *Modell für Erweiterungen vom Typ 2 oder 3*

Abbildung 6.10: *Allgemeine Modelle für die Erweiterungen der MMS*

6.4.1 Aktivierung von Erweiterungen

Auswertung der Spezifikationen

Bei der Aktivierung einer Erweiterung wird (siehe Abschnitt 5.4.2) zunächst die entsprechende Spezifikation geladen und interpretiert. Im Anschluss daran wird die Erweiterung vom Display-Manager (implementiert in der Klasse `DisplayManager`) geladen. Dann erfolgt

die Instanziierung durch die entsprechende Fabrik-Methode, die Registrierung der Erweiterung beim Request-Dispatcher (siehe Abschnitt 5.4.1) und die Registrierung der von der MMS exportierten Schnittstellen zum Zugriff auf MMS- und Systemfunktionen. Im Anschluss daran wird die `Setup()`-Methode aufgerufen, die die Initialisierung der Erweiterung durchführt.

Registrierung der Schnittstellen

Die Registrierung der Schnittstellen auf die von der MMS exportierten Systemfunktionen erfolgt unmittelbar vor dem Aufruf der `Setup()`-Methode. Dadurch ist es möglich bereits bei der Initialisierung auf alle von der MMS zur Verfügung gestellten Funktionalitäten zurückzugreifen. Die Registrierung der Schnittstellen wird über die Methode `SetExposedInterfaces()` der Basisklasse `ViewPluginBase` (siehe Abbildung 6.7) aller Erweiterungen durchgeführt. Dieser sind dazu die Zeiger aller Schnittstellen in der in Abschnitt 5.4.1 festgelegten Struktur `InterfaceContainer` zu übergeben. Die Vorgänge bei der Registrierung sind in Abbildung 6.11 als Sequenzdiagramm dargestellt.

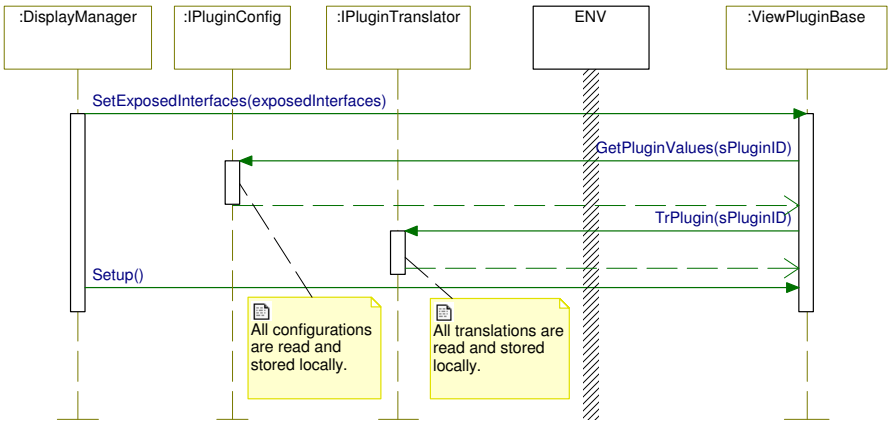


Abbildung 6.11: *Die Vorgänge bei der Initialisierung einer Erweiterung*

Auf der linken Seite sind in Abbildung 6.11 die Instanzen dargestellt, die zur MMS gehören. Auf der rechten Seite die Instanz der Klasse `ViewPluginBase`, welche die konkrete Erweiterung repräsentiert. Zunächst wird eine Kopie der übergebenen Struktur mit den Schnittstellen in der Erweiterung abgelegt, um im Anschluss daran über die entsprechenden Schnittstellen der Übersetzungskomponente und der Konfigurationskomponente der MMS sowohl die komplette Übersetzung der Erweiterung als auch sämtliche Konfigurationswerte in lokalen Abbildungstabellen zu speichern. Durch die lokale Kopie ist es möglich zur Laufzeit direkt mit den Informationen zu arbeiten, ohne diese bei jedem Zugriff über die MMS akquirieren zu müssen (dazu mehr im Abschnitt 6.4.2).

Initialisierung der Erweiterungen

Der Aufbau und die in der `Setup()`-Methode durchzuführenden Initialisierungsschritte sind generell beliebig und können somit vom Entwickler frei implementiert werden. Als sehr nützlich hat sich die Ausgabe der aktuellen Werte für alle Konfigurationswerte, die von der Erweiterung über die Spezifikation festgelegt wurden, erwiesen. Dabei kann die von der MMS zur Verfügung gestellte Funktionalität zur Ausgabe von Meldungen über

die Methode `LogEvent()` direkt eingesetzt werden. Auf diese Weise lassen sich alle Werte einfach zur Laufzeit kontrollieren.

6.4.2 Kapselung der MMS Schnittstellen

Alle von der MMS zur Verfügung gestellten Systemfunktionen werden den Erweiterungen über die Schnittstellen zur Verfügung gestellt, deren Registrierung im vorherigen Abschnitt beschrieben wurde. Dieser Abschnitt widmet sich nun der Integration und der Kapselung der Schnittstellen in den Erweiterungen.

Zugriff auf die Ressourcenverwaltung

Wie in Abschnitt 5.2 beschrieben wurde, liefert die von der MMS exportierte Ressourcenverwaltung die Proxies zu Systemressourcen als `CORBA::Object` an die Erweiterungen. Die Konvertierung auf den konkreten Schnittstellentyp wird dabei nicht durchgeführt. Diese Vorgehensweise macht die MMS vollständig unabhängig von den möglicherweise von Erweiterungen benötigten Systemkomponenten. Auch zukünftige Neuentwicklungen von Systemkomponenten können so ohne Anpassungen an der MMS durch Erweiterungen angesprochen werden.

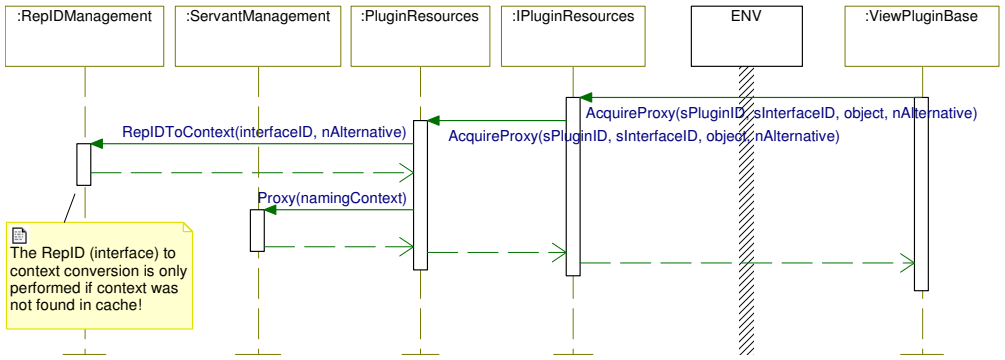


Abbildung 6.12: *Vorgänge bei der Ressourcen-Akquise durch eine Erweiterung*

Konvertierung der akquirierten Proxies

Das Fehlen der Konvertierung von Proxy-Schnittstellen in der MMS macht es jedoch notwendig, diese in den Erweiterungen durchzuführen. Dazu wird die Template-Methode `AcquireResource()` in der Basisklasse `ViewPluginBase` für alle Erweiterungen implementiert, die den Zugriff auf die von der MMS gelieferte Schnittstelle kapselt. Wenn eine Erweiterung Zugriff auf eine Ressource benötigt, kann sie die Template-Methode mit der Klasse der korrespondierenden Proxy-Schnittstelle typisieren und aufrufen. Die zusätzlichen Argumente sind entweder die Repository-ID und die zu verwendende Alternative oder der Namenskontext. Abbildung 6.12 zeigt den Vorgang bei Verwendung der Repository-ID. Zunächst erfolgt die Konvertierung der Repository-ID in den eindeutigen

Kontext der Ressource, um dann mit Hilfe der `ServantManagement` Instanz den Proxy zu akquirieren und an die Erweiterung durchzureichen. Dort erfolgt dann die Konvertierung auf den konkreten Schnittstellentyp der Ressource.

Die `AcquireResource()` Methode, der der Namenskontext direkt übergeben wird, ist als Rückfallmöglichkeit zu sehen, wenn das Akquirieren der Ressource über die flexiblere Akquirierungsmethode mittels Trading-Service fehlschlägt (diese Methodik wurde ausführlich im Abschnitt 5.2 entwickelt und beschrieben).

Zugriff auf Konfigurationswerte

Zusätzlich zu den bereits beschriebenen Konfigurationswerten (siehe Abschnitt 6.3.1) ist den Erweiterungen ein lesender Zugriff auf globale Konfigurationswerte zu ermöglichen. Die grundlegende Funktionalität dazu wurde bereits im Abschnitt 5.4.1 entwickelt und über die entsprechende Schnittstelle exportiert.

lesender Konfigurationszugriff

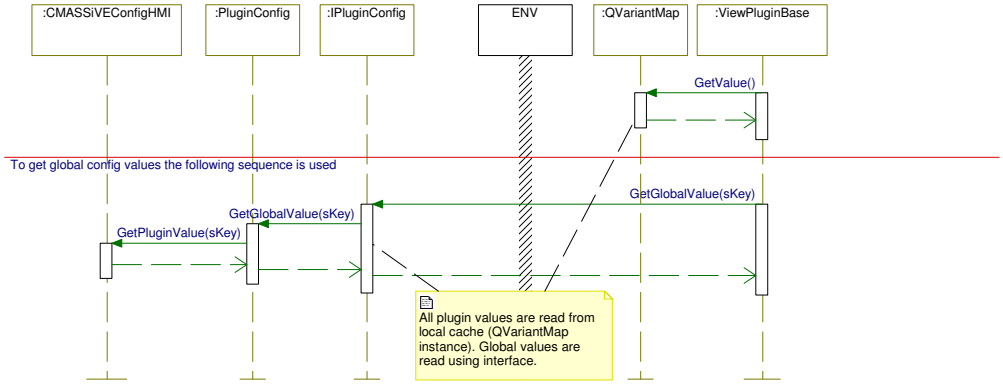


Abbildung 6.13: *Konfigurationswerte der Erweiterung werden aus dem lokalen Zwischenspeicher gelesen, alle anderen über die Schnittstelle zur MMS*

Sämtliche Konfigurationswerte werden über Zeichenketten identifiziert. Die Kapselung der MMS Schnittstelle erfolgt in der Methode `GetSettingsValue()`, welche über ein notwendiges und ein optionales Argument zu parametrieren ist. Das erste Argument legt den Identifikator des Konfigurationswertes fest. Über das zweite optionale Argument wird festgelegt, ob es sich um einen lokalen oder globalen Konfigurationswert handelt (bei fehlender Angabe wir der Identifikator als lokaler Wert interpretiert). Abbildung 6.13 zeigt die möglichen Aufrufsequenzen.

Ausgabe von Meldungen

Zuordnung von Meldungen

Bei der Kapselung der Log-Ausgaben ist beim Aufruf der entsprechenden Schnittstellenmethoden lediglich die ID der konkreten Erweiterung zu übergeben, da diese in den Log-Ausgaben verwendet wird, um die Zuordnung der Meldungen zu vereinfachen. Darüber hinaus erfolgt eine Umsetzung des Datentyps, mit dem die Art der auszugebenden Meldung näher spezifiziert wird (Info-, Warnung-, Fehler- oder Debug-Meldung). Das ist notwendig, da innerhalb der Erweiterungen mit einem eigenen Datentyp gearbeitet wird, der mit zur Schnittstellenspezifikation gehört. Nur so ist es möglich, die Erweiterungen vom Meldungs-Ausgabemodul in MASSiVE unabhängig zu machen.

Übersetzung

Die Initialisierung der Übersetzungsfunktionalität wird bereits während der Registrierung einer Erweiterung durchgeführt (siehe Abschnitt 6.4.1). Alle zur Verfügung stehenden Übersetzungen der aktuell in der MMS eingestellten Sprache werden dabei in einer lokalen Abbildungstabelle abgelegt.

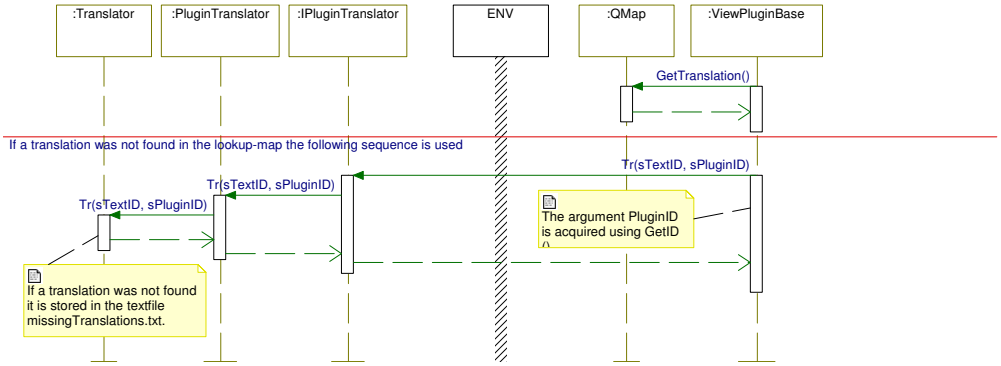


Abbildung 6.14: *Vorgänge, die innerhalb einer Erweiterung zur Übersetzung von Zeichenketten durchgeführt werden*

Übersetzung zur Laufzeit

Zur Laufzeit müssen alle sichtbaren Texte in dieser Tabelle nachgeschlagen und angezeigt werden. Dazu wird in der Basisklasse der Erweiterungen `ViewPluginBase` die Methode `Tr()` implementiert. Diese Methode erhält als einziges Argument die Zeichenkette des originalen Textes (in der Regel der englische Text) und gibt die übersetzte Zeichenkette zurück. Das genaue Vorgehen wurde bereits im Abschnitt 5.3.2 beschrieben. Abbildung 6.14 zeigt eine mögliche Aufrufsequenz.

6.4.3 Deaktivierung von Erweiterungen

Aufräumen durch Destruktoren

Bei der Deaktivierung einer Erweiterung sind keine umfangreichen Vorkehrungen zu treffen. Die internen Aufräumarbeiten erfolgen in den Destruktoren der konkreten Erweiterungen. Bei den Erweiterungen vom Typ 1 wird darüber hinaus die Methode `SetInoperative()` aufgerufen, um den Thread (sofern einer gestartet wurde) zu stoppen. Für alle Erweiterungen ist darüber hinaus nur die Registrierung beim Request-Dispatcher aufzuheben und die Instanz der Erweiterung zu löschen.

Das, wobei unsere Berechnungen versagen, nennen wir Zufall.

Albert Einstein - Physiker und Nobelpreisträger

7

Adaption durch KDD

Auf Basis der in den vorherigen Kapiteln entwickelten Schnittstellen und Spezifikationen wird im vorliegenden Kapitel anhand des in Abschnitt 4.2.4 vorgestellten WinEpi Algorithmus eine Erweiterung realisiert, mittels derer die MMS in die Lage versetzt wird, die zur Verfügung stehenden Szenarien in Abhängigkeit von den Gewohnheiten des Benutzer anzuzeigen. Dazu erfolgt die Untersuchung der vom Benutzer initiierten Ereignisse auf wiederkehrende Muster.

Vier sequentielle Schritte

Nach den im Abschnitt 2.4 beschriebenen Methodiken des KDD besteht der Vorgang der Datenanalyse und Wissensextraktion aus vier sequentiellen Schritten. Der erste Schritt beschreibt die Erstellung der Datenbasis. Im Anschluss daran kommen die Algorithmen der Datenauswertung zum Einsatz. Das in diesem Schritt extrahierte Wissen wird in einer Wissensbasis abgelegt und dient im folgenden Schritt der Inferenz als Basis. Im letzten Schritt werden die Methoden zum Lernen, also zur dynamischen Anpassung der Wissensbasis, implementiert.

Die Folgenden Abschnitte bilden diese Vorgehensweise bei der Implementierung von KDD-Systemen auf die MMS der MASSiVE-Architektur ab.

7.1 Datenselektion und -akquise

Datenquellen

Für den Aufbau der Datenbasis ist es zunächst notwendig, die möglichen Datenquellen zu spezifizieren. In der MASSiVE-Architektur stehen in diesem Zusammenhang drei möglichen Datenquellen zur Diskussion: statisches Systemwissen, die globale Konfiguration und die spezifische Datenerfassung durch die MMS oder deren Erweiterungen. Die folgenden Abschnitte erläutern welche der möglichen Datenquellen für den zu realisierenden KDD-Prozess in Frage kommen.

7.1.1 Statisches Systemwissen

Unter statischem Systemwissen versteht man alle Informationen, die offline erstellt oder bearbeitet werden und den Komponenten der Architektur zur Laufzeit nur lesend zur Verfügung stehen. Im Fall von MASSiVE sind diese Informationen in der globalen Konfiguration und der Systemdatenbank abgelegt. Die Tauglichkeit dieser Informationen ist für KDD nur sehr begrenzt, da keine Veränderung der Daten zur Laufzeit erfolgt. Das heißt, die Informationen bilden nicht den aktuellen System-, Benutzer- oder Umgebungszustand ab.

Globale Konfiguration

Konfiguration

Die globale Konfiguration von MASSiVE enthält neben den Konfigurationsdaten für die Erweiterungen der MMS überwiegend CORBA-Einstellungen, Parameter für das Planungs- und Ausführungsverhalten des Sequenzers sowie die Systemkonfiguration.

Systemdatenbank

Systemdatenbank

Bei der Modellierung der Systemdatenbank wurde festgelegt, dass diese neben statischen Systemdaten auch dynamische beziehungsweise quasi-statische Daten aufnehmen soll. Dazu erfolgte die Aufteilung der Systemdatenbank in drei Datenbanken:

- Die **APK[1]-SDB[2]** enthält statische Informationen über die zur Verfügung stehenden Szenarien. Darüber hinaus sind dort auch die Übersetzungen der Informationstexte zu allen Szenarien zu finden (siehe Abschnitt 5.4.1).
- In der **RTK[3]-QSDB[4]** befinden sich Laufzeit-Informationen, die quasi-statisch abgelegt sind. Das heißt, diese Daten können zur Laufzeit erstellt oder verändert werden, behalten aber darüber hinaus ihre Gültigkeit. Unter diesen Daten erfolgt auch die Ablage der Nutzerdaten (siehe Abschnitt 5.4.1).
- Die **TRE[5]-SDB** beinhaltet das gesamte vom Sequenzer benötigte Planungswissen. Diese Datenbank ist genau wie die APK-SDB Datenbank statisch.

Zusammenfassend lässt sich in Bezug auf die statischen Systemdaten festhalten, dass diese Informationen nicht für einen KDD-Prozess geeignet sind. Auf die quasi-statische Datenbank wird dennoch im Abschnitt 7.1.3 nochmals eingegangen.

[1] **A**-**P**riori **K**nowledge
[2] **S**tatic **D**ata**b**ase
[3] **R**un-**T**ime **K**nowledge
[4] **Q**uasi **S**tatic **D**ata**b**ase
[5] **T**ask **r**epresenting **e**lements

7.1.2 Analyse der Datenströme im System

Neben dem statischen Systemwissen sind die dynamischen Datenströme in der MAS-SiVE-Architektur auf ihre Einsatzmöglichkeit für den KDD-Prozess zu untersuchen. Bei diesen handelt es sich um die Kommunikationskanäle zwischen den einzelnen Komponenten in der Architektur.

Betrachtet man in diesem Zusammenhang die Kommunikation zwischen den Komponenten innerhalb des Skill-Netzwerks der reaktiven Ebene, so stellt man fest, dass diese Kommunikation keinerlei verwertbare Nutzdaten transportiert. Die bei den Skill-Aufrufen transportierten Informationen bestehen lediglich aus Zeichenketten, die der Identifikation von Daten innerhalb des Weltmodells dienen. Die gesamte Kommunikation zwischen den Skill-Servern erfolgt über das Weltmodell, das alle Zugriffe auf die abgelegten Daten verwaltet. Der Einsatz der Daten als Datenbasis für einen KDD-Prozess scheitert daran, dass innerhalb des Weltmodells kein Bezug zum Ausführungskontext der Datenzugriffe durch die Skill-Server hergestellt werden kann. Letzteres ist aber notwendig, um auf Basis der Daten Rückschlüsse über die ausgeführten Systemaufgaben und deren Beziehungen untereinander zu bilden.

keine nutzbaren Informationen

Neben der Kommunikation der Skill-Server im Skill-Netzwerk erfolgt die Kommunikation mit den zugeordneten Hardware-Servern in der reaktiven Ebene. Dabei wird der Datenaustausch nicht über das Weltmodell, sondern direkt über die exportierten Schnittstellen der Hardware-Server abgewickelt. Wie bereits im Zusammenhang mit dem Weltmodell besteht auch innerhalb der Hardware-Server keine Beziehung zum Ausführungskontext. Das Gleiche gilt auch für die direkte Steuerung, welche durch die MMS initiiert und über die Hardware-Server der reaktiven Ebene ausgeführt wird, so dass auch diese Daten nicht als Datenbasis für einen KDD-Prozess in Frage kommen.

7.1.3 Datenerfassung in der MMS

Wie sich in den vorherigen Abschnitten gezeigt hat, lassen sich weder das statische Systemwissen noch die dynamischen Datenströme in der reaktiven Ebene als Datenbasis für KDD-Prozesse nutzen. Daher ist es notwendig eine Datenbasis über Mechanismen innerhalb der MMS zu erstellen.

Erstellung in der MMS

Dazu kommt das in Abschnitt 5.3.1 eingeführte Attribut in der Spezifikation von Anfragen zum Einsatz. Über dieses Attribut lassen sich Anfragen markieren, die dadurch nach ihrem Eintreffen in der MMS serialisiert und als Datenbasis abgelegt werden. Da diese Daten zur Laufzeit erstellt und verändert werden können, müssen sie in der dafür vorgesehenen Datenbank für quasi statische Systemdaten, der RTK-QSDB-Datenbank, aufgenommen werden.

Damit der in Anforderung 3.14 formulierten Notwendigkeit, die aufgezeichneten Nutzerdaten einem Benutzer zuordnen zu können, Rechnung getragen werden kann, wird an

Nutzer-ID

dieser Stelle eine Nutzer-ID eingeführt. Diese wird bei der Erstellung der Datenbasis mit jedem einzelnen Datensatz abgelegt und realisiert so die Zuordnung. Die Nutzer-ID wird somit zu einer Kennung des Benutzers, die von diesem vor der ersten Benutzung zu konfigurieren ist. Aus diesem Grund ist die Systemkonfiguration um die Nutzer-ID zu erweitern.

Durch die beschriebene Serialisierung werden alle Datenfelder soweit möglich in Zeichenketten konvertiert und über Doppelpunkte getrennt konkateniert. Zusätzlich zur Ergebniszeichenkette dieses Vorgangs wird ein absoluter und ein relativer Zeitstempel (Unix Zeitstempel) erzeugt. Zusammen mit der Nutzer-ID entsteht so ein Datensatz mit vier Attributen, der die Struktur der zu verwendenden Datenbanktabelle `usageData` festlegt, wie Abbildung 7.1 als ERM[6] (siehe [Wik10, Stichwort: „Entity-Relationship-Modell“]) zeigt.

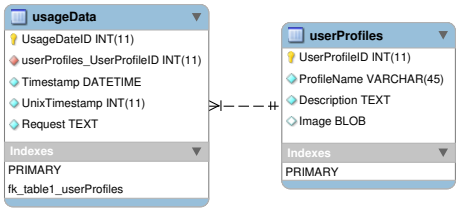


Abbildung 7.1: *ERM der SQL-Objekte für die Datenerfassung der MMS*

Abbildung auf ERM

Neben den bereits beschriebenen Attributen enthält die Datenbanktabelle `usageData` ein weiteres Attribut, das als Primärschlüssel verwendet wird, sowie die integrierte Nutzer-ID als Fremdschlüssel. Diese wird über eine nicht-identifizierende[7] „1 zu n“-Relation aus der Tabelle `userProfiles` referenziert. Letztere dient der Ablage einer Beschreibung und eines Bildes zu allen Benutzern des Systems. Das Attribut `UserProfileID` entspricht der Nutzer-ID.

Neben der soeben beschriebenen Möglichkeit zur Erstellung einer Datenbasis durch die MMS ist nach Anforderung 3.14 die Realisation von individuellen Datenbasen für die Erweiterungen notwendig. Die dazu in Abschnitt 5.4.1 entwickelte Funktionalität steht bereits allen Erweiterungen über eine Schnittstelle zur Verfügung.

Eine Tabelle für jede Datenbasis

Da die individuellen Datenbasen der Erweiterungen über eine unterschiedliche Anzahl von Attributen verfügen können, ist es nicht möglich diese auf eine identische Tabellenstruktur abzubilden. Daher wird für jede Datenbasis eine neue Tabelle erstellt, die über eine Relation auf die Nutzer-ID verweist. So können die Erweiterungen in die Lage versetzt werden, gezielt die aufgezeichneten Daten zum aktuellen Benutzer zu betrachten oder alle in der Datenbasis abgelegten (so wie es über die Schnittstelle vorgesehen ist). Zur Verdeutlichung dieser Vorgehensweise zeigt Abbildung 7.2 das ERM von zwei möglichen Datenbasen.

[6] **E**ntity **R**elationship **M**odell
[7] Nicht als Teil des Primärschlüssels aufgenommene

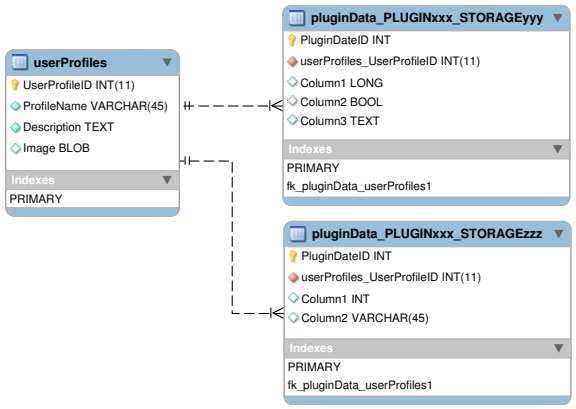


Abbildung 7.2: *Beispiel ERM der SQL-Objekte für zwei Datenbasen*

kodierte Tabellennamen

Die Zuordnungen der Tabellen zu den entsprechenden Erweiterungen zur Laufzeit, erfolgt über einen kodierten Tabellennamen. Dieser wird eingeleitet mit der Zeichenkette „pluginData“, gefolgt vom Namen der Erweiterung (im Beispiel „PLUGINxxx“) und abgeschlossen mit der ID der korrespondierenden Datenbasis (im Beispiel „STORAGEyyy“ beziehungsweise „STORAGEzzz“). Die einzelnen Elemente werden dabei durch Unterstriche getrennt, wie die folgenden Beispiele für die fiktiven Erweiterungen „Plugin1“ und „Plugin2“ zeigen.

Beispiel 7.1. *Mögliche Tabellennamen der Datenbasen für zwei fiktive Erweiterungen.*

```
1 pluginData_PLUGINPlugin1_STORAGEMiningResults
2 pluginData_PLUGINPlugin1_STORAGEAssociationRules
3 pluginData_PLUGINPlugin2_STORAGEUsageData
```

7.2 Datenaufbereitung

Aufzeichnung von Testdaten

Nach der Betrachtung von möglichen Datenquellen und der Beschreibung von Funktionalitäten zur Erstellung einer Datenbasis widmet sich der folgende Abschnitt der Analyse und Aufbereitung von akquirierten Daten. Die in diesem Zusammenhang vorgestellten Algorithmen werden zu Test- und Evaluierungszwecken direkt auf eine Datenbasis angewandt, die im Rahmen der Hannover-Messe 2010 durch die MMS angelegt wurde. Zu diesem Zweck sind mehrere Anfragen in der Anfragespezifikation der MMS für die Akquise markiert worden, wie der folgende Auszug aus der Spezifikation zeigt (das Attribut `usedForMining` in den Zeilen 10 und 15 ist wahr).

```
1 <rl:Request>
2   <type:ID descriptor="ScenarioControl"
3            target="Plugin"
4            level="HMI"
5            requestType="Info"
6            requestMode="Request" />
```

```
  <type:Date name="COMMAND"
             type="QString"
             direction="in"
             usedForMining="true"
             mandatory="true" />
  <type:Date name="NAME"
             type="QString"
             direction="in"
             usedForMining="true"
             mandatory="false" />
</rl:Request>
```

Die Daten wurden zwei Tage vor der Messe im Rahmen der normalen Systemtests, am Tag direkt nach dem Aufbau des Systems auf der Messe sowie an fünf Messetagen aufgenommen. Nach der Serialisierung besitzen die Daten das folgende Format:

```
REQUEST:START_TASK:fetch_meal_getoutsideonly
REQUEST:START_USER_INTERACTION:AdjustObjectRelToObject
```

Verteilung der Testdaten

Abbildung 7.3 zeigt die Verteilung der Daten in der Datenbasis. Die horizontale Achse ist die Zeitachse an der die Zeitstempel aufgetragen sind und die vertikale Achse stellt die in den Datensätzen aufgetretenen Anfragen dar. Dabei steht jede Zeile für genau einen Ereignistyp. Die Beschriftung wird hier zur Verbesserung der Übersichtlichkeit nicht dargestellt.

Die erste Datenanalyse erfolgt mit dem Data-Mining Werkzeug RapidMiner durch direkten Datenbankzugriff auf die entsprechenden Tabellen der RTK-QSDB.

7.2.1 Vorbetrachtungen

Betrachtet man die Datenbasis in Abbildung 7.3 so sind deutlich die einzelnen Test- und Messetage zu erkennen. Darüber hinaus fällt auf, dass der dritte und der sechste Veranstaltungstag deutlich weniger aufgezeichnete Daten als die übrigen aufweisen[8]. Im Fall des dritten Tages lässt sich das damit begründen, dass nach dem Aufbau auf der Messe nur wenige Tests durchgeführt wurden. Am sechsten Tag konnte das System aufgrund eines Hardwarefehlers nur eingeschränkt betrieben werden.

geringe Ereignisdichte

Bevor sich Daten erfolgreich mit einem Algorithmus wie dem WinEpi-Algorithmus aus Abschnitt 4.2.4 auswerten lassen, sind weitere Vorbetrachtungen notwendig. Vergleicht man dazu die Abbildung 7.3 mit der Abbildung 4.9 auf Seite 70, so zeigt sich, dass die Ereignisdichte in Letzterer deutlich höher ist.

Über einen ganzen Tag kann die Aufnahme von Anfragen durch die MMS bei der eingesetzten zeitlichen Auflösung von $1s$ zu maximal 86400 unterscheidbaren Ereignissen führen. Da die im Abschnitt 4.2.4 beschriebenen Algorithmen jeden Zeitpunkt auswerten,

[8] Tage ohne Messwerte werden nicht weiter betrachtet

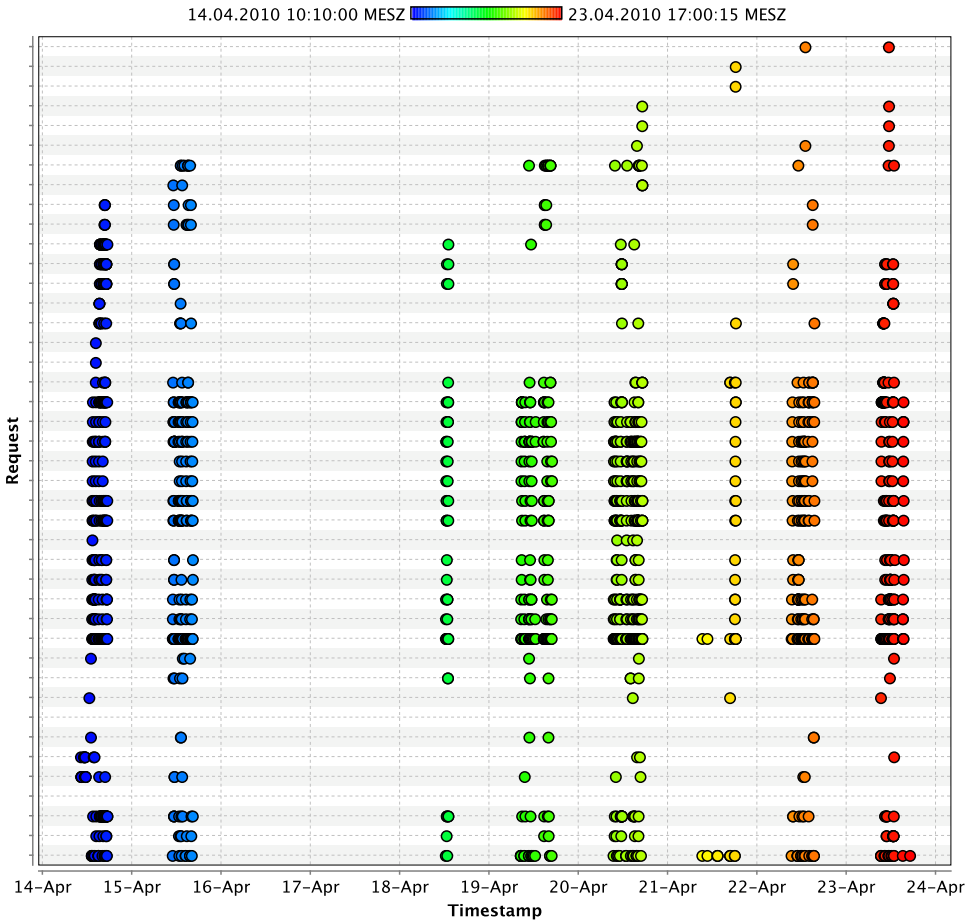


Abbildung 7.3: *Auf der Messeveranstaltung erhobene Datenbasis*

verlängert das die Dauer der Analyse ohne dabei nennenswerte Muster für die ereignisfreien Zeitpunkte aufzuzeigen.

7.2.2 Datenbereinigung

Um eine Lösung für diese Problematik zu entwickeln ist zunächst das angestrebte Analyseziel zu betrachten, welches in der Erkennung von Mustern innerhalb der aufgezeichneten Anfragedaten liegt. Die Muster werden als häufige Episoden erkannt und können so eingesetzt werden, um unterschiedliche Anfragen zueinander in Beziehung zu setzen und Aussagen in der Form „Wenn Ereignis A eingetreten ist, dann tritt Ereignis B danach mit einer Wahrscheinlichkeit von $P(B|A)$ ein“ zu treffen. Da der zeitliche Ablauf und damit die Reihenfolge der Ereignisse in den Episoden dabei eine besondere Rolle spielt sind in diesem Zusammenhang nur die seriellen Episoden von Interesse. Daher ist es möglich vor der Analyse alle Zeitpunkte, zu denen keine Ereignisse existieren, aus der Datenbasis zu

relative Abfolge relevant

entfernen und einen neuen, fortlaufenden Zeitstempel zu erzeugen. Abbildung 7.4 zeigt die dementsprechend angepasste Datenbasis.

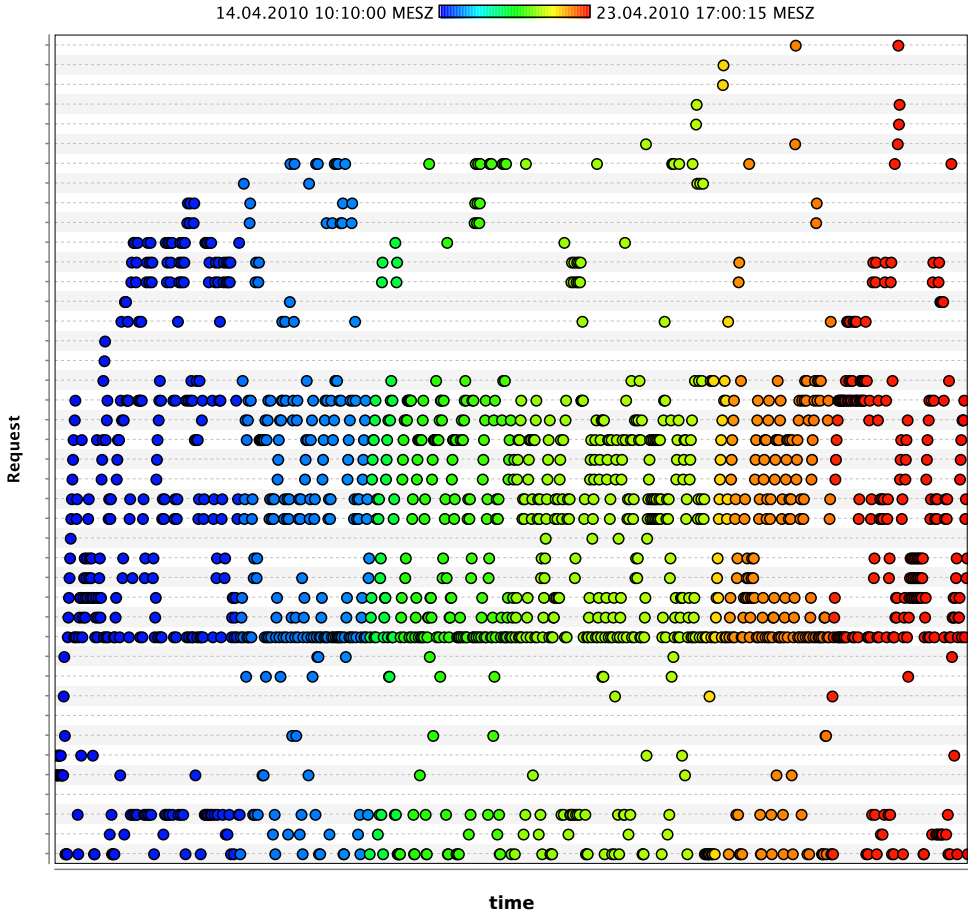


Abbildung 7.4: *Datenbasis nach Erstellung eines neuen, fortlaufenden Zeitstempels*

Reduzierung der Datenbasis

Durch die Generierung eines neuen Zeitstempels reduzierte sich die Ereignissequenz mit einer Länge von circa 800000 (10 Tage) Zeitpunkten auf eine Sequenz mit nur circa 1500 Zeitpunkten. Ein Nebeneffekt dieser Bereinigung der Daten ist das Entfernen des globalen zeitlichen Bezugs von Ereignissen zu Wochentagen oder Tageszeiten. Das ist nicht gewünscht, da es so nicht mehr möglich ist die erstellten Regeln in Abhängigkeit von der Tageszeit oder des Wochentages anzuwenden. Gerade letzteres ist aber das gesetzte Ziel der Adaption der MMS an den Benutzer.

Analyse von Zeitintervallen

Bei genauerer Betrachtung des zeitlichen Bezugs stellt sich die Frage, ob der genaue Zeitpunkt eines Ereignisses für die erfolgreiche Analyse überhaupt von Interesse ist, denn die extrahierten häufigen Episoden lassen sich nicht einem einzigen Zeitpunkt, sondern nur einen Zeitintervall zuordnen. Dieses Zeitintervall beginnt zum Zeitpunkt an dem das erste Ereignis der Episode eintritt und endet zum Zeitpunkt an dem das letzte Ereignis der Episode eintritt. Aufgrund dieser Betrachtung liegt es nahe, die gesamte

Datenbasis zuvor in mehrere Datenbasen aufzuteilen, die jeweils die Ereignisse für ein ganz bestimmtes Zeitintervall beinhalten. Durch den Einsatz mehrerer Zeitintervalle, die 24 Stunden lückenlos abdecken, lassen sich alle Episoden eindeutig auf eine bestimmte Tageszeit abbilden. Fünf mögliche Zeitintervalle, die dieser Grundidee gerecht werden, zeigt Tabelle 7.1. Die Analyse der Datenbasis kann bei dieser Vorgehensweise erst nach der Aufteilung der Daten erfolgen. Die Datenbereinigung kann davor oder danach erfolgen.

Intervall	t_s	t_e
Morgens	$6^{00}Uhr$	$10^{00}Uhr$
Mittags	$10^{00}Uhr$	$14^{00}Uhr$
Nachmittags	$14^{00}Uhr$	$18^{00}Uhr$
Abends	$18^{00}Uhr$	$22^{00}Uhr$
Nachts	$22^{00}Uhr$	$6^{00}Uhr$

Tabelle 7.1: *Mögliche Zeitintervalle zur Repräsentation der Tageszeiten*

7.3 Wissensextraktion durch Mustererkennung

Auf Basis des WinEpi Algorithmus (siehe Abschnitt 4.2.4) wird in diesem Abschnitt die Auswertung der aufgezeichneten Datenbasis erfolgen. Die Auswertung beschränkt sich ungeachtet des schlechteren Laufzeitverhalten auf den Einsatz dieses Algorithmus, da er weniger Speicherressourcen benötigt. Durch die Realisierung einer Erweiterung auf Basis dieses Algorithmus kann so sichergestellt sein, dass die Erweiterung auch bei langfristigem Einsatz nicht zu Ressourcenengpässen führt. Die höhere Zeitkomplexität soll in diesem Zusammenhang durch Implementierung des Algorithmus in einem eigenen Ausführungskontext, parallel zum Ausführungskontext der MMS, kompensiert werden.

parallele Analyse

In den folgenden Abschnitten wird zunächst die Abbildung des Algorithmus auf Klassen, mittels derer die Implementation erfolgt, beschrieben. Im Anschluss daran folgt der Entwurf einer Testanwendung, über die der Algorithmus einfach und mit unterschiedlichen Datenbasen getestet werden kann. Die Untersuchung des Analyseverhaltens bei Anwendung auf unterschiedliche Testdaten sowie auf die aufgezeichnete Datenbasis schließt diesen Abschnitt ab.

7.3.1 Integration des WinEpi-Algorithmus

Die Implementierung des WinEpi Algorithmus wird durch Kapselung der in Algorithmus 1 auf Seite 75 vorgestellten Funktionalitäten realisiert. Diese teilen sich in zwei Hauptfunktionalitäten auf: die automatenbasierte Detektion von häufigen Episoden sowie die Generierung von Kandidaten. Abbildung 7.5 zeigt ein Flussdiagramm zur Veranschaulichung des Algorithmenablaufs.

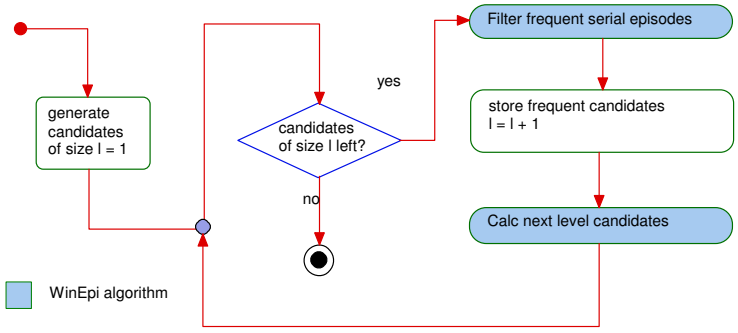


Abbildung 7.5: *Flussdiagramm zu den Hauptfunktionalitäten des WinEpi Algorithmus (siehe Algorithmus 1)*

Kandidatengenerierung und Filterung

Zunächst werden die Kandidaten $\mathbb{C}_1$ der Länge $l = 1$ erzeugt. Mit diesen Episoden erfolgt der Start der zyklischen Datenanalyse durch den WinEpi Algorithmus. In einer Schleife werden zunächst die häufigen Episoden heraus gefiltert um im Anschluss daraus die Kandidaten der Länge $l+1$ zu erzeugen. Dieser Vorgang wird zyklisch wiederholt, so lange bis keine Kandidaten mehr existieren. Zur einfachen Integration des Algorithmus in die zu erstellende Erweiterung erfolgt die Kapselung der Hauptfunktionalitäten in der in Abbildung 7.6 dargestellten Klasse.

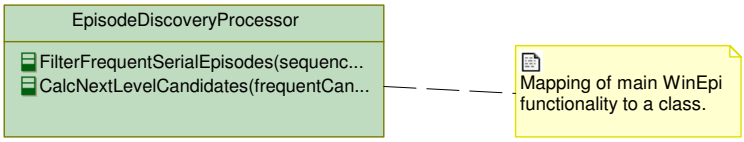


Abbildung 7.6: *Abbildung der WinEpi Hauptfunktionalitäten auf eine Klasse*

7.3.2 Abbildung der Hilfsdatentypen auf Klassen

Datentyp für Ereignisse

Vor der Implementierung sind die vom Algorithmus verwendeten Daten als Datentypen zu modellieren. Zunächst soll dazu der Vorgang des Ladens der Datenbasis betrachtet werden. Die Daten liegen in Form von Ereignissequenzen vor, die aus einzelnen Ereignissen bestehen welche über den Ereignistyp A und den Zeitstempel t definiert sind. Der Ereignistyp wird in Form der Typdefinition `EventType_T` abgebildet, so ist es möglich, die Ereignistypen flexibel zu gestalten. Als Standartwert kommt der Datentyp `std::string` aus der C++ Standardbibliothek zum Einsatz. Dadurch lassen sich sowohl Zeichenketten als auch die in den Beispielen aus Abschnitt 4.2.3 verwendeten Buchstaben als Typ der Ereignisse einsetzen.

Die Abbildung der Ereignisse wird durch den von der Standardbibliothek zur Verfügung gestellten Vorlagentyp `std::pair<>` realisiert. Die Typisierung erfolgt dabei in Form der

Typdefinition `DataPair_T` mit einer Ganzzahl als Zeitstempel und der zuvor eingeführten Typdefinition `EventType_T` als Ereignistyp.

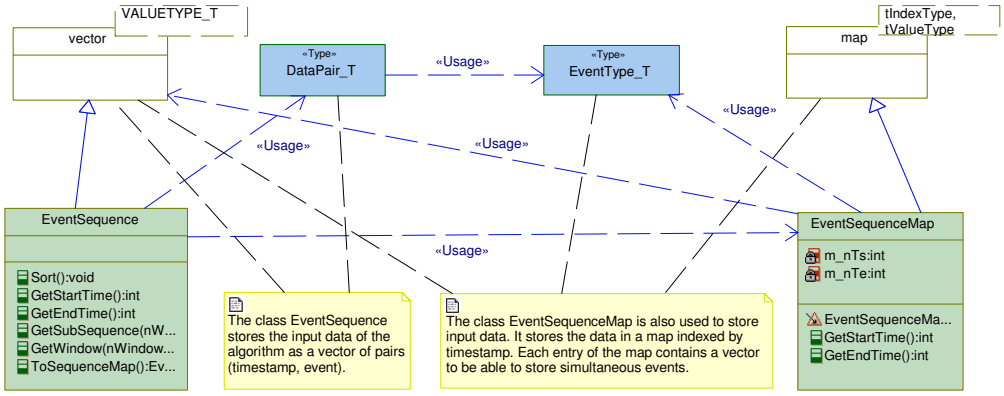


Abbildung 7.7: *Abbildung von Ereignissequenzen und Fenstern (zum Beispiel $\mathbb{S}$ und $\mathbb{W}$) auf Klassen*

Die Zusammenfassung vieler Ereignisse in einer Ereignissequenz wird durch zwei Klassen realisiert. Die erste Klasse (`EventSequence`) verwendet dazu einen Vektor von Elementen des zuvor eingeführten Datentyps `DataPair_T`. Der Einsatz des Vektors besitzt den Vorteil, dass die eingelesenen Daten direkt darauf abgebildet werden können. Jedes Element des Vektors entspricht genau einem Datensatz. In Bezug auf die Indizierung weist der Einsatz eines Vektors jedoch einen Nachteil auf: es ist keine direkte Indizierung eines Ereignisses zu einem bestimmten Zeitpunkt t möglich. Aus diesem Grund implementiert die Klasse `EventSequence` eine Methode, mittels derer die gesamte Sequenz in eine Abbildungstabelle[9] konvertiert werden kann. Das Ergebnis dieser Konvertierung ist eine Instanz der Klasse `EventSequenceMap`. Abbildung 7.7 zeigt das Modell dieser Datenabbildung.

Datentyp für Ereignissequenzen

Beide Klassen bieten als Grundfunktionalität die Rückgabe der Start- und Endzeit (T_s und T_e) der gespeicherten Ereignissequenz. Darüber hinaus implementiert die Klasse `EventSequence` Funktionalitäten zur Rückgabe eines Fensters $\mathbb{W}$ auf die gespeicherte Ereignissequenz sowie die Möglichkeit die Daten anhand ihres Zeitstempels zu sortieren. Die Rückgabe des Fensters erfolgt dabei entweder als Wert vom Typ `EventSequence` (bei Verwendung der Methode `GetSubSequence()`) oder als Wert vom Typ `EventSequenceMap` (bei Verwendung der Methode `GetWindow()`).

Neben der Abbildung der Eingabedaten ist es notwendig, die innerhalb vom WinEpi Algorithmus verwendeten Datenstrukturen zu modellieren. Als Erstes wären hier die Episoden und Mengen von Episoden zu nennen. Abbildung 7.8 zeigt das Modell dieser Abbildung.

Datentyp für Episoden und Mengen

Die Klasse `Episode` steht in einer Kompositionsbeziehung zu einem Vektor mit Werten vom Typ `EventType_T`. Neben Methoden zum Hinzufügen von Ereignistypen zu der

[9] Es kommt die Instanz der Standardklasse `std::map<indexType, valueType>` zum Einsatz.

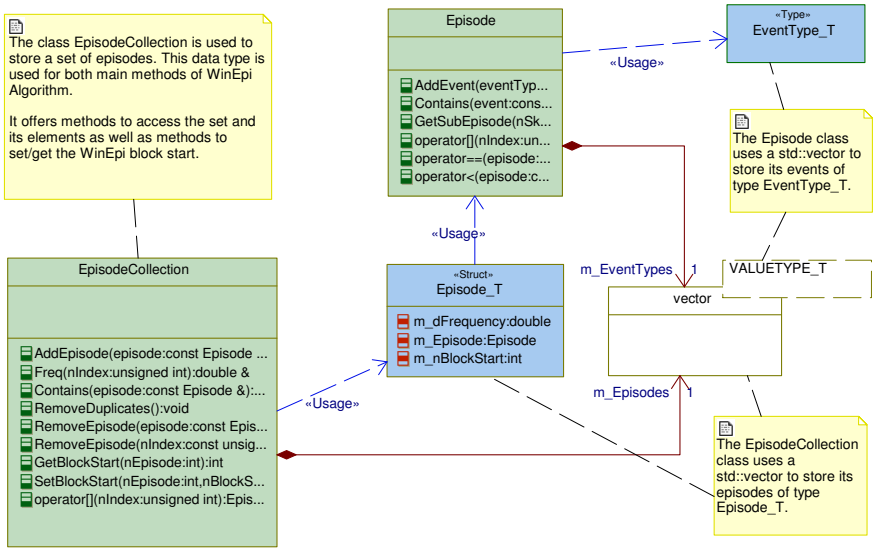


Abbildung 7.8: *Abbildung von Episoden (zum Beispiel α) und Mengen von Episoden (zum Beispiel $\mathbb{C}$ und $\mathbb{F}$) auf Klassen*

Episode besitzt sie eine Methode zur Rückgabe einer Unterepisode und eine Methode um einen bestimmten Ereignistyp innerhalb der Episode zu finden. Darüber hinaus existieren nur noch Operatoren, die den einfachen Zugriff und Vergleiche von Episoden ermöglichen. Über die Klasse `EpisodeCollection` können mehrere Episoden zu einer Menge zusammengefasst werden. Dazu steht diese Klasse wiederum in einer Kompositionsbeziehung zu einem Vektor, der die Episoden enthält. Sowohl die Menge an Kandidaten $\mathbb{C}$ als auch die Menge von häufigen Episoden $\mathbb{F}$ wird als Instanz dieser Klasse realisiert. Damit steht die Abbildung von allen Datentypen, die in den Schnittstellen der Hauptmethoden des WinEpi Algorithmus (`CalcNectLevelCandidates()` und `FilterFrequentSerialEpisodes()`) benötigt werden, zur Verfügung.

Datentyp für Automaten

Die Erkennung von häufigen Episoden und damit die Erkennung von Mustern in der Ereignissequenz basiert auf dem Einsatz von Automaten (siehe Abschnitt 4.2.4). Diese werden auf die Struktur `Automaton` abgebildet. Die Struktur enthält als erstes Attribut den Indexwert, über den der aktuelle Zustand des Automaten beschrieben wird. Dabei handelt es sich um eine Ganzzahl, die angibt, auf welches Ereignis einer Episode der Automat wartet. Das zweite Attribut ist ebenfalls eine Ganzzahl. Über diese erfolgt die Zuordnung zu einer Klasse, die die Zustandswechsel eines jeden Automaten protokolliert. Diese Klasse wird später ausführlich beschrieben.

Zuordnung von Automat zu Episode

Neben den Attributen besitzt die Klasse `Automaton` eine Kompositionsbeziehung zur Klasse `Episode`. Dadurch erfolgt die Zuordnung eines Automaten zu der Episode deren Vorkommen der Automat analysiert. Innerhalb eines Durchlaufs des WinEpi Algorithmus werden alle Automaten in einer Liste verwaltet. Diese Liste wird durch die Klasse `AutomatonCollection` auf Basis der Klasse `std::vector` realsiert. Abbildung 7.9 zeigt den gesamten Entwurf der Abbildung.

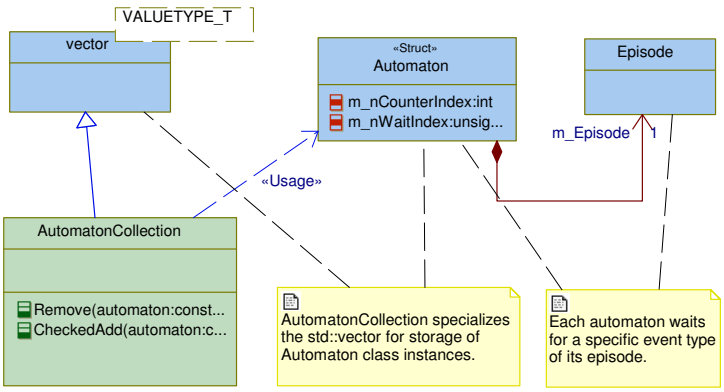


Abbildung 7.9: *Abbildung von Automaten und Automatenlisten*

Nach Algorithmus 1 werden die Zustandswechsel der Automaten über eine Menge von Transitionen festgehalten und ausgewertet. Die in [MTV97] vorgeschlagene Verwendung von Transitionen soll hier keine direkte Anwendung finden, sondern durch Instanzen der Klasse `Counter`, welche in Abbildung 7.10 dargestellt ist, ersetzt werden.

Counter speichert Ein- und Austritt in Fenster

Die Implementation der Klasse `Counter` zählt nicht die Transitionen der Automaten, sondern speichert das Ein- und Austreten einer Episode in ein Fenster. Dazu wird zu Beginn der Analyse eine Abbildungstabelle auf Basis der `std::map` erzeugt, in der für jeden Kandidaten eine Instanz der Klasse `Counter` abgelegt und über die Episode indiziert wird. In jedem Bearbeitungsschritt kann so festgehalten werden, in welchen Fenstern eine Episode enthalten ist. Auf Basis dieser Daten ist es möglich die Häufigkeit einer Episode zu berechnen.

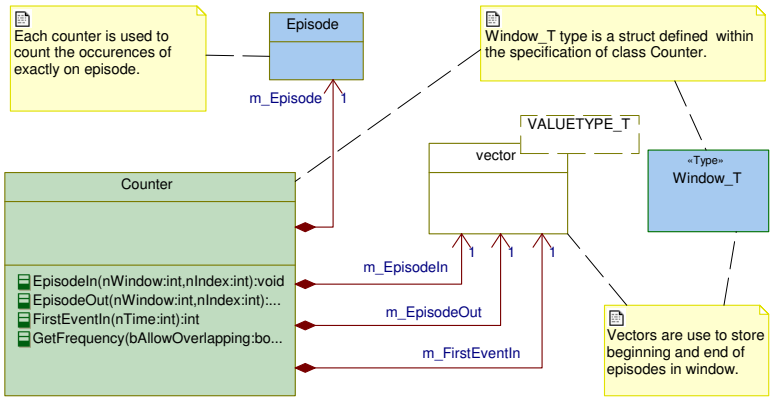


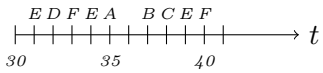
Abbildung 7.10: *Abbildung der Klasse zur Auswertung der Zustandswechsel*

Die Klasse `Counter` besitzt dazu drei Methoden, mittels derer die Zustandswechsel festgehalten werden. Über die Methode `FirstEventIn()` wird der Zeitpunkt festgehalten, zu dem das erste Ereignis der korrespondierenden Episode in das aktuelle Zeitfenster kommt. Sobald die Episode komplett im Zeitfenster ist, wird der Zeitpunkt über die

Verwaltung der Zustandswechsel

Methode `EpisodeIn()` markiert. Wenn die Episode das Fenster wieder verlässt, wird das durch die Methode `EpisodeOut()` festgehalten. Darüber hinaus überprüft die Methode, ob sich die Episode zuvor vollständig im Fenster befand, und markiert die aufgenommenen Werte in diesem Fall als gültig. Das Ergebnis dieser Vorgehensweise ist eine Liste mit Intervallen, in denen sich die Episode vollständig im Fenster befindet. Das folgende Beispiel verdeutlicht diese Zusammenhänge.

Beispiel 7.2. *Gegeben sei die folgende Ereignissequenz* $s = (\mathbb{S}, 31, 41)$ *mit neun Ereignissen aus der Menge möglicher Ereignistypen* $\mathbb{E} = \{'A', 'B', 'C', 'D', 'E', 'F'\}$*:*



Bei einer Fenstergröße von $N = 4$ *ergeben sich* $|\omega(\mathbb{S}, 4)| = 41 - 31 + 4 - 1 = 13$ *Fenster, wie folgende Tabelle veranschaulicht.*

	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43
x_i				E	D	F	E	A		B	C	E	F			
1	(			)												
2		(			)											
3			(			)										
4				(			)									
5					(			)								
6						(			)							
7							(			)						
8								(			)					
9									(			)				
10										(			)			
11											(			)		
12												(			)	
13													(			)

Die farbig hervorgehobenen Fenster sind jene, die Episode $\alpha = 'F'$ *beinhalten. Die zu dieser Episode korrespondierende Instanz der Klasse* `Counter` *speichert für* α *die folgenden gültigen Übergänge.*

```
1 in: 3  valid  out: 7  valid
2 in: 10  valid  out: 14  valid
```

Das heißt, Episode α *ist im dritten Fenster das erste mal vollständig enthalten. In Fenster sieben ist die Episode dann nicht mehr enthalten. Das gleiche gilt für das zweite Intervall, welches sich zwischen dem zehnten und vierzehnten Fenster befindet.*

Häufigkeitsberechnung

Aus den so gewonnenen Daten lässt sich mit Hilfe der Fensternummern x_i direkt die absolute Häufigkeit für jede der Episoden, also für alle Kandidaten eines Durchlaufs berechnen. Dazu addiert man die Differenzen der Fensternummer des jeweils letzten (x_i^{end}) und des jeweils ersten (x_i^{start}) Fensters für alle ermittelten Intervalle. Die An-

zahl der ermittelten Intervalle wird dabei mit X bezeichnet. Gleichung 7.1 zeigt diesen Zusammenhang.

$$|\{\mathbb{W} \in \omega(\mathbb{S}, N) \quad | \quad \alpha \text{ occurs in } \mathbb{W}\}| = \sum_{i=1}^{X} \left(x_i^{end} - x_i^{start} \right) \tag{7.1}$$
$$= 8$$

Setzt man Gleichung 7.1 in Gleichung 4.3 von Seite 76 ein, so erhält man für die relative Häufigkeit folgenden Ausdruck (mit $width(\mathbb{W}) = N$).

$$f(\alpha, \mathbb{S}, N) = \frac{\sum_{i=1}^{X} x_i^{end} - x_i^{start}}{|\omega(\mathbb{S}, N)|} \tag{7.2}$$

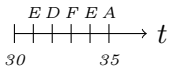
Bezogen auf oben angegebenes Beispiel ergibt sich für α eine relative Häufigkeit von

$$f(\alpha, \mathbb{S}, 4) = \frac{8}{13}$$
$$= 0,6154$$
$$= 61,5\%$$

Bei der zuvor beschriebenen Vorgehensweise ist noch ein Sonderfall zu betrachten. Sollte eine Episode mehrfach innerhalb eines Fensters auftreten, so kommt es zur Überlappung der von der Klasse `Counter` ermittelten Intervalle, wie folgendes Beispiel demonstriert.

Sonderfall: Überlappung

Beispiel 7.3. *Gegeben sei die folgende Ereignissequenz* $s = (\mathbb{S}, 31, 36)$ *mit fünf Ereignissen aus der Menge möglicher Ereignistypen* $\mathbb{E} = \{'A', 'D', 'E', 'F'\}$*:*



Bei einer Fenstergröße von $N = 4$ *ergeben sich* $|\omega(\mathbb{S}, 4)| = 36 - 31 + 4 - 1 = 8$ *Fenster, wie folgende Tabelle veranschaulicht.*

	28	29	30	31	32	33	34	35	36	37	38
x_i				E	D	F	E	A			
1	(				)						
2		(				)					
3			(				)				
4				(				)			
5					(				)		
6						(				)	
7							(				)
8								(			)

Die farbig hervorgehobenen Fenster sind jene, die Episode $\alpha = 'E'$ *beinhalten. Die zu dieser Episode korrespondierende Instanz der Klasse* `Counter` *speichert für* α *die folgenden gültigen Übergänge.*

```
in: 1  valid  out: 5  valid
in: 4  valid  out: 8  valid
```

Die Besonderheit ist in diesem Zusammenhang, dass das Fenster mit der Nummer vier die gesuchte Episode zweimal enthält. Insgesamt ist die Episode in sieben Fenstern enthalten, die Berechnung nach Gleichung 7.1 ergibt jedoch acht Fenster.

Zusammenfassung von Intervallen

Aus diesem Grund erfolgt in der Methode `GetFrequency()` bei der Berechnung der Häufigkeit die Überprüfung der Intervalle auf Überlappungen. Liegt eine Überlappung vor, so werden die Intervalle zusammengefasst.

```
in: 1  valid  out: 8  valid
```

Im oben angegebenen Fall werden die zwei Intervalle so zusammengefasst, dass sie die Fenster eins bis sieben enthalten. Die Berechnung nach Gleichung 7.1 ergibt dann sieben Fenster und ist somit wieder korrekt.

$$\begin{aligned}\sum_{i=1}^{X}\left(x_i^{end} - x_i^{start}\right) &= 8 - 1\\ &= 7\end{aligned}$$

7.3.3 Entwurf einer Testanwendung

Erzeugung von Testdaten und Analyse

Zur Realisierung von Testläufen des WinEpi Algorithmus wird eine Kommandozeilenanwendung zum Einsatz kommen. Diese kann einfach über Kommandozeilenargumente parametriert werden und soll neben der Datenanalyse unterschiedliche Funktionalitäten wie das Lesen von externen Datenquellen oder die Generierung von Zufallsdaten bieten. Dabei soll es möglich sein, den zufällig generierten Daten zu Testzwecken eine Episode einzuprägen, die mit einer festgelegten Häufigkeit auftritt. Alle weiteren Ereignisses innerhalb der Testdaten sind gleichverteilt.

Die Integration des WinEpi-Algorithmus in die Testanwendung erfolgt über die in den vorherigen Abschnitten entwickelten Klassen. Die Funktionsweise ist in Abbildung 7.11 als Flussdiagramm dargestellt.

Steuerung über Kommandozeile

Nach dem Start werden zunächst die Schalter ausgewertet, über die die Datenquelle spezifiziert wird. In Abhängigkeit von diesen Schaltern erfolgt dann der Lade- oder Generierungsvorgang mit anschließender Sortierung der Daten. Im Anschluss daran startet nach der Generierung der Kandidaten die Datenanalyse durch Aufruf der Methode `FilterFrequentSerialEpisodes()`. Diese erhält als Argument die Kandidaten $\mathbb{C}_l$ und ermittelt daraus die Menge der häufigen Kandidaten $\mathbb{F}_l$. Letztere werden gespeichert und stellen später einen Teil des Analyseergebnisses dar. Zusätzlich wird der Level erhöht, um dann aus $\mathbb{F}_l$ die Kandidaten für den nächsten Durchlauf $\mathbb{C}_{l+1}$ zu erzeugen. Der Algorithmus endet, wenn der Aufruf der Methode `CalcNextLevelCandidates()` eine leere

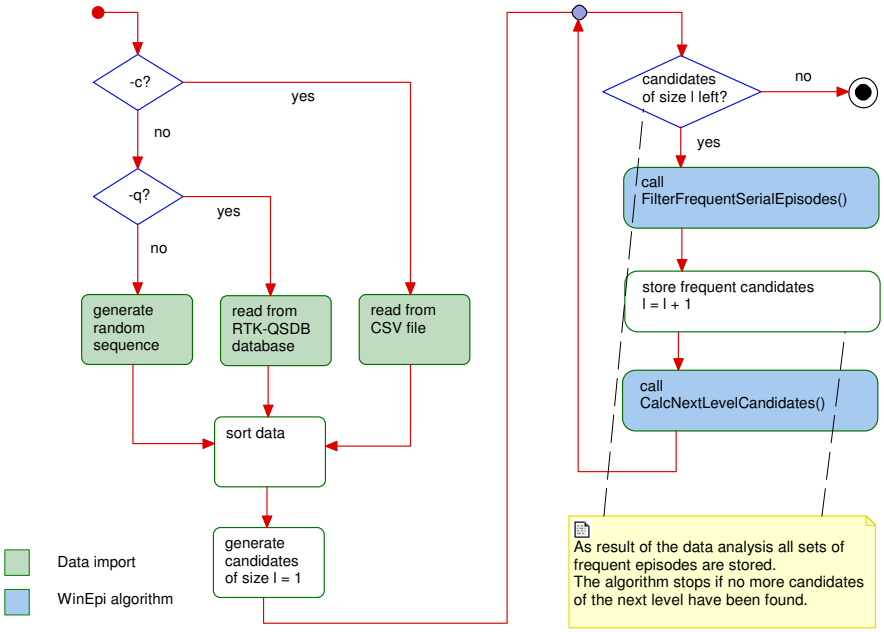


Abbildung 7.11: *Flussdiagramm zum WinEpi Algorithmus (siehe Algorithmus 1)*

Menge zurückgibt. Abbildung 7.12 zeigt den Entwurf der Testanwendung als Klassendiagramm.

Alle Funktionalitäten der Anwendung sind über Kommandozeilenargumente (Optionen und Schalter) aktivierbar. Es stehen die folgenden Optionen zur Verfügung:

CSV lesen

-c Über diese Option wird die Anwendung im CSV[10]-Modus gestartet. In diesem Modus wird die zu analysierende Datenbasis aus einer externen CSV-Datei gelesen. Dabei muss zusätzlich zur Option der Name der zu lesenden CSV-Datei angegeben werden.

SQL lesen

-q Diese Option startet die Anwendung im SQL-Modus. In diesem Modus wird die Datenbasis aus der RTK-QSDB-Datenbank geladen. Neben der Option ist der Dateiname anzugeben, in der die zu verwendende SQL-Abfrage enthalten ist.

Daten erzeugen

-r Steht keine auszuwertende Datenbasis zur Verfügung, so kann die Anwendung über diese Option im Zufalls-Modus betrieben werden. Dabei werden n Ereignisse erzeugt, die gleichverteilt sind. Zusätzlich zu der Option ist die Anzahl der zu erzeugenden Ereignisse anzugeben. Sollen die erzeugten Daten mehrmals zur Analyse verwendet werden, so können diese über die Option `-o` in eine CSV-Datei geschrieben werden.

Ausgabe

-o Über diese Option wird der Name der Ausgabedatei angegeben, in der die zufällig erzeugten Ereignisse geschrieben werden sollen. Diese Option ist nur in Verbindung mit der Option `-r` sinnvoll.

[10] **C**omma-**S**eparated **V**alues

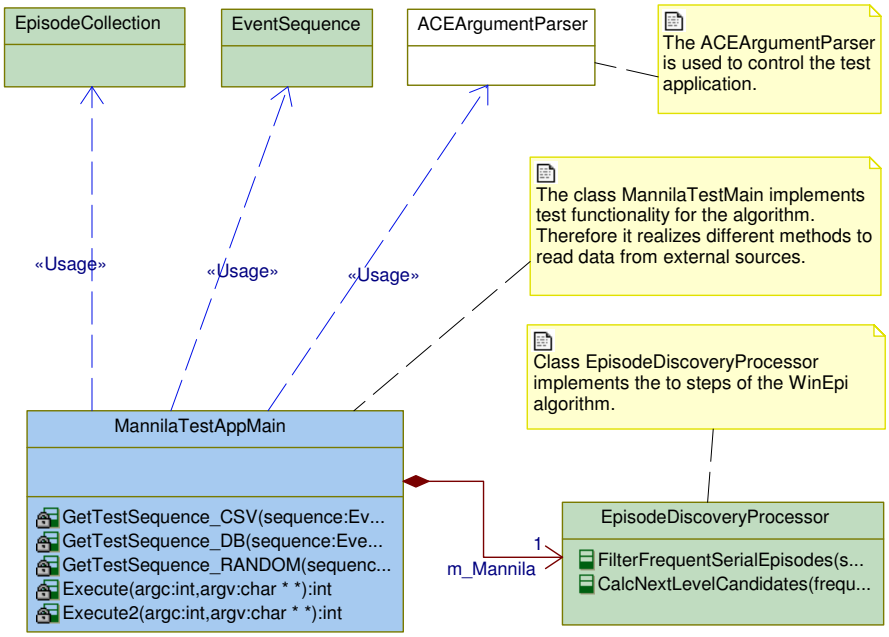


Abbildung 7.12: *Entwurf der Testanwendung für den WinEpi Algorithmus*

Episode einprägen

-s Über diese Option kann die Länge der in die zufällig erzeugten Daten eingeprägten Episode angegeben werden. Wird diese Option nicht verwendet, so wird keine Episode eingeprägt. Die Ereignisse der Episode werden dabei zufällig aus den in der Ereignissequenz vorhandenen Ereignissen ausgewählt. Diese Option ist nur in Verbindung mit der Option `-r` sinnvoll.

Häufigkeit

-p Über diese Option wird die Anzahl der Vorkommen der eingeprägten Episode bestimmt. Diese Option ist nur in Verbindung mit der Option `-r` sinnvoll.

WinEpi Schwellwert

-m Diese Option setzt den Schwellwert min_f durch den entschieden wird, ob eine Episode häufig ist oder nicht. Wird die Option nicht verwendet, so kommt der Standardwert $min_f = 0,4$ zur Anwendung.

Fensterbreite

-w Diese Option legt die Breite des gleitenden Fensters $N = width(\mathbb{W})$ fest. Wird die Option nicht verwendet, so kommt der Standardwert $N = width(\mathbb{W}) = 4$ zur Anwendung.

Neben den Optionen dienen die folgenden Schalter der weiteren Einflussnahme auf die Ausführung des WinEpi Algorithmus.

Zeitstempel generieren

-g Für alle Daten der bearbeiteten Datenbasis werden neue, fortlaufende Zeitstempel generiert. Dadurch ist es möglich, Zeitpunkte, zu denen kein Ereignis vorliegt, aus der Sequenz zu entfernen. Die Reihenfolge der Ereignisse bleibt erhalten, jedoch verändern sich die zeitlichen Bezüge zwischen unterschiedlichen Ereignissen (siehe Abschnitt 7.2.2).

-D Standardmäßig erlaubt die WinEpi-Implementierung keine doppelt vorkommenden Ereignisse innerhalb einer Episode. Es werden also nur injektive Episoden detektiert. Über diesen Schalter ist das Verhalten umkehrbar. doppelte Ereignisse zulassen

-d Durch Angabe dieses Schalters wird die Anwendung im Debug-Modus gestartet. Die Anwendung erzeugt in diesem Modus deutlich mehr Textausgaben auf der Konsole. Der Schalter dient der Fehlersuche und wird im normalen Anwendungsfall nicht benötigt. Debug-Modus

7.3.4 Anwendung auf Testdaten

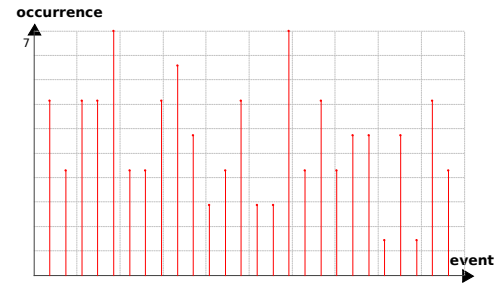
Bevor der implementierte WinEpi-Algorithmus auf die aufgezeichneten realen Daten angewandt wird, ist zunächst über einige Tests die korrekte Funktionalität sicherzustellen. Dazu sollen generierte Daten analysiert werden. Als erstes erfolgt in diesem Zusammenhang die Analyse von Daten, die zuvor von der Testanwendung über die folgenden Anweisungen erzeugt werden. Testdaten generieren

```
$ mannilaAlgorithm -oData100.csv -r100
$ mannilaAlgorithm -oData1000.csv -r1000
$ mannilaAlgorithm -oData10000.csv -r10000
$ mannilaAlgorithm -oData100000.csv -r100000
```

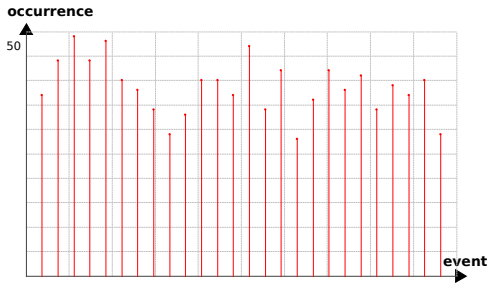
Zur Generierung der Testdatensätze wird die entsprechende Funktionalität der Testanwendung eingesetzt. Das heißt, die Analyse erfolgt zunächst auf Basis von gleichverteilten Daten. Der Zeitstempel dieser Daten wird mit Null beginnend fortlaufend erzeugt. Zu jedem Zeitstempel wird genau ein zufälliges Ereignis generiert. Als Menge von möglichen Ereignissen wird das Alphabet mit seinen 26 Buchstaben verwendet. Um zu überprüfen, in wie fern die erzeugten Daten tatsächlich einer Gleichverteilung genügen, werden diese zunächst mit dem KDD Werkzeug RapidMiner analysiert. Abbildung 7.13 zeigt qualitativ die Häufigkeitsverteilung sowie den Mittelwert und die Standardabweichung der erzeugten Datensätze. Voranalyse mit Rapid-Miner

Zusätzlich zum Mittelwert und der Standardabweichung stellt Tabelle 7.2 die Variationskoeffizienten c_v (normierte Standardnormalverteilung) der vier Verteilungen einander gegenüber. Vergleicht man diese für die Zufallsdatensätze mit 100, 1000, 10000 und 100000 Ereignissen so wird deutlich, dass eine annähernde Gleichverteilung erst bei einer hohen Anzahl von generierten Ereignissen erreicht wird. Aus diesem Grund werden für die folgenden Testläufe nur Datensätze mit mindestens 10000 Datensätzen verwendet.

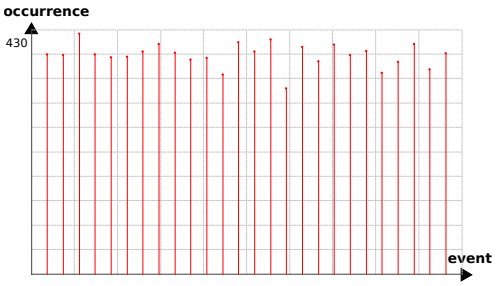
Die erste Anwendung des implementierten WinEpi-Algorithmus wird mit einer Fensterbreite von $N = 4$ durchgeführt. Es zeigt sich, dass die Analyse bereits nach dem ersten Durchlauf aufgrund fehlender Kandidaten abbricht. Dadurch beinhaltet die Ergebnismenge nur Episoden der Länge $l = 1$, deren Häufigkeitsverteilung direkt mit dem Analyseergebnis von Rapid-Miner verglichen werden kann. WinEpi-Anwendung auf Testdaten



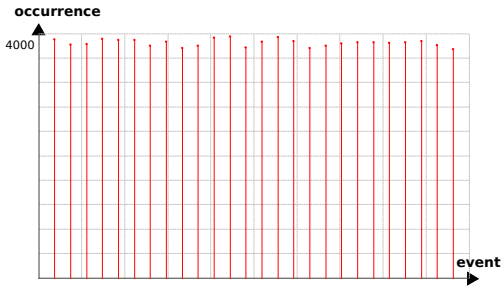
(a) *Ereignissequenz mit 100 Ereignissen ($\bar{x} = 3,84$, $\sigma_X = 1,62$)*



(b) *Ereignissequenz mit 1000 Ereignissen ($\bar{x} = 38,46$, $\sigma_X = 5.54$)*



(c) *Ereignissequenz mit 10000 Ereignissen ($\bar{x} = 384,61$, $\sigma_X = 20,69$)*



(d) *Ereignissequenz mit 100000 Ereignissen ($\bar{x} = 3846,15$, $\sigma_X = 57,01$)*

Abbildung 7.13: *Qualitative Häufigkeitsverteilung der generierten Zufallsdatensätze*

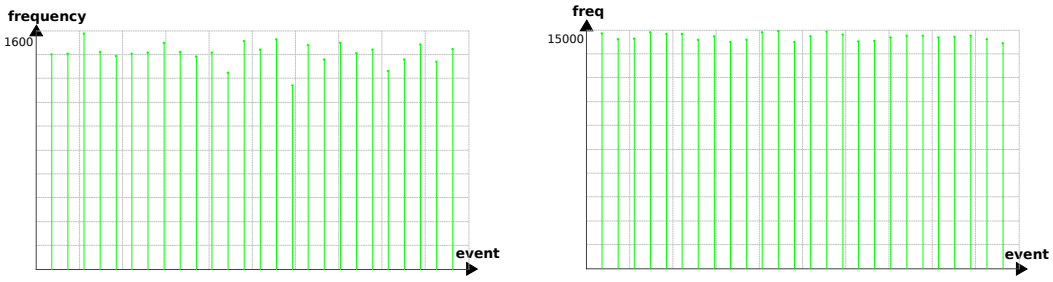
n	$\bar{x}$	σ_X	c_v
100	3,84	1,62	41,93%
1000	38,46	5,54	14,40%
10000	384,61	20,69	5,37%
100000	3846,15	57,01	1,48%

Tabelle 7.2: *Variationskoeffizienten c_v der generierten Datensätze*

Wie deutlich zu erkennen ist, sind beide Häufigkeitsverteilungen qualitativ identisch. Die deutlich höheren absoluten Werte für die Häufigkeit der Ereignisse bei der Analyse über den WinEpi-Algorithmus liegen darin begründet, dass bei dessen Anwendung nicht die absolute Anzahl, sondern die Vorkommnisse der Episoden in Fenstern gezählt wird. Abbildung 7.14 zeigt die Häufigkeitsverteilung nach Analyse der Datensätze durch den WinEpi Algorithmus.

Die bei der Analyse verwendeten Kommandozeilenargumente der Testanwendung lauteten wie folgt:

```
1 $ mannilaAlgorithm -cData10000.csv -w4 -m0.4
2 $ mannilaAlgorithm -cData100000.csv -w4 -m0.4
```



(a) *Ereignissequenz mit 10000 Ereignissen* **(b)** *Ereignissequenz mit 100000 Ereignissen*

Abbildung 7.14: *Auswertung der Vorkommnisse von Episoden durch den WinEpi Algorithmus mit* $N = 4$ *und* $min_f = 40\%$

Tabelle 7.3 zeigt die Analyseergebnisse der Testdatensätze für zwei unterschiedliche Fensterbreiten. Aus den Daten ist zu erkennen, dass die Laufzeit des Algorithmus mit steigender Anzahl an Ereignissen überproportional zunimmt. Die Größe des Fensters wirkt sich dagegen kaum auf die Laufzeit aus. Dieser Zusammenhang wird klar, wenn man Definition 4.10 betrachtet. Danach existieren genau $T_e^s - T_s^s + N - 1$ Fenster der Breite N auf Ereignissequenz s. Darin geht die Breite der Fenster nur als Summand ein, so dass bei Änderung der Fensterbreite die Anzahl der Fenster nur um die Differenz der Fensterbreite variiert. Vergleicht man die Variationskoeffizienten mit denen vor der WinEpi Analyse, so zeigen sich auch hier keine nennenswerten Änderungen. Die Eigenschaften der Verteilung wurden also durch die Anwendung des Algorithmus nicht nennenswert beeinflusst.

Laufzeit und Fensterbreite

n	N	$\bar{x}$	σ_X	$\bar{f}$	c_v	Laufzeit
10000	4	1446, 80	74, 00	14, 47%	5, 11%	$271ms$
100000	4	14536, 92	215, 23	14, 54%	1, 48%	$3319ms$
10000	8	2681, 64	133, 68	26, 80%	4, 99%	$279ms$
100000	8	26987, 46	387, 00	26, 99%	1, 43%	$3965ms$

Tabelle 7.3: *Analyse der Testdaten mit unterschiedlicher Fensterbreite bei festem Schwellwert* $min_f = 40\%$

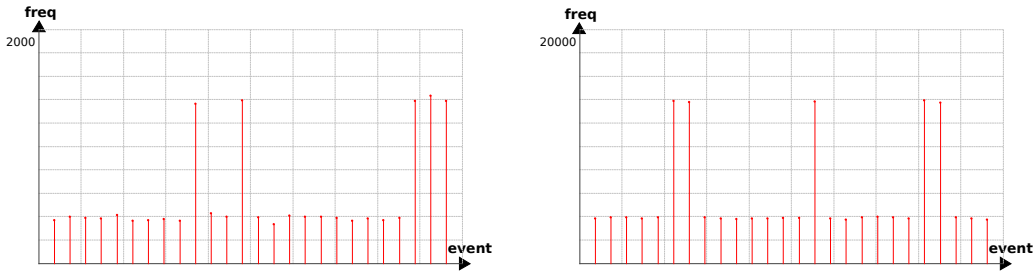
Es zeigte sich bereits, dass die Ausführung des WinEpi-Algorithmus nach dem ersten Durchlauf mit einer leeren Menge als Ergebnis unterbrochen wird. Es können also keine häufigen Episoden gefunden werden, deren Häufigkeit den Schwellwert von $min_f = 40\%$ erreicht oder überschreitet. Wie aus Tabelle 7.3 hervor geht, wächst die mittlere Anzahl von Vorkommnissen der Ereignisse (und damit auch die mittlere relative Häufigkeit) nahezu proportional mit der Fensterbreite. Dieser Zusammenhang wird später nochmal aufgegriffen. Es ist zu erwarten, dass mit weiterer Vergrößerung der Fensterbreite der Schwellwert min_f überschritten wird. In diesem Fall werden dann alle Episoden als häufig eingestuft und die Abbruchbedingung erst nach mehreren Durchläufen erreicht, obwohl dabei aufgrund der Gleichverteilung weiterhin keine sinnvollen Informationen

Einfluss der Fensterbreite

abgeleitet werden können. Dieser Zusammenhang macht deutlich, wie wichtig die Wahl der Fensterbreite in Bezug auf die Analyse von Daten ist.

Einprägung einer Episode

Für die Analyse weiterer Testdaten erfolgt die Einprägung einer häufigen Episode mit dem Ziel der Untersuchung der Analysefähigkeit des Algorithmus. Die Einprägung wird durch die Testanwendung vorgenommen, wobei zufällig die Ereignisse einer Episode erzeugt und dem Datensatz hinzugefügt werden. Die Länge der Episode sowie die Häufigkeit, mit der diese Episode eingeprägt wird, ist vorher anzugeben. Die Ereignisse, aus denen die Episode besteht, werden zufällig ausgewählt, wobei Dopplungen ausgeschlossen sind. Abbildung 7.15 zeigt die Häufigkeitsverteilung der erzeugten Testdaten.



(a) *Ereignissequenz mit 10000 Ereignissen* **(b)** *Ereignissequenz mit 100000 Ereignissen*

Abbildung 7.15: *Qualitative Häufigkeitsverteilung der generierten Zufallsdatensätze mit eingeprägter Episode der Länge* 5

Die zwei Testdatensätze wurden mit unterschiedlicher Einprägung generiert. Der Datensatz mit 10000 Ereignissen erhält die Episode $\alpha =' BWGUN'$ und der Datensatz mit 100000 Ereignissen die Episode $\beta =' PVWLT'$ als Einprägung. Die dazu eingesetzten Aufrufe der Testanwendung lauten wie folgt:

```
1 $ mannilaAlgorithm -oData10000.csv -r10000 -p1000 -s5
2 $ mannilaAlgorithm -oData100000.csv -r100000 -p10000 -s5
```

Histogrammanalyse

Die in Abbildung 7.15 dargestellten Häufigkeitsverteilungen zeigen deutlich, dass die Anzahl der Vorkommnisse von fünf Ereignissen über der Anzahl der anderen liegt. Bevor die Daten durch den WinEpi-Algorithmus analysiert werden, sollen zunächst die zu erwartenden Analyseergebnisse diskutiert werden.

Da die eingeprägten Episoden eine zeitliche Ausdehnung von fünf Ereignissen besitzen (alle Ereignisse der Episode sind aufeinander folgend), müssen diese nach dem fünften Durchlauf des Algorithmus erkannt werden (siehe Abschnitt 4.2.4 auf Seite 75). Da die Ereignisse der Ereignissequenz, abgesehen von den Ereignissen der eingeprägten Episode, gleichverteilt sind, sollte darüber hinaus kein weiterer Durchlauf erfolgen.

Abschätzung der Fensterbreite

Des Weiteren führt die zeitliche Ausdehnung der eingeprägten Episode dazu, dass der WinEpi-Algorithmus die Episode nur dann erkennen kann, wenn die Fensterbreite min-

destens fünf Zeiteinheiten entspricht. Um zu einer genaueren Abschätzung der minimalen Fensterbreite zu kommen dienen folgende Betrachtungen.

Nach Gleichung 4.1 gilt für die Anzahl an Fenstern $\omega(\mathbb{S}, N)$ der Zusammenhang $|\mathbb{S}| + N - 1$. Das heißt, für eine Episode α die nur genau einmal in einem Fenster auftritt gilt für die relative Häufigkeit folgender Zusammenhang.

$$f_\alpha = \frac{1}{|\mathbb{S}| + N - 1}$$

Da eine Episode nur dann als häufig detektiert wird, wenn sie den Schwellwert min_f erreicht oder überschreitet, muss sie daher mindestens in min_{Win} Fenstern detektiert werden.

$$\begin{aligned} min_{Win} &= \frac{min_f}{\frac{1}{|\mathbb{S}|+N-1}} \\ &= min_f \cdot (|\mathbb{S}| + N - 1) \end{aligned} \tag{7.3}$$

Ist bei der Analyse einer Ereignissequenz die zeitliche Ausdehnung der gesuchten (eingeprägten) Episode identisch mit der Fensterbreite, so kann ein Vorkommen der Episode nur in einem Fenster detektiert werden, nicht jedoch in zwei oder mehr aufeinander folgenden Fenstern. Wird eine Episode in diesem Fall also n_{stamp}-mal in die Testdaten eingeprägt, so wird sie in genau n_{stamp} Fenstern detektiert. Ist die Fensterbreite dagegen um eine Zeiteinheit größer, so kann ein Vorkommen in genau zwei aufeinander folgenden Fenstern detektiert werden, was den bereits vermuteten nahezu proportionalen Zusammenhang bestätigt.

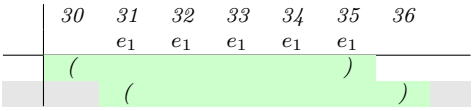
Detektion von Episoden

Nach diesen Vorüberlegungen dient das folgende Beispiel der Abschätzung der notwendigen Fensterbreite zur Detektion der eingeprägten Episode am Beispiel der Sequenz mit 10000 Ereignissen.

Beispiel 7.4. *Die generierten Testdatensätze umfassen* 10000 *zeitlich aufeinander folgende Ereignisse. Die eingeprägte Episode besitzt eine zeitliche Ausdehnung von fünf Zeiteinheiten. Bei einem Schwellwert* $min_f = 0,4$ *muss die Episode nach Gleichung 7.3 in mindestens* 4002 *Fenstern detektiert werden:*

$$\begin{aligned} min_{Win} &= min_f \cdot (|\mathbb{S}| + N - 1) \\ &= 0,4 \cdot (10000 + 5 - 1) \\ &= 4001,6 \end{aligned}$$

Da die Episode 1000*-mal eingeprägt wurde, kann sie bei einer Fensterbreite von* $N = 5$ *nur* 1000*-mal detektiert werden. Vergrößert man das Fenster um eine Zeiteinheit (von fünf auf sechs), so kann jede Episode, die nicht am Anfang der Sequenz beginnt oder am Ende der Sequenz endet, in zwei aufeinander folgenden Fenstern detektiert werden, wie folgende Abbildung veranschaulicht.*



Die Anzahl der Vorkommnisse erhöht sich damit um 1000. *Um die notwendige Anzahl an Vorkommnissen von* $4001,6$ *zu erreichen muss die Fenstergröße also vier mal erhöht werden. Man erhält dadurch als Abschätzung für die minimale Fenstergröße* $N = 5+4 = 9$.

Analyse der Testdaten

Nach dieser Abschätzung kann die Analyse der Testdaten erfolgen. Diese wird dabei jeweils mit den Fensterbreiten 4, 6, 8 und 10 Zeiteinheiten durchgeführt. In beiden Fällen sollte die eingeprägte Episode nach den vorherigen Überlegungen erst im letzten Analysedurchlauf erfolgreich detektiert werden. Für die Analyse der Ereignissequenz mit 10000 Ereignissen kommen die folgenden Aufrufe der Testanwendung zum Einsatz.

```
$ mannilaAlgorithm -cData10000.csv -w4
$ mannilaAlgorithm -cData10000.csv -w6
$ mannilaAlgorithm -cData10000.csv -w8
$ mannilaAlgorithm -cData10000.csv -w10
```

Die Analyse der Ereignissequenz mit 100000 Ereignissen wird analog dazu mit folgenden Aufrufen der Testanwendung durchgeführt.

```
$ mannilaAlgorithm -cData100000.csv -w4
$ mannilaAlgorithm -cData100000.csv -w6
$ mannilaAlgorithm -cData100000.csv -w8
$ mannilaAlgorithm -cData100000.csv -w10
```

Die Tabellen 7.4a und 7.4b zeigen die Analyseergebnisse für die zwei Ereignissequenzen mit den eingeprägten Episoden. Die Analyse erfolgte in beiden Fällen mit vier unterschiedlichen Fensterbreiten ($N = 4, ..., 10$). In den Tabellen ist sowohl die Anzahl der resultierenden Kandidaten als auch die Anzahl der häufigen Episoden für jeden Durchlauf angegeben. Darüber hinaus zeigt die dritte Spalte die Gesamtlaufzeit der Analyse.

Es ist deutlich zu erkennen, dass unabhängig von der verwendeten Fenstergröße im ersten Durchlauf des Algorithmus fünf häufige Episoden detektiert werden. Die naheliegende Vermutung, dass es sich bei diesen Episoden um die Ereignisse aus der eingeprägten Episode handelt, bestätigt sich bei Betrachtung der in Tabelle 7.5a und 7.5b dargestellten Ergebnismengen $\mathbb{F}_i$.

Wie in Abschnitt 7.3.1 beschrieben wurde, generiert der Algorithmus auf Basis aller in der Ereignissequenz auftretenden Ereignisse zunächst $\mathbb{C}_1$, die Kandidaten der Länge $l = 1$. Diese Menge besitzt aufgrund des Einsatzes von Buchstaben aus dem Alphabet zur Repräsentation der Ereignisse eine Länge von 26 und ist für beide Ereignissequenzen identisch und unabhängig von der Fenstergröße.

WinEpi Abbruchbedingung

Besonders anschaulich geht die Funktionsweise der Abbruchbedingung des Algorithmus

N	$\lvert\mathbb{C}_l\rvert$	$\lvert\mathbb{F}_l\rvert$	Laufzeit
4	$\lvert\mathbb{C}_1\rvert = 26$ $\lvert\mathbb{C}_2\rvert = 20$	$\lvert\mathbb{F}_1\rvert = 5$ $\lvert\mathbb{F}_2\rvert = 0$	1172ms
6	$\lvert\mathbb{C}_1\rvert = 26$ $\lvert\mathbb{C}_2\rvert = 20$ $\lvert\mathbb{C}_3\rvert = 0$	$\lvert\mathbb{F}_1\rvert = 5$ $\lvert\mathbb{F}_2\rvert = 4$ $\lvert\mathbb{F}_3\rvert = 0$	1846ms
8	$\lvert\mathbb{C}_1\rvert = 26$ $\lvert\mathbb{C}_2\rvert = 20$ $\lvert\mathbb{C}_3\rvert = 7$ $\lvert\mathbb{C}_4\rvert = 2$ $\lvert\mathbb{C}_5\rvert = 0$	$\lvert\mathbb{F}_1\rvert = 5$ $\lvert\mathbb{F}_2\rvert = 9$ $\lvert\mathbb{F}_3\rvert = 7$ $\lvert\mathbb{F}_4\rvert = 2$ $\lvert\mathbb{F}_5\rvert = 0$	3285ms
10	$\lvert\mathbb{C}_1\rvert = 26$ $\lvert\mathbb{C}_2\rvert = 20$ $\lvert\mathbb{C}_3\rvert = 60$ $\lvert\mathbb{C}_4\rvert = 5$ $\lvert\mathbb{C}_5\rvert = 1$ $\lvert\mathbb{C}_6\rvert = 0$	$\lvert\mathbb{F}_1\rvert = 5$ $\lvert\mathbb{F}_2\rvert = 20$ $\lvert\mathbb{F}_3\rvert = 10$ $\lvert\mathbb{F}_4\rvert = 5$ $\lvert\mathbb{F}_5\rvert = 1$ $\lvert\mathbb{F}_6\rvert = 0$	8034ms

(a) $n = 10000$, $\alpha =' BWGUN'$

N	$\lvert\mathbb{C}_l\rvert$	$\lvert\mathbb{F}_l\rvert$	Laufzeit
4	$\lvert\mathbb{C}_1\rvert = 26$ $\lvert\mathbb{C}_2\rvert = 20$	$\lvert\mathbb{F}_1\rvert = 5$ $\lvert\mathbb{F}_2\rvert = 0$	29832ms
6	$\lvert\mathbb{C}_1\rvert = 26$ $\lvert\mathbb{C}_2\rvert = 20$ $\lvert\mathbb{C}_3\rvert = 0$	$\lvert\mathbb{F}_1\rvert = 5$ $\lvert\mathbb{F}_2\rvert = 4$ $\lvert\mathbb{F}_3\rvert = 0$	54152ms
8	$\lvert\mathbb{C}_1\rvert = 26$ $\lvert\mathbb{C}_2\rvert = 20$ $\lvert\mathbb{C}_3\rvert = 7$ $\lvert\mathbb{C}_4\rvert = 1$	$\lvert\mathbb{F}_1\rvert = 5$ $\lvert\mathbb{F}_2\rvert = 9$ $\lvert\mathbb{F}_3\rvert = 6$ $\lvert\mathbb{F}_4\rvert = 0$	87880ms
10	$\lvert\mathbb{C}_1\rvert = 26$ $\lvert\mathbb{C}_2\rvert = 20$ $\lvert\mathbb{C}_3\rvert = 51$ $\lvert\mathbb{C}_4\rvert = 5$ $\lvert\mathbb{C}_5\rvert = 1$ $\lvert\mathbb{C}_6\rvert = 0$	$\lvert\mathbb{F}_1\rvert = 5$ $\lvert\mathbb{F}_2\rvert = 19$ $\lvert\mathbb{F}_3\rvert = 10$ $\lvert\mathbb{F}_4\rvert = 5$ $\lvert\mathbb{F}_5\rvert = 1$ $\lvert\mathbb{F}_6\rvert = 0$	250723ms

(b) $n = 100000$, $\beta =' PVWLT'$

Tabelle 7.4: *Sequenzanalysen mit* $min_f = 40\%$

aus den Tabellen 7.4a und 7.4b hervor. Es existieren zwei Möglichkeiten, die zum Abbruch der zyklischen Ausführung des WinEpi Algorithmus führen können. Die erste tritt ein, wenn im aktuellen Durchgang unter den Kandidaten keine Episode als häufig detektiert wird. Dieser Fall ist daran zu erkennen, dass $\mathbb{C}_l > 0$ und $\mathbb{F}_l = 0$ ist. Diese Abbruchbedingung liegt zum Beispiel in der Tabelle 7.4b bei den Durchgängen für $N = 4$ und $N = 8$ vor.

Die zweite Möglichkeit des Analyseabbruchs tritt dann ein, wenn aus den häufigen Episoden $\mathbb{F}_l$ keine Kandidaten $\mathbb{C}_{l+1}$ hervor gehen. Dieser Fall ist daran zu erkennen, dass sowohl $\mathbb{C}_l = 0$ als auch $\mathbb{F}_l = 0$ ist, wie es in den Durchläufen für $N = 6$, $N = 8$ und $N = 10$ in Tabelle 7.4a der Fall ist. Die Gründe dafür liegen in der Regel darin, dass Satz 4.1 auf Seite 78 zur Anwendung kommt (siehe Abschnitt 4.2.4).

innere Ordnung

Weiterhin lassen sich die zuvor formulierten Annahmen über die Analyseergebnisse anhand der Tabellen 7.4a und 7.4b bestätigen. In beiden Fällen wird die eingeprägte Episode nach genau fünf Durchgängen detektiert und in beiden Fällen erst bei einer Fensterbreite von $N > 8$. Es ist festzustellen, dass die innere Ordnung der eingeprägten Episoden in der jeweiligen Ergebnismenge $\mathbb{F}_5$ korrekt wiedergegeben wird, obwohl die Reihenfolge der detektierten Episoden der Länge eins in $\mathbb{F}_1$ zunächst willkürlich ist. Das heißt, die Detektion von seriellen Episoden über die eingeführten Automaten arbeitet fehlerfrei.

$\mathbb{F}_1$	$\{'G','U','N','B','W'\}$
$\mathbb{F}_2$	$\{'GU','GN','GB','GW','UG',$ $'UN','UB','UW','NG','NU',$ $'NB','NW','BG','BU','BN',$ $'BW','WG','WU','WN','WB'\}$
$\mathbb{F}_3$	$\{'GUN','BGU','BGN','BUN','BWG',$ $'BWU','BWN','WGU','WGN','WUN'\}$
$\mathbb{F}_4$	$\{'BGUN','BWGU','BWGN','BWUN','WGUN'\}$
$\mathbb{F}_5$	$\{'BWGUN'\}$

(a) $n = 10000$, $\alpha =' BWGUN'$

$\mathbb{F}_1$	$\{'P','W','T','V','L'\}$
$\mathbb{F}_2$	$\{'PW','PT','PV','PL','WP',$ $'WT','WV','WL','TP','TW',$ $'TV','TL','VP','VW','VT',$ $'VL','LP','LT','LV'\}$
$\mathbb{F}_3$	$\{'PWT','PWL','PVW','PVT','PVL',$ $'PLT','WLT','VWT','VWL','VLT'\}$
$\mathbb{F}_4$	$\{'PWLT','PVWT','PVWL','PVLT','VWLT'\}$
$\mathbb{F}_5$	$\{'PVWLT'\}$

(b) $n = 100000$, $\beta =' PVWLT'$

Tabelle 7.5: *Häufige Episoden der Sequenzanalyse mit* $min_f = 40\%$ *und* $N = 10$

7.3.5 Anwendung auf reale Daten

Ausgangspunkt der Analyse von realen Daten werden die Datensätze aus der in Abschnitt 7.2.2 vorgefilterten Datenbasis sein. Die aufgezeichneten Ereignisse lassen sich in vier unterschiedliche Gruppen aufteilen:

- Ereignisse, die beim Starten von Systemhandlungen durch den Benutzer initiiert wurden
- Ereignisse, die beim Stoppen von Systemhandlungen durch den Benutzer initiiert wurden
- Ereignissen, die zum Starten von Benutzerinteraktionen durch den Sequenzer initiiert wurden
- Ereignissen, die zum Stoppen von Benutzerinteraktionen durch den Sequenzer initiiert wurden

Analyse der Start-Ereignisse

Da die Stopp-Ereignisse direkt mit einem einleitenden Start-Ereignis korrespondieren, wird deren Analyse zu keinem Informationsgewinn führen, so dass deren Analyse nicht notwendig ist. Des Weiteren kann auf die Analyse der Ereignisse, die in Verbindung mit Benutzerinteraktionen auftreten verzichtet werden, da diese nicht vom Endbenutzer,

sondern vom System initiiert werden. Um das Ziel der Anpassung an den Endbenutzer zu erreichen, sind daher lediglich die Start-Ereignisse zu analysieren, die mit dem Start von Systemhandlungen korrespondieren.

Auf die in Abschnitt 7.2.2 beschriebene Aufteilung der Daten in (den Tageszeiten entsprechende) Intervalle wird bei der folgenden Analyse bewusst verzichtet. Da auf der Messe, auf der die Daten erhoben wurden, in allen Vorführungen dasselbe Szenario präsentiert wurde, ließen sich keine Unterschiede der Systemnutzung in Abhängigkeit von der Tageszeit ableiten.

Keine Intervallbearbeitung

Der Zugriff auf die Datenbasis in der RTK-QSDB Datenbank erfolgt über den SQL-Modus der Testanwendung. Die Filterung der Ereignisse übernimmt dabei die folgende SQL-Anweisung:

```
SELECT u.‘UnixTimestamp‘,
       SUBSTRING_INDEX(u.‘Request‘, ’:’, -1) as Request
FROM usageData u
WHERE (u.‘userProfiles_UserProfileID‘ = 1)
      and (SUBSTRING_INDEX(u.‘Request‘, ’:’, 2)
           = ’REQUEST:START_TASK’)
ORDER BY u.‘UnixTimestamp‘;
```

Neben der Filterung aller Ereignistypen, die nicht mit in die Analyse übernommen werden sollen, übernimmt die SQL-Anweisung auch die Auswahl des Benutzers, dessen Nutzungsverhalten zu analysieren ist. Diese Auswahl erfolgt über die entsprechende Nutzer-ID `userProfiles_UserProfileID`. Das Ergebnis der Abfrage ist eine Ereignissequenz mit 691 Ereignissen, die über die folgenden Kommandos mit der Testanwendung analysiert wird.

Datenakquise im SQL-Modus

```
$ mannilaAlgorithm -q rtk-usageData.sql -m 0.4 -w 4 -g
$ mannilaAlgorithm -q rtk-usageData.sql -m 0.4 -w 6 -g
$ mannilaAlgorithm -q rtk-usageData.sql -m 0.4 -w 8 -g
$ mannilaAlgorithm -q rtk-usageData.sql -m 0.4 -w 10 -g
$ mannilaAlgorithm -q rtk-usageData.sql -m 0.4 -w 12 -g
$ mannilaAlgorithm -q rtk-usageData.sql -m 0.4 -w 14 -g
$ mannilaAlgorithm -q rtk-usageData.sql -m 0.4 -w 16 -g
$ mannilaAlgorithm -q rtk-usageData.sql -m 0.4 -w 18 -g
$ mannilaAlgorithm -q rtk-usageData.sql -m 0.4 -w 20 -g
```

Die Datenbasis besitzt eine zeitliche Auflösung von $1s$ und enthält die serialisierten Daten aller zur Auswertung markierter Anfragen aus der MMS (siehe Abschnitt 7.2), wodurch deren Reihenfolge automatisch korrekt abgebildet wird. Liegen zwei Ereignisse sehr dicht beieinander ($\Delta t < 1s$), so ist es möglich, dass diese in der Datenbasis den gleichen Zeitstempel erhalten. Da die Auswertung innerhalb des Algorithmus ebenfalls serialisiert abläuft, werden die Ereignisse nacheinander bearbeitet, so dass durch die Abbildung auf die zeitliche Auflösung der Datenbasis keine Informationen über die sequentielle Abfolge von Ereignissen verloren gehen.

Bei der Analyse ergeben sich zunächst die elf Ereignisse, die innerhalb der Datenbasis vorkommen. Diese werden als Kandidaten $\mathbb{C}_1$ für den ersten Durchlauf des WinEpi Algorithmus verwendet.

$$\begin{aligned}\mathbb{C}_1 = \{&'fetch\_meal\_getoutsideonly', 'cook\_meal\_prepare',\\ &'cook\_meal\_withopendoor\_mealtrayislifted',\\ &'cook\_meal\_mealtrayisininsertloc',\\ &'fetch\_meal\_from\_oven\_withopendoor\_getoutsideonly',\\ &'fetch\_meal\_from\_oven\_putdownonly', 'eat\_meal\_support',\\ &'eat\_meal\_support\_finish', 'eat\_meal\_support\_continue',\\ &'fetch\_meal\_from\_oven\_closedooronly', 'fetch\_meal\_from\_tray'\}\end{aligned}$$

mögliche Ereignisse

Diese elf Ereignisse aus $\mathbb{C}_1$ werden im folgenden beschrieben, um später die Plausibilität der detektierten Episoden überprüfen zu können. Dabei ist zu beachten, dass es sich bei diesen Systemaufgaben um Teilsequenzen handelt. Vollständige Szenarien können derzeit noch nicht von der MASSiVE Architektur bearbeitet werden[11]. Da der WinEpi Algorithmus nur mit den symbolischen Namen der Teilsequenzen in Form von Zeichenketten arbeitet, ist er aber auch später bei Einsatz von kompletten Szenarien unverändert einsetzbar.

- `fetch_meal_getoutsideonly` Das System detektiert das Mahlzeitentablett innerhalb des geöffneten Kühlschranks, bewegt den Greifer des Roboterarms in eine Greifposition und greift das Mahlzeitentablett. Abschließend wird das Tablett vom Roboterarm angehoben.

- `fetch_meal_from_tray` Das System detektiert das Mahlzeitentablett auf dem Tablett vor dem Benutzer. Im Anschluss daran wird das Tablett gegriffen und angehoben.

- `cook_meal_prepare` Der Roboterarm wird mit dem gegriffenen Mahlzeitentablett nach rechts (aus Sicht des Benutzers) bewegt. Der Benutzer kann das System jetzt vor der Mikrowelle positionieren ohne das der Roboterarm im Weg ist.

- `cook_meal_withopendoor_mealtrayislifted` Die Mikrowelle wird automatisch geöffnet und der Roboterarm bewegt seinen Greifer mit dem gegriffenen Mahlzeitentablett in die Mikrowelle. Im Anschluss daran schließt der Roboterarm mit seinem Greifer die Tür der Mikrowelle und startet den Aufwärmvorgang.

- `cook_meal_mealtrayisininsertloc` Der Roboterarm bewegt seinen Greifer mit dem gegriffenen Mahlzeitentablett in die Mikrowelle. Im Anschluss daran schließt der Roboterarm die Tür der Mikrowelle und startet den Aufwärmvorgang.

- `fetch_meal_from_oven_withopendoor_getoutsideonly` Das System öffnet die Mikrowelle, detektiert das Mahlzeitentablett darin, bewegt den Greifer in eine mögliche

[11] Stand: Mai 2011

Greifposition und schließt ihn. Im Anschluss daran wird das Mahlzeitentablett angehoben und aus der Mikrowelle bewegt.

- `fetch_meal_from_oven_closedooronly` Das System detektiert die Mikrowelle und schließt deren Tür mit dem Greifer des Roboterarmes. Das Mahlzeitentablett befindet sich während des Vorganges auf dem Tablett vor dem Benutzer.

- `fetch_meal_from_oven_putdownonly` Der Roboterarm bewegt das gegriffene Mahlzeitentablett über das Tablett vor dem Benutzer und stellt es darauf ab.

- `eat_meal_support` Das System detektiert den im Mahlzeitentablett integrierten Löffel und bewegt den Greifer in eine mögliche Greifposition. Der Greifer wird geschlossen und der Löffel aus dem Mahlzeitentablett entnommen. Im Anschluss daran kann der Löffel in den Teller geführt werden, um Nahrung aufzunehmen. Abschließend wird der Löffel vor dem Mund des Benutzers ausgerichtet, so dass dieser die Nahrung vom Löffel aufnehmen kann.

- `eat_meal_support_continue` Der leere Löffel vor dem Mund des Benutzers wird erneut zum Teller geführt und mit Nahrung gefüllt um diese dem Benutzer anzureichen.

- `eat_meal_support_finish` Der leere Löffel vor dem Mund des Benutzers wird nach der Detektion des Mahzeitentabletts wieder in diesem abgelegt.

Laboraufbau: Küche

Die den Teilhandlungen zugrunde liegende Benutzer- und Systemumgebung ist der Laboraufbau einer Küche. Die Einrichtung der Küche besteht aus einem Kühlschrank und einer Mikrowelle. Im Kühlschrank befindet sich ein Mahlzeitentablett mit einem vorbereiteten Gericht. Der Nutzer sitzt im FRIEND System und hat das Tablett des Rollstuhls vor sich. Sowohl der Kühlschrank als auch die Mikrowelle besitzen einen elektromechanischen Türöffner, der durch den Benutzer über eine entsprechende Erweiterung der MMS bedient werden kann.

Analyse der Messedaten

Die Tabellen 7.6a und 7.6b zeigen die Ergebnisse der Analyse aller in der Datenbasis enthaltenen Datensätze. Bei allen Analysevorgängen wird mit einem Schwellwert von $min_f = 40\%$ gearbeitet. Dieser Wert wird von [MTV97] als Standardwert empfohlen und hat sich auch in den Testläufen bewährt.

Ausführungszeiten

Betrachtet man die Ausführungszeiten, so liegen diese bis zu einer Fensterbreite von $N = 14$ unter einer Sekunde. Bei einer Fenstergröße von $N > 14$ steigt die Ausführungszeit schnell an. Das hängt damit zusammen, dass durch die Vergrößerung der Fensterbreite auch die Häufigkeit von Kandidaten steigt und somit mehrere häufige Kandidaten für den jeweils nächsten Schritt zu berücksichtigen sind. Dieser Zusammenhang führt darüber hinaus zur Notwendigkeit von weiteren Durchläufen, was ebenfalls die Ausführungszeit beeinflusst. Vor der Integration des WinEpi Algorithmus als Erweiterung in die MMS muss daher ein Kompromiss aus möglichst gutem Laufzeitverhalten und der zu erwartenden Informationsmenge gefunden werden.

N	$\lvert\mathbb{C}_l\rvert$	$\lvert\mathbb{F}_l\rvert$	Laufzeit
4	$\lvert\mathbb{C}_1\rvert = 11$	$\lvert\mathbb{F}_1\rvert = 0$	22ms
6	$\lvert\mathbb{C}_1\rvert = 11$ $\lvert\mathbb{C}_2\rvert = 30$	$\lvert\mathbb{F}_1\rvert = 6$ $\lvert\mathbb{F}_2\rvert = 0$	180ms
8	$\lvert\mathbb{C}_1\rvert = 11$ $\lvert\mathbb{C}_2\rvert = 42$ $\lvert\mathbb{C}_3\rvert = 0$	$\lvert\mathbb{F}_1\rvert = 7$ $\lvert\mathbb{F}_2\rvert = 4$ $\lvert\mathbb{F}_3\rvert = 0$	231ms
10	$\lvert\mathbb{C}_1\rvert = 11$ $\lvert\mathbb{C}_2\rvert = 42$ $\lvert\mathbb{C}_3\rvert = 2$ $\lvert\mathbb{C}_4\rvert = 0$	$\lvert\mathbb{F}_1\rvert = 7$ $\lvert\mathbb{F}_2\rvert = 8$ $\lvert\mathbb{F}_3\rvert = 2$ $\lvert\mathbb{F}_4\rvert = 0$	174ms
12	$\lvert\mathbb{C}_1\rvert = 11$ $\lvert\mathbb{C}_2\rvert = 42$ $\lvert\mathbb{C}_3\rvert = 9$ $\lvert\mathbb{C}_4\rvert = 1$ $\lvert\mathbb{C}_5\rvert = 0$	$\lvert\mathbb{F}_1\rvert = 7$ $\lvert\mathbb{F}_2\rvert = 14$ $\lvert\mathbb{F}_3\rvert = 7$ $\lvert\mathbb{F}_4\rvert = 1$ $\lvert\mathbb{F}_5\rvert = 0$	321ms
14	$\lvert\mathbb{C}_1\rvert = 11$ $\lvert\mathbb{C}_2\rvert = 42$ $\lvert\mathbb{C}_3\rvert = 44$ $\lvert\mathbb{C}_4\rvert = 2$ $\lvert\mathbb{C}_5\rvert = 0$	$\lvert\mathbb{F}_1\rvert = 7$ $\lvert\mathbb{F}_2\rvert = 25$ $\lvert\mathbb{F}_3\rvert = 13$ $\lvert\mathbb{F}_4\rvert = 2$ $\lvert\mathbb{F}_5\rvert = 0$	698ms

(a) $N = 4, 6, 8, 10, 12, 14$

N	$\lvert\mathbb{C}_l\rvert$	$\lvert\mathbb{F}_l\rvert$	Laufzeit
16	$\lvert\mathbb{C}_1\rvert = 11$ $\lvert\mathbb{C}_2\rvert = 56$ $\lvert\mathbb{C}_3\rvert = 149$ $\lvert\mathbb{C}_4\rvert = 12$ $\lvert\mathbb{C}_5\rvert = 2$ $\lvert\mathbb{C}_6\rvert = 0$	$\lvert\mathbb{F}_1\rvert = 8$ $\lvert\mathbb{F}_2\rvert = 37$ $\lvert\mathbb{F}_3\rvert = 33$ $\lvert\mathbb{F}_4\rvert = 11$ $\lvert\mathbb{F}_5\rvert = 2$ $\lvert\mathbb{F}_6\rvert = 0$	2838ms
18	$\lvert\mathbb{C}_1\rvert = 11$ $\lvert\mathbb{C}_2\rvert = 56$ $\lvert\mathbb{C}_3\rvert = 182$ $\lvert\mathbb{C}_4\rvert = 29$ $\lvert\mathbb{C}_5\rvert = 4$ $\lvert\mathbb{C}_6\rvert = 0$	$\lvert\mathbb{F}_1\rvert = 8$ $\lvert\mathbb{F}_2\rvert = 41$ $\lvert\mathbb{F}_3\rvert = 52$ $\lvert\mathbb{F}_4\rvert = 24$ $\lvert\mathbb{F}_5\rvert = 4$ $\lvert\mathbb{F}_6\rvert = 0$	5254ms
20	$\lvert\mathbb{C}_1\rvert = 11$ $\lvert\mathbb{C}_2\rvert = 56$ $\lvert\mathbb{C}_3\rvert = 211$ $\lvert\mathbb{C}_4\rvert = 66$ $\lvert\mathbb{C}_5\rvert = 11$ $\lvert\mathbb{C}_6\rvert = 1$ $\lvert\mathbb{C}_7\rvert = 0$	$\lvert\mathbb{F}_1\rvert = 8$ $\lvert\mathbb{F}_2\rvert = 44$ $\lvert\mathbb{F}_3\rvert = 81$ $\lvert\mathbb{F}_4\rvert = 48$ $\lvert\mathbb{F}_5\rvert = 11$ $\lvert\mathbb{F}_6\rvert = 1$ $\lvert\mathbb{F}_7\rvert = 0$	9992ms

(b) $N = 16, 18, 20$

Tabelle 7.6: *Sequenzanalysen Hannover Messe 2010 ($min_f = 40\%$ und $n = 691$)*

Abbildung der Szenarien

Bei der Sequenzanalyse liegen die Information in den als häufig erkannten Episoden. Das Ziel ist dabei möglichst viele Ereignisse zu ermitteln, die zueinander in Beziehung stehen. Das heißt, der Informationsgehalt wächst mit der Länge der detektierten Episoden. Dieser Zusammenhang lässt sich anschaulich erklären indem die Zielsetzung des WinEpi Einsatzes in der MMS betrachtet wird: die häufigen Episoden geben Sequenzen vor, die als besonders häufig einzustufen sind, und können so genutzt werden, um aus bereits eingetretenen Ereignissen Rückschlüsse auf folgende Ereignisse zu ziehen.

Nach Satz 4.1 sind alle Unterepisoden einer häufigen Episode ebenfalls häufig. Daher ist sichergestellt, dass alle Unterepisoden von $\mathbb{F}_n$ in $\mathbb{F}_{n-1}$ enthalten sind. Aus diesem Grund gehen keine Informationen verloren, wenn nur die häufigen Episoden betrachtet werden, die aus dem jeweils letzten Durchgang des WinEpi Algorithmus hervorgehen. Vor der Auflistung dieser Episodenmengen werden alle vorkommenden Ereignisse zunächst auf Buchstaben abgebildet, um die Übersichtlichkeit zu erhöhen. Die verwendete Abbildung ist in Tabelle 7.7 zu finden.

Der Zusammenhang, dass mit einem breiteren Fenster die Anzahl der als häufig detektierten Episoden und damit auch deren Länge steigt wird durch die in Tabelle 7.8 dargestellten Analyseergebnisse bestätigt. Dabei werden die Ergebnismengen $\mathbb{F}$ des jeweils letzten Analysedurchlaufs nach der Abbildung auf die zuvor eingeführten Buchstaben dargestellt. Die Fenstergröße wird von $N = 4$ Schrittweise auf $N = 20$ erhöht.

Ereignis	abgebildet auf
`fetch_meal_getoutsideonly`	„A“
`cook_meal_prepare`	„B“
`cook_meal_withopendoor_mealtrayislifted`	„C“
`cook_meal_mealtrayisininsertloc`	„D“
`fetch_meal_from_oven_withopendoor_getoutsideonly`	„E“
`fetch_meal_from_oven_putdownonly`	„F“
`eat_meal_support`	„G“
`eat_meal_support_finish`	„H“
`eat_meal_support_continue`	„I“
`fetch_meal_from_oven_closedooronly`	„J“
`fetch_meal_from_tray`	„K“

Tabelle 7.7: *Abbildung der Ereignisse auf Buchstaben*

Zur Untersuchung der Plausibilität der detektierten häufigen Episoden wird die zuvor eingeführte Abbildung der Ereignisse auf Buchstaben wieder rückgängig gemacht. Das führt im Fall der Episodenmenge $\mathbb{F}_6^{N=20}$ zu den folgenden Ereignissen:

$$\begin{aligned}\mathbb{F}_6^{N=20} = \{'&fetch\_meal\_getoutsideonly\\ &cook\_meal\_prepare\\ &cook\_meal\_withopendoor\_mealtrayislifted\\ &fetch\_meal\_from\_oven\_withopendoor\_getoutsideonly\\ &fetch\_meal\_from\_oven\_putdownonly\\ &eat\_meal\_support|'\}\end{aligned}$$

Plausibilitätsüberprüfung

Das der als häufig detektierten Episode entsprechende Szenario umfasst also das Detektieren, Greifen und Anheben des Mahlzeitentabletts im geöffneten Kühlschrank. Im Anschluss daran folgt die Bewegung des Greifers mit dem Mahlzeitentablett an eine freie Position rechts im Arbeitsraum, um das Öffnen der Mikrowelle zu ermöglichen. Danach wird das gegriffene Tablett in der Mikrowelle abgesetzt, die Mikrowelle geschlossen und der Aufwärmvorgang gestartet. Nach dem Aufwärmvorgang wird die Mikrowelle geöffnet, das Mahlzeitentablett detektiert, gegriffen und herausgehoben. Der Roboterarm setzt das Tablett dann vor dem Benutzer ab und beginnt mit der Unterstützung beim Essen. Somit handelt es sich bei der als häufig detektierten Abfolge von Systemaufgaben um

$\mathbb{F}_l^N$	
$\mathbb{F}_1^{N=4}$	$\{\}$
$\mathbb{F}_1^{N=6}$	$\{'A',' B',' C',' E',' F',' G'\}$
$\mathbb{F}_2^{N=8}$	$\{'AB',' BC',' CE',' EF'\}$
$\mathbb{F}_3^{N=10}$	$\{'ABC',' BCE'\}$
$\mathbb{F}_4^{N=12}$	$\{'ABCE'\}$
$\mathbb{F}_4^{N=14}$	$\{'ABCE',' BCEF'\}$
$\mathbb{F}_5^{N=16}$	$\{'ABCEF',' BCEFG'\}$
$\mathbb{F}_5^{N=18}$	$\{'ABCEF',' ABCEG',' ABEFG',' BCEFG'\}$
$\mathbb{F}_6^{N=20}$	$\{'ABCEFG'\}$

Tabelle 7.8: *Episoden nach der Abbildung*

die Einzelschritte des auf der Messeveranstaltung vorgeführten ADL-Szenarios wie der vergleich mit [Uni09] bestätigt.

redundante Informationen

Die Ergebnismengen $\mathbb{F}_1^{N=4}$ bis $\mathbb{F}_6^{N=20}$ zeigen, dass die einzelnen Schritte redundante Informationen über den sequentiellen Aufbau der Episoden beinhalten. Die Informationen wiederholen sich zum Teil von Analyse zu Analyse. Dieser Zusammenhang wird im folgenden Abschnitt dazu verwendet, um daraus eine Vorgehensweise für die Integration des Algorithmus in eine Erweiterung der MMS zu entwickeln.

7.4 Modellierung als Erweiterung

Durch die im Abschnitt 7.2.2 vorgestellte Vorgehensweise bei der Datenbereinigung ist es möglich, die Länge der Ereignissequenz signifikant zu reduzieren. Dadurch kann auch die Ausführungszeit des WinEpi Algorithmus verkürzt werden. Dennoch ist es notwendig die Auswertung innerhalb der zu entwickelnden Erweiterung parallel zum Ausführungskontext der MMS zu bearbeiten.

Wie die WinEpi Analyse der Daten im vorherigen Abschnitt gezeigt hat, werden erst nach dem ersten Durchlauf des Algorithmus, bei einer Fensterlänge von $N = 6$, häufige Episoden gefunden. Da diese Episoden jedoch die Länge eins besitzen, können daraus keine häufigen Ausführungssequenzen abgleitet werden. Das heißt, die WinEpi Analyse muss mit einer Fensterbreite von mindesten $N = 8$ beginnen, um zumindest häufige Episoden der Länge zwei zu erhalten.

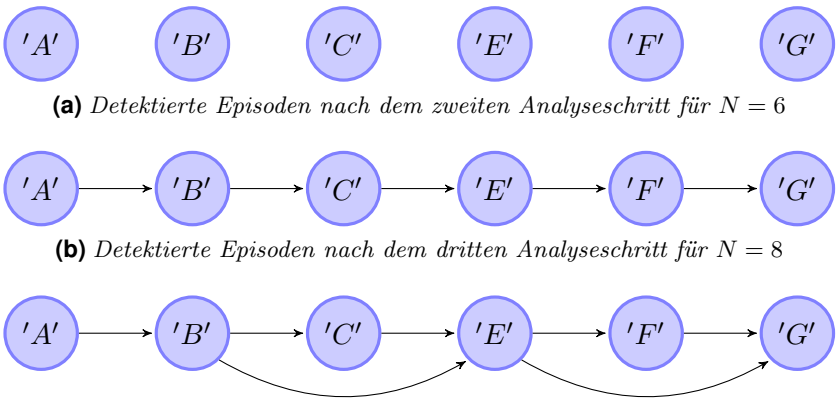
Einfluss des Laufzeitverhaltens

Darüber hinaus haben die Testläufe gezeigt, dass die Ausführungszeit mit steigender Fensterbreite zunimmt, so dass bei der Analyse mit einer großen Fensterbreite wie $N = 20$ erst nach ca. $10s$ einsetzbare Messwerte zur Verfügung stehen. Die Erweiterung wäre in diesem Fall nach der Start der MMS nicht in der Lage eine Abschätzung über das Nutzungsverhalten des Nutzers zu treffen. Auch wenn diese Zeit nicht als problematisch anzusehen ist, so ist weiterhin zu bedenken, dass über die Fensterbreite auch die Länge der maximal zu erkennenden Episoden begrenzt wird. Daher sollte die Fensterbreite so groß wie möglich und im günstigsten Fall nicht fest vorgegeben werden.

Es zeigt sich, dass die als häufig detektierten Episoden (beziehungsweise deren Abhängigkeiten untereinander) mit jedem Durchlauf genauer werden. Um diesen Zusammenhang zu verdeutlichen zeigt Abbildung 7.16 die grafische Korrespondenz der Ergebnisse nach Abbildung 4.11 auf Seite 73.

Entscheidungsbaum generieren

In der Darstellung in Abbildung 7.16 werden die Einzelergebnisse der jeweiligen Analyseschritte zu einem Graphen zusammengefasst. Durch diese Vorgehensweise ergibt sich bereits früh die Möglichkeit den entstandenen Graphen als Entscheidungsbaum zu nutzen und so für Informationen zur Auswertung des Nutzerverhaltens heranzuziehen. Dabei werden die Ergebnisse jedes Analysedurchlaufs in den Graphen integriert, so dass der



(a) *Detektierte Episoden nach dem zweiten Analyseschritt für $N = 6$*

(b) *Detektierte Episoden nach dem dritten Analyseschritt für $N = 8$*

(c) *Detektierte Episoden nach dem letzten Analyseschritt für $N = 20$*

Abbildung 7.16: *Zusammenfassung der Informationen über alle detektierten Episoden*

Entscheidungsbaum mit der Zeit immer genauer die Episoden und deren sequentielle Beziehungen untereinander abbildet.

Neben den Überlegungen zur Anwendung des Algorithmus innerhalb der zu entwickelnden Erweiterung muss noch die Erweiterung selbst entworfen werden. Dieser Entwurf erfolgt dabei nach dem in Kapitel 6 vorgestellten Schema für die Entwicklung von Erweiterungen und ist in Abbildung 7.17 dargestellt.

modellbasierter Entwurf

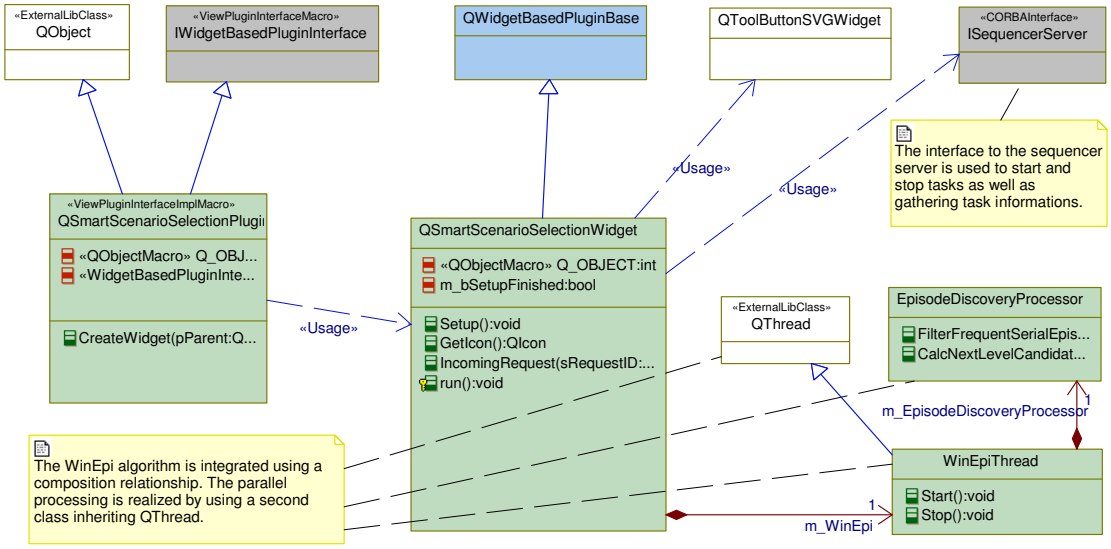


Abbildung 7.17: *UML-Entwurf der Erweiterung zur Integration des WinEpi Algorithmus in die MMS*

Beim Laden einer Erweiterung wird nach Eintragung der MMS-Schnittstellen die `Setup()`-Methode aufgerufen (siehe Abschnitt 6.4.1). Dieser Aufruf erfolgt im Ausführungskontext der MMS, so dass umfangreiche Berechnungen in dieser Methode den Startvor-

parallele Ausführung

gang der MMS verzögern würden. Aus diesem Grund ist es notwendig, die Datenanalyse parallel zum Ausführungskontext der MMS auszuführen. Daher wird Funktionalität des WinEpi Algorithmus in der Klasse `WinEpiThread` gekapselt und über eine Kompositionsbeziehung in die konkrete Instanz der Erweiterung integriert. Um der Forderung nach Parallelisierung nachzukommen, erbt die Klasse `WinEpiThread` die Funktionalitäten der Klasse `QThread`. Dadurch erfolgt die gesamte Analyse in einem separaten Ausführungskontext. Die für die Erweiterung zum Einsatz kommende Spezifikation sowie die entsprechenden Konfigurationswerte nach Abschnitt 6.3 sind den entsprechenden Tabellen in Anhang A.2.3 beziehungsweise Anhang A.2.4 zu entnehmen.

sequentielle Integration

Um bereits möglichst früh verlässliche Daten über die häufigen Episoden zu erhalten, erfolgt die Auswertung der aufgezeichneten Datenbasis zyklisch. Dabei wird in jedem Schritt die Fensterbreite N erhöht. Die Ergebnisse jeder Ausführung werden dazu eingesetzt den Entscheidungsbaum nach Abbildung 7.16 zu erstellen. Zur Laufzeit können dem Benutzer auf Basis dieses Entscheidungsbaums mögliche, folgende (Teil-)Szenarien zur Ausführung präsentiert werden.

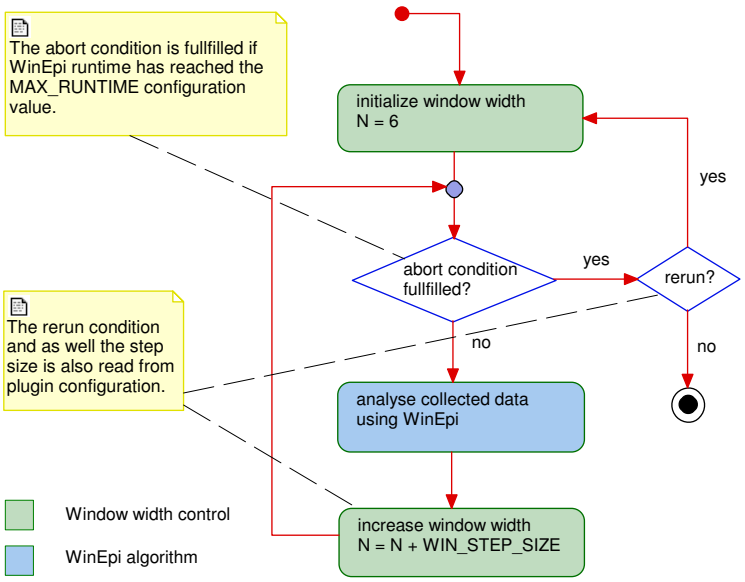


Abbildung 7.18: *Flussdiagramm zur Integration des WinEpi Algorithmus in eine MMS-Erweiterung*

Da die Ausführungszeit mit jedem Durchlauf zunimmt, erfolgt nach jedem die Überprüfung der Laufzeit. Überschreitet diese einen zuvor festgelegten Wert, so endet die zyklische Ausführung. Der bis dahin erstellte Entscheidungsbaum wird in der Datenbasis der Erweiterung abgelegt und steht so später wieder zur Verfügung. Weiterhin kann der Algorithmus so konfiguriert werden, dass er nach Eintreten der Abbruchbedingung von vorn gestartet wird. Der in der Datenbasis abgelegte Entscheidungsbaum wird dann neu erstellt. Durch diese Vorgehensweise ist es möglich zur Laufzeit des System eine einfache Form lernenden Verhaltens zu realisieren. Den gesamten Ablauf skizziert Abbildung 7.18 in Form eines Flussdiagrammes.

7.5 Herstellung des zeitlichen Bezugs

Durch die Generierung eines neuen Zeitstempels für die Ereignisse in der Datenbasis geht der absolute zeitliche Bezug der Ereignisse verloren (siehe Abschnitt 7.2.2). Der vorgestellte Lösungsansatz sieht die Aufteilung aller Ereignisse in die fünf in Tabelle 7.1 dargestellten Zeitintervalle vor. Diese Aufteilung muss erfolgen, bevor die realen Zeitstempel der Ereignisse verloren gehen. Aus diesem Grund werden alle Ereignisse in der Datenbasis zunächst dem entsprechenden Intervall zugeordnet um dann die Analyse über den WinEpi-Algorithmus getrennt für jedes Intervall durchzuführen. Dazu ist das Flussdiagramm der WinEpi-Integration um eine weitere, innere Schleife zu ergänzen, die über die Zeitintervalle iteriert. Abbildung 7.19 zeigt diese Ergänzung.

Beachtung des zeitlichen Bezugs

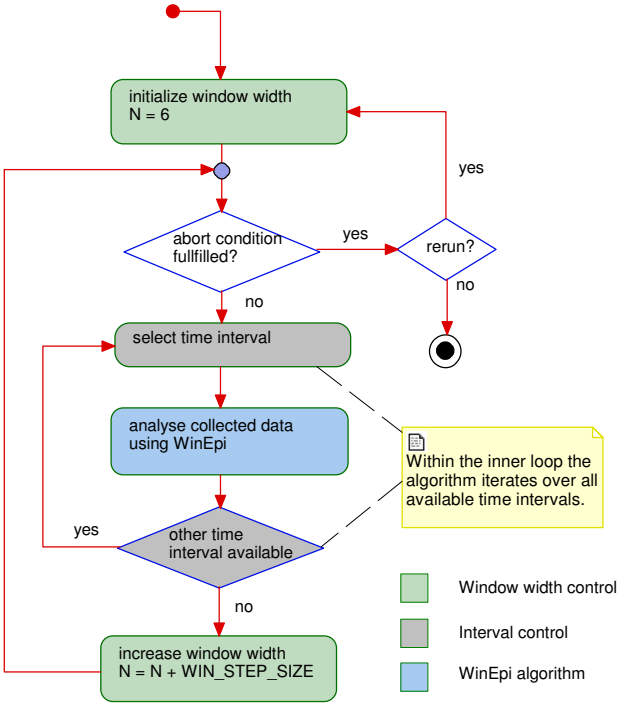


Abbildung 7.19: *Erweitertes Flussdiagramm zur Integration des WinEpi Algorithmus in eine MMS-Erweiterung*

Nach der Überprüfung der Laufzeit des Algorithmus als übergeordnete Abbruchbedingung wird das zu bearbeitende Zeitintervall ausgewählt. Im Anschluss daran folgt die Datenanalyse. Direkt im Anschluss an die Datenanalyse wird der Vorgang so lange mit der gleichen Fensterbreite N wiederholt, bis die Daten aller Zeitintervalle bearbeitet wurden.

Probleme kann man niemals mit derselben Denkweise lösen, durch die sie entstanden sind.

Albert Einstein - Physiker und Nobelpreisträger

8

Zusammenfassung und Ausblick

Im Rahmen der vorliegenden Arbeit wurde das Ziel verfolgt, mit der MMS ein Rahmenwerk zu schaffen, das adaptive Benutzerinteraktion zwischen Benutzern und Rehabilitationsrobotern erlaubt. Dazu wurden ausgehend von den in Abschnitt 1.2 gestellten Fragen offene Probleme herausgestellt und mögliche, aufeinander aufbauende Lösungskonzepte entwickelt, deren Zusammenhänge in Abbildung 8.1 dargestellt sind.

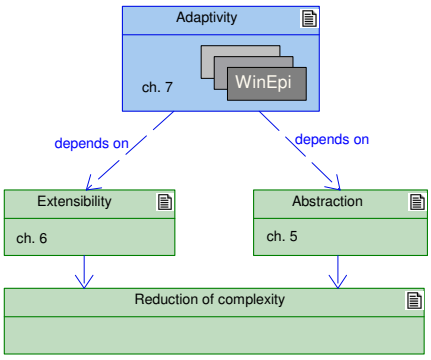


Abbildung 8.1: *Zusammenhänge zwischen den Lösungskonzepten*

Die dazu im Verlauf von Kapitel 5, Kapitel 6 und Kapitel 7 schrittweise entwickelten Anpassungen an der eingesetzten Softwarearchitektur und beschriebenen Lösungskonzepte werden im Folgenden zusammenfassend dargestellt.

Reduzierung der Architekturkomplexität

Basierend auf der Evaluation der zum Einsatz kommenden Softwarearchitektur wurde gezeigt, wie durch den Einsatz einer Spezifikationssprache für die Systemressourcen die Komplexität beim Aufbau des Skill-Netzwerkes in der reaktiven Ebene verringert werden konnte. Basierend auf dem entwickelten XML-Dialekt kann jede Serveranwendung in

Verbindung mit der XML-Schema Instanz ihre korrespondierende Spezifikation beim Anwendungsstart zunächst validieren und im Anschluss daran auswerten.

Ein weiterer Vorteil ergibt sich durch die Möglichkeit unterschiedliche Spezifikationen für mehrere Instanzen einer Serveranwendung einzusetzen, so ist es möglich, für jede Instanz spezifische Parameter festzulegen. Die Parameter sind dazu auf die definierten Datentypen abzubilden und können dann völlig frei festgelegt werden.

Eine weitere Reduktion der Komplexität konnte durch die Abbildung des in der MMS eingesetzten MVC-Entwurfsmusters auf die „Signal“ und „Slot“ Architektur von Qt erreicht werden. Die Anzahl der MMS-Anwendungskomponenten ließ sich dadurch auf eine reduzieren und dadurch sowohl das Startverhalten als auch der robuste Betrieb deutlich verbessern. Ein positiver Nebeneffekt der Reduzierung: die Entwicklung von Funktionalitäten für die MMS wurde deutlich vereinfacht, ein Zusammenhang, der später nochmals aufgegriffen wird.

Abstraktion vom Robotersystem

Die Untersuchungen zeigten, dass durch die bestehende Vorgehensweise bei der Initialisierung des Kommunikationsnetzwerkes in der reaktiven Ebene Abhängigkeiten zwischen den einzelnen Schichten der Architektur, bis hin zur MMS bestanden. Diese konnten, ebenfalls durch Einsatz der entwickelten Spezifikationssprache, aufgelöst werden. Dadurch wurde es möglich, die MMS vollständig von den Systemressourcen zu entkoppeln und eine Schnittstelle zu schaffen, die auf Basis eines neuen Lokalisierungsdienstes die dynamische Akquise von Ressourcen zur Laufzeit ermöglicht. Die MMS konnte dadurch in die Lage versetzt werden, Systemressourcen auf Basis ihrer Schnittstellen zu akquirieren, ohne den bisher zur Lokalisierung benötigten Namenskontext zu kennen.

Der neu integriere Lokalisierungsdienst bietet zudem die erweiterte Funktionalität der Überwachung des Zustandes von Systemressourcen. Dazu können Komponenten über einen Call-Back-Mechanismus Suchmuster bei dem Lokalisierungsdienst registrieren anhand derer die Statusinformationen zu den, dem Muster entsprechenden, Systemressourcen akquiriert und fortlaufend oder bei Statusänderung zur Verfügung gestellt werden. Diese Funktionalität bildete später die Grundlage für das adaptive Verhalten der MMS in Bezug auf System- oder Umgebungsänderungen.

Diese weitgehende Abstraktion führte zu einem deutlich robusteren Zugriff auf die zur Verfügung stehenden Systemressourcen und vereinfachte durch Entwicklung entsprechender, in Klassen gekapselter Funktionalitäten darüber hinaus den Verbindungsauf- und -abbau.

Erweiterbarkeit der MMS

Durch die Aufteilung der MMS-Funktionalitäten in systemspezifische und anwendungsorientierte Funktionen konnte ein Rahmenwerk geschaffen werden, das die systemspezifischen Funktionalitäten wie Methoden zur Kommunikation, zur Akquise von Systemressourcen und zum Zugriff auf System- und Spezifikationsdaten, um nur die wichtigsten zu nennen, implementiert und über spezielle Schnittstellen exportiert. Die anwendungsorientierten Funktionalitäten wurden dagegen in Module, den so genannten Erweiterungen, ausgelagert, deren robuste Integration durch den Einsatz von XML-basierten Spezifikationen erreicht werden konnte.

Durch die bereits eingangs beschriebene Abbildung des eingesetzten MVC-Entwurfsmusters auf die von Qt bereitgestellte „Signal“ und „Slot“ Architektur konnte darüber hinaus der zuvor für die Integration von Eingabegeräten eingesetzte Ereigniskanal ersetzt werden. Durch diese Vorgehensweise und die Nachrichten-basierte Spezifikation der gesamten Kommunikation zwischen Erweiterungen sowie zwischen Erweiterungen und der MMS war es möglich, einen robusteren Betrieb der Eingabegeräte zu erlauben und deren Integration durch Entwickler deutlich zu vereinfachen.

Basierend auf der Modularisierung und der Möglichkeit, Erweiterungen zur Laufzeit dynamisch zu aktivieren wurde es möglich, den kontextabhängigen Aufbau der MMS zu erlauben. Dazu können zum Beispiel die Informationen des neuen Lokalisierungsdienstes eingesetzt werden, um den Aufbau der Benutzeroberfläche dynamisch an die zur Verfügung stehenden Systemressourcen anzupassen, indem Erweiterungen nur dann und nur so lange aktiviert werden, wie die von ihnen benötigten Ressourcen zur Verfügung stehen.

Neben der Realisation aller Eingabegeräte und der grafischen Erweiterungen für die dialogbasierte Interaktion mit dem Benutzer ermöglichte die generische Schnittstelle zu den Ressourcen in den unteren Schichten der Softwarearchitektur auch die Entwicklung von Erweiterungen zur Unterstützung der Entwickler. In diesem Zusammenhang wurde exemplarisch eine Erweiterung zur Anzeige aller Systemmeldungen und eine Erweiterung zur Darstellung des 3D-Umgebungsmodells implementiert.

Adaptivität der MMS

Die Adaptivität der MMS basiert grundlegend auf den Entwicklungen zur Abstraktion und Erweiterbarkeit. Nur durch die konsequente Modularisierung und den entkoppelten Zugriff auf Systemressourcen wird es möglich, den gesamten Aufbau der Benutzeroberfläche an den System- und Benutzerzustand dynamisch anzupassen. Dazu erhielten die Erweiterungen die Möglichkeit des Zugriffs auf System- und Planungsdaten sowie die Möglichkeit zur Erstellung einer beliebigen Anzahl von Datenbasen mittels derer die Realisation und Integration von KI-Algorithmen ermöglicht wurde.

Zur Vereinfachung der Implementation von Methoden zur Anpassung der MMS an den Benutzer wurde darüber hinaus die Möglichkeit geschaffen Benutzerprofile auf Basis der Kommunikation zwischen den Erweiterungen und der MMS automatisiert zu erstellen.

Wie anhand der exemplarischen Implementation des WinEpi-Algorithmus gezeigt werden konnte, können diese Daten sehr einfach für die Auswertung durch KDD-Methoden eingesetzt werden und so zum Beispiel der Analyse des Nutzungsverhaltens dienen, um auf Basis dieser Daten eine Vorhersage über zukünftig durch den Benutzer ausgeführte (Teil-) Szenarien zu treffen. Durch diese Möglichkeit kann die intelligente Auswahl oder Darstellung von Szenarien realisiert und eine individuelle Anpassung an den Benutzer erreicht werden.

Ausblick

Die im Rahmen der vorliegenden Arbeit entwickelte Abstraktionsschicht zwischen der MMS und den übrigen Schichten baut auf der im FRIEND-Projekt entwickelten Softwarearchitektur MASSiVE auf und ist somit in der derzeitigen Form auf den Einsatz in Verbindung mit dieser Architektur beschränkt. Da die Zugriffe auf Komponenten der Architektur ausschließlich über genau definierte Schnittstellen erfolgen, ist es möglich die Anpassung an andere Softwarearchitekturen durch Neuimplementation der Schnittstellen zu erreichen. Um diesen Vorgang weiter zu vereinfachen wäre die zukünftige Kapselung der Schnittstellenimplementationen in statischen oder dynamischen Bibliotheken denkbar.

Betrachtet man den exemplarisch implementierten Algorithmus zur Mustererkennung, so beschränkt sich dieser auf die Untersuchung von endlichen, zur Laufzeit der Untersuchung bereits zur Verfügung stehenden, Ereignissequenzen. Daher ist diese Methode effektiv für die offline Analyse der erstellten Datenbasen einzusetzen, scheitert aber bei der online Datenanalyse. Die Problematik besteht darin, dass die Daten vom Algorithmus mehrfach durchlaufen werden müssen und die online Anwendung auf wachsende Ereignissequenzen in jedem Durchlauf zu veränderten Häufigkeitsverhältnissen der betrachteten Daten führen würden. Das heißt, dass die Häufigkeit von bereits als häufig erkannten Episoden sinken und ebenso die Häufigkeit von als nicht häufig erkannten Episoden steigen könnte. Als eine Lösung dieser Problematik wird in [MM02, HK06] der „Lossy Counting“ Algorithmus vorgestellt, welcher zukünftig als Alternative zum WinEpi-Algorithmus eingesetzt werden könnte.

Im weiteren Verlauf der Entwicklungen sollten weitere Algorithmen als Erweiterung in der MMS entwickelt und implementiert werden, um diese in Hinblick auf ihre Einsatzfähigkeit zu überprüfen und einander gegenüber zu stellen. Besonders die Gegenüberstellung wird aufgrund der Möglichkeiten, Erweiterungen beliebig zu aktivieren, zu deaktivieren oder parallel zu betreiben, gut durch die MMS unterstützt. Als Aussichtsreich erscheint in diesem Zusammenhang die Kombination unterschiedlicher Algorithmen, so könnten basierend auf der in Abschnitt 7.4 beschriebenen Methodik zur Erstellung

eines einfachen Entscheidungsbaumes Algorithmen zum Entscheidungsbaumlernen angewandt werden. Diese Vorgehensweise würde es erlauben, nach der offline Erstellung des Entscheidungsbaumes über die WinEpi-Analyse durch einen Algorithmus wie den in [BKI08] vorgestellten, die Datenauswertung online fortzuführen.

A

Anhang

A.1 Entwicklung von Erweiterungen

Dieser Abschnitt beschreibt das Vorgehen bei der Entwicklung von neuen Erweiterungen für die MMS und kann in diesem Zusammenhang als Referenz verstanden werden. Die Beschreibung ist aufgeteilt in mehrere Schritte die in Abhängigkeit von der zu erstellenden Erweiterung durchgeführt werden müssen. Tabelle A.1 zeigt für welche Erweiterung welche Schritte erforderlich sind.

	A.1.1	A.1.2	A.1.3	A.1.4	A.1.5	A.1.6	A.1.7	A.1.8	A.1.9	A.1.10	A.1.11
Typ 1	•	•		•	•	•					•
Typ 2	•	•	•[1]	•	•		•				•
Typ 3	•	•	•[1]	•	•		•	•	•	•	•

Tabelle A.1: *Notwendige Schritte bei der Entwicklung von Erweiterungen*

Für die folgenden Beschreibungen wird als Name der zu erstellenden Erweiterung der Name „<PluginID>“ als Platzhalter verwendet. Dieser ist durch die ID der neu zu erstellenden Erweiterung zu ersetzen.

A.1.1 Schritt 1: Festlegen der Eckdaten

Bevor mit der Implementation begonnen werden kann, muss zunächst der Typ der Erweiterung festgelegt werden. Dabei ist zu betrachten, ob diese eine grafische Komponente besitzen wird (Typ 2) oder ob es sich um eine Erweiterung für ein Ein-/Ausgabegerät handelt (Typ 1). Soll mit der zu entwickelnden Erweiterung eine Benutzerinteraktion realisiert werden, so ist eine grafische Komponente notwendig. Darüber hinaus muss die

[1] Die Erstellung des grafischen Elementes mittels Qt-Designer ist nur notwendig wenn diese nicht zur Laufzeit dynamisch erzeugt wird. Das Erstellen der Qt-Ressourcen-Datei muss in jedem Fall durchgeführt werden.

Implementation der Nutzerintegration als Systemskill (Typ 3) in der entsprechenden Systemkomponente durchgeführt werden (siehe Abschnitt 5.4.3).

A.1.2 Schritt 2: Erstellen der Spezifikation

Im Anschluss an die vorbereitenden Überlegung ist die Spezifikation der Erweiterung zu erstellen. Als Name für die Spezifikationsdatei ist „<PluginID>.psl“ zu verwenden. Es ist darauf zu achten, dass die Spezifikation sowohl gegen das XML-Schema als auch das XML-Schematron validiert, sodass sichergestellt ist, dass die Spezifikation syntaktisch und semantisch korrekt ist. Die ausführliche Beschreibung des XML Schemas und des XML Schematrons ist im Abschnitt 6.3 zu finden. Zur Validierung kann entweder ein externes XML-Validierungswerkzeug verwendet werden oder die MMS. Soll die MMS verwendet werden so ist die Spezifikation lediglich in das entsprechende Spezifikationsverzeichnis zu legen und die MMS zu starten. Zu Beginn des Startvorganges werden alle gefundenen Spezifikation automatisch validiert und dabei aufgetretene Fehler mit einer Meldung quittiert.

Typ	Dateiname
Typ 1	`libIn<PluginID>Plugin.so` oder[2] `libOut<PluginID>Plugin.so`
Typ 2	`libIO<PluginID>Plugin.so`
Typ 3	`libUI<PluginID>Plugin.so`

Tabelle A.2: *Name der neuen Erweiterung in Abhängigkeit vom Typ*

Als Ausgangsbasis kann die Kopie einer bestehenden Spezifikationsdatei verwendet werden. Zunächst sind die den Vorüberlegungen entsprechenden Attribute festzulegen (siehe Abbildung 6.6 im Abschnitt 6.3.1 auf Seite 126). Besonders wichtig ist dabei die Änderung des Attributs „name“ auf den Wert „<PluginID>“ und die Anpassung des Attributs „fileName“ entsprechend der Tabelle A.2. Im Anschluss daran müssen die Konfigurationswerte sowie deren Standardwert für die Erweiterung spezifiziert werden. Die letzten Elemente beschreiben die benötigten System-Ressourcen, die Grammatik für die Sprachsteuerung[3] und die Übersetzungen. Letztere müssen nicht notwendigerweise vorab spezifiziert werden. Einfacher ist es in der Regel die Übersetzung später fertig zu stellen (siehe Abschnitt A.1.11).

A.1.3 Schritt 3: Grafisches Element erstellen

Zur Erstellung eines grafischen Elementes gibt es grundsätzlich zwei Wege. Der einfachste besteht darin das Element über den Qt-Designer zu erstellen. Als Ausgangspunkt sollte dabei wieder die Vorlage einer bestehenden Erweiterung verwendet werden. Dazu sind die

[2]Hier unterscheidet sich der Name zusätzlich darin ob es sich um ein Ein- oder Ausgabegerät handelt.
[3]Die Grammatik ist optional und wird derzeit noch nicht ausgewertet.

entsprechenden Dateien „<PluginID>.ui“ und „<PluginID>.qrc“. Die erste Datei enthält dabei die Vorlage für das grafische Element und die zweite sämtliche darin verwendeten Abbildungen beziehungsweise Piktogramme.

Nachdem die Dateien kopiert wurden ist die Qt-Designer-Vorlage zu öffnen um darin die korrekte, ebenfalls kopierte, Ressourcen-Datei zu referenzieren. Darüber hinaus ist der Objektname des Dialoges in „<PluginID>Ui“ abzuändern. Abschließend erfolgt der Entwurf des grafischen Elementes mit den Werkzeugen des Qt-Designers.

A.1.4 Schritt 4: Implementation

Auch bei der Implementation der neuen Erweiterung sollte als Vorlage das Modell einer bestehenden Erweiterung verwendet werden. Dazu ist im Modell zunächst das entsprechende Paket zu kopieren. Die Oberpakete für die unterschiedlichen Erweiterungen sind dabei folgendermaßen organisiert:

```
massive
  Design
    HMI
      HumanMachineInterface
        PluginSystem
          WidgetBased
->          SkillWidgets
->          Widgets
->        WidgetLess
```

Innerhalb des Pakets „SkillWidgets“ sind alle Erweiterungen vom Typ 3, im Paket „Widgets“ alle Erweiterungen vom Typ 2 und im Paket „WidgetLess“ alle Erweiterungen vom Typ 1 zu platzieren.

Nachdem das kopierte Paket in „<PluginID>“ umbenannt wurde, sind neben dem Namen des Klassendiagrammes die Namen der Klassen wie folgt anzupassen:

- `Q<PluginID>Plugin` - Diese Klasse repräsentiert die konkrete Fabrik zur Instanziierung der Erweiterung durch das Qt-Rahmenwerk.
- `Q<PluginID>` - Diese Klasse repräsentiert die konkrete Erweiterung (bei Erweiterungen vom Typ 1).
- `Q<PluginID>Widget` - Diese Klasse repräsentiert die konkrete Erweiterung (bei Erweiterungen vom Typ 2 oder 3).
- `Ui/<PluginID>Ui` - Diese Klasse repräsentiert die im Qt-Designer erstelle Vorlage für das grafische Element (bei Erweiterungen vom Typ 2 oder 3, die die grafischen Elemente nicht dynamisch zur Laufzeit erzeugen).

Als nächster Schritt ist in der Fabrikmethode der konkreten Fabrik `Q<PluginID>Plugin` der Name der zu instantiierenden Klasse anzupassen. Besonders wichtig ist dabei, dass dem Konstruktor der Instanz der konkreten Erweiterung der korrekte Name „<PluginID>“ als erstes Argument übergeben wird. Bei Erweiterungen vom Typ 3 ist dieser Name als Konstante im Paket „UserInteractionSkillIDs“ anzulegen. Diese Konstante ist dann sowohl in der Fabrikmethode als Konstruktorargument zu verwenden als auch später in dem Skill, da darüber zur Laufzeit die Verbindung zwischen Skill und Erweiterung hergestellt wird.

```
massive
  Design
    HMI
->    UserInteractionSkillIDs
```

Weitere Anpassung sind wie folgt durchzuführen:

- Anpassen des Konstruktors der Klasse `Q<PluginID>Widget` indem die notwendigen Signal- & Slot-Verbindungen hergestellt werden. Darüber hinaus können in diesem Schritt bereits weitere, für die konkrete Erweiterung benötigte, Signal und Slots angelegt werden. Besonders wichtig ist in diesem Zusammen der Aufruf der Methode `setupUi()` um die vom Qt-Designer erzeugte Vorlage des grafischen Elementes zu initialisieren, soweit sie denn eingesetzt wird. Diese Methode sollte direkt als erstes im Konstruktor aufgerufen werden!

- Implementation der Methode `Setup()` der Klasse `Q<PluginID>Widget`. Hier sollten alle nötigen Initialisierungen durchgeführt werden. Dazu gehören unter anderem die Initialisierung der Dialogelemente und das Übersetzten aller nutzer-sichtbaren Texte über die `Tr()`-Methode. Letzteres kann auch zunächst ausgelassen und später mit Unterstützung der MMS durchgeführt werden (siehe Abschnitt A.1.11).

A.1.5 Schritt 5: Anlegen der Modellkomponente

Wie bereits in den vorherigen Schritten bedient man sich auch beim Anlegen der Modellkomponente der Kopie einer bestehenden Erweiterung als Vorlage. Nachdem die Vorlage kopiert und der Name entsprechend Tabelle A.2 angepasst wurde, müssen noch folgende Anpassungen vorgenommen werden:

- In den Konfigurationswerten zu der neuen Komponente im Reiter „Directory“ ist der Pfad auf `../build/<NAME>` entsprechend des zuvor festgelegten Namens anzupassen.

- Im Reiter „Properties“ ist der Konfigurationswert `PRE_DEPENDENCIES` entsprechend der verwendeten Namen anzupassen.

- Die Namen der zusätzlich zur Komponenten hinzugelinkten Datein müssen angepasst werden. Das betrifft in der Regel nur Dateien die vom Qt-Metaobjekt-Kompiler verwendet werden.

- Die letzte wichtige Anpassung besteht in der Aktualisierung des Scopes der neuen Komponenten. In diesem ist das Paket der als Vorlage dienenden Erweiterung zu deaktivieren und das Paket mit den neu erstellten Klassen zu aktivieren.

Im Anschluss daran kann die Komponente der neuen Erweiterung das erste Mal generiert und kompiliert werden, um zu testen ob sich Fehler eingeschlichen haben.

A.1.6 Schritt 6: Erster Test von Ein- und Ausgabegeräten

Beim Test von Erweiterung des Typs 1 ist zunächst darauf zu achten, dass die Erweiterung erfolgreich geladen, initialisiert und angezeigt wird. Das ist bei diesen Erweiterungen am entsprechenden Piktogramm in der Statuszeile zu erkennen. Die erfolgreiche Initialisierung ist darüber hinaus in der Regel an den Ausgaben der aktuellen Konfigurationswerte in der Meldungsliste (sofern diese in der `Setup()` Methode implementiert wurden) zu erkennen.

A.1.7 Schritt 7: Erster Test von grafischen Erweiterungen

Um bereits vor der Verwendung einer neuen grafischen Erweiterung (Typ 2 oder Typ 3) die grundlegende Funktionsweise zu testen kann die zu diesem Zweck entwickelte Anwendung „UserInteractionWidgetTestApp“ verwendet werden. Sie bietet die Möglichkeit eine Erweiterung zu landen und anzuzeigen. Darüber hinaus werden alle bereits für die Übersetzung vorbereiteten Texte angezeigt (sie werden umschlossen von „Tr(<TEXT>)“) sowie die Meldungen der Erweiterung und deren Aufrufe zur Akquise von Ressourcen protokolliert.

A.1.8 Schritt 8: Anlegen eines Skills

Beim Anlegen eines Skills sollte wieder mit der Kopie eines bestehenden Skills begonnen werden. Dazu navigiert man im Modell zur Klasse `UserInteractionSkillServer_impl` und kopiert eine Methode aus der Gruppe `SequencerSkill_impls`, die als Vorlage dienen soll.

```
massive
  Design
    HMI
->    UserInteractionSkillServer
```

Im Anschluss daran ist der Name des Skills anzupassen. Hier ist der gleiche Name zu verwenden wie er auch für die Erweiterung verwendet wurde, also „<PluginID>“. Des Weiteren ist zu Anfang in der Skill-Methode der Aufruf der Methode `SetSkillID()` dahingehend anzupassen, dass er als Argument die zuvor angelegte Konstante als Argument erhält (siehe Abschnitt A.1.4).

Abschließend muss die Skill-Methode noch in die Schnittstelle des Skill-Servers aufgenommen werden. Dazu kopiert man wieder einen bereits bestehende Methode in der Schnittstellenklasse `IUserInteractionSkillServer` und benennt sie wie zuvor um.

Zu allen Skills müssen zur korrekten Verwendung in Verbindung mit der MMS noch Informationen in der MASSiVE Datenbank abgelegt werden. Diese befinden sich in der Tabelle `uiinfos` der Datenbank `massive_apk_sdb`. Die Eintragung erfolgt dabei mehrsprachig unter Angabe der Sprach-ID und der Skill- beziehungsweise Plugin-ID als Schlüssel. Die zur Verfügung stehenden Sprachen sind der Tabelle `languages` der gleichen Datenbank zu entnehmen.

A.1.9 Schritt 9: Erweiterung der Skill-Testanwendung

Bevor der neue Skill in Verbindung mit der MMS getestet werden kann muss die dazu entwickelte Testanwendung `UserInteractionSkillServerTestApp` erweitert werden. Dazu ist zunächst in der Klasse `UserInteractionSkillServerTestApp` des gleichnamigen Pakets die Methode `SLT_Test()` um einen entsprechenden Aufruf zu erweitern.

```
massive
  Design
    HMI
      UserInteractionSkillServer
->      UserInteractionSkillServerTestApp
```

Des Weiteren muss eine neue Methode mit dem Namen „<PluginID>“ erstellt werden, die den Skill-Aufruf kapselt. Abschließend ist die in der grafischen Testanwendung eingesetzte Aufzählungsliste um den neuen Skill zu erweitern. Das erfolgt im Konstruktor der Klasse `UserInteractionSkillServerTestApp`.

A.1.10 Schritt 10: Testlauf des Skills

Für einen Testlauf des Skills muss die MMS, der UserInteraction-Skill-Server und die Testanwendung gestartet werden. Werden vom Skill weitere Ressourcen benötigt, so müssen diese ebenfalls gestartet werden, oder der Skill-Server ist im Simulationsmodus zu betreiben (siehe Hilfe des Skill-Servers). Sind alle Anwendungen gestartet so kann mit der Testanwendung ein Skill per Doppelklick in die Aufzählungsliste gestartet werden. Über die MMS sind dann die Skill-Funktionalitäten zu testen.

A.1.11 Schritt 11: Internationalisierung

Nach dem erfolgreichen Test einer Erweiterung kann das Startverzeichnis der MMS auf die Existenz der Datei „missingTranslations.txt“ hin untersucht werden. Ist sie vorhanden, so hat die MMS zur Laufzeit fehlende Übersetzungen entdeckt und diese in der Datei protokolliert. Der Inhalt der Datei kann dann direkt als Vorlage verwendet werden, um in der entsprechenden Spezifikation die fehlenden Übersetzungen nachzutragen.

A.2 Tabellen

A.2.1 Spezifikationen der Systemressourcen

Dieser Abschnitt beschreibt alle zur Verfügung stehenden Systemressourcen und die Zuordnung zu den von Ihnen benötigten Ressourcen.

Sicherheitssystem

Eine zentrale Rolle im Sicherheitssystem der MASSiVE-Architektur nimmt die „Watchdog“-Komponente ein. Diese Komponente besitzt die Aufgabe einer externen Elektronik zyklisch ein Signal zu senden bevor ein zuvor festgelegtes Zeitintervall abgelaufen ist. Wird die Zeit überschritten so trennt die externe Elektronik die Versorgungsspannung des Robotersystems. Dadurch wird sichergestellt das der Roboterarm bei einem Systemabsturz nicht unkontrolliert bewegt wird.

@name	WatchdogServer			
pd:Resources	pr:Resource (1 rows)	@name `LoggingServer`	@mandatory false	@type `LoggingServer`
pd:ConfigValues	cv:ConfigValue (0 rows)	@name	@type	@value

Tabelle A.3: *Spezifikation für „WatchdogServer“-Komponente*

Ressourcen der Planungsebene

Die Ressourcen Sequenzer, Weltmodell und MVR (siehe [PFG05, FIG06]) sind Ressourcen der Planungsebene. Sie sind übergeordnete Komponenten innerhalb der MASSiVE-Architektur. Jede von Ihnen nimmt innerhalb der Architektur eine Schlüsselrolle ein und sorgt so für das korrekte Zusammenspiel der Einzelkomponenten. Daher werden diese Komponenten auch nicht über das Ressourcenmanagement angesprochen. Die notwendigen Information zum Verbindungsaufbau zu diesen Komponenten sind über das Konfigurationssystem zu erfragen. Eine Spezifikationsdatei existiert für diese Ressourcen nicht.

Ressource für Benutzerinteraktionen

Die Benutzerinteraktion gehört logisch zur MMS ist jedoch in das Skillnetzwerk der reaktiven Ebene eingebunden. Es handelt sich dabei um einen Skill-Server, über den die vom Sequenzer der Architektur initiierten Benutzerinteraktioen durchgeführt werden. Dazu steht diese Komponente in direktem Kontakt der MMS.

@name	UserInteractionSkillServer			

pd:Resources	pr:Resource
	(7 rows)
		@name
		`LoggingServer`
		`Robotarm`
		`MachineVisionSkillServer`
		`TraySkillServer`
		`ManipulatorSkillServer`
		`CalculationSkillServer`
		`WorldModel`
			@mandatory
			false
			true
			true
			true
			true
			true
			true
				@type
				`LoggingServer`
				`DataServer`
				`SkillServer`
				`SkillServer`
				`SkillServer`
				`SkillServer`
				`WorldModelServer`

pd:ConfigValues	cv:ConfigValue (0 rows)	@name	@type	@value

Tabelle A.4: *Spezifikation für „UserInteractionSkillServer“-Komponente*

Funktionale Gruppe für die Manipulation

Alle Systemressourcen die in Verbindung mit der Bewegungsplanung in der MASSiVE-Architektur stehen befinden sich in dieser funktionalen Gruppe. Die Hauptkomponente ist der Manipulator Skill-Server. Dieser implementiert die gesamte Bewegungsplanung des 7-Gelenk Roboterarmes und kommt bei jeder Verwendung des Roboterarms zum Einsatz.

@name	ManipulatorSkillServer			

pd:Resources	pr:Resource
	(9 rows)
		@name
		`LoggingServer`
		`Robotarm`
		`Gripper`
		`FTSensor`
		`Robotarm`
		`ManipulatorSkillServer`
		`CalculationSkillServer`
		`ManipulatorSkillServer`
		`WorldModel`
			@mandatory
			false
			true
			true
			true
			true
			true
			true
			true
			true
				@type
				`LoggingServer`
				`HardwareServer`
				`HardwareServer`
				`HardwareServer`
				`DataServer`
				`SkillServer`
				`SkillServer`
				`MirrorSkillServer`
				`WorldModelServer`

pd:ConfigValues	cv:ConfigValue (0 rows)	@name	@type	@value

Tabelle A.5: *Spezifikation für „ManipulatorSkillServer“-Komponente*

Dabei wird bei allen Bewegungen der im Arm integrierte Kraft-Momenten-Sensor eingesetzt um Kollisionen des Greifers mit der Umgebung zu detektieren und entsprechend ausweichen zu können.

@name	FTSensorHardwareServer			
pd:Resources	pr:Resource (1 rows)	@name `LoggingServer`	@mandatory false	@type `LoggingServer`
pd:ConfigValues	cv:ConfigValue (0 rows)	@name	@type	@value

Tabelle A.6: *Spezifikation für „FTSensorHardwareServer“-Komponente*

Die Bewegungsplanung wird dabei durch die MVR[4]-Server-Komponente (siehe [FIG06]) unterstützt. Diese Verwaltet ein dreidimensionales Modell der Roboterumgebung und bietet Funktionalitäten zur Abstandsberechnung. Weiterführende Informationen über die integrierte Bewegungsplanung sind [Ojd09] zu entnehmen.

@name	MVRServer			

pd:Resources	pr:Resource
	(2 rows)
		@name
		`LoggingServer`
		`WorldModel`
			@mandatory
			false
			true
				@type
				`LoggingServer`
				`WorldModelServer`

pd:ConfigValues	cv:ConfigValue (0 rows)	@name	@type	@value

Tabelle A.7: *Spezifikation für „MVRServer“-Komponente*

Zeitaufwendige Berechnungen erfolgen in der Calculation-Skill-Server Komponente. Die Implementation dieser Berechnungen in einer separaten Komponente begünstigt die Verteilung der Rechenlast auf dem Computersystem.

@name	CalculationSkillServer			

pd:Resources	pr:Resource
	(6 rows)
		@name
		`LoggingServer`
		`Robotarm`
		`CalculationSkillServer`
		`ManipulatorSkillServer`
		`CalculationSkillServer`
		`WorldModel`
			@mandatory
			false
			true
			true
			true
			true
			true
				@type
				`LoggingServer`
				`DataServer`
				`SkillServer`
				`SkillServer`
				`MirrorSkillServer`
				`WorldModelServer`

pd:ConfigValues	cv:ConfigValue (0 rows)	@name	@type	@value

Tabelle A.8: *Spezifikation für „CalculationSkillServer“-Komponente*

[4] **M**odeled **V**irtual **R**eality

Für die verschiedenen bisher aufgebauten Demonstratoren (FRIEND 1, FRIEND 2 und FRIEND 3) wurden unterschiedliche Generationen von Robotersystemen eingesetzt. Die folgenden Ressourcen dienen der Ansteuerung dieser Robotersysteme. Die CORBA-Schnittstellen dieser Ressourcen sind identisch so dass die Ressourcen gegeneinander austauschbar sind.

@name	RobotarmHardwareServerLWA3			
pd:Resources	pr:Resource (1 rows)	@name `LoggingServer`	@mandatory false	@type `LoggingServer`
pd:ConfigValues	cv:ConfigValue (0 rows)	@name	@type	@value

Tabelle A.9: *Spezifikation für „RobotarmHardwareServerLWA3“-Komponente*

@name	RobotarmHardwareServerRobIK			
pd:Resources	pr:Resource (1 rows)	@name `Logging`	@mandatory false	@type `LoggingServer`
pd:ConfigValues	cv:ConfigValue (0 rows)	@name	@type	@value

Tabelle A.10: *Spezifikation für „RobotarmHardwareServerRobIK“-Komponente*

@name	RobotarmHardwareServer			
pd:Resources	pr:Resource (1 rows)	@name `LoggingServer`	@mandatory false	@type `LoggingServer`
pd:ConfigValues	cv:ConfigValue (0 rows)	@name	@type	@value

Tabelle A.11: *Spezifikation für „RobotarmHardwareServer“-Komponente*

Darüber hinaus bietet die Manipulator Skill-Server Ressource Funktionalitäten zur Steuerung von Greifersystemen. Die folgenden Ressourcen dienen der Integration dieser Greifersysteme in die MASSiVE-Architektur. Alle diese Ressourcen besitzen ebenso wie die Robotersysteme die gleiche CORBA-Schnittstelle und sind gegeneinander austauschbar.

@name	GripperHardwareServerCAN			
pd:Resources	pr:Resource (1 rows)	@name `LoggingServer`	@mandatory false	@type `LoggingServer`
pd:ConfigValues	cv:ConfigValue (0 rows)	@name	@type	@value

Tabelle A.12: *Spezifikation für „GripperHardwareServerCAN“-Komponente*

@name	GripperHardwareServerPG70			
pd:Resources	pr:Resource (1 rows)	@name `LoggingServer`	@mandatory false	@type `LoggingServer`
pd:ConfigValues	cv:ConfigValue (0 rows)	@name	@type	@value

Tabelle A.13: *Spezifikation für „GripperHardwareServerPG70“-Komponente*

@name	GripperHardwareServer			
pd:Resources	pr:Resource (1 rows)	@name `LoggingServer`	@mandatory false	@type `LoggingServer`
pd:ConfigValues	cv:ConfigValue (0 rows)	@name	@type	@value

Tabelle A.14: *Spezifikation für „GripperHardwareServer“-Komponente*

Die folgenden Ressourcen wurden zu Testzwecken entwickelt und dienen der Simulation des Teilsystems aus Roboter und Greifer.

@name	RobotarmHardwareServerSimulator			
pd:Resources	pr:Resource (1 rows)	@name `Logging`	@mandatory false	@type `LoggingServer`
pd:ConfigValues	cv:ConfigValue (0 rows)	@name	@type	@value

Tabelle A.15: *Spezifikation für „RobotarmHardwareServerSimulator“-Komponente*

@name	GripperHardwareServerSimulator			
pd:Resources	pr:Resource (1 rows)	@name `LoggingServer`	@mandatory false	@type `LoggingServer`
pd:ConfigValues	cv:ConfigValue (0 rows)	@name	@type	@value

Tabelle A.16: *Spezifikation für „GripperHardwareServerSimulator“-Komponente*

Funktionale Gruppe für die Bildverarbeitung

Eine Kernkomponente der MASSiVE-Architektur ist die digitale Bildverarbeitung. Diese ist in Form des ROVIS[5]-Rahmenwerks in der Machine-Vision-Skill-Server Komponente

[5] **RO**bust machine **VI**sion for **S**ervice robotics

implementiert. Die Ressource ist in der Lage über unterschiedliche Kamerasysteme die Umgebung zu erfassen und Objekte zu lokalisieren.

@name	MachineVisionSkillServer			

pd:Resources	pr:Resource
	(4 rows)
		@name
		`LoggingServer`
		`Bumblebee`
		`PTH`
		`WorldModel`
			@mandatory
			false
			true
			true
			true
				@type
				`LoggingServer`
				`HardwareServer`
				`HardwareServer`
				`WorldModelServer`

pd:ConfigValues	cv:ConfigValue (0 rows)	@name	@type	@value

Tabelle A.17: *Spezifikation für „MachineVisionSkillServer“-Komponente*

Das Stereokamerasystem „Bumblebee“ ist das Hauptkamerasystem der FRIEND 2 und FRIEND 3 Demonstratoren. Es ist über eine PTH[6]-Einheit in einer Position über dem Benutzer an den Systemen montiert.

@name	BumblebeeCameraHardwareServer			
pd:Resources	pr:Resource (1 rows)	@name `LoggingServer`	@mandatory false	@type `LoggingServer`
pd:ConfigValues	cv:ConfigValue (0 rows)	@name	@type	@value

Tabelle A.18: *Spezifikation für „BumblebeeCameraHardwareServer“-Komponente*

Für die verschiedenen Demonstratoren kommen unterschiedliche PTH-Module zum Einsatz die über verschiedene Nullpunkte verfügen. Zur Kompensation dieser Unterschiede wurde in der Spezifikation der PTH-Module das erste Mal die Konfigurationsmöglichkeit durch die Spezifikationen eingesetzt. Zu sehen ist nachfolgend exemplarisch die Spezifikationen für das PTH-Modul des FRIEND-Demonstrators „Nemo“.

@name	PTHHardwareServer			
pd:Resources	pr:Resource (1 rows)	@name `Logging`	@mandatory false	@type `LoggingServer`

pd:ConfigValues	cv:ConfigValue
	(3 rows)
		@name
		`ROTATION_OFFSET_PAN`
		`ROTATION_OFFSET_TILT`
		`PTHS_INIT_STRING`
			@type
			double
			double
			string
				@value
				`0.0`
				`0.0`
				`ESD:1,250`

Tabelle A.19: *Spezifikation für „PTHHardwareServer“-Komponente*

[6] **P**an-**T**ilt **H**ead

Eine weitere Kamera wird derzeit eingesetzt um eine Systemkalibrierung durchzuführen. Das Ziel ist die Kompensation von Ungenauigkeiten durch die gefedert gelagerten Elemente des Rollstuhles.

@name	UEyeCameraHardwareServer			
pd:Resources	pr:Resource (1 rows)	@name `LoggingServer`	@mandatory false	@type `LoggingServer`
pd:ConfigValues	cv:ConfigValue (0 rows)	@name	@type	@value

Tabelle A.20: *Spezifikation für „UEyeCameraHardwareServer“-Komponente*

Zur Unterstützung des Prozesses der initialen Umgebungserfassung wurden unterschiedliche 3D-Kameras in das System integriert. Die Auswertung und Verwendung der damit akquirierten Aufnahmen ist jedoch derzeit noch nicht abgeschlossen.

@name	SR3100CameraHardwareServer			
pd:Resources	pr:Resource (1 rows)	@name `LoggingServer`	@mandatory false	@type `LoggingServer`
pd:ConfigValues	cv:ConfigValue (0 rows)	@name	@type	@value

Tabelle A.21: *Spezifikation für „SR3100CameraHardwareServer“-Komponente*

@name	PMDvision03CameraHardwareServer			
pd:Resources	pr:Resource (1 rows)	@name `LoggingServer`	@mandatory false	@type `LoggingServer`
pd:ConfigValues	cv:ConfigValue (0 rows)	@name	@type	@value

Tabelle A.22: *Spezifikation für „PMDvision03CameraHardwareServer“-Komponente*

Zur Unterstützung der Greifvorgänge ist geplant eine weitere Kamera in das Greifersystem zu integrieren, um so detaillierte Aufnahmen der zu greifenden Objekte in Kontaktnähe zu erhalten.

@name	HandCameraHardwareServer			
pd:Resources	pr:Resource (1 rows)	@name `LoggingServer`	@mandatory false	@type `LoggingServer`
pd:ConfigValues	cv:ConfigValue (0 rows)	@name	@type	@value

Tabelle A.23: *Spezifikation für „HandCameraHardwareServer“-Komponente*

Funktionale Gruppe für das intelligente Tablett

Das intelligente Tablett der FRIEND-Demonstratoren besteht aus einer taktilen Oberfläche zur Detektion der Position von abgestellten Gegenständen. Die Funktionalität der Auswertung der taktilen Oberfläche ist dabei in der Hauptkomponente dieser Gruppe, dem Tray-Skill-Server implementiert.

@name	TraySkillServer			

pd:Resources	pr:Resource
	(10 rows)
		@name
		`LoggingServer`
		`Skin`
		`Scale`
		`TraySkillServer`
		`MachineVisionSkillServer`
		`CalculationSkillServer`
		`ManipulatorSkillServer`
		`UserInteractionSkillServer`
		`TraySkillServer`
		`WorldModel`
			@mandatory
			false
			true
			true
			true
			true
			true
			true
			true
			true
			true
				@type
				`LoggingServer`
				`HardwareServer`
				`HardwareServer`
				`SkillServer`
				`SkillServer`
				`SkillServer`
				`SkillServer`
				`SkillServer`
				`MirrorSkillServer`
				`WorldModelServer`

pd:ConfigValues	cv:ConfigValue (0 rows)	@name	@type	@value

Tabelle A.24: *Spezifikation für „TraySkillServer“-Komponente*

Im Laufe der Entwicklung erfolgten Versuche mit unterschiedlichen taktilen Oberflächen. Das Ziel war es ein System zu entwickeln das auch unter stark wechselnden Umgebungslichtverhältnissen gute Ergebnisse bei der Positionserkennung liefert.

@name	SkinHardwareServer			
pd:Resources	pr:Resource (1 rows)	@name `LoggingServer`	@mandatory false	@type `LoggingServer`
pd:ConfigValues	cv:ConfigValue (0 rows)	@name	@type	@value

Tabelle A.25: *Spezifikation für „SkinHardwareServer“-Komponente*

@name	PESkinHardwareServer			
pd:Resources	pr:Resource (1 rows)	@name `LoggingServer`	@mandatory false	@type `LoggingServer`
pd:ConfigValues	cv:ConfigValue (0 rows)	@name	@type	@value

Tabelle A.26: *Spezifikation für „PESkinHardwareServer“-Komponente*

@name	UKIRSkinHardwareServer			
pd:Resources	pr:Resource (1 rows)	@name `LoggingServer`	@mandatory false	@type `LoggingServer`

pd:ConfigValues	cv:ConfigValue
	(2 rows)
		@name
		`ACTIVE_MODE`
		`PASSIVE_MODE`
			@type
			bool
			bool
				@value
				`true`
				`true`

Tabelle A.27: *Spezifikation für „UKIRSkinHardwareServer“-Komponente*

@name	IRSkinHardwareServer			
pd:Resources	pr:Resource (1 rows)	@name `LoggingServer`	@mandatory false	@type `LoggingServer`
pd:ConfigValues	cv:ConfigValue (0 rows)	@name	@type	@value

Tabelle A.28: *Spezifikation für „IRSkinHardwareServer“-Komponente*

Bei dem bereits nicht mehr im Einsatz befindlichen FRIEND 1 Demonstrator war zusätzlich eine elektronische Waage integriert.

@name	ScaleHardwareServer			
pd:Resources	pr:Resource (1 rows)	@name `LoggingServer`	@mandatory false	@type `LoggingServer`
pd:ConfigValues	cv:ConfigValue (0 rows)	@name	@type	@value

Tabelle A.29: *Spezifikation für „ScaleHardwareServer“-Komponente*

Funktionale Gruppe für die Rollstuhlsteuerung

Sämtliche Funktionalitäten zur Steuerung der eingesetzten Elektrorollstühle vom Typ „Nemo vertical“ sind in der Wheelchair-Skill-Server Komponente realisiert.

@name	WheelchairSkillServer			

pd:Resources	pr:Resource
	(4 rows)
		@name
		`LoggingServer`
		`WheelchairSkillServer`
		`MeyraSerialHardwareServer`
		`MeyraBusHardwareServer`
			@mandatory
			false
			true
			true
			false
				@type
				`LoggingServer`
				`SkillServer`
				`HardwareServer`
				`HardwareServer`

pd:ConfigValues	cv:ConfigValue (0 rows)	@name	@type	@value

Tabelle A.30: *Spezifikation für „WheelchairSkillServer“-Komponente*

Die folgende Ressource dient dabei dem Zugriff auf die Bedienfunktionen des Rollstuhls über eine zur Verfügung gestellte serielle Schnittstelle. Die vom Bedienteil zur Verfügung gestellten Funktionen werden dabei vollständig auf die Ressource abgebildet. Darunter sind Funktionalitäten zur Abfrage der aktuellen Auslenkung des Joysticks, zur Umschaltung zwischen Mauscursor-Steuerung (Deaktivierung der Rollstuhlantriebselektronik) und der Rollstuhlsteuerung, zum Zugriff auf die Verstellfunktionen des Rollstuhls sowie der Zugriff auf die Sicherheitsfunktionen wie Licht und Hupe.

@name	MeyraSerialHardwareServer			
pd:Resources	pr:Resource (1 rows)	@name `Logging`	@mandatory false	@type `LoggingServer`
pd:ConfigValues	cv:ConfigValue (0 rows)	@name	@type	@value

Tabelle A.31: *Spezifikation für „MeyraSerialHardwareServer“-Komponente*

Die folgende Ressource liefert Zugriff auf den CAN[7]-Bus des Meyra Rollstuhls. Darüber kann neben dem Ladezustand der Rollstuhlbatterien auch die aktuelle Auslenkung des Joysticks ermittelt werden. Eine Steuerung, also schreibender Zugriff auf den Bus, ist über diese Ressource nicht möglich.

@name	MeyraBusHardwareServer			
pd:Resources	pr:Resource (1 rows)	@name `LoggingServer`	@mandatory false	@type `LoggingServer`
pd:ConfigValues	cv:ConfigValue (0 rows)	@name	@type	@value

Tabelle A.32: *Spezifikation für „MeyraBusHardwareServer“-Komponente*

Funktionale Gruppe für die Umgebungssteuerung/-Sensorik

In dieser funktionalen Gruppe sind alle Ressourcen zusammengefasst die in Verbindung mit der intelligenten Umgebung stehen. Hauptsächlich sind das die Elemente der für den Test von Systemszenarien eingesetzten Küche mit fern-steuerbarer Mikrowelle und fern-steuerbaren Türöffnern. Die implementierten Funktionalitäten werden dabei von den entsprechenden Skill-Servern implementiert.

[7] **C**ontroller-**A**rea **N**etwork

@name	DoorOpenerSkillServer			

pd:Resources	pr:Resource
	(5 rows)
		@name
		`LoggingServer`
		`IRController`
		`IRDoorOpener`
		`DoorOpenerSkillServer`
		`WorldModel`
			@mandatory
			false
			true
			true
			true
			true
				@type
				`LoggingServer`
				`HardwareServer`
				`HardwareServer`
				`SkillServer`
				`WorldModelServer`

pd:ConfigValues	cv:ConfigValue (0 rows)	@name	@type	@value

Tabelle A.33: *Spezifikation für „DoorOpenerSkillServer“-Komponente*

@name	MicrowaveSkillServer			

pd:Resources	pr:Resource
	(5 rows)
		@name
		`LoggingServer`
		`IRController`
		`IRMicrowave`
		`MicrowaveSkillServer`
		`WorldModel`
			@mandatory
			false
			true
			true
			true
			true
				@type
				`LoggingServer`
				`HardwareServer`
				`HardwareServer`
				`SkillServer`
				`WorldModelServer`

pd:ConfigValues	cv:ConfigValue (0 rows)	@name	@type	@value

Tabelle A.34: *Spezifikation für „MicrowaveSkillServer“-Komponente*

Die Anbindung an die Elektronik erfolgt in beiden Fällen über ein Infrarotsystem, welches über den IR-Controller-Hardware-Server angesprochen wird.

@name	IRControllerHardwareServer			
pd:Resources	pr:Resource (1 rows)	@name `LoggingServer`	@mandatory false	@type `LoggingServer`
pd:ConfigValues	cv:ConfigValue (0 rows)	@name	@type	@value

Tabelle A.35: *Spezifikation für „IRControllerHardwareServer“-Komponente*

Die Ansteuerung der eingesetzten Elektronik zum Betätigen von Lichtschaltern erfolgt durch die folgende Ressource.

@name	IRSwitchHardwareServer			
pd:Resources	pr:Resource (1 rows)	@name `LoggingServer`	@mandatory false	@type `LoggingServer`
pd:ConfigValues	cv:ConfigValue (0 rows)	@name	@type	@value

Tabelle A.36: *Spezifikation für „IRSwitchHardwareServer“-Komponente*

Der Türöffner kann in unterschiedliche Türen integriert werden. Im Rahmen des Projektes wurde er zum Beispiel für eine Kühlschranktür eingesetzt.

@name	IRDoorOpenerHardwareServer			
pd:Resources	pr:Resource (1 rows)	@name `LoggingServer`	@mandatory false	@type `LoggingServer`
pd:ConfigValues	cv:ConfigValue (0 rows)	@name	@type	@value

Tabelle A.37: *Spezifikation für „IRDoorOpenerHardwareServer“-Komponente*

Genau wie die Ressource zum Öffnen von Türen nutzt auch die Ressource zum Steuern einer Mikrowelle Infrarot-Datenübertragung.

@name	IRMicrowaveHardwareServer			
pd:Resources	pr:Resource (1 rows)	@name `LoggingServer`	@mandatory false	@type `LoggingServer`
pd:ConfigValues	cv:ConfigValue (0 rows)	@name	@type	@value

Tabelle A.38: *Spezifikation für „IRMicrowaveHardwareServer“-Komponente*

Im Rahmen eines zweijährigen Projektes erfolgte die Untersuchung in wie fern die MASSiVE-Architektur für die Steuerung einer Anlage im industriellen Umfeld eingesetzt werden kann. Dabei kam ein optischer Profilsensor zum Einsatz. Die Integration erfolgte über den entsprechenden Skill-Server.

@name	ProfileSensorHardwareServer			
pd:Resources	pr:Resource (1 rows)	@name `Logging`	@mandatory false	@type `LoggingServer`
pd:ConfigValues	cv:ConfigValue (0 rows)	@name	@type	@value

Tabelle A.39: *Spezifikation für „ProfileSensorHardwareServer“-Komponente*

Die Ansteuerung des Sensorkopfes übernahm dabei der Profile-Sensor-Hardware-Server, welcher mittels TCP/IP-Verbindung mit dem Sensor kommuniziert.

@name	ProfileSensorSkillServer			

pd:Resources	pr:Resource
	(2 rows)
		@name
		`LoggingServer`
		`ProfileSensorSkillServer`
			@mandatory
			false
			true
				@type
				`LoggingServer`
				`SkillServer`

pd:ConfigValues	cv:ConfigValue (0 rows)	@name	@type	@value

Tabelle A.40: *Spezifikation für „ProfileSensorSkillServer“-Komponente*

Im Rahmen von [Fre09] wurde zu Testzwecken eine weitere Ressource integriert. Diese erlaubt den Zugriff auf eine externe USB Wetterstation. Sie bietet derzeit nur die Möglichkeit die aktuelle Temperatur und die aktuelle relative Luftfeuchte auszulesen.

@name	TempSensorHardwareServer			
pd:Resources	pr:Resource (1 rows)	@name `Logging`	@mandatory false	@type `LoggingServer`
pd:ConfigValues	cv:ConfigValue (0 rows)	@name	@type	@value

Tabelle A.41: *Spezifikation für „TempSensorHardwareServer“-Komponente*

Funktionale Gruppe für die mobile Plattformen

In weiteren Projekten erfolgte die Untersuchung der Einsatzmöglichkeiten von mobilen Plattformen. Dazu wurde eine Komponente entwickelt die Kartendaten für die autonome Navigation für mobile Plattformen bietet.

@name	MapServer			

pd:Resources	pr:Resource
	(2 rows)
		@name
		`LoggingServer`
		`WorldModel`
			@mandatory
			false
			true
				@type
				`LoggingServer`
				`WorldModelServer`

pd:ConfigValues	cv:ConfigValue (0 rows)	@name	@type	@value

Tabelle A.42: *Spezifikation für „MapServer“-Komponente*

Des Weiteren erfolgte die Implementation der Steuerungsfunktionalität und der Navigationsalgorithmen innerhalb entsprechenden Skill-Server.

@name	MobilePlatformSkillServer			

pd:Resources	pr:Resource
	(3 rows)
		@name
		`LoggingServer`
		`MP470`
		`WorldModel`
			@mandatory
			false
			true
			true
				@type
				`LoggingServer`
				`HardwareServer`
				`WorldModelServer`

pd:ConfigValues	cv:ConfigValue (0 rows)	@name	@type	@value

Tabelle A.43: *Spezifikation für „MobilePlatformSkillServer“-Komponente*

@name	PltfNavigationSkillServer			

pd:Resources	pr:Resource
	(2 rows)
		@name
		`LoggingServer`
		`PltfNavigationSkillServer`
			@mandatory
			false
			true
				@type
				`LoggingServer`
				`SkillServer`

pd:ConfigValues	cv:ConfigValue (0 rows)	@name	@type	@value

Tabelle A.44: *Spezifikation für „PltfNavigationSkillServer“-Komponente*

Die Ansteuerung der Hardware erfolgt durch die Hardware-Server.

@name	MP470HardwareServer			
pd:Resources	pr:Resource (1 rows)	@name `Logging`	@mandatory false	@type `LoggingServer`
pd:ConfigValues	cv:ConfigValue (0 rows)	@name	@type	@value

Tabelle A.45: *Spezifikation für „MP470HardwareServer“-Komponente*

@name	RWMobilePlatformHardwareServer			
pd:Resources	pr:Resource (1 rows)	@name `LoggingServer`	@mandatory false	@type `LoggingServer`
pd:ConfigValues	cv:ConfigValue (0 rows)	@name	@type	@value

Tabelle A.46: *Spezifikation für „RWMobilePlatformHardwareServer“-Komponente*

A.2.2 Konfigurationswerte der Systemressourcen

Da die Möglichkeit zur Festlegung von Konfigurationsdaten für Ressourcen erst im Rahmen dieser Arbeit realisiert wurde, existieren nach derzeitigem Stand nur für zwei Ressourcen Konfigurationswerte. Es handelt sich dabei um die Ressource zur Steuerung von PTH-Modulen und die Ressource für die Auswertung der taktilen Haut des intelligenten Tabletts.

Die PTH-Module werden über einen CAN-Bus am System betrieben und nutzen je nach System unterschiedliche Bus-Adressen. Des Weiteren besitzen die Module unterschiedliche Null-Positionen so dass es notwendig ist die für Berechnungen genutzten Positionsdaten um die Abweichung zu korrigieren. Die dazu spezifizierten Konfigurationswerte sind Tabelle A.47

Ressource	**PTHHardwareServer**
`ROTATION_OFFSET_PAN`	`[double]` Dieses Konfigurationspaar dient der Festlegung des Rotationsoffsets für das PTH-Modul in Schwenkrichtung („Pan“). Der Offset wird vor der Verarbeitung mit den Rotationsdaten des Moduls verrechnet.
`ROTATION_OFFSET_TILT`	`[double]` Dieses Konfigurationspaar dient der Festlegung des Rotationsoffsets für das PTH-Modul in Neigungsrichtung („Tilt“). Der Offset wird vor der Verarbeitung mit den Rotationsdaten des Moduls verrechnet.
`PTHS_INIT_STRING`	`[string]` Adresse zur Spezifikation des korrekten CAN-Busses.

Tabelle A.47: *Konfigurationswerte für „PTHHardwareServer“-Komponente*

Die taktile Oberfläche des intelligenten Tabletts besteht aus einer Vielzahl von Infrarot-Empfangs und -Sendedioden, die im Matrixraster angeordnet sind. Werden Objekte auf der Oberfläche abgestellt, so kann über deren Reflexion (im aktiven Modus) oder Schattenwurf (im passiven Modus) die Position und Form ermittelt werden.

Ressource	**UKIRSkinHardwareServer**
`ACTIVE_MODE`	`[bool]` Dieses Konfigurationspaar schaltet die Ressource in den aktiven Modus. Das heißt, das intelligente Tablett emittiert Infrarotlicht und detektiert den reflektierten Anteil des Lichts um die Position der abgestellten Objekte zu ermitteln.
`PASSIVE_MODE`	`[bool]` Über dieses Konfigurationspaar wird die Ressource im passiven Modus betrieben. Dabei erfolgt lediglich die Auswertung des Infrarotanteils im einfallenden Umgebungslicht zur Ermittelung von Objektpositionen.

Tabelle A.48: *Konfigurationswerte für „UKIRSkinHardwareServer“-Komponente*

A.2.3 Spezifikationen der entwickelten Erweiterungen

Die folgenden Abschnitte erläutern die im Rahmen dieser Arbeit entwickelten Erweiterungen und beschreiben deren Spezifikation in den angegebenen Tabellen. Dabei ist zu beachten, dass das optionale Attribut „name“ der zugeordneten Ressourcen nicht mit angegeben wurde.

Ein- und Ausgabe-Erweiterungen

Die erste Gruppe beinhaltet Erweiterungen, die die Ein-/Ausgabe Funktionalität der MMS erweitern (Erweiterungen vom Typ 1, siehe Abschnitt 6.2.1). Diese besitzen keine grafische Komponente und werden lediglich mittels eines kleinen Piktogrammes im Statusbereich der MMS angezeigt. Bei diesem Typ entscheidet das Standard-Konfigurationspaar `DEFAULT` darüber, ob die Erweiterung automatisch gestartet wird oder nicht.

Automatische Sprachverarbeitung Eine besonders interessante Alternative zur Steuerung von technischen Systemen stellt die Spracheingabe dar. Diese besitzt einige Vorteile gegenüber Standardeingabemedien, insbesondere bei Nutzern mit Behinderung. Durch diese Erweiterung wird Spracheingabefunktionalität für die MMS der MASSiVE Architektur realisiert. Dabei kommt die sprecherunabhängige Sprachverarbeitungssoftware (ASR) der Fa. Loquendo [Loq10a] zum Einsatz. Neben großen Vokabularien unterstützt sie eine vielzahl von Sprachen und ist robust gegenüber Umgebungsgeräuschen. Ein weiterer Vorteil, der darüber hinaus die einfache Integration begünstigt, besteht in der Verfügbarkeit für unterschiedliche Plattformen und dem ebenfalls zur Verfügung stehendem SDK[8] für die Sprache C++.

Da die Sprachverarbeitungslösung (in der verwendeten Version) keine integrierte Audioaufnahmemöglichkeit bietet, wird zusätzlich die ALSA[9]-Bibliothek verwendet, die diese Lücke schließt. Tabelle A.49 zeigt die festgelegte Spezifikation für diese Erweiterung.

Brain-Computer-Interface Diese Erweiterung integriert ein SSVEP[10]-basiertes Brain-Computer-Interface (BCI) in die MMS. Das eingesetzte SSVEP BCI ermöglicht in der Anwendung als Eingabegerät bis zu sechs Freiheitsgrade, welche in unterschiedlichen Einsatzmodi verwendet werden. Die Konfiguration dieser unterschiedlichen Modi erfolgt über die Konfigurationsschnittstelle zur MMS. Tabelle A.50 zeigt die Spezifikation der Erweiterung.

[8] **S**oftware **D**evelopment **K**it
[9] **A**dvanced **L**inux **S**ound **A**rchitecture
[10] **S**teady **S**tate **V**isually **E**voked **P**otential

@name	AutomaticSpeechRecognition			
@fileName	libInAutomaticSpeechRecognitionPlugin.so			
@type	WidgetLess			
@dynamicLoad	false			
pd:ConfigValues	cv:ConfigValue (6 rows)	@name	@type	cv:Default
		DEFAULT	bool	false
		ASR_BUFFER_SIZE	uint	1024
		ASR_CHANNEL_NUMBER	uint	1
		ASR_CONFIDENCE_LEVEL	double	0.5
		ASR_CONFIG_FILE	QString	../../etc/HMI/ASR-TT[...]
		ASR_GRAMMAR_PATH	QString	../../etc/HMI/ASR-TT[...]
pd:Resources	pr:Resource (0 rows)	@type	@mandatory	@interface

Tabelle A.49: *Spezifikation für „AutomaticSpeechRecognition“-Erweiterung*

@name	BrainComputerInterface			
@fileName	libInBrainComputerInterfacePlugin.so			
@type	WidgetLess			
@dynamicLoad	false			
pd:ConfigValues	cv:ConfigValue (5 rows)	@name	@type	cv:Default
		DEFAULT	bool	true
		PORT	uint	1234
		TCP_MODE	bool	false
		UDP_TABBING	bool	false
		UDP_EVENT	bool	true
pd:Resources	pr:Resource (0 rows)	@type	@mandatory	@interface

Tabelle A.50: *Spezifikation für „BrainComputerInterface“-Erweiterung*

Die zuvor beschriebenen Modi unterscheiden sich zum einen in der Art der Datenübertragung und zum anderen in der verwendeten Steuerungsart. Es stehen folgende Modi zu Verfügung:

TCP Modus Bei Einsatz des TCP Modus erfolgt die Datenübertragung von der BCI Komponente ([VCF+07, Grä10]) zur Erweiterung mittels Ethernet TCP Paketen. Die Auswertung in der MMS erfolgt über den im Abschnitt 5.4.1 vorgestellten Modus der direkten Cursorsteuerung. Es werden sechs BCI-Freiheitsgrade ausgewertet deren Abbildung auf Eingabekommandos Tabelle A.51 zu entnehmen ist.

Bytesequenz	Abbildung	Funktion
0x30, 0x30	CONT_MOVE_STOP	stop der kontinuierlichen Bewegung
0x30, 0x31	CONT_MOVE_LEFT	kontinuierliche Bewegung nach links
0x30, 0x32	CONT_MOVE_RIGHT	kontinuierliche Bewegung nach rechts
0x30, 0x33	CONT_MOVE_UP	kontinuierliche Bewegung nach oben
0x30, 0x34	CONT_MOVE_DOWN	kontinuierliche Bewegung nach unten
0x30, 0x35	CLICK	Mausklick

Tabelle A.51: *Abbildung der Freiheitsgrade im TCP Modus*

Da diese Art der Verbindung durch die Verbindungsorientierte Art der Datenübertragung anfällig gegenüber kurzzeitigen Verbindungsabbrüchen ist erfolgte die Realisierung der drei UDP-basierten Steuerungsmodi.

UDP-Event Modus Bei Einsatz des UDP-Event Modus kommt die ereignisorientierte Steuerung der MMS zum Einsatz. Dafür werden nur fünf BCI-Freiheitsgrade in Kommandos umgesetzt. Durch den Einsatz einer UDP-Verbindung wird die Datenübertragen weniger störungsanfällig.

Bytesequenz	Abbildung	Funktion
0x30, 0x31	`CURSOR_LEFT`	Bewegung nach links
0x30, 0x32	`CURSOR_RIGHT`	Bewegung nach rechts
0x30, 0x33	`CURSOR_UP`	Bewegung nach oben
0x30, 0x34	`CURSOR_DOWN`	Bewegung nach unten
0x30, 0x35	`CLICK`	Mausklick

Tabelle A.52: *Abbildung der Freiheitsgrade bei ereignisorientierter Steuerung*

UDP-Tabbing Modus Der UDP-Tabbing Modus steuert die MMS indem, wie im Abschnitt 5.4.1 beschrieben wurde, der Druck auf die Tabulatortaste simuliert und so eine Änderung des Schaltflächenfukus erreicht wird. Dieser Modus benötigt lediglich drei BCI-Freiheitsgrade, deren Abbildung in Tabelle A.53 zu sehen ist.

Bytesequenz	Abbildung	Funktion
0x30, 0x31	`TAB_PREVIOUS`	Bewegung nach links
0x30, 0x32	`TAB_NEXT`	Bewegung nach rechts
0x30, 0x33	`CLICK`	Mausklick

Tabelle A.53: *Abbildung der Freiheitsgrade bei tabulatorbasierter Steuerung*

UDP-Stream Modus Der UDP-Stream Modus ist mit dem zuvor beschriebenen TCP Modus, abgesehen von der eingesetzten Ethernet-Verbindungsart, identisch. Die Abbildung der Kommandos entspricht daher den in Tabelle A.51 angegebenen.

Meyra Joysticksteuerung - „MeyraJoystick“ Diese Erweiterung ermöglicht die Integration des Steuerungsjoysticks der Rollstuhlelektronik. Es erfolgt die Auswertung der Auslenkung in x- und y-Richtung unter Verwendung des Hardwareservers MeyraSerialHardwareServer. Tabelle A.54 zeigt die Spezifikation dieser Erweiterung.

Tastatursteuerung - „KeyboardControl“ Die Tastatursteuerung wurde zunächst zu Testzwecken implementiert. In dieser Erweiterung werden Tastendrücke auf MMS Eingabekommandos statisch abgebildet. In der aktuellen Implementation erfolgt dabei die Abbildung

@name	MeyraJoystick			
@fileName	libInMeyraJoystickPlugin.so			
@type	WidgetLess			
@dynamicLoad	true			
pd:ConfigValues	cv:ConfigValue	@name	@type	cv:Default
	(4 rows)	DEFAULT	bool	false
		DEBUG_LOG	bool	false
		JOYSTICK_SLEEP_TIME	uint	100
		LEFT_RIGHT_SWAP	bool	false
pd:Resources	pr:Resource	@type	@mandatory	@interface
	(1 rows)	HardwareServer	true	IMeyraSerialHardwareServer

Tabelle A.54: *Spezifikation für „MeyraJoystick“-Erweiterung*

auf die für die direkte Cursorsteuerung notwendigen Kommandos. Abbildung A.55 zeigt die Spezifikation dieser Erweiterung.

@name	KeyboardControl			
@fileName	libInKeyboardControlPlugin.so			
@type	WidgetLess			
@dynamicLoad	false			
pd:ConfigValues	cv:ConfigValue	@name	@type	cv:Default
	(1 rows)	DEFAULT	bool	true
pd:Resources	pr:Resource	@type	@mandatory	@interface
	(0 rows)			

Tabelle A.55: *Spezifikation für „KeyboardControl“-Erweiterung*

Neben dem Anschluss einer Tastatur kann diese Erweiterung auch dafür eingesetzt werden über einen Sensoradapter spezielle an einen Benutzer angepasste Taster zu integrieren. Innerhalb der Erweiterung werden die durch den Sensoradapter erfassten Eingabekommandos dann auf von der MMS zur Verfügung gestellte Steuerungskommandos abgebildet.

Synthetisierte Sprachausgabe - „TextToSpeech“ Die Sprachausgabe dient der Unterstützung des Benutzers mittels akustischer Rückmeldungen. Alle Nachrichten, die in der MMS als Nutzer-Nachricht grafisch dargestellt werden, werden auch über Text-to-Speech Synthese (TTS[11]) und das Soundsystem des Hauptrechners ausgegeben. Zum Einsatz kommt dabei die Sprachsynthesesoftware der Fa. Loquendo [Loq10c]. Genau wie die entsprechende Spracherkennungslösung ist diese für unterschiedliche Plattformen und inklusive C++ SDK erhältlich. Tabelle A.56 zeigt die Spezifikation der Erweiterung.

[11] **T**ext **T**o **S**peech

@name	TextToSpeechLTTS7			
@fileName	libOutTextToSpeechLTTS7Plugin.so			
@type	WidgetLess			
@dynamicLoad	false			
pd:ConfigValues	cv:ConfigValue (4 rows)	@name	@type	cv:Default
		DEFAULT	bool	true
		TTS_CONFIG_FILE	QString	../../etc/HMI/ASR-TT[...]
		TTS_SAMPLE_RATE	uint	32000
		TTS_SPEAKER	QString	en-US:Kate|en-GB:Kat[...]
pd:Resources	pr:Resource (0 rows)	@type	@mandatory	@interface

Tabelle A.56: *Spezifikation für „TextToSpeechLTTS7“-Erweiterung*

Grafische Erweiterungen

Die zweite Gruppe beinhaltet Erweiterungen, die die grafische Oberfläche der MMS um Elemente erweitern (Erweiterungen vom Typ 2, siehe Abschnitt 6.2.2). Diese Erweiterungen erzeugen jeweils genau ein Element, das in die Oberfläche integriert wird. Über die Position, an der das Element in der MMS integriert werden soll, entscheiden die Angaben in der Spezifikation der Erweiterung. Dazu dienen die zuvor eingeführten und in Abbildung 6.6 aus Seite 126 dargestellten Attribute `type`, `subWidget` und `dock`. In der Festlegung der Spezifikation wurden diese Werte als optionale Angaben deklariert, da sie ausschließlich für diese Gruppe von Erweiterungen zur Laufzeit interpretiert werden. Einzige Ausnahme bildet hier die Untergruppe an Erweiterungen mittels derer die Benutzerinteraktionen realisiert werden. Diese werden über das Attribut `skill` spezifiziert und immer im Hauptbereich der MMS als Reiter dargestellt, unabhängig von den Werten der übrigen Positions-Attribute (siehe Abschnitt A.2.3). Wie bei den Ein- und Ausgabegerät-Erweiterungen entscheidet auch bei diesen Erweiterungen das Standard-Konfigurationspaar `DEFAULT` darüber, ob die Erweiterung automatisch gestartet wird oder nicht.

Panel zur Systemsteuerung - „ControlPanel“ Das Steuerungspanel enthält alle global erreichbaren Systemfunktionen, die für das FRIEND III System notwendig sind. Darunter fällt das Starten und Stoppen der Szenarien, der Benutzereingriff, die Umschaltung zwischen Rollstuhlbewegung und Maussteuerung (diese Funktionalität ist in Verbindung mit der Verwendung des Meyra-Joysticks notwendig; vgl. Abschnitt A.2.3) sowie die Möglichkeit zum Beenden der MMS-Anwendung. Tabelle A.57 zeigt die Spezifikation dieser Erweiterung.

Anzeige des Rollstuhlbatteriestatus - „BatteryState“ Diese Erweiterung wertet den Ladezustand der Batterie des Systems aus. Die Daten werden dabei vom Systembus des Rollstuhls über den dafür vorgesehenen MeyraBusHardwareServer akquiriert. Tabelle A.58 zeigt die Spezifikation der Erweiterung.

@name	ControlPanel			
@fileName	libIOControlPanelPlugin.so			
@type	WidgetBased			
@dynamicLoad	false			
pd:ConfigValues	cv:ConfigValue (1 rows)	@name	@type	cv:Default
		DEFAULT	bool	true
pd:Resources	pr:Resource (2 rows)	@type	@mandatory	@interface
		HardwareServer	false	IMeyraSerialHardwareServer
		Server	false	ISequencerServer

Tabelle A.57: *Spezifikation für „ControlPanel“-Erweiterung*

@name	BatteryState			
@fileName	libIOBatteryStatePlugin.so			
@type	Toolbar			
@dynamicLoad	true			
pd:ConfigValues	cv:ConfigValue (2 rows)	@name	@type	cv:Default
		DEFAULT	bool	false
		REFRESH_RATE	uint	10000
pd:Resources	pr:Resource (1 rows)	@type	@mandatory	@interface
		HardwareServer	true	IMeyraBusHardwareServer

Tabelle A.58: *Spezifikation für „BatteryState“-Erweiterung*

Anzeige von Systemmeldungen - „LogViewer“ Der „LogViewer“ wurde zur Vereinfachung der Systemtests als Erweiterung eingeführt. Die Darstellung erfolgt in der MMS als Reiter im Haupt-Widget Bereich. Nach dem Laden der Erweiterung öffnet diese die in der Konfiguration angegebene Textdatei. Danach werden alle neu in die Datei geschriebenen Meldungen im Widget der Erweiterung dargestellt. Zu beachten ist dabei das die bereits in der Datei enthaltenen Meldungen nicht dargestellt werden. Tabelle A.59 zeigt die Spezifikation der Erweiterung.

@name	LogViewer			
@fileName	libIOLogViewerPlugin.so			
@type	WidgetBased			
@dynamicLoad	false			
pd:ConfigValues	cv:ConfigValue (2 rows)	@name	@type	cv:Default
		DEFAULT	bool	true
		LOG_FILE	QString	../../log/MASSiVE.lo[...]
pd:Resources	pr:Resource (0 rows)	@type	@mandatory	@interface

Tabelle A.59: *Spezifikation für „LogViewer“-Erweiterung*

Anzeige des Umgebungsmodells - „MVRViewer“ Diese Erweiterung integriert den MVRViewer (siehe [FIG06]) in die MMS. Dieses Werkzeug stand zuvor lediglich als eigenständige Anwendung zur Verfügung und kann so als Reiter im Haupt-Widget Bereich angezeigt werden. Ebenfalls zur Unterstützung der Systemtests realisiert, zeigt die Erweiterung das aktuelle im MASSiVE Weltmodell gespeicherte und in der MVR aufbereitete Umgebungsmodell. Die Spezifikation zeigt Tabelle A.60.

@name	MVRViewer			
@fileName	libIOMVRViewerPlugin.so			
@type	WidgetBased			
@dynamicLoad	false			
pd:ConfigValues	cv:ConfigValue (2 rows)	@name	@type	cv:Default
		DEFAULT	bool	true
		SCANNING_TIME	int	100
pd:Resources	pr:Resource (1 rows)	@type	@mandatory	@interface
		DataServer	true	IMVRServer

Tabelle A.60: *Spezifikation für „MVRViewer“-Erweiterung*

Steuerung eines Kühlschranks - „FridgeControl“ Die Erweiterung zur Kontrolle des in der intelligenten Umgebung integrierten Kühlschranks erfolgt über diese Erweiterung. Sie bietet als Funktionalität derzeit lediglich das Öffnen und das Schließen der Kühlschranktür mittels automatischem Türöffner. Die Darstellung erfolgt als grafisches Element im Unter-Widget Bereich unter den Reitern in der MMS. Tabelle A.61 zeigt die Spezifikation der Erweiterung.

@name	FridgeControl			
@fileName	libIOFridgeControlPlugin.so			
@type	WidgetBased			
@dynamicLoad	true			
pd:ConfigValues	cv:ConfigValue (1 rows)	@name	@type	cv:Default
		DEFAULT	bool	true
pd:Resources	pr:Resource (1 rows)	@type	@mandatory	@interface
		HardwareServer	true	IIRDoorOpenerHardwareServer

Tabelle A.61: *Spezifikation für „FridgeControl“-Erweiterung*

Steuerung einer Mikrowelle - „MicrowaveOven“ Ähnlich wie die Erweiterung zur Kontrolle des Kühlschranks dient auch diese Erweiterung der Steuerung eines Elementes der intelligenten Umgebung. In diesem Fall kann mittels der Erweiterung eine speziell angepasste Mikrowelle bedient werden. Es werden Funktionalitäten zum Öffnen der Tür, Einstellen der Garzeit und Starten des Garvorganges angeboten. Die Darstellung erfolgt als grafisches Element im Unter-Widget Bereich unter den Reitern in der MMS. Tabelle A.62 zeigt die Spezifikation der Erweiterung.

@name	MicrowaveOven			
@fileName	libIOMicrowaveOvenPlugin.so			
@type	WidgetBased			
@dynamicLoad	true			
pd:ConfigValues	cv:ConfigValue (1 rows)	@name	@type	cv:Default
		DEFAULT	bool	true
pd:Resources	pr:Resource (1 rows)	@type	@mandatory	@interface
		HardwareServer	true	IIRMicrowaveHardwareServer

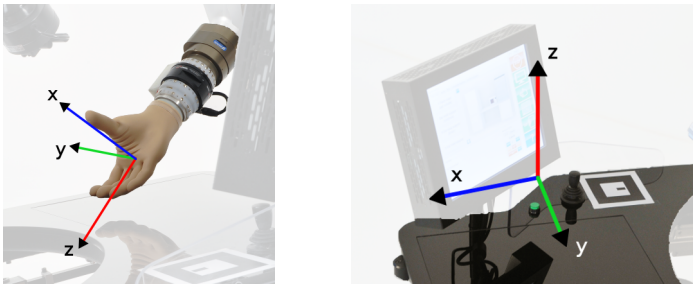
Tabelle A.62: *Spezifikation für „MicrowaveOven“-Erweiterung*

Direkte Roboterkontrolle - „RobotControl“ Die Erweiterung zur Roboterkontrolle versetzt den Benutzer in die Lage einen am System installierten Roboterarm direkt zu steuern. Diese Art der Steuerung verursacht eine hohe kognitive Belastung und sollte im Regelfall nicht notwendig werden. Nur wenn das System einmal nicht in der Lage sein sollte eine Aufgabe autonom auszuführen, weil zum Beispiel kein passendes Szenario zur Verfügung steht, so kann der Benutzer diese Aufgabe dennoch manuell ausführen. Die Erweiterung wird im Haupt-Widget Bereich der MMS als Reiter angezeigt. Die Tabelle A.63 zeigt die Spezifikation.

@name	RobotControl			
@fileName	libIORobotControlPlugin.so			
@type	WidgetBased			
@dynamicLoad	false			
pd:ConfigValues	cv:ConfigValue (4 rows)	@name	@type	cv:Default
		`DEFAULT`	bool	`false`
		`STEP_VALUES`	QString	`1#2#4`
		`ROBOT_SPEED`	uint	`10`
		`CONFIGURATIONS`	QString	`test.START#ReadyPose[...]`
pd:Resources	pr:Resource (2 rows)	@type	@mandatory	@interface
		`SkillServer`	true	`IManipulatorSkillServer`
		`DataServer`	true	`IWorldModelServer`

Tabelle A.63: *Spezifikation für „RobotControl“-Erweiterung*

Die direkte Steuerung kann in zwei unterschiedlichen Koordinatensystemen erfolgen. Zum einen besteht die Möglichkeit der Bewegung in Greiferkoordinaten, dabei werden alle Bewegungskommandos als relative Angaben bezogen auf den Ursprung des Greifers gewertet (siehe Abbildung A.1a). Als Alternative steht die Steuerung in Weltkoordinaten zur Verfügung. Bei dieser in Abbildung A.1b dargestellten Steuerungsart werden alle Kommandos als relative Bewegungsangaben bezogen auf den Mittelpunkt der Roboterbasis interpretiert.



(a) *Bewegung relativ zum Ursprung des Greifers*

(b) *Bewegung relativ zum Ursprung der Roboterbasis*

Abbildung A.1: *Mögliche Koordinatensysteme*

Es können translatorische und rotatorische Bewegungen mit unterschiedlichen Schrittweiten/Winkeln ausgeführt werden. Die Umschaltung zwischen den Modi erfolgt in einem weiteren Reiter, den die Erweiterung zusätzlich zu den Hauptelementen darstellt. Zusätzlich ist das Öffnen und Schließen des Greifers möglich.

Ein darüber hinaus eingeführter dritter Reiter ermöglicht es fest vorkonfigurierte Positionen mit dem Roboterarm anzufahren. Da diese Steuerungsmöglicheit in erster Linie zu Testzwecken realisiert wurde erfolgt die Festlegung der Positionen über die statische Systemdatenbank (referenziert mittels symbolischer Namen) von MASSiVE und ist zur Laufzeit nicht durch den Nutzer änderbar.

Szenarienauswahl - „ScenarioSelection“ Die Erweiterung für die Szenarienauswahl ermöglicht das Starten von über die MASSiVE Architektur autonom ausgeführten Szenarien (siehe Abschnitt 2.1.3). Die Szenarien werden dabei entsprechend der Strukturierung in der statischen Systemdatenbank als Gruppen dargestellt, jede Gruppe in einem separaten Reiter. Die Anzahl der Szenarien, die pro Reiter maximal angezeigt wird, kann über die Konfiguration festgelegt werden. Sollten mehr Szenarien in der Datenbank spezifiziert sein als maximal angezeigt werden sollen, so werden automatisch Schaltflächen zum Blättern freigeschaltet. Die Erweiterung wird im Haupt-Widget Bereich der MMS als Reiter angezeigt. Tabelle A.64 zeigt die Spezifikation.

@name	ScenarioSelection			
@fileName	libIOScenarioSelectionPlugin.so			
@type	WidgetBased			
@dynamicLoad	false			
pd:ConfigValues	cv:ConfigValue (3 rows)	@name	@type	cv:Default
		`DEFAULT`	bool	`true`
		`FOLLOW_ON_CLICK`	bool	`false`
		`NUM_OF_TASKS`	uint	`4`
pd:Resources	pr:Resource (1 rows)	@type	@mandatory	@interface
		`Server`	false	`ISequencerServer`

Tabelle A.64: *Spezifikation für „ScenarioSelection“-Erweiterung*

Intelligente Szenarienauswahl - „SmartScenarioSelection“ Diese Erweiterung entspricht in Bezug auf die grafische Komponente im wesentlichen der zuvor beschriebenen Erweiterung „ScenarioSelection“. Die Anzeige der Szenarien erfolgt jedoch erst nach einer Filterung: es werden nur die Szenarien angezeigt deren Auswahl im aktuellen Nutzungskontext als wahrscheinlich angesehen werden kann. Die möglichen Berechnungen dazu wurden ausführlich in Kapitel 7 beschrieben. Tabelle A.65 zeigt die Spezifikation.

Anzeige des Szenarienstatus - „TaskStatus“ Die Ausgabe des Status von aktuell ausgeführten MASSiVE-Szenarien übernimmt die Statusanzeige. Sie zeigt für jedes Szenario einen farbcodierten Prozessbalken, den Namen des Szenarios, den derzeit ausgeführten System-Skill und eine Schaltfläche um die Ausführung umgehend zu beenden. Die Farbcodierung des Prozessbalken gibt Auskunft über den aktuellen Ausführungsstatus des Szenarios. Die maximale Anzahl an darzustellenden Szenarien wird über die Konfiguration festgelegt. Den möglichen Stati sind dabei die in Tabelle A.66 angegebenen Farben zugeordnet.

@name	SmartScenarioSelection			
@fileName	libIOSmartScenarioSelectionPlugin.so			
@type	WidgetBased			
@dynamicLoad	false			
pd:ConfigValues	cv:ConfigValue (7 rows)	@name	@type	cv:Default
		DEFAULT	bool	false
		NUM_OF_TASKS	uint	4
		GENERATE_NEW_TIMESTAMP	bool	true
		MIN_FREQUENCY	double	0.4
		WIN_STEP_SIZE	uint	2
		MAX_RUNTIME	uint	600
		AUTO_RERUN	bool	true
pd:Resources	pr:Resource (1 rows)	@type	@mandatory	@interface
		Server	false	ISequencerServer

Tabelle A.65: *Spezifikation für „SmartScenarioSelection“-Erweiterung*

Ausführungsstatus	Farbcodierung
TASK_FINISHED_SUCCESS TASK_RUNNING	grün
TASK_ABORTED TASK_FINISHED_FAILURE	rot
TASK_USER_INTERACTION	gelb
TASK_NOT_RUNNING TASK_SCHEDULED TASK_QUEUED	grau

Tabelle A.66: *Zuordung der Farbcodierungen*

Die Erweiterung wird im Unter-Widget Bereich der MMS angezeigt. Tabelle A.67 zeigt die Spezifikation.

@name	TaskStatus			
@fileName	libIOTaskStatusPlugin.so			
@type	WidgetBased			
@dynamicLoad	true			
pd:ConfigValues	cv:ConfigValue (3 rows)	@name	@type	cv:Default
		DEFAULT	bool	true
		NUMBER_OF_TASKS	uint	3
		REFRESH_RATE	uint	1000
pd:Resources	pr:Resource (1 rows)	@type	@mandatory	@interface
		Server	false	ISequencerServer

Tabelle A.67: *Spezifikation für „TaskStatus“-Erweiterung*

Wetterstation - „WeatherStation“ Die Wetterstation wurde als Testerweiterung im Rahmen von [Fre09] realisiert. Sie bietet die Möglichkeit eine externe USB Wetterstation abzufragen und stellt die aktuellen Werte für Temperatur und Luftfeuchtigkeit dar. Die Erweiterung wird im Unter-Widget Bereich der MMS angezeigt. Tabelle A.68 zeigt die Spezifikation.

@name	WeatherStation			
@fileName	libIOWeatherStationPlugin.so			
@type	WidgetBased			
@dynamicLoad	true			
pd:ConfigValues	cv:ConfigValue (2 rows)	@name	@type	cv:Default
		`DEFAULT`	bool	`false`
		`REFRESH_RATE`	uint	`1`
pd:Resources	pr:Resource (1 rows)	@type	@mandatory	@interface
		`HardwareServer`	true	`ITempSensorHardwareServer`

Tabelle A.68: *Spezifikation für „WeatherStation“-Erweiterung*

Sitzeinstellungen für den Rollstuhl - „WheelchairAdjustments“ Über diese Erweiterung sollen die Sitzeinstellungen für den Rollstuhl zugänglich gemacht werden. Dabei handelt es sich überwiegend um Sitzverstellfunktionen und zwei speziell für das FRIEND-System integrierte Funktionen zur Bedienung der Schwenkarme für Roboterarm und Monitor. Die Erweiterung wird im Haupt-Widget Bereich der MMS als Reiter angezeigt. Tabelle A.69 zeigt die Spezifikation.

@name	WheelchairAdjustments			
@fileName	libIOWheelchairAdjustmentsPlugin.so			
@type	WidgetBased			
@dynamicLoad	false			
pd:ConfigValues	cv:ConfigValue (1 rows)	@name	@type	cv:Default
		`DEFAULT`	bool	`false`
pd:Resources	pr:Resource (1 rows)	@type	@mandatory	@interface
		`HardwareServer`	true	`IMeyraSerialHardwareServer`

Tabelle A.69: *Spezifikation für „WheelchairAdjustments“-Erweiterung*

Ersatz der Rollstuhlfunktionsanzeige - „WheelchairDisplay“ Diese Erweiterung soll die Anzeige- und Funktionselemente des im FRIEND-Projekt eingesetzten Rollstuhls ersetzen. Die Erweiterung enthält Elemente zur Darstellung der aktuellen Fahrgeschwindigkeit, Elemente, die den Status der Rollstuhlsicherheitsfunktionen (Beleuchtung und Hupe) wiederspiegeln, sowie Schaltflächen um letztere ein- und auszuschalten. Die Erweiterung wird im Unter-Widget Bereich dargestellt. Die Spezifikation zeigt Tabelle A.70.

Intelligente Texteingabe - „Speller“ Diese Erweiterung dient der Eingabe von Text durch den Benutzer. Zum Einsatz kommt dabei eine Bildschirmtastatur die an den in [Grä10, Kapitel 12] vorgestellten Entwurf angelehnt ist. Das Ziel ist die Möglichkeit der Eingabe von einfachen Texten ohne Tastatur. Die Anordnung der Schaltflächen wurde in Abhängigkeit von der Häufigkeit mit der bestimmte Buchstaben in deutschen Texten vorkommen gewählt, so dass vom Mittelpunkt aus gesehen immer möglichst wenig Schritte notwendig sind um eine Texteingabe erfolgreich abzuschließen. Im Unterschied zu der in [Grä10] realisierten Texteingabe wurde die Erweiterung dahingehend opti-

@name	WheelchairDisplay			
@fileName	libIOWheelchairDisplayPlugin.so			
@type	WidgetBased			
@dynamicLoad	false			
pd:ConfigValues	cv:ConfigValue (1 rows)	@name	@type	cv:Default
		DEFAULT	bool	false
pd:Resources	pr:Resource (2 rows)	@type	@mandatory	@interface
		HardwareServer	true	IMeyraBusHardwareServer
		HardwareServer	false	IMeyraSerialHardwareServer

Tabelle A.70: *Spezifikation für „WheelchairDisplay“-Erweiterung*

miert, so dass auch Texte in gemischter Groß- und Kleinschreibung eingegeben werden können.

Generell kann die Eingabe von Texten in der MMS von anderen Erweiterungen initiiert werden. Das ist zum Beispiel dann von Interesse wenn eine Benutzerinteraktion auftritt oder eine Erweiterung die Texteingabe erfordert. Diese Erweiterung wird im Haupt-Widget Bereich als Reiter dargestellt. Die Spezifikation zeigt Tabelle A.71.

@name	Speller			
@fileName	libIOSpellerPlugin.so			
@type	WidgetBased			
@dynamicLoad	false			
pd:ConfigValues	cv:ConfigValue (3 rows)	@name	@type	cv:Default
		DEFAULT	bool	true
		RESET_TO_CENTER	bool	false
		UNSHIFT_AFTER_LETTER	bool	true
pd:Resources	pr:Resource (0 rows)	@type	@mandatory	@interface

Tabelle A.71: *Spezifikation für „Speller“-Erweiterung*

Benutzerinteraktionen

Die Implementation der Benutzerinteraktionen (Erweiterungen vom Typ 3, siehe Abschnitt 6.2.3) erfolgte überwiegend in Form von Beispielimplementationen, da die konkreten Implementationen parallel zu der Entwicklung der entsprechenden autonomen Skills von den Entwicklern auszuführen sind. Ausgenommen sind hier nur drei Benutzerinteraktionen die von den Beispielimplementationen abweichen. Aus diesem Grund erfolgt in diesem Abschnitt keine ausführliche Beschreibung jeder einzelnen Benutzerinteraktion sondern nur die Vorstellung der Beispielimplementationen sowie Referenzen zu den Hintergründen der übrigen Benutzerinteraktionen. Derzeit existieren zwei unterschiedliche Beispielimplementationen. Tabelle A.92 auf Seite 225 beschreibt, welche dieser Beispielimplementationen für welche Erweiterung eingesetzt wurde.

Im Gegensatz zu den bereits beschriebenen Erweiterungen von Typ 1 und 2 besitzen die Erweiterungen für Benutzerinteraktionen kein Konfigurationspaar `DEFAULT`. Das hängt damit zusammen, dass über das Ein- und Ausblenden dieser Erweiterungen zur Laufzeit in Abhängigkeit vom aktuellen Systemstatus entschieden wird. Die Anzeige einer Benutzerinteraktion wird vom Sequenzer des Systems initiiert.

Eine weitere Abweichung zu den bereits beschriebenen Erweiterungen besteht in den notwendigen Ressourcen. Erweiterungen, die eine Benutzerinteraktion implementieren, erhalten ausschließlich Zugriff auf die Ressource `IUserInteractionSkillServer`, welche in die Reactive-Ebene der MASSiVE Architektur eingebettet ist. Sie stellt die Verbindung zwischen der Planungskomponente von MASSiVE und der entsprechenden Erweiterung her. Dieser Zusammenhang wurde bereits im Abschnitt 5.4.3 beschrieben und als Anforderung 6.5 formuliert.

Beispielimplementation: Direkte Roboterkontrolle Diese Implementation wird für alle manipulativen Benutzerinteraktionen eingesetzt. Der Aufbau der Dialoge ist nahezu identisch mit dem Aufbau der Erweiterung für die direkte Roboterkontrolle, einzig die Steuerungsmöglichkeiten für den Greifer und Möglichkeit des Anfahrens von festen Roboterpositionen fehlen. Tabelle A.72 zeigt die Spezifikation exemplarisch durch die Benutzerinteraktion „AdjustGripper“.

@name	AdjustGripper			
@fileName	libUIAdjustGripperPlugin.so			
@type	WidgetBased			
@dynamicLoad	false			
pd:ConfigValues	cv:ConfigValue	@name	@type	cv:Default
	(5 rows)	`MANIPULATE`	uint	`0`
		`STEP`	int	`1`
		`COORDINATE_SYSTEM`	uint	`0`
		`MODE`	uint	`1`
		`STEP_VALUES`	QString	`1#2#4`
pd:Resources	pr:Resource	@type	@mandatory	@interface
	(1 rows)	`SkillServer`	true	`IUserInteractionSkillServer`

Tabelle A.72: *Spezifikation für „AdjustGripper“-Erweiterung*

Beispielimplementation: Festlegung des Skillergebnisses Für alle Benutzerinteraktionen, die autonome Methoden des Bildverarbeitungsrahmenwerks ROVIS (siehe [Gri10]) ersetzen sollen, kommt eine triviale Beispielimplementation zum Einsatz die dem Benutzer lediglich alle möglichen Skill-Rückgabewerte zur Auswahl anbietet. Nachdem der Nutzer einen dieser Rückgabewerte ausgewählt hat, kehrt der Skill umgehend mit diesem Ergebnis zurück und die Erweiterung wird wieder entfernt. Tabelle A.73 zeigt die Spezifikation dieser Erweiterungen am Beispiel der Benutzerinteraktion „AskUser“.

@name	AskUser			
@fileName	libUIAskUserPlugin.so			
@type	WidgetBased			
@dynamicLoad	false			
pd:ConfigValues	cv:ConfigValue (0 rows)	@name	@type	cv:Default
pd:Resources	pr:Resource (1 rows)	@type `SkillServer`	@mandatory true	@interface `IUserInteractionSkillServer`

Tabelle A.73: *Spezifikation für „AskUser“-Erweiterung*

Systemfaktum erfragen - „AskFact“ Hintergrund dieser Erweiterung ist die initiale Erfassung der Systemumgebung durch die Planungsebene von MASSiVE. Diese als „initial monitoring“ bezeichnete Phase (siehe [Pre03]) der Systeminitialisierung wird vor der Ausführung eines autonomen Szenarios durchlaufen, um den Ist-Zustand aller Umgebungsobjekte mittels Methoden der Bildverarbeitung zu erfassen. Sollten dabei Fehler auftreten, die die Erfassung eines oder mehrerer Fakten von Umgebungsobjekten verhindern, so wird der Benutzer in diese Aufgabe mit eingebunden. Zu diesem Zweck erfolgt dann der Aufruf einer Benutzerinteraktion mittels derer der Benutzer die aktuellen Fakten angibt. Diese Erweiterung befindet sich derzeit im Entwicklungsstadium. Tabelle A.74 zeigt die derzeitige Spezifikation.

@name	AskFact			
@fileName	libUIAskFactPlugin.so			
@type	WidgetBased			
@dynamicLoad	false			
pd:ConfigValues	cv:ConfigValue (0 rows)	@name	@type	cv:Default
pd:Resources	pr:Resource (1 rows)	@type `SkillServer`	@mandatory true	@interface `IUserInteractionSkillServer`

Tabelle A.74: *Spezifikation für „AskFact“-Erweiterung*

Benutzergestützte Bildsegmentierung - „ObjectSegmentationByUser“ Diese Erweiterung findet im selben Kontext Anwendung wie die zuvor beschriebene. Hier geht es darum in der „initial monitoring“ Phase Umgebungsobjekte durch den Benutzer erkennen zu lassen. Konkret heißt das, dass der Nutzer mittels geführter Segmentierung die Objekte, die in der aktuellen Systemumgebung für die auszuführende Aufgabe von Interesse sind, segmentiert damit sie im Anschluss daran von der Bildverarbeitung erkant und lokalisiert werden können. Das genaue dabei angewandte Verfahren ist in [Lan08] beschrieben. Die Spezifikation dieser Erweiterung ist in Tabelle A.75 dargestellt.

Essensunterstützung - „UserTakesFood“ Die Essensunterstützung ist eine Erweiterung die bei der Nahrungsaufnahme des Benutzers im FRIEND-System zum Einsatz kommt. Dabei wird über die Benutzerinteraktion das Besteck gesteuert zum Teller geführt, um Nahrungsmittel aufzunehmen und dann dem Benutzer anzureichen. Da bei dieser In-

@name	ObjectSegmentationByUser			
@fileName	libUIObjectSegmentationByUserPlugin.so			
@type	WidgetBased			
@dynamicLoad	false			
pd:ConfigValues	cv:ConfigValue (0 rows)	@name	@type	cv:Default
pd:Resources	pr:Resource (1 rows)	@type `SkillServer`	@mandatory true	@interface `IUserInteractionSkillServer`

Tabelle A.75: *Spezifikation für „ObjectSegmentationByUser“-Erweiterung*

teraktion der direkte Kontakt zum Benutzer notwendig ist und die Systemsteuerung mittels Sprache oder Joystick nicht realisierbar ist, erfolgte in [Bau08] die Untersuchung der Möglichkeiten einer Kraft-Momenten-Sensor gestützten Steuerung dieses Vorganges. Diese Erweiterung ist das Ergebnis dieser Untersuchungen. Tabelle A.76 zeigt die Spezifikation dieser Erweiterung.

@name	UserTakesFood			
@fileName	libUIUserTakesFoodPlugin.so			
@type	WidgetBased			
@dynamicLoad	false			
pd:ConfigValues	cv:ConfigValue (0 rows)	@name	@type	cv:Default
pd:Resources	pr:Resource (1 rows)	@type `SkillServer`	@mandatory true	@interface `IUserInteractionSkillServer`

Tabelle A.76: *Spezifikation für „UserTakesFood“-Erweiterung*

A.2.4 Konfigurationswerte der entwickelten Erweiterungen

Die folgenden Tabellen beschreiben die zur Verfügung stehenden Konfigurationswerte der entwickelten Erweiterungen. Der Konfigurationswert `DEFAULT`, der für alle Erweiterungen des Typs 1 und 2 zur Verfügung steht, wurde dabei jedoch ausgelassen.

Erweiterungen vom Typ 1

ID	**AutomaticSpeechRecognition**
`ASR_BUFFER_SIZE`	`[uint]` Dieser Wert legt die Größe des Zwischenspeichers für die Audiaufnahme fest. Sie wird intern von der ALSA-Bibliothek in der Funktion `snd_pcm_readi(...)` weiter verarbeitet. Voreingestellt ist eine Größe von 1024 Bytes. Weiterführende Informationen sind [als10] zu entnehmen.
`ASR_CHANNEL_NUMBER`	`[uint]` Dieser Wert legt die Kanalnummer für die Audioaufnahme fest. Er wird intern von der ALSA-Bibliothek in der Funktion `snd_pcm_hw_params_set_channels(...)` weiter verarbeitet. Voreingestellt ist Kanal 1. Weiterführende Informationen sind [als10] zu entnehmen.
`ASR_CONFIDENCE_LEVEL`	`[double]` Dieser Wert wird in der Loquendo ASR API[12]-Funktion `lasrxSetConfidenceLevel(...)` weiter verarbeitet. Eine genaue Beschreibung ist dem SDK-Handbuch [Loq10b] zu entnehmen.
`ASR_CONFIG_FILE`	`[QString]` Über `ASR_CONFIG_FILE` wird die Konfigurationsdatei festgelegt, die zur Laufzeit von der ASR Engine eingelesen und ausgewertet wird. Die genaue Beschreibung des Wertes und der Aufbau der Datei ist dem SDK-Hanbuch [Loq10b] zu entnehmen.
`ASR_GRAMMAR_PATH`	`[QString]` Das Konfigurationspaar `ASR_GRAMMAR_PATH` gibt den Pfad zu den Grammatikdateien an. Hier wird keine feste Datei vorgegeben, da die Datei zur Laufzeit in Abhängigkeit von der aktuell eingestellten Sprache der MMS ausgewählt wird. Die genaue Beschreibung des Wertes und der Aufbau der Datei ist dem SDK-Hanbuch [Loq10b] zu entnehmen.

Tabelle A.77: *Konfigurationswerte für „AutomaticSpeechRecognition“-Erweiterung*

ID	**BrainComputerInterface**
PORT	[uint] Über diesen Konfigurationswert erfolgt die Festlegung des Ethernet-Ports für die Datenübertragung zwischen der BCI-Komponente und der Erweiterung. Als Voreinstellung ist die Portnummer 1234 festgelegt.
TCP_MODE	[bool] Wird dieser Wert auf wahr gesetzt, so arbeitet die Erweiterung im TCP Modus. Es wird ein TCP Port mit der angegebenen Nummer geöffnet, der auf eingehende Verbindungen wartet. Ist dieser Wert nicht wahr, so arbeitet die Erweiterung automatisch im UDP-Stream Modus. Dieses ist gleichzeitig die Voreinstellung.
UDP_TABBING	[bool] Wird dieser Wert auf wahr gesetzt, so arbeitet die Erweiterung im UDP-Tabbing Modus. Es wird ein UDP-Port mit der angegebnen Port-Nummer geöffnet der auf eingehende Verbindungen wartet. In der Voreinstellung is dieser Modus deaktiviert.
UDP_EVENT	[bool] Wird dieser Wert auf wahr gesetzt, so arbeitet die Erweiterung im UDP-Event Modus. Es wird ein UDP-Port mit der angegebnen Port-Nummer geöffnet der auf eingehende Verbindungen wartet. In der Voreinstellung is dieser Modus aktiviert.

Tabelle A.78: *Konfigurationswerte für „BrainComputerInterface“-Erweiterung*

ID	**MeyraJoystick**
LEFT_RIGHT_SWAP	[bool] Dieses Konfigurationspaar dient der Einstellung der Joystickauslenkung nach links beziehungsweise rechts, eine Funktionalität die die Nutzung von Joysticks bei denen die seitliche Richtung vertauscht ist ermöglicht.
DEBUG_LOG	[bool] Über diesen Wert besteht zu Testzwecken die Möglichkeit ausführliche Textmeldungen zur Laufzeit zu aktivieren. Da das Einschalten in Abhängigkeit von der Zykluszeit jedoch zur sehr vielen Meldungen führt, ist es standardmäßig deaktiviert.
JOYSTICK_SLEEP_TIME	[uint] Die Zykluszeit, mit der die aktuellen Joystickdaten vom entsprechenden HardwareServer akquiriert werden, kann über dieses Konfigurationspaar festgelegt werden. Zum Ende eines jeden Zyklus wird die Ausführung der Joystickabfrage für die angegebene Zeit (in Millisekunden) unterbrochen und die CPU für andere Prozesse freigegeben. Nach Ablauf der Zeit beginnt der nächste Zyklus. Voreingestellt sind $100ms$.

Tabelle A.79: *Konfigurationswerte für „MeyraJoystick“-Erweiterung*

ID	**TextToSpeechLTTS7**
`TTS_CONFIG_FILE`	`[QString]` Dieser Wert wird bei der Initialisierung der Sprachsynthese-Engine von Loquendo als Aufrufparameter für die Funktion `ttsNewSession(...)` verwendet. Weiteres dazu ist der SDK-Dokumentation [Loq10d] zu entnehmen.
`TTS_SAMPLE_RATE`	`[uint]` Dieser Wert wird bei der Initialisierung der Sprachsynthese-Engine von Loquendo als Aufrufparameter für die Funktion `ttsSetAudio(...)` verwendet. Weiteres dazu ist der SDK-Dokumentation [Loq10d] zu entnehmen.
`TTS_SPEAKER`	`[QString]` Dieser Wert wird bei der Initialisierung der Sprachsynthese-Engine von Loquendo als Aufrufparameter für die Funktion `ttsLoadPersona(...)` verwendet. Weiteres dazu ist der SDK-Dokumentation [Loq10d] zu entnehmen.

Tabelle A.80: *Konfigurationswerte für „TextToSpeechLTTS7“-Erweiterung*

Erweiterungen vom Typ 2

ID	**BatteryState**
`REFRESH_RATE`	`[uint]` Über diesen Wert erfolgt die Festlegung des Abfrageintervalls in ms. Voreingestellt ist ein Intervall von $10s$.

Tabelle A.81: *Konfigurationswerte für „BatteryState“-Erweiterung*

ID	**LogViewer**
`LOG_FILE`	`[QString]` Über diesen Wert erfolgt die Festlegung der zu öffnenden Log-Datei. Es sind sowohl absolute als auch relative Pfade erlaubt. Voreingestellt ist die standardmäßig von der MASSiVE-Architektur verwendete Log-Datei `../../log/MASSiVE.log`.

Tabelle A.82: *Konfigurationswerte für „LogViewer“-Erweiterung*

ID	**MVRViewer**
`SCANNING_TIME`	`[uint]` Über diesen Wert wird die Wiederholfrequenz festgelegt mit der das zur Darstellung verwendete OpenGL Element aktualisiert wird. Voreingestellt ist ein Wert von $100ms$, was einer Bildwiederholfrequenz von ca. $10Hz$ entspricht (bei Vernachlässigung der Zeit, die für die OpenGL Aktualisierung benötigt wird, ermöglicht $100ms$ als Zykluszeit maximal 10 Bilder pro Sekunde).

Tabelle A.83: *Konfigurationswerte für „MVRViewer“-Erweiterung*

ID	**RobotControl**
`STEP_VALUES`	`[QString]` Über diesen Wert werden die möglichen Schrittweiten für die Steuerungskommandos festgelegt. Dabei erfolgt die Angabe über eine Zeichenkette die zur Laufzeit an den Markern `#` aufgeteilt wird um die drei möglichen Werte zu erhalten. Voreingestellt ist die Schrittweitendefinition `1#2#4`, was den Werten $1cm$, $2cm$ und $3cm$ (beziehungsweise 1°, 2° oder 4°) entspricht.
`ROBOT_SPEED`	`[uint]` Mit diesem Konfigurationspaar erfolgt die Festlegung der zur Verfügung stehenden festen Roboterpositionen. Auch hier handelt es sich um ein Konfigurationspaar mit Datentyp Zeichenkette, die am Marker `#` aufgesplittet wird. Voreingestellt ist die Konfigurationsdefinition `test.START#ReadyPoseUp` was den symbolischen Namen `test.START` und `ReadyPoseUp` entspricht. Auf diese Weise können maximal 8 Konfigurationen festgelegt werden.
`CONFIGURATIONS`	`[QString]` Über dieses Ganzzahl-Konfigurationspaar wird die maximale Bewegungsgeschwindigkeit jedes einzelnen Gelenkes des Roboters in Grad pro Sekunde festgelegt. Voreingestellt ist eine Geschwindigkeit von $10° \frac{1}{s}$.

Tabelle A.84: *Konfigurationswerte für „RobotControl“-Erweiterung*

ID	**ScenarioSelection**
`FOLLOW_ON_CLICK`	`[bool]` Über dieses Konfigurationspaar kann das automatische Blättern ein- und ausgeschaltet werden. Wird es eingeschaltet, so erfolgt bei Auswahl der obersten oder untersten in dem Reiter angezeigten Szenarios automatisch das Blättern zur vorherigen oder zur nächsten Seite. In der Voreinstellung ist diese Funktionalität ausgeschaltet.
`NUM_OF_TASKS`	`[uint]` Mit diesem Wert erfolgt die Festlegung der maximal zur gleichen Zeit in einem Reiter angezeigten Szenarien. Die Voreinstellung ist 4.

Tabelle A.85: *Konfigurationswerte für „ScenarioSelection“-Erweiterung*

ID	**TaskStatus**
`NUMBER_OF_TASKS`	`[uint]` Über diesen Wert erfolgt die Festlegung der maximalen Anzahl an Szenarien, die ausgegeben werden sollen. Sollte darüber hinaus der Status von weiteren Szenarien vom System gemeldet werden, so werden diese ignoriert und nicht dargestellt.
`REFRESH_RATE`	`[uint]` Der Wert dieses Konfigurationspaares gibt die Aktualisierungszeit an mit der die Statusliste aktualisiert wird. Voreingestellt ist ein Wert von $1s$.

Tabelle A.86: *Konfigurationswerte für „TaskStatus“-Erweiterung*

ID	**SmartScenarioSelection**
NUM_OF_TASKS	[uint] Mit diesem Wert erfolgt die Festlegung der maximal zur gleichen Zeit in einem Reiter angezeigten Szenarien. Die Voreinstellung ist 4.
GENERATE_NEW_TIMESTAMP	[bool] Mit diesem Wert kann das automatische Erzeugen eines neuen Zeitstempels für die aufgezeichneten Nutzungsdaten vor der Analyse mit dem integrierten WinEpi Algorithmus ein- und ausgeschaltet werden. In der Voreinstellung ist die Generierung eingeschaltet.
MIN_FREQUENCY	[double] Mit diesem Wert erfolgt die Festlegung des Häufigkeitsschwellwertes der innerhalb des WinEpi Algorithmus ausgewertet wird (siehe Abschnitt 4.2.4 oder Kapitel 7. Die Voreinstellung ist 0.4.
WIN_STEP_SIZE	[uint] Mit diesem Wert erfolgt die Festlegung der Anzahl an Zeiteinheiten um die die Fensterbreite bei jedem WinEpi Durchlauf erhöht wird. Die Voreinstellung ist 2.
MAX_RUNTIME	[uint] Mit diesem Wert erfolgt die Festlegung der maximalen Laufzeit des WinEpi Algorithmus nach deren Ablaufen die Ausführung und Erhöhung der Fensterbreite abgebrochen wird. Die Einheit des Wertes ist Sekunden und die Voreinstellung $600s$.
AUTO_RERUN	[bool] Ist dieser Wert wahr so wird nach Abbruch der Analyse durch den WinEpi Algorithmus aufgrund des Erreichens oder Überschreitens von MAX_RUNTIME die Ausführung erneut begonnen. In der Voreinstellung ist dieser Wert wahr.

Tabelle A.87: *Konfigurationswerte für „SmartScenarioSelection“-Erweiterung*

ID	**WeatherStation**
REFRESH_RATE	[uint] Der Wert dieses Konfigurationspaares gibt die Aktualisierungszeit an, mit der die Wetterdaten aktualisiert werden. Voreingestellt ist ein Wert von $1s$.

Tabelle A.88: *Konfigurationswerte für „WeatherStation“-Erweiterung*

ID	**WebBrowser**
START_PAGE	[QString] Über dieses Konfigurationspaar vom Typ Zeichenkette wird die URL[13] der anzuzeigenden Startseite festgelegt. Voreingestellt ist http://www2.iat.uni-bremen.de/mantis.

Tabelle A.89: *Konfigurationswerte für „WebBrowser“-Erweiterung*

ID	**Speller**
RESET_TO_CENTER	[bool] Wird der Wert dieses Konfigurationspaares auf wahr gesetzt, so wird nach Eingabe eines Buchstabens automatisch der Mauscursor wieder auf den mittleren Buchstaben gesetzt. Dieses Verhalten entspricht der in [Grä10] vorgestellten „Speller“ Anwendung, ist aber in der Voreinstellung dieser Erweiterung deaktiviert.
UNSHIFT_AFTER_LETTER	[bool] Ist dieser Wert `wahr` so wird bei Verwendung von Groß- und Kleinschreibung nach der Eingabe eines Groß geschriebenen Buchstabens automatisch zurück auf Kleinschreibung geschaltet. In der Voreinstellung ist diese Funktionalität aktiviert.

Tabelle A.90: *Konfigurationswerte für „Speller“-Erweiterung*

Erweiterungen vom Typ 3

ID	**AdjustGripper**
MANIPULATE	[uint] Über dieses Konfigurationspaar wird die standardmäßig bei der Darstellung des Dialoges ausgewählte Bewegungsrichtung vorgegeben. Wird der Wert auf 0 gesetzt so ist die x-Richtung vorgegeben, bei einem Wert von 1 die y-Richtung und bei allen anderen Werten die z-Richtung. Voreingestellt ist 0, also die x-Richtung.
STEP_VALUES	[QString] Dieses Konfigurationspaar legt die zur Verfügung stehenden Schrittweiten (Schrittwinkel) fest. Dazu wird die angegebene Zeichenkette am Trennzeichen `#` aufgesplittet. Voreingestellt ist die Zeichenkette `1#2#4` was den Schrittweiten (Schrittwinkeln) $1cm$, $2cm$ und $4cm$ ($1°$, $2°$ und $4°$) entspricht.
STEP	[int] Über dieses Konfigurationspaar wird die standardmäßig bei Darstellung des Dialoges ausgewählte Schrittweite (Schrittwinkel) vorgegeben. Dabei ist der angegebene Wert als Index (Null-basiert) auf die mit dem Konfigurationspaar `STEP_VALUES` festgelegten Werte zu verstehen. In der Standardkonfiguration führt ein Wert von 1 zu einer Schrittweite (Schrittwinkel) von $2cm$ ($2°$).
COORDINATE_SYSTEM	[uint] Mit diesem Konfigurationswert erfolgt die Festlegung des zur Verwendenden Bezugskoordinatensystems. Ist dieser Wert 1 so erfolgt die Bewegung im Greiferkoordinatensystem (siehe Abbildung A.1a auf Seite 211). Für alle anderen Werte erfolgt die Bewegung im Weltkoordinatensystem (siehe Abbildung A.1b auf Seite 211). Voreingestellt ist 0 (Weltkoordinatensystem).
MODE	[uint] Dises Konfigurationspaar legt fest, ob eine translatorische oder rotatorische Beweung ausgeführt werden soll. Ein Wert von 0 führt zu rotatorischen Bewegungen. Alle anderen Werte führen zu tranlatorischen Bewegungen. Voreingestellt ist ein Wert von 1 (translatorische Bewegung).

Tabelle A.91: *Konfigurationswerte für die Beispielimplementation „Direkte Roboterkontrolle“*

A.2.5 Beispielimplementationen der Benutzerinteraktionen

Beispielimplementation	Benutzerinteraktion
Direkte Robotersteuerung	AdjustGripper AdjustObjectRelToObject AdjustToObject MoveObjectIn MoveObjectOut MoveObjectToFreeLocation MoveToFreeLocation PullDoorToOpen PullHandle PushDoorToClose PushHandle PushToLoc VisualInspection
Rückgabewert	AskUser ConfirmOpenGripper ConfirmOrFinishBookScan DetermineCookingInfo DetermineDepartLocation DetermineGrippedObjectBySCam DetermineInsertLocation DetermineObjectBySCam DetermineObjectDepartLocation HSGet2DPositionByUser SearchLocationOnPlatformByUser TestKeypad UserDrinks
SPEZIAL	AskFact Determine2DPositionByUser ObjectSegmentationByUser ThumbThroughBook UserTakesFood

Tabelle A.92: *Verwendete Beispielimplementationen der Benutzerinteraktionen*

Literaturverzeichnis

[ACC06] Aipperspach, R. ; Cohen, E. ; Canny, J.: Modeling human behavior from simple sensors in the home. In: *Lecture Notes in Computer Science* 3968 (2006), S. 337. ISBN 3–540–33894–2

[AIS93] Agrawal, Rakesh ; Imieliński, Tomasz ; Swami, Arun: Mining association rules between sets of items in large databases. In: *Proceedings of the 1993 ACM SIGMOD Conference*. New York, NY, USA : ACM, 1993. – ISBN 0–89791–592–5, S. 207–216

[Alf09] Alfred Kärcher Vertriebs-GmbH: *RoboCleaner*. `http://www.robocleaner.de`, 2009. – Friedrich-List-Straße 4

[als10] alsa-project.org: *Advanced Linux Sound Architecture (ALSA) project homepage*. `http://www.alsa-project.org/main/index.php/Main_Page`, 2010

[AP94] Aamodt, A. ; Plaza, E.: Case-based reasoning: Foundational issues, methodological variations, and system approaches. In: *AI communications* 7 (1994), Nr. 1, S. 39–59. – ISSN 0921–7126

[App10] Apple Inc.: *Apple*. `http://www.apple.com`, 2010. – Apple Inc. 1 Infinite Loop, MS60-DR, Cupertino, California, USA, 95014

[AS94] Agrawal, R. ; Srikant, R.: Fast algorithms for mining association rules. 1215 (1994), S. 487–499

[AS95] Agrawal, Rakesh ; Srikant, Ramakrishnan: Mining Sequential Patterns. 1 (1995), S. 3–14

[Ass09] Assistive Innovations bv: *Esshilfe - Mealtime Partner, Armstütze - DAS, Roboterarm - iARM*. `http://assistive.nl`, 2009. – Kervel 4, 6942 SC DIDAM

[Bau08] Bauer, Andreas: *Kraft-Momenten-Sensor-gestützte Benutzerinteraktionen in einem Szenario der Rehabilitationsrobotik*. Otto-Hahn-Alle 1, Universität Bremen, IAT, Diplomarbeit, März 2008

[BEL09] BELROBOTICS - Belgium Robotic Systems: *Belrobotics*. `http://www.belrobotics.com/`, 2009. – Avenue Lavoisier 16 B, B-1300 Wavre, Belgium

[Bie93] Bieberstein, N.: *CASE-Tools: Auswahl - Bewertung - Einsatz*. 1993

[BKI08] Beierle, Christoph ; Kern-Isberner, Gabriele: *Methoden wissensbasierter Systeme: Grundlagen, Algorithmen, Anwendungen*. 4., verb. Aufl. Wiesbaden : Vieweg + Teubner, 2008

[BWK+03] Böhme, Hans-Joachim ; Wilhelm, Torsten ; Key, Jürgen ; Schauer, Carsten ; Schröter, Christof ; Gross, Horst-Michael ; Hempel, Torsten: An Approach to Multi-modal Human-Machine Interaction for Intelligent Service Robots. In: *Robotics and Autonomous Systems* (2003), S. 83–96

[CCC09] Costa, Raul ; Cachulo, Nuno ; Cortez, Paulo: An Intelligent Alarm Management System for Large-Scale Telecommunication Companies. In: *Progress in Artificial Intelligence* Bd. 5816, Springer Berlin / Heidelberg, 2009 (Lecture Notes in Computer Science). – ISBN 978–3–642–04685–8, 386-399

[CEM03] Chatley, R. ; Eisenbach, S. ; Magee, J.: Modelling a framework for plugins. In: *Specification and verification of component-based systems* (2003)

[Cyr05] Cyriacks, Marco: *Entwicklung einer Mensch-Maschine-Schnittstelle zur teilautonomen Aufgabenbearbeitung mit einem Rehabilitationsroboter*. Otto-Hahn-Allee 1, Universität Bremen, Diplomarbeit, Dezember 2005

[Dah06] Dahm, Markus: *Grundlagen der Mensch-Computer-Interaktion*. Pearson Studium, 2006

[Dil06] Dilger, W.: *Einführung in die Künstliche Intelligenz*. Chemnitz University of Technology, Faculty of Computer Science, Chair of Artificial Intelligence (Künstliche Intelligenz), April 2006

[DK08] Diankov, R. ; Kuffner, J.: Openrave: A planning architecture for autonomous robotics. In: *Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-08-34* (2008)

[Eng89] Engelberger, J. F.: *Robotics in service*. The MIT press, 1989

[EUR09] EUROBOTS GmbH: *Wiper Blitz 4WD*. `http://www.lawnbot.de/`, 2009. – Münsterstr. 336, 40470 Düsseldorf

[Exa09] Exact Dynamics: *Exact Dynamics - iARM*. `http://www.exactdynamics.nl`, 2009. – Kervel 4; NL-6942 SC; Didam, The Netherlands

[FIG06] Feuser, Johannes ; Ivlev, Oleg ; Gräser, Axel: Mapped Virtual Reality for a safe manipulation in rehabilitation robotics. In: *Technology and Disability* 18 (2006), Nr. 4, S. 195–200. – ISSN 1055–4181

[FOC09] FOCAL MEDITECH BV: *Personal Robot BRIDGIT*. http://www.focalmeditech.nl, 2009. – Droogdokkeneiland 19; 5026SP Tilburg; The Netherlands

[Fre09] FREITAG, Max: *Dynamische Aktorik- und Sensorikerfassung in CORBA-basierter, hybrider Mehrschichtenarchitektur*. Otto-Hahn-Allee 1, Universität Bremen, Diplomarbeit, Juli 2009

[Fri09] FRIENDLY ROBOTICS®: *Robomow®*. `http://www.friendlyrobotics.de/robomow`, 2009. – Kemnather Str. 7, 95469 Speichersdorf

[GHJV95] GAMMA, Erich ; HELM, Richard ; JOHNSON, Ralph ; VLISSIDES, John: *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995

[Grä10] GRÄSER, Axel ; VOLOSYAK, Ivan (Hrsg.): *BRAINROBOT - Methods and Applications for Brain Computer Interfaces*. Brain Computer Interface. Bremen : Shaker Verlag, 2010 (Publication Series of the Institute of Automation, University of Bremen)

[Gri10] GRIGORESCU, Sorin M.: *Robust Machine Vision for Service Robotics*. Otto-Hahn-Allee 1, 28359 Bremen, Germany, University of Bremen, Diss., 2010

[HC08] HUANG, Kuo-Yu ; CHANG, Chia-Hui: Efficient mining of frequent episodes from complex sequences. In: *Information Systems* 33 (2008), Nr. 1, 96-114. `http://dx.doi.org/10.1016/j.is.2007.07.003`. – DOI 10.1016/j.is.2007.07.003. – ISSN 0306–4379

[Hei04] HEINECKE, Andreas M.: *Mensch-Computer-Interaktion: mit 19 Tabellen*. München [u.a.] : Fachbuchverl. Leipzig im Carl-Hanser-Verl., 2004

[HK06] HAN, Jiawei ; KAMBER, Micheline: *Data Mining: Concepts and Techniques*. 2nd edition. Morgan Kaufmann, 2006

[HM04] HUNT, Andrew ; MCGLASHAN, Scott: *Speech Recognition Grammar Specification Version 1.0*. `http://www.w3.org/TR/speech-grammar/`, March 2004. – W3C Recommendation 16 March 2004

[Hol03] HOLLNAGEL, E.: Is affective computing an oxymoron? In: *International Journal of Human-Computer Studies* 59 (2003), Nr. 1-2, S. 65–70

[HS95] HOUTSMA, M. ; SWAMI, A.: Set-oriented mining for association rules in relational databases. In: *11th International Conference on Data Engineering* (1995), S. 25–33. ISBN 0–8186–6910–1

[Hud03] Hudlicka, E.: To feel or not to feel: The role of affect in human–computer interaction. In: *International Journal of Human-Computer Studies* 59 (2003), Nr. 1-2, S. 1–32

[HV99] Henning, M. ; Vinoski, S.: *Advanced CORBA programming with C++*. Addison-Wesley, 1999 (Professional Computing Series)

[ISO06a] ISO (International Organization for Standardization): Document Schema Definition Languages (DSDL); Part 3: Rule-based validation - Schematron / ISO/IEC. 2006 (1). – International standard. – Information technology

[ISO06b] ISO (International Organization for Standardization): *ISO 3166-1:2006 - Codes for the representation of names of countries and their subdivisions*. `http://www.iso.org/iso`, 2006. – ISO copyright office, Case postale 56, CH-1211 Geneva 20

[ISO06c] ISO (International Organization for Standardization): *ISO 9241-110:2006 - Ergonomics of human-system interaction – Part 110: Dialogue principles*. `http://www.iso.org/iso/iso_catalogue/`, 2006. – ISO copyright office, Case postale 56, CH-1211 Geneva 20

[ISO10] ISO (International Organization for Standardization): *The Leading Implementation of ISO Schematron*. `http://www.schematron.com`, 2010. – Information technology

[JGH+07] Jacob, Robert ; Girouard, Audrey ; Hirshfield, Leanne M. ; Horn, Michael ; Shaer, Orit ; Solovey, Erin T. ; Zigelbaum, Jamie: CHI2006: what is the next generation of human-computer interaction? In: *interactions* 14 (2007), Nr. 3, S. 53–58. `http://dx.doi.org/http://doi.acm.org/10.1145/1242421.1242459`. – DOI http://doi.acm.org/10.1145/1242421.1242459. – ISSN 1072–5520

[JGH+08] Jacob, Robert J. K. ; Girouard, Audrey ; Hirshfield, Leanne M. ; Horn, Michael S. ; Shaer, Orit ; Solovey, Erin T. ; Zigelbaum, Jamie: Reality-based interaction: a framework for post-WIMP interfaces. In: *CHI '08: Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*. New York, NY, USA : ACM, 2008. – ISBN 978–1–60558–011–1, S. 201–210

[Kle09] Klein & More AG + Co. KG: *Haushaltsroboter*. `http://www.irobot.com/de/home_robots.cfm`, 2009. – iRobot, 8 Crosby Drive, Bedford, MA 01730

[Kle10] Kleinert, Jan: So gehts: Benutzeroberflächen modern und portabel gestalten - Aufpoliert. In: *Linux Magazin* (2010), Nr. 03/10, S. 35

[KLRMS02] Kantety, R. V. ; La Rota, M. ; Matthews, D. E. ; Sorrells, M. E.: Data mining for simple sequence repeats in expressed sequence tags from barley, maize, rice, sorghum and wheat. In: *Plant Molecular Biology* 48 (2002), Nr. 5, S. 501–510. – ISSN 0167–4412

[KPI95] Kawamura, K. ; Pack, R. T. ; Iskarous, M.: Design philosophy for service robots. In: *Systems, Man and Cybernetics - Intelligent Systems for the 21st Century* Bd. 4, IEEE, oct. 1995, S. 3736–3741

[KQ99] Kohavi, Ron ; Quinlan, Ross: Decision Tree Discovery. In: *IN HANDBOOK OF DATA MINING AND KNOWLEDGE DISCOVERY*, University Press, 1999, S. 267–276

[KS07] Kramer, J. ; Scheutz, M.: Development environments for autonomous mobile robots: A survey. In: *Autonomous Robots* 22 (2007), Nr. 2, S. 101–132

[KT08] Kelly, S. ; Tolvanen, J. P.: *Domain-specific modeling: enabling full code generation.* Wiley-IEEE Computer Society Pr, 2008

[Lan08] Lange, Uwe: *Benutzergestützte Objektlokalisierung in Szenarien der Rehabilitationsrobotik über ein Stereokamerasystem.* Otto-Hahn-Alle 1, Universität Bremen, IAT, Diplomarbeit, April 2008

[LHB10] Leishman, F. ; Horn, O. ; Bourhis, G.: Smart wheelchair control through a deictic approach. In: *Robotics and Autonomous Systems* 58 (2010), Nr. 10, 1149-1158. `http://dx.doi.org/DOI:10.1016/j.robot.2010.06.007`. – DOI DOI: 10.1016/j.robot.2010.06.007. – ISSN 0921–8890

[LKMY02] Larranaga, P. ; Kuijpers, C. M. H. ; Murga, R. H. ; Yurramendi, Y.: Learning Bayesian network structures by searching for the best ordering with genetic algorithms. In: *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on* 26 (2002), Nr. 4, S. 487–493. – ISSN 1083–4427

[Loq10a] Loquendo - Vocal Technology and Services: *Loquendo ASR - Automatic speech recognition.* `http://www.loquendo.com/de/technology/asr.htm`, 2010. – Loquendo; via Olivetti, 6 - 10148 Torino - Italy

[Loq10b] Loquendo - Vocal Technology and Services: *Loquendo ASR 7.1 - Programmer's Guide.* 7.1.5. Loquendo; via Olivetti, 6 - 10148 Torino - Italy, 2010

[Loq10c] Loquendo - Vocal Technology and Services: *Loquendo TTS - Text-to-Speech.* `http://www.loquendo.com/de/technology/tts.htm`, 2010. – Loquendo; via Olivetti, 6 - 10148 Torino - Italy

[Loq10d] Loquendo - Vocal Technology and Services: *Loquendo TTS programmer's guide*. 7. Loquendo; via Olivetti, 6 - 10148 Torino - Italy, 2010

[LSU07] Laxman, Srivatsan ; Sastry, P. S. ; Unnikrishnan, K. P.: A fast algorithm for finding frequent episodes in event streams. In: *13th International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA : ACM, 2007. – ISBN 978–1–59593–609–7, S. 410–419

[LXM08] LaRosa, C. ; Xiong, L. ; Mandelberg, K.: Frequent pattern mining for kernel trace data. In: *SAC 2008* (2008), S. 880–885

[Mae10] Maemo Development Platform: *The Home of the Maemo Community*. `http://www.maemo.org/`, 2010. – Maemo Development Platform; Att: Tero Kojo; Itämerenkatu 11 - 13; 00180 HELSINKI; PL/P.O.Box 407; 00045 NOKIA GROUP; Finland

[Mar99] Marquardt, K.: Patterns for Plug-ins. In: *Pattern Languages of Program Design 5* (1999), S. 301

[Mar03] Martens, Christian: *Teilautonome Aufgabenbearbeitung bei Rehabilitationsrobotern mit Manipulator*. Otto-Hahn-Allee 1, Institut für Automatisierungstechnik - Universität Bremen, Diss., 2003

[MB10] Magnus, Nils ; Bantle, Ulrich: Statt Mann und Maus - Evolution und Revolution bei der grafischen Benutzerführung. In: *Linux Magazin* (2010), Nr. 03/10, S. 36–38

[MDBM10] Montesano, L. ; Diaz, M. ; Bhaskar, S. ; Minguez, J.: Towards an Intelligent Wheelchair System for Users With Cerebral Palsy. In: *Neural Systems and Rehabilitation Engineering, IEEE Transactions on* 18 (2010), apr., Nr. 2, S. 193–202. `http://dx.doi.org/10.1109/TNSRE.2009.2039592`. – DOI 10.1109/TNSRE.2009.2039592. – ISSN 1534–4320

[Min04] Mintert, Stefan: Man spricht XML - Was die Extensible Markup Language ist und was nicht. In: *iX Special - Das XML Kompendium* (2004), Nr. 01/2004, S. 6–10

[MM02] Manku, Gurmeet S. ; Motwani, Rajeev: Approximate frequency counts over data streams. In: *Proceedings of the 28th international conference on Very Large Data Bases*. Hong Kong, Chin : VLDB Endowment, 2002, S. 346–357

[MMS03] Mayer, Johannes ; Melzer, Ingo ; Schweiggert, Franz: Lightweight Plug-In-Based Application Development. In: *Objects, Components, Architectures, Services, and Applications for a Networked World* Bd. 2591, Springer Berlin / Heidelberg, 2003 (Lecture Notes in Computer Science). – ISBN 978–3–540–00737–1, 87-102

[MPFG05] Martens, Christian ; Prenzel, Oliver ; Feuser, Johannes ; Gräser, Axel: MASSiVE: Multi-Layer Architecture for Semi-Autonomous Service-Robots with Verified Task Execution. (2005), S. 107–112

[MRL+01] Martens, C. ; Ruchel, N. ; Lang, O. ; Ivlev, O. ; Graser, A.: A FRIEND for assisting handicapped people. In: *IEEE Robotics & Automation Magazine* 8 (2001), Nr. 1, S. 57–65. – ISSN 1070–9932

[MTV97] Mannila, Heikki ; Toivonen, Hannu ; Verkano, A. I.: Discovery of frequent episodes in event sequences. In: *13th International Conference on Knowledge Discovery and Data Mining* 1 (1997), Nr. 3, S. 259–289. `http://dx.doi.org/10.1023/A:1009748302351`. – DOI 10.1023/A:1009748302351

[MWK+06] Mierswa, Ingo ; Wurst, Michael ; Klinkenberg, Ralf ; Scholz, Martin ; Euler, Timm: YALE: rapid prototyping for complex data mining tasks. In: Ungar, L. (Hrsg.) ; Craven, M. (Hrsg.) ; Gunopulos, D. (Hrsg.) ; Eliassi-Rad, T. (Hrsg.): *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. New York, NY, USA : Acm, 2006. – ISBN 1–59593–339–5, S. 935–940

[Mye03] Myers, Gene: Advances in DNA sequencing. In: *PCK50: Proceedings of the Paris C. Kanellakis memorial workshop on Principles of computing & knowledge*. New York, NY, USA : ACM, 2003 (PCK50). – ISBN 1–58113–604–8, 9-9

[Nok10] Nokia Corporation: *Qt Reference Documentation - Qt 4.6.2*. `http://doc.qt.nokia.com/4.6/index.html`, 2010

[Obj10] Object Management Group, Inc.: *OMG's Internet Inter-ORB Protocol - A Brief Description*. `http://www.omg.org/news/whitepapers/iiop.htm`, 2010. – 140 Kendrick Street, Building A Suite 300, Needham, MA 02494, U.S.A.

[Ojd09] Ojdanic, Darko: *Using Cartesian Space for Manipulator Motion Planning - Application in Service Robotics*. Otto-Hahn-Allee 1, 28359 Bremen, Germany, University of Bremen, Diss., 2009

[Pan10] Pantke, Karl-Heinz ; Pantke, Karl-Heinz (Hrsg.): *Mensch und Maschine - Wie Brain-Computer-Interfaces und andere Innovationen gelähmten Menschen kommunizieren helfen*. Frankfurt am Main : Mabuse Verlag, 2010

[PCH10] Piper, A. M. ; Campbell, R. ; Hollan, J. D.: Exploring the Accessibility and Appeal of Surface Computing for Older Adult Health Care Support. (2010)

[PFG05] Prenzel, O. ; Feuser, J. ; Graser, A.: Rehabilitation robot in intelligent home environment-software architecture and implementation of a distributed system. (2005), S. 530–535

[PMC+07] Prenzel, Oliver ; Martens, Christian ; Cyriacks, Marco ; Wang, Chao ; Gräser, Axel: System-controlled user interaction within the service robotic control architecture massive. In: *Robotica* 25 (2007), Nr. 2, S. 237–244. http://dx.doi.org/http://dx.doi.org/10.1017/S0263574707003347. – DOI http://dx.doi.org/10.1017/S0263574707003347. – ISSN 0263–5747

[Pre03] Prenzel, Oliver: *Teilautonome Umgebungserfassung zur automatischen Befehlsbearbeitung mit einem Rehabilitationsroboter.* Otto-Hahn-Alle 1, Universität Bremen, Diplomarbeit, 2003

[Pre09] Prenzel, Oliver: *Process Model for the Development of Semi-Autonomous Service Robots.* Otto-Hahn-Alle 1, Institute of Automation - University of Bremen, Diss., 2009

[PT99] Percus, Allon G. ; Torney, David C.: Greedy algorithms for optimized DNA sequencing. In: *SODA '99: Proceedings of the tenth annual ACM-SIAM symposium on Discrete algorithms.* Philadelphia, PA, USA : Society for Industrial and Applied Mathematics, 1999. – ISBN 0–89871–434–6, S. 955–956

[QGC+09] Quigley, M. ; Gerkey, B. ; Conley, K. ; Faust, J. ; Foote, T. ; Leibs, J. ; Berger, E. ; Wheeler, R. ; Ng, A.: ROS: an open-source Robot Operating System. In: *International Conference on Robotics and Automation* (2009)

[Rap10] Rapid-I: *RapidMiner 5.0 - Benutzerhandbuch.* 5.0. Stockumer Str. 475; 44227 Dortmund; Germany, 2010

[RBSM10] Reis, Luis ; Braga, Rodrigo ; Sousa, Márcio ; Moreira, Antonio: IntellWheels MMI: A Flexible Interface for an Intelligent Wheelchair. In: Baltes, Jacky (Hrsg.) ; Lagoudakis, Michail (Hrsg.) ; Naruse, Tadashi (Hrsg.) ; Ghidary, Saeed (Hrsg.): *RoboCup 2009: Robot Soccer World Cup XIII* Bd. 5949. Springer Berlin / Heidelberg, 2010, Kapitel Lecture Notes in Computer Science, S. 296–307

[RSI98] Rowe, Dennis W. ; Sibert, John ; Irwin, Don: Heart rate variability: indicator of user state as an aid to human-computer interaction. In: *CHI '98: Proceedings of the SIGCHI conference on Human factors in computing systems.* New York, NY, USA : ACM Press/Addison-Wesley Publishing Co., 1998. – ISBN 0–201–30987–4, S. 480–487

[Sch10] Schroeder, S.: Introduction to MeeGo. In: *Pervasive Computing, IEEE* 9 (2010), april, Nr. 4, S. 4–7. http://dx.doi.org/10.1109/MPRV.2010.82.

– DOI 10.1109/MPRV.2010.82. – ISSN 1536–1268

[SCH11] SCHUNK GmbH & Co. KG - Spann- und Greiftechnik: *Service Robotics - Modular Robotics*. http://www.schunk.de, 2011. – Bahnhofstr. 106 - 134, 74348 Lauffen/Neckar

[SKS+10] Shaer, O. ; Kol, G. ; Strait, M. ; Fan, C. ; Grevet, C. ; Elfenbein, S.: G-nome Surfer: a Tabletop Interface for Collaborative Exploration of Genomic Data. (2010)

[SLB99] Song, Won-Kyung ; Lee, Heyoung ; Bien, Zeungnam: KARES: Intelligent wheelchair-mounted robotic arm system using vision and force sensor. In: *Robotics and Autonomous Systems* 28 (1999), Nr. 1, 83-94. http://dx.doi.org/DOI:10.1016/S0921-8890(99)00031-7. – DOI DOI: 10.1016/S0921–8890(99)00031–7. – ISSN 0921–8890. – Robotics Applications at FLINS'98

[Son09] Sony Entertainment Robot Europe: *Sony AIBO Europe*. http://support.sony-europe.com/aibo/index.asp?language=de, 2009

[Syn10] SyncRO soft ltd: *<oXygen/> XML Editor*. http://www.oxygenxml.com, 2010. – SyncRO soft ltd; str. Fermei; nr. 42; Craiova 200782; Romania

[The11] The Apache Software Foundation: *Xerces-C++*. http://xerces.apache.org/xerces-c/, 2011

[Top02] Topping, M.: An overview of the development of Handy 1, a rehabilitation robot to assist the severely disabled. In: *Journal of Intelligent & Robotic Systems* 34 (2002), Nr. 3, S. 253–263. – ISSN 0921–0296

[UG10] Usakli, A. B. ; Gurkan, S.: Design of a Novel Efficient Human-Computer Interface: An Electrooculagram Based Virtual Keyboard. In: *Instrumentation and Measurement, IEEE Transactions on* 59 (2010), aug., Nr. 8, S. 2099–2108. http://dx.doi.org/10.1109/TIM.2009.2030923. – DOI 10.1109/TIM.2009.2030923. – ISSN 0018–9456

[Uni09] University of Bremen - Institute of Automation: *ADL (Activities of Daily Life) Scenario*. http://www.iat.uni-bremen.de/sixcms/detail.php?id=589, 2009. – This video presents parts of the ADL (Activities of Daily Life) Scenario.

[VCF+07] Valbuena, Diana ; Cyriacks, Marco ; Friman, Ola ; Volosyak, Ivan ; Graeser, Axel: Brain-Computer Interface for high-level control of rehabilitation robotic systems. In: *10th International Conference on Rehabilitation Robotics* (2007)

[VPA+00] Viano, Gianni ; Parodi, Andrea ; Alty, James ; Khalil, Chris ; Angulo, Inaki ; Biglino, Daniele ; Crampes, Michel ; Vaudry, Christophe ;

DAURENSAN, Veronique ; LACHAUD, Philippe: Adaptive user interface for process control based on multi-agent approach. In: *AVI '00: Proceedings of the working conference on Advanced visual interfaces*. New York, NY, USA : ACM Press, 2000. – ISBN 1–58113–252–2, S. 201–204

[Wik10] WIKIMEDIA FOUNDATION INC.: *Die freie online Enzyklopädie*. `http://de.wikipedia.org/wiki`, 2010. – P.O. Box 78350, San Francisco, CA 94107-8350, United States of America

[Wit02] WITTIG, Frank: *Maschinelles Lernen Bayes'scher Netze für benutzeradaptive Systeme*, Universität Saarbrücken, Diss., 2002. – Online-Ressource

[ZHL+98] ZAIANE, Osmar R. ; HAN, Jiawei ; LI, Ze-Nian ; CHEE, Sonny H. ; CHIANG, Jenny Y.: MultiMediaMiner: a system prototype for multimedia data mining. In: *Proceedings of the 1998 ACM SIGMOD international conference on Management of data*. New York, NY, USA : ACM, 1998 (SIGMOD '98). – ISBN 0–89791–995–5, 581-583

Abbildungsverzeichnis

Tabellenverzeichnis