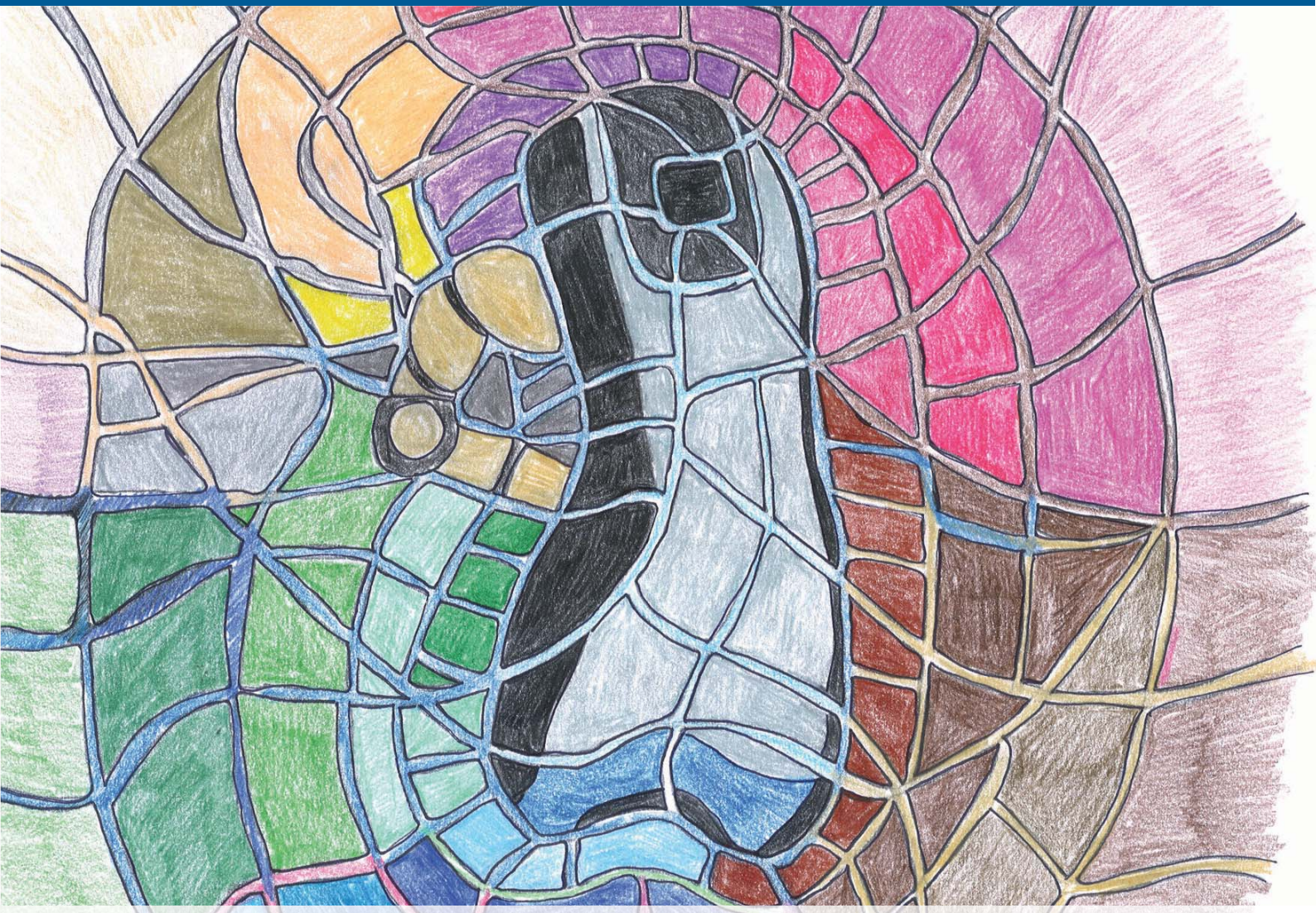
STUTTGARTER BEITRÄGE ZUR PRODUKTIONSFORSCHUNG

ULRICH REISER

Eine webbasierte Integrations- und Testplattform zur Unterstützung des verteilten Entwicklungsprozesses von komplexen Serviceroboter-Applikationen



Universität Stuttgart

Fraunhofer IPA

STUTTGARTER BEITRÄGE ZUR PRODUKTIONSFORSCHUNG BAND 29

Herausgeber:
Univ.-Prof. Dr.-Ing. Thomas Bauernhansl
Univ.-Prof. Dr.-Ing. Dr. h.c. mult. Alexander Verl
Univ.-Prof. a. D. Dr.-Ing. Prof. E.h. Dr.-Ing. E.h. Dr. h.c. mult. Engelbert Westkämper

Ulrich Reiser

Eine webbasierte Integrations- und Testplattform zur Unterstützung des verteilten Entwicklungsprozesses von komplexen Serviceroboter-Applikationen

FRAUNHOFER VERLAG

Kontaktadresse:
Fraunhofer-Institut für Produktionstechnik und Automatisierung IPA, Stuttgart
Nobelstraße 12, 70569 Stuttgart
Telefon 0711 970-00, Telefax 0711 970-1399
info@ipa.fraunhofer.de, www.ipa.fraunhofer.de

STUTTGARTER BEITRÄGE ZUR PRODUKTIONSFORSCHUNG

Herausgeber:
Univ.-Prof. Dr.-Ing. Thomas Bauernhansl
Univ.-Prof. Dr.-Ing. Dr. h.c. mult. Alexander Verl
Univ.-Prof. a. D. Dr.-Ing. Prof. E.h. Dr.-Ing. E.h. Dr. h.c. mult. Engelbert Westkämper

Fraunhofer-Institut für Produktionstechnik und Automatisierung IPA, Stuttgart
Institut für Industrielle Fertigung und Fabrikbetrieb (IFF) der Universität Stuttgart
Institut für Steuerungstechnik der Werkzeugmaschinen und Fertigungseinrichtungen (ISW)
der Universität Stuttgart

Titelbild: © Olga Reiser

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar.
ISSN: 2195-2892
ISBN (Print): 978-3-8396-0675-9

D 93

Zugl.: Stuttgart, Univ., Diss., 2013

Druck: Mediendienstleistungen des Fraunhofer-Informationszentrum Raum und Bau IRB, Stuttgart
Für den Druck des Buches wurde chlor- und säurefreies Papier verwendet.

Fraunhofer-Informationszentrum Raum und Bau IRB
Postfach 80 04 69, 70504 Stuttgart
Nobelstraße 12, 70569 Stuttgart
Telefon 0711 970-2500
Telefax 0711 970-2508
E-Mail verlag@fraunhofer.de
URL http://verlag.fraunhofer.de

GELEITWORT DER HERAUSGEBER

Produktionswissenschaftliche Forschungsfragen entstehen in der Regel im Anwendungszusammenhang, die Produktionsforschung ist also weitgehend erfahrungsbasiert. Der wissenschaftliche Anspruch der „Stuttgarter Beiträge zur Produktionsforschung" liegt unter anderem darin, Dissertation für Dissertation ein übergreifendes ganzheitliches Theoriegebäude der Produktion zu erstellen.

Die Herausgeber dieser Dissertations-Reihe leiten gemeinsam das Fraunhofer-Institut für Produktionstechnik und Automatisierung IPA und jeweils ein Institut der Fakultät für Konstruktions-, Produktions- und Fahrzeugtechnik an der Universität Stuttgart.

Die von ihnen betreuten Dissertationen sind der marktorientierten Nachhaltigkeit verpflichtet, ihr Ansatz ist systemisch und interdisziplinär. Die Autoren bearbeiten anspruchsvolle Forschungsfragen im Spannungsfeld zwischen theoretischen Grundlagen und industrieller Anwendung.

Die „Stuttgarter Beiträge zur Produktionsforschung" ersetzt die Reihen „IPA-IAO Forschung und Praxis" (Hrsg. H.J. Warnecke / H.-J. Bullinger / E. Westkämper / D. Spath) bzw. ISW Forschung und Praxis (Hrsg. G. Stute / G. Pritschow / A. Verl). In den vergangenen Jahrzehnten sind darin über 800 Dissertationen erschienen.

Der Strukturwandel in den Industrien unseres Landes muss auch in der Forschung in einen globalen Zusammenhang gestellt werden. Der reine Fokus auf Erkenntnisgewinn ist zu eindimensional. Die „Stuttgarter Beiträge zur Produktionsforschung" zielen also darauf ab, mittelfristig Lösungen für den Markt anzubieten. Daher konzentrieren sich die Stuttgarter produktionstechnischen Institute auf das Thema ganzheitliche Produktion in den Kernindustrien Deutschlands. Die leitende Forschungsfrage der Arbeiten ist: Wie können wir nachhaltig mit einem hohen Wertschöpfungsanteil in Deutschland für einen globalen Markt produzieren?

Wir wünschen den Autoren, dass ihre „Stuttgarter Beiträge zur Produktionsforschung" in der breiten Fachwelt als substanziell wahrgenommen werden und so die Produktionsforschung weltweit voranbringen.

Alexander Verl Thomas Bauernhansl Engelbert Westkämper

Eine webbasierte Integrations- und Testplattform zur Unterstützung des verteilten Entwicklungsprozesses von komplexen Serviceroboter-Applikationen

Von der Fakultät Konstruktions-, Produktions- und Fahrzeugtechnik der Universität Stuttgart zur Erlangung der Würde eines Doktor-Ingenieurs (Dr.-Ing.) genehmigte Abhandlung

Vorgelegt von

Dipl.-Ing. Ulrich Reiser
aus Albstadt

Hauptberichter:	Prof. Dr.-Ing. Dr. h.c. mult. Alexander Verl
Mitberichter:	Prof. Dr.-Ing. Heinz Wörn
Tag der mündlichen Prüfung:	26. September 2013

Institut für Steuerungstechnik der Werkzeugmaschinen
und Fertigungseinrichtungen der Universität Stuttgart
2014

Vorwort des Verfassers

Die vorliegende Arbeit ist während meiner Tätigkeit als wissenschaftlicher Mitarbeiter am Fraunhofer Institut für Produktionstechnik und Automatisierung (IPA) entstanden. Sie wurde teilweise mit Mitteln des Ministeriums für Bildung und Forschung (BMBF) im Rahmen des Forschungsprojekts DESIRE (Deutsche Servicerobotik Initiative) unter dem Kennzeichen 01IME01A sowie des 7. Rahmenprogramms der europäischen Union im Rahmen des Forschungsprojekts BRICS (Best Practice in Robotics) unter dem Kennzeichen FP7-ICT-231940-BRICS gefördert.

Mein herzlicher Dank gilt Herrn Prof. Dr.-Ing. Dr. h.c. mult. Alexander Verl für die Unterstützung meiner Arbeit und die Übernahme des Hauptberichts. Weiterhin bedanke ich mich bei Herrn Prof. Dr.-Ing. Heinz Wörn für die Übernahme des Mitberichts und die sehr angenehme Zusammenarbeit.

Weiterhin möchte ich mich bei Herrn Dipl.-Ing. Martin Hägele M.S., dem Leiter der Abteilung Roboter- und Assistenzsysteme, bedanken. Seine jahrelange Unterstützung und sein Vertrauen, das er mir stets entgegen brachte, haben mich fortwährend motiviert.

Für die Durchsicht sowie die kritischen und konstruktiven Kommentare zu meiner Arbeit danke ich Herrn Prof. Dr.-Ing. Christian Schlegel, Frau Dr.-Ing. Dipl.-Inform. Birgit Graf, Herrn Dipl.-Inform. Winfried Baum, Herrn Dr. Armin Reiser, Frau Luzia Schuhmacher M.A. und meinem lieben Vater.

Ich danke meinen Kollegen und dem DESIRE-Projektteam für den Zuspruch, den ich während der Themenfindung erfahren habe. Insbesondere bedanke ich mich bei den von mir betreuten Studenten, die im Laufe von 5 Jahren während Praktika, Diplom- und Forschungsarbeiten wertvolle Beiträge zu meiner Arbeit geleistet haben.

Schließlich danke ich von Herzen meiner Frau Olga. Ohne Ihre jahrelange Unterstützung und die Schaffung der notwendigen Freiräume an den vielen Wochenenden auch nach der Geburt unserer beiden Töchter Elisabeth und Helen wäre diese Arbeit nicht möglich gewesen.

Albstadt, im Dezember 2013 Ulrich Reiser

Kurzinhalt

Die Entwicklung von Robotersystemen zeichnet sich aus durch einen immer größeren Anteil der Integration bestehender Technologien unter anderem aus den Bereichen Automatisierungstechnik, Maschinenbau und Informatik. Insbesondere für komplexe Serviceroboter, die in dynamischen und unstrukturierten Umgebungen eingesetzt werden und daher mit einer Vielzahl von Soft- und Hardwarekomponenten ausgestattet sind, werden Methoden zur Vereinfachung der Systemintegration benötigt. Der Bedarf an Abstraktion der Komplexität besteht umso mehr, als diese Systeme in der Regel in multi-disziplinären Teams an verteilten Standorten entwickelt werden.

In der vorliegenden Arbeit werden Methoden und Konzepte zur Vereinfachung des Entwicklungs- und Integrationsprozesses von komponenten-basierten Applikationen auf komplexen Servicerobotern mit verteilten Rechnerarchitekturen vorgestellt. Zur Identifikation der Besonderheiten der Roboterentwicklung werden typische Entwicklungsszenarien in der Servicerobotik-Forschung analysiert.

Neben der Konzeption eines Vorgehensmodells für die verteilte Entwicklung und Integration stehen Technologien zur Abstraktion der Deployment-Aktivitäten im Fokus, die die werkzeuggestützte Installation und Aktualisierung, die Konfiguration für das Zielsystem, die Zielumgebung und eventuelle Nutzerpräferenzen sowie die Bedienung der Applikation umfassen. Durch diese Technologien und Konzepte soll eine Verbesserung der Rollentrennung von Komponentenentwicklern, Applikationsentwicklern und Systemintegratoren sowie die Effizienzsteigerung des Integrationsprozesses durch räumliche und zeitliche Entkopplung der Integrations- und Testaktivitäten erzielt werden.

Als Basis für die Konzeption der abstrakten Deployment-Werkzeuge wird unter anderem eine erweiterbare domänenspezifische Sprache zur Ressourcenmodellierung des Zielsystems “komplexer Serviceroboter” und eine abstrakte Laufzeitumgebung zum hardwareunabhängigen Zugriff auf Robotersysteme entwickelt. Die einzelnen Deployment-Werkzeuge zur Planung, Installation, Konfiguration und Aktivierung werden in Form einer webbasierten Integrations- und Testplattform als geschlossene Deployment-Werkzeugkette integriert.

Zur quantitativen Evaluierung der entwickelten Werkzeuge wird zunächst eine mathematische Formulierung des Integrationsaufwands auf der Basis von Kostenschätzmodellen der Softwaretechnik erstellt. Mit Hilfe dieser Evaluierungsmethodik wird am Beispiel eines großen Verbundforschungsprojektes die Effektivität der entwickelten Werkzeuge nachgewiesen. Die räumliche und zeitliche Entkopplung der Integrations- und Testaktivitäten sowie die damit einhergehende Verbesserung der Rollentrennung der einzelnen Entwickler wird durch graphische Auswertungen anschaulich belegt.

Short summary

The development of robot systems depends more and more on the integration of existing technologies e.g., from automation, mechanical engineering and computer science. In particular, complex service robots that are applied outside of production halls in dynamic and unstructured environments consist of a multiplicity of software and hardware components, requiring methods to reduce system integration. The demand for the abstraction of this complexity is increased by the fact that these robot systems are typically developed in multi-disciplinary teams at different sites.

In this thesis, methods and concepts are presented that simplify the distributed development and integration process of component based applications for complex service robots that perform computation on a distributed system. For the identification of the specialties of the robot development process, typical development scenarios in robotics research are analyzed.

In addition to the conception of a dedicated process model for distributed development and integration, this work focuses on the abstraction of the deployment activities, incorporating tool supported installation, update, configuration for target platform, target environment and possible user preferences, as well as activation and deactivation of the application. These measures aim at the separation of roles of component developers, application developers and system integrators, as well as a substantial increment in the efficiency of the robot development process through temporal and spacial decoupling of integration and test activities of the single developers.

For the conception of the tools to abstract the deployment activities, an extensible domain specific language is introduced to model the available resources of the target system "complex service robot." Furthermore an abstract runtime environment for a hardware-agnostic system access is developed. All tools for the single deployment steps (planning, installation, update, configuration, activation / deactivation) are integrated in a web-based integration and test platform to provide a complete deployment tool-chain.

In order to quantitatively evaluate the developed tools, a formal representation of the integration effort based on software cost estimation models is derived. Using this formal representation, the efficacy of the tools and methods developed in this thesis is evaluated

in the context of a joint robotics research project. The temporal and spacial decoupling of the integration and test activities as well as the coinciding improvement of the separation of roles is shown in graphical evaluation charts.

Inhaltsverzeichnis

Abkürzungsverzeichnis **xiii**

1 Einleitung **1**
1.1 Motivation und Problemstellung 1
1.2 Zielsetzung und Vorgehensweise 3

2 Grundlagen und Ausgangssituation **6**
2.1 Definitionen 6
2.2 Grundlagen 10
2.2.1 Integrationsmodelle und -strategien 10
2.2.2 Kooperative Softwareentwicklung 12
2.2.3 Modelle zur Aufwandsschätzung für die Entwicklung und Integration von Software-Komponenten 13
2.3 Szenarien für verteilte Integration und Tests 14
2.3.1 Definition von Parametern für verteilte Entwicklungsszenarien 14
2.3.2 Szenario 1: Verbund-Forschungsprojekt 16
2.3.3 Szenario 2: Servicerobotik-Teststand 17
2.3.4 Szenario 3: Diagnose und Fernwartung 19
2.3.5 Szenario 4: Projektübergreifende lokale Nutzung 21
2.3.6 Zusammenfassung 22

3 Analyse der Aufgabenstellung und Ableitung von Anforderungen **23**
3.1 Analyse der Integrations- und Test-Aktivitäten von komponenten-basierten Applikationen für die Entwicklungsdomäne Servicerobotik 23
3.1.1 Analyse der erforderlichen Arbeitsschritte für die Implementierung einer Servicerobotik-Applikation 25
3.1.2 Vorgehensmodell für die verteilte Integration 30
3.1.3 Analyse der Entwicklerrollen 31
3.2 Analyse der Anforderungen durch die räumliche Verteilung der Integrationsarbeit 31
3.3 Analyse der Szenarien für verteilte Integration 33

3.3.1 Szenario 1: Verbund-Forschungsprojekt . . . 33
3.3.2 Szenario 2: Servicerobotik-Teststand . . . 34
3.3.3 Szenario 3: Diagnose und Fernwartung . . . 35
3.3.4 Szenario 4: Projektübergreifende lokale Nutzung . . . 35
3.4 Strukturierung der Anforderungen . . . 36

4 Stand der Technik **41**
4.1 Werkzeuge und Methoden für Integration und Test komponenten-basierter Applikationen . . . 41
4.1.1 Unterstützung der Bindung von Komponenten an das Zielframework . 41
4.1.2 Unterstützung der komponenten-basierten Applikationsentwicklung . 42
4.1.3 Unterstützung zur Durchführung von Komponenten-, Integrations- und Systemtests . . . 44
4.1.4 Folgerungen . . . 45
4.2 Vorgehensmodelle zur Unterstützung des verteilten Entwicklungs- und Integrationsprozesses . . . 45
4.2.1 Sequenzielle Phasenmodelle . . . 45
4.2.2 Agile Entwicklungsmethoden . . . 47
4.2.3 Iterative und inkrementelle Vorgehensmodelle . . . 48
4.2.4 Folgerungen . . . 50
4.3 Deployment-Technologien . . . 52
4.3.1 Deployment-Technologien aus der Industrie . . . 52
4.3.2 Deployment-Methoden aus der Forschung . . . 53
4.3.3 Folgerungen . . . 57
4.4 Laufzeitunterstützung zur Bedienung und Wartung von Robotersystemen . . 57
4.4.1 Kommandozeileninterpreter . . . 59
4.4.2 Webbasierte Ausführungsumgebungen . . . 59
4.4.3 Teleoperationsumgebungen . . . 60
4.4.4 Ferndiagnosesysteme . . . 61
4.4.5 Folgerungen . . . 62
4.5 Kooperationswerkzeuge . . . 62
4.5.1 Unterstützung der Kollaboration in verteilten Projekten . . . 62
4.5.2 Werkzeuge zur kollaborativen Softwareerstellung . . . 64
4.5.3 Integrierte kommerzielle Kollaborations-Plattformen . . . 65
4.5.4 Folgerungen . . . 66

5 Lösungskonzeption **67**
5.1 Entwicklung des Vorgehensmodells für die verteilte Entwicklung von komplexen Servicerobotern . . . 67

5.1.1 Variantenbildung . . . 67
5.1.2 Konzeption des DDT-Vorgehensmodells . . . 69
5.1.3 Definition der DDT-Rollen . . . 70
5.1.4 Entwicklung der Modell-Phasen . . . 73
5.1.5 Entwicklung von Werkzeugen zur Unterstützung der Prozesse im Vorgehensmodell . . . 75
5.2 Entwicklung von Deployment-Werkzeugen für komplexe Serviceroboter . . . 77
5.2.1 Diskussion: Modellierung von Applikation und Zieldomäne mit Hilfe von Ontologien . . . 77
5.2.2 Entwicklung eines Algorithmus für die Deployment-Planung von Servicerobotik-Applikationen . . . 78
5.2.3 Entwicklung von Werkzeugen zur Unterstützung der Deployment-Aktivitäten . . . 85
5.3 Laufzeitwerkzeuge zur besseren Bedienbarkeit für verteilte Integration und Test . . . 89
5.3.1 Diskussion: Zentraler vs. dezentraler Zugriff . . . 90
5.3.2 Abstraktionsgrade der Laufzeitumgebung . . . 92
5.4 Zusammenführung der einzelnen Module . . . 94

6 Realisierung der Integrations- und Testplattform 96
6.1 Einbettung des Vorgehensmodells . . . 98
6.1.1 Implementierung des Vorgehensmodells in EPF . . . 98
6.2 Realisierung des Deployment-Moduls . . . 101
6.2.1 Deployment-Planung . . . 101
6.2.2 Installation und Aktualisierung . . . 106
6.3 Realisierung der Werkzeuge zur Laufzeitunterstützung . . . 109
6.3.1 Das Robotcontrol-Plugin . . . 109
6.3.2 Das HardwareMonitor-Plugin . . . 115
6.4 Auswahl eines geeigneten Übertragungsprotokolls für 3D-Applikationen . . . 117

7 Auswertung 120
7.1 Abgleich mit den Anforderungen . . . 120
7.1.1 Zielerfüllung der allgemeinen Anforderungen . . . 120
7.1.2 Zielerfüllung des Vorgehensmodells . . . 121
7.1.3 Zielerfüllung der Deployment-Werkzeuge . . . 122
7.1.4 Zielerfüllung der Laufzeitwerkzeuge . . . 123
7.1.5 Zielerfüllung der Kooperationswerkzeuge . . . 124
7.2 Evaluation des Webportals im Kontext eines Forschungsprojekts . . . 125

8 Zusammenfassung und Ausblick **135**

9 Literaturverzeichnis **139**

10 Anhang A: Referenz-Fallstudie DESIRE **163**

10.1 Struktur und Chronologie des Projektes 163

10.2 Messwerte und Parameter für Aufwandsschätzung für die Integration der DESIRE Komponenten 168

Abkürzungsverzeichnis

Namen und Begriffe

Zeichen	Bedeutung
API	Application Programming Interface
ASM	Abstract State Machine
BCM	BRICS Component Model
BRICS	Best-of-Robotics
CBSD	Component Based Software Development
CCM	CORBA Component Model
CI	Continuous Integration
CORBA	Common Object Request Broker Architecture
COTS	Commercial off-the-shelf
CPL	CRAM Plan Language
CPU	Central Processing Unit
CRAM	Cognitive Robot Abstract Machine
DMS	Dokumentenmanagementsystem
DSL	Domain Specific Language
EPF	Eclipse Process Framework
HSM	Hybrid State Machines
IDE	Integrated Development Environment
IDL	Interface Definition Language
IMU	Inertial Measurement Unit
KI	Künstliche Intelligenz
KMS	Kraft-Momenten-Sensor
MDE	Model-Driven Engineering
MTBF	Mean Time Between Failure
MTTR	Mean Time To Repair
NDI	Non-developmental-item
PIM	Platform Independent Model
RAM	Random Access Memory (Arbeitsspeicher)

rFSM	reduced Finite State Machine
RTE	Runtime Environment
SCA	Service Component Architecture
SDK	Software Development Kit
SE	Software Engineering
SPEM	Software Process Engineering Metamodel
SPS	Speicherprogrammierbare Steuerung
SWIG	Simplified Wrapper and Interface Generator
TCB	Task Coordination Block
TDD	Test-Driven Development
UPNP	Universal Plug And Play
USB	Universal Serial Bus
WfMS	Workflow-Management-System
XP	Extreme Programming

Formelverzeichnis

Zeichen	**Bedeutung**
Formelzeichen für die Deployment-Planung	
A	Applikationsgraph
C_{cw}	Menge aller Software-Komponenten
C_{hw}	Menge aller Hardware-Komponenten
D	Graph der Zieldomäne (des Zielsystems)
E^A	Menge der Kommunikationskanäle des Applikationsgraphen
E^D	Menge der Kommunikationskanäle des Domänengraphen
E_c	Menge aller Kommunikationskanäle, die mit der Komponente c verbunden sind
$I^{A \mapsto D}$	Menge aller möglichen vollständigen Deployments einer Applikation A in die Domäne D
$\tilde{I}^{A \mapsto D}$	Menge aller gültigen Deployments einer Applikation A in die Domäne D
H	Menge aller Rechner im Zielsystem
$P(x)$	Menge aller Eigenschaften eines Elements x (Kanal bzw. Komponente)
P_{quant}	Menge aller quantitativen Eigenschaften der Zieldomäne
P_{cap}	Menge aller Eigenschaften der Zieldomäne bzgl. Kapazität
P_{min}	Menge aller Minimalwert-Eigenschaften der Zieldomäne
P_{max}	Menge aller Maximalwert-Eigenschaften der Zieldomäne

P_{att}	Menge aller attributiven Eigenschaften der Zieldomäne
P_{sel}	Menge aller selektiven Eigenschaften der Zieldomäne
$R(x)$	Menge aller Anforderungen des Elements x (Kanal bzw. Komponente)
R_{quant}	Menge aller quantitativen Anforderungen der Applikation
R_{cap}	Menge aller Anforderungen der Applikation bzgl. Kapazität
R_{min}	Menge aller Minimalwert-Anforderungen der Applikation
R_{max}	Menge aller Maximalwert-Anforderungen der Applikation
R_{att}	Menge aller attributiven Anforderungen der Applikation
R_{sel}	Menge aller selektiven Anforderungen der Applikation
V^A	Menge der Komponenten des Applikationsgraphen
V^D	Menge der Komponenten des Domänengraphen
$Z_D(c^A)$	Menge aller möglichen Kandidaten für das Deployment einer Komponente c^A in die Domäne D
$M(x^A, x^D)$	Matching Funktion der Anforderungsmenge R einer Komponente x^A gegen die Eigenschaftsmenge P einer Komponente x^D
$\mu(r, p)$	Matching Funktion für die Anforderung r einer Komponente der Applikation mit der Eigenschaft p einer Komponente des Zielsystems
$\delta(c^A, c^D)$	Gibt zurück, ob eine Komponente c^A auf einem Host c^D deployed werden kann
$\lambda(x)$	Funktion, die die Bezeichnung eines Mengenelements x zurückgibt
$v(x)$	Funktion, die den Zahlenwert eines Mengenelements x zurückgibt
c	Komponente
$i^{A \mapsto D}$	Konkretes Deployment einer Applikation A auf eine Domäne D
$j^{E^A \mapsto E^D}(i^{A \mapsto D})$	Relation für die Zuordnungen der Kommunikationskanäle von Applikation A auf Zieldomäne D, gegeben ein bestimmtes Deployment $i^{A \mapsto D}$

Abkürzungen für die Aufwandschätzung für die Integration existierender Komponenten

AAF	Modifikationsaufwand für eine Komponente
AAM	Code-Anpassungsmultiplikator
ACAP	Leistungsfähigkeit der System-Architekten
APEX	Vorerfahrung mit der Applikation
BRAK	Anteil des Integrationscodes für eine Komponente, der aufgrund von Anforderungsänderungen neugeschrieben werden muss
CM	Anteil des zu verändernden Komponenten-Codes
CPLX	Komplexität der Systemmodule
DATA	Größe der verwendeten Datenbank

DM	Anteil des zu verändernden Komponenten-Designs
DOCU	Erforderlicher Dokumentationsgrad
FLEX	Vom Auftraggeber zugestandene Flexibilität bezüglich des Entwicklungsprozesses (Development Flexibility)
IM	Anteil des zu verändernden Gesamttest- und Integrationscodes durch die Integration der neuen Komponente
K	Skalierungsfaktor für die Komplexität des Projekts
KASLOC	Größe der zu integrierenden Komponente in Tausenden von Quellcodezeilen
KNSLOC	Erforderlicher Integrationscode ("glue code") für die Integration der Komponente in Tausenden von Quellcodezeilen
LTEX	Erfahrung des Personals mit den einzusetzenden Werkzeugen und Programmiersprachen
M	Faktor zur Berücksichtigung von Produkt-, Personal-, Hardware- und Projektattributen
PM_{NDI}	Schätzung der gesamten Integrationsarbeit in Personenmonaten
PCAP	Leistungsfähigkeit der Programmierer
PCON	Kontinuität des Personals
PLEX	Vorerfahrung mit der Zielplattform / in der Zieldomäne
PMAT	Reglementierungsgrad der Prozesse der Organisation basierend auf der CMM Skala
PVOL	Volatilität der Entwicklungsplattform
PREC	Erfahrungsgrad einer Organisation mit dem zu schätzenden Projekttyp
RELY	Erforderliche Zuverlässigkeit des Gesamtsystems
RESL	Ausmaß der durchgeführten Risikoanalyse und Architekturkonzeption (Architecture/risk resolution)
RUSE	Erforderlicher Anteil von wiederverwendbaren Komponenten
SCED	Zeitliche Rahmenbedingungen
SITE	Verteilungsgrad des Entwicklungsteams und verfügbare Kommunikationswerkzeuge
Size	Größe der zu entwickelnden Software bzw. Applikation
STOR	Speichereinschränkungen der Entwicklungsplattform
SU	Erforderliches Verständnis der zu integrierenden Komponente
TEAM	Faktor zur Berücksichtigung der Qualität der Zusammenarbeit der Projekt-Akteure
TIME	Randbedingungen hinsichtlich Ausführungszeit
TOOL	Verwendung von Software-Werkzeugen
UNFM	Erfahrung des Programmierers mit der zu integrierenden Komponente

Formelzeichen zur Auswertung der Integrationsplattform

t	Projektmonat
t_d	Arbeitstag
PL	Gesamtlaufzeit des Projekts in Monaten
$d(t)$	Anzahl der Arbeitstage im Projektmonat t
$W_\mathrm{I}(t)$	Integrationsarbeit in Personenmonaten, die im Projektmonat t geleistet werden
$W_\mathrm{I,r}(t)$	Integrationsarbeit, die aus der Ferne geleistet wird
$W_\mathrm{I,l}(t)$	Integrationsarbeit, die lokal geleistet wird (Meetingaufwand)
$W_{\mathrm{I,l_{rel}}}(t)$	Relativer Meetingaufwand pro Eincheckvorgang
$MT_\mathrm{I,l}(t)$	Reine Anwesenheitszeit der Projektpartner während der Integrationstreffen in Manntagen
$M_\mathrm{it}(t)$	Menge der Integrationstreffen in einem Projektmonat t
$n_\mathrm{it}(m_i)$	Teilnehmeranzahl eines Integrationstreffens m_i
$n_\mathrm{c}(t)$	Anzahl der Eincheckvorgänge in das gemeinsame Projektrepository
$n_\mathrm{wp}(t_\mathrm{d})$	Anzahl der Webportalnutzer an einem Tag t_d
$V_\mathrm{I,l}(t)$	Verlustterm zur Berücksichtigung von Teamverlusten
$\bar{f}_v$	durchschnittlicher Verlustfaktor aufgrund von Teamverlusten
k_c	Proportionalitätsfaktor zwischen Integrationsarbeit und Eincheckvorgängen

1 Einleitung

1.1 Motivation und Problemstellung

Serviceroboter dringen mehr und mehr in verschiedene Lebensbereiche vor: professionelle Serviceroboter kommen zur Gebäudereinigung oder als Melkroboter zum Einsatz und Serviceroboter für den häuslichen Bereich sind in Form von autonomen Staubsaugern, Rasenmähern bereits heute im Alltag vieler Menschen angekommen. Der World Robotics Report prognostiziert für die kommenden Jahre weiterhin hohe Wachstumsraten in beiden Bereichen [Int12]. Komplexe Serviceroboter, die mit einem oder mehreren Manipulatoren, einer Vielzahl von Sensoren und einer mobilen Basis ausgestattet sind (z.B. Care-O-bot 3 [RCF+09], Armar [ARA+06], HRP [KKK+04, KAW07], ASIMO [Hon04], iCub [MSV+08], PR2 [Wil10], die DESIRE Technologieplattform [VM12]) und somit potenziell ein weit größeres Aufgabenspektrum abdecken könnten, sind heute jedoch noch als Forschungsplattformen im Einsatz. Bisher konnten diese Serviceroboter trotz jahrzehntelanger Forschung (u.a. [MOR03], [DES09]) noch nicht in die Praxis überführt werden. Doch woran liegt das? Sicher spielen die hohen Kosten solcher Plattformen, ungeklärte Haftungsfragen sowie die unstrukturierten und nicht-abgeschlossenen Umgebungsbedingungen eine Rolle. Eine weitere Ursache für die fehlenden Umsetzungsbeispiele stellen jedoch auch die hohen Aufwände für die Applikationsentwicklung auf komplexen Servicerobotern dar, für die heute meist ein interdisziplinäres Expertenteam erforderlich ist. Nach Fertigstellung der Applikation fällt zudem oft kontinuierlicher Konfigurationsaufwand an, um die Anwendung an sich ändernde Umgebungsbedingungen (z.B. neue Objekte) anzupassen.

Um den Entwicklungsaufwand trotz der hohen Komplexität der Systeme zu reduzieren, werden Anwendungen für komplexe Serviceroboter in der Regel mit Hilfe von komponentenbasierten Entwicklungsmethoden programmiert, die sich durch die Möglichkeit der Wiederverwendung bereits entwickelter Komponenten auszeichnen. In den letzten Jahren sind sehr viele Open-Source Software-Bibliotheken (z.B. OpenCV [Bra02], CARMEN [MRT+08], WEKA [WFT+01], MSL [LaV00]) und Software-Frameworks und Middlewares (z.B. ROS [QGC+09], SmartSoft [LSS09], Player/Stage [GVS+01], ORCA [MBK06], MAIRIE [CLMB07], GOSTAI [GOS03], OROCOS [Bru01], MSRS [Micf], XCF [WFBS04], MCA2 [SKAD01]) so-

wie Architekturen (z.B. [SGH+97], [Gat98], [ACF+98], [AS05], [VNE+00]) für die Robotik entstanden, die einen reichen Fundus an bereits verfügbarer Grundfunktionalität bieten.

Die Integration dieser existierenden Komponenten auf eine Servicerobotik-Zielplattform wird im Allgemeinen jedoch durch unterschiedliche Hardware- und Kommunikationsarchitekturen erschwert. Bestehende Komponenten sind selten plattformunabhängig, sondern meist in die Kommunikationsstruktur der Entwickler eingebettet, so dass sie zur Wiederverwendung zunächst in die neue Middleware portiert werden müssen. Für die Auswahl und Konfiguration der passenden Software-Komponenten sind zudem meist Experten aus vielen verschiedenen Forschungsgebieten (u.a. Bildverarbeitung, Regelungstechnik, Mensch-Maschine Interaktion, Softwaretechnik, Mechatronik, Künstliche Intelligenz, Maschinelles Lernen, etc.) erforderlich, da bereits eine unüberschaubare Anzahl von Komponenten in Open-Source Frameworks existieren, deren Qualität für den Nicht-Fachmann schwer zu bewerten ist. Diese Experten müssen in interdisziplinären, heterogenen und oft räumlich verteilten Teams kooperieren, so dass sich neben den technischen Problemen auch Herausforderungen auf der organisatorischen Ebene ergeben. Typischerweise wird die Applikationsentwicklung für komplexe Serviceroboter zum heutigen Stand im Rahmen von Forschungsprojekten durchgeführt, z.B. in öffentlich geförderten Verbundprojekten internationaler Forschungseinrichtungen, zum Teil mit industrieller Beteiligung. Eine größer angelegte industrielle Forschung und Entwicklung wird momentan auf dem Gebiet Servicerobotik kaum durchgeführt, da die Vermarktungschancen für komplexe Servicerobotersysteme von der Industrie in der nächsten Zukunft noch eher verhalten gesehen werden. In der Automobil- oder Luftfahrtbranche treiben große Standardisierungsgremien mit breiter industrieller Beteiligung die Schaffung von Standards zum leichteren Austausch von Software-Komponenten innerhalb ihrer Domäne voran, z.B. OSEK/VDX [OSE94] (Definition von Standards für Echtzeitsysteme und Kommunikationsstrukturen in der Automobilelektronik), Autosar [AUT04] (Spezifikation einer einheitlichen Software-Architektur mit einheitlichen Beschreibungs- und Konfigurationsformaten für Embedded Software im Automobil). In der (Service-)Robotik fehlt es hingegen an Standardisierung sowohl hinsichtlich einheitlicher Komponentenschnittstellen als auch hinsichtlich standardisierter Entwicklungsprozesse. Der meist sehr hohe Innovationsgrad und die seltene Anwendung von Software-Vorgehensmodellen im Forschungsumfeld erschweren weiterhin einen effizienten Entwicklungsprozess [NRP12].

Eine Trennung der Entwicklerrollen in Komponenten- und Applikationsentwickler sowie Systemintegratoren ist aufgrund der Vermischung der Aspekte Algorithmik, Kommunikation und Konfiguration verbreiteter Komponenten-Frameworks zur Zeit in der Praxis kaum möglich [SSL12], so dass sich die einzelnen Teammitglieder jeweils ein sehr detailliertes Wissen bezüglich ihrer Software-Komponente, der Kommunikationsstruktur der gemeinsamen

Architektur sowie der Zielhardware aneignen müssen und sich oft nicht auf ihr Fachgebiet konzentrieren können. Da sich die Servicerobotik durch eine hohe Abhängigkeit der Komponenten untereinander auszeichnet [SHLS09], ist die Durchführung selbst von einfachen Schnittstellen- und Integrationstests oft nur in Form großer Integrationstreffen möglich. Die Vielfalt der Komponenten bringt es schließlich auch mit sich, dass neue Applikationen für den Serviceroboter nur von einem Expertenteam entwickelt werden können, auch wenn kaum Komponenten von Grund auf neu entwickelt werden müssen. Oft ist selbst der Betrieb des Roboters mit einem hohem Personalaufwand verbunden.

Zusammenfassend lässt sich also feststellen: es gibt heute ein viel größeres Angebot an fertigen Software- und Hardware-Komponenten für die Entwicklung von Servicerobotik-Lösungen als vor ein paar Jahren. Somit ist es aus heutiger Sicht weder wissenschaftlich noch wirtschaftlich sinnvoll, wie früher alle Komponenten des Roboters selbst zu entwickeln. Die Integration dieser existierenden Komponenten zu komplexeren Applikationen ist jedoch aktuell noch sehr aufwändig. Es besteht folglich ein unmittelbarer Bedarf an Integrationstechnologien, die die Entwicklung von Serviceroboter-Applikationen erheblich beschleunigen und deren Konfiguration stark vereinfachen können.

1.2 Zielsetzung und Vorgehensweise

Ziel dieser Arbeit ist die Effizienzsteigerung der Applikationsentwicklung für komplexe Serviceroboter (mit vielen Freiheitsgraden) in interdisziplinären, ggf. räumlich verteilten Entwicklerteams.

Insbesondere sollen Forschungsprojekte im Bereich Servicerobotik unterstützt werden, die durch die folgenden Eigenschaften klassifiziert werden:

- Die Zieldomäne Servicerobotik ist charakterisiert durch einen hohen Integrationsanteil bestehender Software- und Hardware-Komponenten, mit hohen Abhängigkeiten untereinander und weitgehend fehlenden Schnittstellenstandards. Der Konfigurations- und Koordinationsaufwand (inkl. Fehlerbehandlung) der Software-Komponenten ist für die Applikationsentwicklung durch nicht abgeschlossene, unstrukturierte Einsatzumgebungen sehr hoch.
- Die Programmierung erfolgt durch das Paradigma der komponenten-basierten Softwareentwicklung.
- Der Entwicklungsprozess wird durch einen hohen Innovationsgrad mit selten standardisierten Vorgehensmodellen bestimmt.

- Die Entwicklungsteams sind gekennzeichnet durch eine hohe Interdisziplinarität und setzen sich oft aus räumlich verteilten Experten zusammen.

Deshalb sollen in dieser Arbeit möglichst universell einsetzbare Methoden und Werkzeuge konzipiert und entwickelt werden, die nicht auf eine bestimmte Software-Architektur oder Hardwareplattform beschränkt sind und die Effizienz der Applikationsentwicklung durch die folgenden Maßnahmen steigern:

(1) Verbesserung der Rollentrennung von Komponentenentwicklern, Applikationsentwicklern und Systemintegratoren

(2) Verbesserung der Bedienbarkeit von komplexen Servicerobotern und Reduktion des erforderlichen Domänenwissens der einzelnen Entwickler

(3) Strukturierung des verteilten Entwicklungsprozesses für die Entwicklung von Serviceroboter-Applikationen

(4) Steigerung der Effizienz im verteilten Entwicklungsteam durch räumliche und zeitliche Entkopplung der Integrationsaktivitäten.

Zunächst werden dazu in Kapitel 2 Begriffsdefinitionen und Grundlagen der komponentenbasierten Entwicklung, Aspekte der kooperativen Softwareentwicklung und spezifische Eigenschaften der Entwicklungsdomäne Servicerobotik eingeführt. Zur Analyse der Anforderungen an die zu entwerfenden Werkzeuge werden in Kapitel 3 weiterhin vier Szenarien der Applikationsentwicklung für komplexe Serviceroboter definiert: die Applikationsentwicklung in einem Forschungsverbundprojekt, die Nutzung eines komplexen Serviceroboters als zentrale Test- und Benchmarkingstation für Forschungsgruppen, der Fernzugriff auf Robotersysteme für die Fehlerdiagnose und Wartung und die projektübergreifende lokale Nutzung einer komplexen Serviceroboterplattform. Dazu werden Kriterien zur Klassifizierung von Forschungsprojekten der Servicerobotik definiert.

In Kapitel 4 wird der Stand der Technik bezüglich existierender Technologien und Werkzeuge zur effizienten Applikationsentwicklung in der Servicerobotik analysiert und mit den erarbeiteten Anforderungen abgeglichen. Die Konzepte für Werkzeuge zur Unterstützung des verteilten Entwicklungsprozesses und der Integration von komponenten-basierten Applikationen auf komplexen Servicerobotern werden in Kapitel 5 erarbeitet. Unter anderem wird ein Vorgehensmodell für den verteilten Entwicklungsprozess entwickelt und eine generische Beschreibungssprache zur Modellierung von Ressourcenanforderungen von Applikationen an die Zieldomäne Serviceroboter konzipiert. Auf dieser Basis wird weiterhin ein Algorithmus entwickelt, der auf Basis dieser Modelle Deploymenvarianten von komponentenbasierten Applikationen auf komplexe Serviceroboter unter Berücksichtigung der Ressour-

cenanforderungen berechnet. Kapitel 6 umfasst die Präsentation der Realisierung der konzipierten Werkzeuge in Form eines zentralen Webportals.

In Kapitel 7 wird der erreichte Nutzen der entwickelten Werkzeuge in einem großen Verbundforschungsprojekt mit Hilfe von Kostenschätzmodellen aus der Softwaretechnik evaluiert.

2 Grundlagen und Ausgangssituation

Dieses Kapitel ist in zwei Hauptteile gegliedert: Zunächst werden im ersten Teil Begriffsdefinitionen und Grundlagen des Softwareentwicklungsprozesses sowie typische Randbedingungen der Entwicklungsdomäne Servicerobotik präsentiert. Im zweiten Teil folgt die Darstellung der Ausgangssituation in Form von Entwicklungsszenarien.

2.1 Definitionen

Serviceroboter: Nach der ISO-Norm 8737:2012 wird ein *Roboter* definiert als ein "aktuierter Mechanismus, der in zwei oder mehr Bewegungsachsen programmierbar ist, zu einem gewissen Grad Autonomie besitzt und sich in seiner Umgebung bewegt, um vorgesehene Aufgaben zu erfüllen" [Intc]. Ein *Serviceroboter* ist "ein Roboter, der Dienstleistungen für Personen oder Ausrüstungsgegenstände erbringt" [Intc], wobei Anwendungen im Bereich der industriellen Automatisierung explizit ausgeschlossen werden.

Komplexe Serviceroboter: Unter die Definition *Serviceroboter* fallen eine Vielzahl technischer Systeme [Wor11], von einfacheren Systemen mit einer eingeschränkten, speziellen Funktionalität, einer definierten Einsatzumgebung und einer geringen Anzahl von Bewegungsachsen (z.B. Staubsaugerroboter, Rasenmäher, Poolreiniger) bis hin zu mobilen Roboterassistenten (z.B. Care-O-bot 3 [RCF+09]) und humanoiden Robotern (z.B. Armar IV [Kar]) mit einer hohen Anzahl von Bewegungsachsen, die viele verschiedene Dienste unter sich ändernden Umgebungsbedingungen erbringen sollen. Der Begriff *komplexer Serviceroboter* wird in dieser Arbeit verwendet für mobile Serviceroboter mit typischerweise mehr als zehn programmierbaren Bewegungsachsen, die mit Umgebungssensoren und mehreren vernetzten Rechnern ausgerüstet sind.

(Software-)Komponente: Eine Komponente ist nach Szyperski [Szy02] eine softwaretechnische Einheit, die vertraglich zugesicherte Schnittstellen und explizite Abhängigkeiten spezifiziert. Komponenten können unabhängig voneinander auf dem Zielsystem gestartet

werden und werden in der Regel in einem speziellen Applikationskontext von Integratoren zusammengestellt. Komponenten bilden die Grundbausteine für die komponenten-basierte Softwareentwicklung.

Repository: Der Begriff Repository bezeichnet im Allgemeinen ein zentral verwaltetes Verzeichnis zur Speicherung und Beschreibung von digitalen Objekten, meist in Form einer Datenbank [Som06]. In dieser Arbeit werden zwei verschiedene Arten von Repositories unterschieden: das Software-Repository, das der Versionsverwaltung von Quellcode dient und das Komponenten-Repository, das Komponenten als installierbare Software-Pakete bereithält.

Deployment: Deployment bezeichnet nach Richard et al. [HHW99] im Allgemeinen die Aktivitäten nach der eigentlichen Entwicklung einer Komponente bzw. einer Applikation. Diese Aktivitäten umfassen vor allem die Installation, die Konfiguration und Aktivierung der Komponente bzw. der Applikation in der operativen Umgebung des Zielsystems.

Für komponenten-basierte Softwaresysteme im Besonderen umfasst das Deployment nach Heydarnoori [Hey08] die folgende Sequenz von Aktivitäten:

- Bereitstellung der Komponenten als Pakete (*Release*)
- Aufnahme der Komponenten ins Repository (*Acquire*)
- Deployment-Planung für das Zielsystem und die Zielapplikation (*Planning*)
- Installation auf dem Zielsystem (*Install*)
- Konfiguration der Komponenten (*Configure*)
- Aktivierung der Applikation (*Activate*)
- Deaktivierung der Applikation (*Deactivate*)
- Aktualisierung der Komponenten (*Update*)
- Deinstallation der Komponenten (*Uninstall*)
- Löschen aus dem Repository (*Retire*)

Integration: Unter dem Begriff Integration wird im Allgemeinen ein “Prozess der Kombination von Teilen oder Objekten” verstanden (vgl. [T.G07]), die in einer bestimmten

Form zusammenarbeiten und ein neues Ganzes bilden. In der Softwaretechnik wird Integration als die Komposition von Software-Komponenten und Hardware-Komponenten zu einem Gesamtsystem definiert [Intb]. Im Rahmen dieser Arbeit wird der Begriff Integration im Sinne der Kombination von verschiedenen (ggf. teilweise bereits existierenden) Software-Komponenten sowie deren Deployment auf ein gemeinsames Zielsystem verwendet.

Komponenten-basierte Softwareentwicklung: Bei der komponenten-basierten Software-Entwicklung (engl. "component based software development", CBSD) (vgl. [HC01]) wird Software durch die Komposition von Software-Komponenten erstellt. Somit lässt sich CBSD in eine höhere Abstraktionsebene als die prozedurale Programmierung (z.B. C) und die objektorientierte Programmierung (z.B. C++) einordnen. Nach Clements [Cle01] verschiebt sich bei CBSD der Fokus von der Programmierung von neuer Software hin zur Integration von bestehenden Komponenten. Daher nimmt das Deployment von Software-Komponenten eine zunehmend wichtige Rolle im Softwareentwicklungsprozess ein (vgl. [Szy02]). In der Servicerobotik-Entwicklung erfährt die komponenten-basierte Entwicklung eine immer größere Verbreitung [ICO+09], [BKM+05], [Bru01], [SHLS09], da die Wiederverwendung vieler existierender Robotik-Komponenten einen erheblich schnelleren Entwicklungsprozess verspricht.

Vorgehensmodell: In der Systementwicklung und im Software-Engineering sind Vorgehensmodelle von zentraler Bedeutung. Vorgehensmodelle beinhalten eine modellhafte, abstrahierende Beschreibung von Vorgehensweisen, Richtlinien, Empfehlungen oder Prozessen für abgegrenzte Problembereiche und auch für eine möglichst große Anzahl von Einzelfällen. Insbesondere beschreibt ein Vorgehensmodell die Folge bzw. das Bündel aller Aktivitäten (in Phasen), die zur Durchführung eines Software-Entwicklungsprojekts erforderlich sind. Vorgehensmodelle für die Systementwicklung von Informationssystemen und allgemein von Anwendungssystemen beschreiben, wie die Prinzipien, Methoden, Verfahren und Werkzeuge der System- und Software-Entwicklung einzusetzen sind (vgl. Balzert, 2008 [Bal08]).

Komponentenmodell: Nach Sommerville [Som06] beinhaltet das Komponentenmodell eine "Reihe von Standards für die Implementierung, Dokumentation und Bereitstellung von Komponenten. Diese behandeln die einzelnen Schnittstellen, die eine Komponente verfügbar macht, die Benennung von Komponenten, die Zusammenarbeit unter Komponenten sowie die Zusammenstellung von Komponenten. Komponentenmodelle bilden die Basis, von der aus die Middleware die Komponentenausführung unterstützt."

Ontologie: In der Informatik definiert eine Ontologie eine Menge von Repräsentations-Primitiven zur Modellierung einer Wissensdomäne. Die Repräsentations-Primitive sind typischerweise Klassen (oder Mengen), Attribute (oder Eigenschaften) und Beziehungen zwischen den einzelnen Klassen. Ontologien werden in der Regel mit Hilfe von Modellierungssprachen spezifiziert, die abstrakt von Datenstrukturen und Implementierungen sind [Gru09]. Weiterhin sollen Ontologien Modelle der Wirklichkeit generieren, um zu einem gewissen Grad automatisierte Entscheidungsprozesse zu ermöglichen [Hay85].

Software-Architektur: Der Begriff Software-Architektur umfasst konzeptuelle Grundlagen, den allgemeinen Aufbau bzw. die hierarchischen Strukturen eines Softwaresystems und die Zusammenhänge zwischen dessen Teilen (vgl. [Bal08]). Die Software-Architektur ist Teil des Softwareentwurfs, innerhalb dessen sie als Grobgliederung der Komponenten entsteht [BDA+04].

Middleware: Der Begriff Middleware (aus dem Englischen für "Diensteschicht" oder "Zwischenanwendung") oder Vermittlungssoftware bezeichnet in der Informatik "anwendungsneutrale Programme, die so zwischen Anwendungen vermitteln, dass die Komplexität dieser Applikationen und ihre Infrastruktur verborgen werden" [RMB00]. Middlewares ermöglichen somit auf einer höheren Architekturebene die Kommunikation zwischen verteilten Komponenten. Oft wird Middleware in der Robotik als Teil komponenten-basierter Entwicklung gesehen, der die Komposition von Subsystemen in große Gesamtsysteme erleichtert [MAJJ08].

(Software-)Framework: Nach Li et al. [LJH06] spezifiziert ein Software-Framework formal die grundsätzliche Organisation der einzelnen Module in einer Applikation. Dies schließt vor allem einen einheitlichen Kommunikationsmechanismus zwischen den Modulen mit ein. Üblicherweise umfasst ein Software-Framework weiterhin in einer Applikationsdomäne wiederverwendbare Software-Bibliotheken, Hilfsprogramme, Werkzeuge und Programmier-Schnittstellen (engl. *application programming interface*, APIs). Im Unterschied zu Software-Bibliotheken erfolgt die Ablaufsteuerung innerhalb des Programms durch das Software-Framework und nicht durch die aufrufende Routine (engl. "Inversion of control") [Rie00].

Software-Bibliotheken: Nach dem IEEE-Standard 6120.12 (1990) [Intb] wird eine Software-Bibliothek wie folgt definiert: "A controlled collection of software and related documentation designed to aid in software development, use, or maintenance." Software-Bibliotheken stellen also im Allgemeinen Implementierungen für eine Klasse von Problemen

bereit. Diese Implementierungen stehen jedoch im Unterschied zu Komponenten nicht als fertig lauffähige Einheiten zur Verfügung, sondern müssen zunächst in einer Komponente bzw. einem Framework integriert werden, um sie für eine Applikation einsetzbar zu machen. Meist bestehen Bibliotheken aus einer Sammlung von einzelnen Klassen.

Softwaretest: Nach Denert und Siedersleben [DS91] ist der Softwaretest grundsätzlich "der überprüfbare und jederzeit wiederholbare Nachweis der Korrektheit eines Softwarebausteines relativ zu vorher festgelegten Anforderungen." Dabei werden in der Literatur für komponenten-basierte Systeme üblicherweise die folgenden Teststufen unterschieden (vgl. Bernhard und Breiteneder, 2009 [BB09]): (1) Komponententests, (2) Integrationstests, (3) Sytemtests und (4) Applikations- bzw. Abnahmetests.

Zur Überprüfung der Funktionalität einer Komponente unabhängig vom Zielsystem dienen **Komponententests**, die in der Regel isoliert durchgeführt werden. Dazu ist in der Regel eine entsprechende Testumgebung erforderlich, die entsprechende Testfälle, Testdaten und ggf. die Simulation einer Hardware-Komponente bereitstellt. Bei **Integrationstest** werden vor allem Schnittstellen zwischen Komponenten und die Ergebnisse über komplette Abläufe geprüft. Weiterhin wird die korrekte Interaktion mit dem Zielsystem (z.B. bzgl Betriebs- oder Dateisystem) sichergestellt. **Applikations- oder Systemtests** schließlich überprüfen die Erfüllung der Anforderungen an das Gesamtsystem bzw. die Zielapplikation. Ist bei den Applikationstests der Anwender bzw. Kunde involviert, spricht man auch von **Abnahmetests**.

2.2 Grundlagen

2.2.1 Integrationsmodelle und -strategien

Das Integrationsmodell beschreibt die technische Vorgehensweise bei der Einbindung existierender Softwarebausteine in eine neue Zielanwendung. In der Literatur wird das enggekoppelte und das lose gekoppelte Integrationsmodell unterschieden [Nes07]. Im **enggekoppelten Integrationsmodell** werden alle für eine Zielapplikation erforderlichen Algorithmen so umgeschrieben (engl. "refactored"), dass sie dieselbe Kommunikationsstruktur, Architektur und dieselben Datentypen verwenden. In der Regel beinhaltet das enggekoppelte Integrationsmodell die Implementierung aller Algorithmen in ein domänenspezifisches Framework und die Verwendung eines gemeinsamen Komponentenmodells. Der Integrationsaufwand bei diesem Modell ist relativ hoch, da bereits existierende Komponenten nicht verwendet werden können bzw. komplett neu geschrieben werden müssen. Der

Vorteil dieses Modells liegt vor allem in der hohen Effizienz, da durch die Verwendung derselben Datentypen wenig Typkonversionen durchgeführt werden müssen. Der integrierte Code ist somit speziell an die Zielapplikation angepasst und kann daher in der Regel nur schwer für andere Applikationen wiederverwendet werden.

Beim **lose gekoppelten Integrationsmodell** werden die Algorithmen (in Objekten oder Komponenten) gekapselt und nicht für ein dediziertes Framework umgeschrieben. Für die Kommunikation zwischen den einzelnen Komponenten müssen entsprechend die Datenformate angepasst werden. Der Vorteil dieses Modells besteht in der schnelleren Integration von Subsystemen an räumlich verteilten Stellen, da der Restrukturierungsaufwand zur Erfüllung der Vorgaben des Frameworks wegfällt und die einzelnen Komponenten weitgehend architektonisch unabhängig voneinander entwickelt werden können. Nachteilig wirkt sich hier jedoch die Notwendigkeit häufiger Datentypkonversionen aus, da die Komponenten intern unterschiedliche Typen verwenden können. Dieses Integrationsmodell empfiehlt sich vor allem für Systeme mit kleinen Komponenten-Schnittstellen und wenig Datenaustausch.

Die Integrationsstrategie beschreibt hingegen die Methodik und zeitliche Vorgehensweise für die Kombination der Einzelkomponenten zum Gesamtsystem. Mit der Größe des Integrationsumfangs nimmt die Problematik der "Isolation von Fehlerwirkungen" im Gesamtsystem zu. Oft werden daher inkrementelle Integrationsstrategien angewandt, um das Risiko einer späten Fehlerfindung zu reduzieren [BB09].

Grundsätzlich lassen sich die folgenden Strategien unterscheiden:

- Top-Down Integration
- Bottom-Up Integration
- Big-Bang Integration

Bei der Top-Down Integration werden zunächst die oberen Schichten (z.B. Benutzerschnittstellen) der einzelnen Komponenten integriert und getestet, wobei die unteren Schichten und die eigentliche Funktionalität simuliert werden. In den weiteren Integrationsschritten werden dann die darunterliegenden Schichten implementiert. Der Vorteil dieser Methode liegt in der sehr frühen Verfügbarkeit der Komponenten-Schnittstellen und damit einer frühen Durchführbarkeit von Systemtests. Bei der Bottom-Up Strategie werden hingegen zunächst die unteren Schichten (z.B. Hardware-Treiber) implementiert. In jedem Integrationsschritt müssen daher dedizierte "Testtreiber" für die jeweilige Schicht erstellt werden. Im Vegleich zur Top-Down Integration entfällt die Erstellung von Simulationen für noch nicht implementierte Schichten. Systemtests können erst durchgeführt werden, wenn die oberste Schicht der Komponenten integriert wurde und diese für Tests zur Verfügung steht.

Die Big-Bang Integration ist im Unterschied zu den beiden vorhergehenden Strategien ein nicht-iteratives Verfahren: Alle Komponenten werden getrennt voneinander implementiert und getestet und erst bei der Verfügbarkeit aller Komponenten gleichzeitig integriert. Aus diesem Grund können die Integrations- sowie Systemtests erst sehr spät im Entwicklungsprozess durchgeführt werden. Weiterhin ist die Lokalisierung der Fehler schwieriger als bei den iterativen Integrationsverfahren. Auf der anderen Seite müssen keine Simulationen oder Testtreiber in frühen Phasen des Entwicklungsprozesses geschrieben werden.

2.2.2 Kooperative Softwareentwicklung

Nach [T.G07] werden die Begriffe und Wirkungszusammenhänge der Kommunikation, Koordination und Kooperation wie folgt definiert:

Kommunikation umfasst den zwischen Kommunikationspartnern stattfindenden Prozess der Übermittlung und des Austausches von Informationen. Dient der Austausch von Informationen zur Abstimmung von Aktivitäten zwischen den Gruppenmitarbeitern, so bildet diese Art der Kommunikation die Grundlage für die Koordination arbeitsteiliger Prozesse.

Koordination beruht auf geeigneten Kommunikationsprozessen und umfasst alle Tätigkeiten, die zur Abstimmung von arbeitsteiligen Aufgaben im Rahmen eines Arbeitsprozesses notwendig sind. Dies umfasst u.a. die Terminkoordination, Vorgangsteuerung und zentrale Koordination von unstrukturierten Abläufen (vgl. auch [SSU01]).

Kooperation bezeichnet jene Koordination, die zur Vereinbarung gemeinsamer Ziele und zur aufeinander abgestimmten Erreichung eines gemeinsamen Arbeitsergebnisses zwischen den beteiligten Personen notwendig ist. Die Ausführungen zum Begriff der Kooperation zeigen, dass die Gestaltung, Abstimmung und Steuerung arbeitsteiliger Aufgaben auf der Basis von Kommunikations- und Koordinationsvorgängen stattfindet.

Eine wichtige Rolle spielen auch sogenannte “Awareness”-Funktionen, die z.B. aktuelle Ereignisse als Informationen – nach Filterung hinsichtlich Interesse, Privatsphäre oder sonstigen Projektregularien – anderen zur Verfügung stellen. Beispiele für solche Funktionalitäten sind z.B. Gruppeneditoren, die bei gleichzeitigem Zugriff auf Dokumente entsprechende Mitteilungen an die jeweiligen Akteure senden, oder News-Feeds oder Blogs, die z.B. die Verfügbarkeit einer neuen Softwarerevision bekannt machen.

2.2.3 Modelle zur Aufwandsschätzung für die Entwicklung und Integration von Software-Komponenten

Die Aufwandsschätzung von Softwareentwicklung hat in den letzten Jahrzehnten mehr und mehr an Bedeutung gewonnen [BAC00], da Software-Projekte in der Vergangenheit oft zu gering budgetiert wurden. Die besondere Herausforderung bei der Erstellung von geeigneten Modellen zur Schätzung von zu erwartenden Projektkosten liegt in der hohen Diversität der einzelnen Projekte sowie im großen Aufwand für die Erhebung objektiver empirischer Daten.

Auch wenn die Kostenschätzung von Softwareprojekten mit Hilfe von algorithmischen Modellen stark von der Qualität und Quantität verfügbarer Kalibrierdaten abhängt und in der Praxis in der Regel nur von großen Unternehmen eingesetzt werden kann, die sich eine umfangreiche Erhebung von lokalen Daten leisten können (vgl. [Som06]), bieten diese Modelle Aufschluss über Schlüsselfaktoren und Kostentreiber von großen Softwareprojekten.

Eines der am häufigsten angewandten Schätzmodelle ist das COCOMO Modell, das 1981 in seiner ersten Version eingeführt [Boe81] und im Laufe von zwei Jahrzehnten zu COCOMO II [BCH+95], [BAB+00] verfeinert und auf aktuelle Software-Vorgehensmodelle (u.a. das Spiralmodell und die inkrementelle Entwicklung) angepasst wurde.

In [ABB+00] wird das Standardverfahren COCOMO II zur Schätzung von Entwicklungskosten durch das COCOTS Kostenmodell für die Einbindung von existierenden „commercial-off-the-shelf“ (COTS) Komponenten in das Zielsystem erweitert. COTS Komponenten sind dadurch charakterisiert, dass dem Applikationsentwickler in der Regel kein Quellcode vorliegt und die zukünftige Entwicklung nicht vom Applikationsentwickler gesteuert werden kann. Diese Eigenschaften lassen sich auch auf Komponenten übertragen, die in großen Forschungsprojekten entwickelt bzw. bereitgestellt werden. Für die Integration von existierenden Komponenten, deren Quellcode verfügbar ist (engl. “non-developmental items”, NDI), entwickelten [AB97] basierend auf COCOMO II ein weiteres Schätzmodell, die “NDI Software Integration” Kostenschätzung.

Tabelle 2.1: Überblick über verschiedene Methoden zur Aufwandsschätzung für die Entwicklung und Integration von Software-Komponenten

	Wiederverwendung existierender Komponenten	Neuentwicklung von Komponenten
Komponenten-Quellcode verfügbar	NDI Software Integration	COCOMO II
Closed-Source Komponenten	COCOTS	COCOMO II

Tabelle 2.1 gibt einen Überblick über die beschriebenen Schätzmethoden und ihre Anwendung. Generell beinhalten die Modelle nur Schätzformeln für den Personalaufwand eines Projektes, so dass Kosten für Hardware oder Software und Reisekosten nicht enthalten sind. In den meisten Projekten stellen jedoch die Personalkosten den Hauptkostenfaktor dar.

2.3 Szenarien für verteilte Integration und Tests

Als Basis für die Anforderungsanalyse werden im Folgenden typische Szenarien für die Entwicklung von Serviceroboter-Applikationen dargestellt, die sich hinsichtlich organisatorischer und technischer Rahmenbedingungen, z.B. der räumlichen Verteilung der Entwickler, unterscheiden. Die Szenarien bewegen sich entsprechend der Ausrichtung der Arbeit im Forschungsumfeld (vgl. Kap.1):

- Verbund-Forschungsprojekt im Bereich Servicerobotik: Nutzung einer gemeinsamen Hardware im verteilten Projektteam
- Servicerobotik Teststand: Referenzplattform für "Benchmarking" und Verifikation
- Diagnose und Fernwartung von Serviceroboter-Plattformen
- Projektübergreifende lokale Nutzung einer komplexen Serviceroboterplattform

Vor der detaillierten Beschreibung dieser Szenarien werden im nächsten Abschnitt formale Parameter zur Spezifikation der Szenarien eingeführt.

2.3.1 Definition von Parametern für verteilte Entwicklungsszenarien

Im Folgenden wird vereinfachend von zwei grundsätzlichen Rollen ausgegangen: den Entwicklern, die keine detaillierten Kenntnisse des Hardwareaufbaus besitzen (d.h. Komponenten- und Applikationsentwickler) und den Systemintegratoren, die mit der Hardware vertraut sind und das Deployment von Software-Komponenten auf der Hardware durchführen bzw. steuern können. Sei SI die Menge aller beteiligten Systemintegratoren, $E = \{e_1, e_2, ..., e_n\}$ die Menge aller beteiligten Komponenten- und Applikationsentwickler, SR die Menge aller verfügbarer Testsysteme und L_{SR}, L_E und L_{SI} die Mengen der Standorte der Serviceroboter, Entwickler und Systemintegratoren. Damit wird $E_l \subset E$ definiert als die Menge aller "lokalen" Entwickler, die einen Roboter an ihrem Standort haben und $E_r = E \backslash E_l$ die Menge der "entfernten" Entwickler, für die dies nicht der Fall ist. Analog werden SI_l und SI_r definiert.

Weiterhin spielen die Abhängigkeiten zwischen den jeweiligen Komponenten der einzelnen Entwickler eine große Rolle. Die Funktion $d(e_i; e_j)$ beschreibt binär das Abhängigkeitsverhältnis zweier Entwickler e_i, e_j:

$$d(e_i; e_j) = \begin{cases} 1, & \text{falls eine oder mehrere Komponenten von } e_i \\ & \text{Ergebnisse einer oder mehrerer Komponenten von } e_j \\ & \text{als Eingangsgröße benötigt} \\ 0, & \text{andernfalls.} \end{cases} \tag{2.1}$$

Die Korrelations- oder Abhängigkeitsmatrix D, aufgebaut aus den einzelnen Abhängigkeitsverhältnissen der einzelnen Komponenten, beschreibt die Verflechtung des Gesamtsystems und $d_g \in [0, 1]$ ein normiertes Abhängigkeitsmaß über alle Entwickler hinweg:

$$D = (d(e_i; e_j))_{i,j} \tag{2.2}$$

$$d_g = \frac{|E| - \mathrm{Rg}(D)}{|E| - 1}, |E| > 1. \tag{2.3}$$

Sind die einzelnen Komponenten und somit die entsprechenden Entwickler beispielsweise komplett unabhängig voneinander (Rang der Abhängigkeitsmatrix $\mathrm{Rg}(D) = |E|$), beträgt $d_g = 0$. Bestehen im anderen Extremfall von jeder Komponente Abhängigkeiten zu allen übrigen Komponenten (Rang der Abhängigkeitsmatrix $\mathrm{Rg}(D) = 1$), ergibt $d_g = 1$.

Eine weitere wichtige Fragestellung für die Integration, das Deployment und den Test der einzelnen Komponenten sowie die Durchführung von System- und Applikationstests auf dem Zielsystem bezieht sich auf die Nutzung von Software-Frameworks und Komponentenmodellen: Existieren innerhalb eines Szenarios mehrere Software-Frameworks oder kann mit einem gemeinsamen Referenzframework gerechnet werden? Deshalb wird als weiteres Kriterium der boolesche Ausdruck $RF \in \{\text{ja,nein}\}$ für die Existenz eines Referenzframeworks eingeführt.

In Tabelle 2.2 werden die Integrations- und Testaktivitäten schließlich hinsichtlich ihrer räumlichen und zeitlichen Kopplung kategorisiert. Die räumliche Kopplung wird dabei relativ zum Zielsystem ausgedrückt. Der Grad der Kopplung korreliert dabei zu einem gewissen Maß mit der Abhängigkeit der Komponenten.

Zur Kategorisierung des Innovationsgrads und der Standardisierung der Entwicklungsprozesse wird auf Definitionen im Kostenschätzmodell COCOMO II [BCH+95] (vgl. auch Abschnitt 2.2.3) zurückgegriffen: Der Parameter *Precedentness* (*PREC*) beschreibt den Erfahrungsgrad der beteiligten Akteure mit dem Projekttyp und der Anwendungsdomäne und der Parameter *Process Maturity* (*PMAT*) beschreibt den Reglementierungsgrad der Pro-

Tabelle 2.2: Raum-Zeit-Klassifizierung für die Durchführung von Integration und Test am Zielsystem

Zeit / Ort	synchron	asynchron
lokal	lokale Integration in Form von Integrationstreffen aller Entwickler.	Integrationsaktivitäten finden direkt am Zielsystem, in Bezug auf die einzelnen Entwickler jedoch zeitlich entkoppelt statt.
verteilt	Integrationsaktivitäten werden in Form von virtuellen Integrationstreffen durchgeführt, bei denen einzelne oder alle Entwickler sich entfernt (z.B. durch ein VPN) verbinden.	Integrationsaktivitäten können räumlich und zeitlich entkoppelt von den Entwicklern separat durchgeführt werden.

zesse der Organisation basierend auf der CMM Skala [PCCW95]. Der Wertebereich der beiden Parameter wird dabei qualitativ ausgedrückt (*sehr niedrig – sehr hoch*).

Diese Parameter werden im Folgenden zur Abgrenzung der betrachteten Szenarien verwendet.

2.3.2 Szenario 1: Verbund-Forschungsprojekt

In großen Forschungsprojekten ist die Anzahl der Entwicklerstandorte in der Regel höher als die Anzahl der verfügbaren Hardwareplattformen, d.h. nicht an jedem Standort ist das Zielsystem verfügbar. Immer häufiger ist es Vorgabe in Robotik-Forschungsprojekten, alle Teilergebnisse zwecks eines möglichst hohen Verwertungspotenzials im Hinblick auf die Umsetzung in alltagstaugliche Applikationen auf einem gemeinsamen Demonstrator zu integrieren (z.B. DESIRE [DES09], SRS [SRS13], ACCOMPANY [ACC11]). Üblicherweise stehen einer geringen Anzahl von Systemintegratoren $|SI|$ eine große Anzahl von Komponentenentwicklern $|E|$ entgegen, die sich mit dem Gesamtsystem nicht im Detail auskennen bzw. auseinandersetzen können. Die einzelnen Komponentenentwickler e_i befinden sich an räumlich verteilten Standorten L_{e_i}, die Systemintegratoren in der Regel am Standort des Zielsystems. Dieses Szenario ist dabei nicht auf die Existenz nur einer Projektplattform beschränkt. Es wird aber angenommen, dass die Anzahl der Entwicklerstandorte $|L_E|$ deutlich größer als die Anzahl der Roboterstandorte $|L_{SR}|$ ist.

- $|E_l| < |E_r|$
- $|SI_l| > |SI_r|$
- $d_g > 0,7$ (typisch)
- $20 < |E| < 30$
- $RF = ja$
- $PMAT$: typischerweise *sehr niedrig* bis *niedrig*
- $PREC$: typischerweise *niedrig*
- räuml./zeitl. Kopplung der Integrationsaktivitäten: lokal/synchron (typisch)

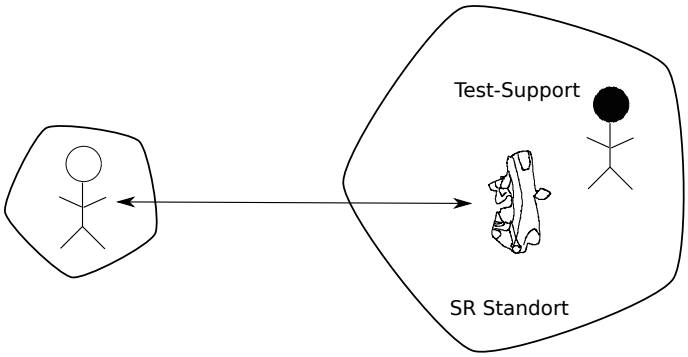
Abbildung 2.1: Klassifikation des Verbund-Forschungsszenarios

Für eine integrierte Zielapplikation auf einer komplexen Serviceroboter Plattform sind die Abhängigkeiten zwischen den einzelnen Komponenten in der Regel sehr hoch. Daher finden Integrations- und Applikationstests typischerweise in Form von lokalen Integrationstreffen statt. Dafür kann jedoch davon ausgegangen werden, dass die Entwicklung der Zielapplikation auf einem gemeinsamen Software-Framework basiert. Die Parameter des Verbund-Forschungsszenarios sind in Abbildung 2.1 zusammengefasst.

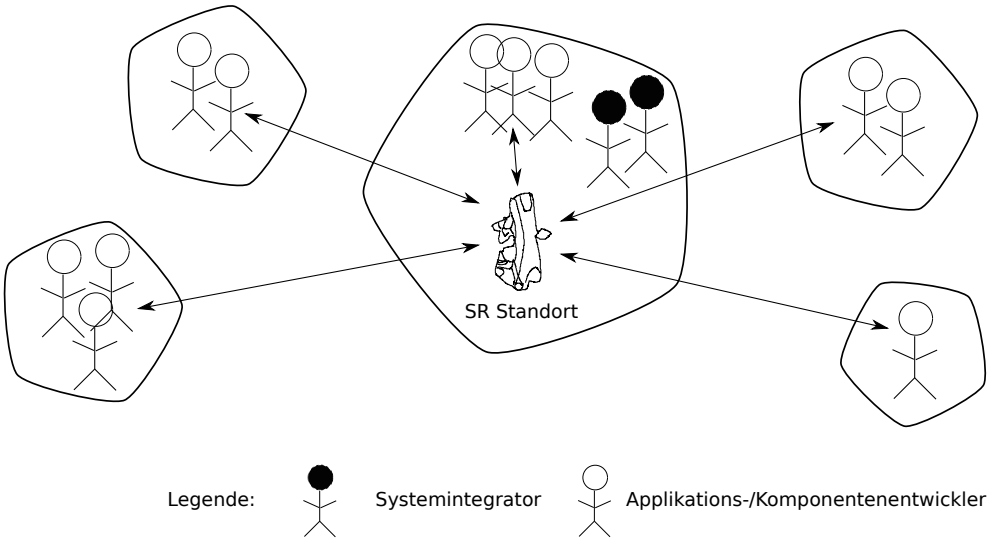
Für die Integrations- und Testphase ergeben sich demnach die in Abbildung 2.2 dargestellten Test-Situationen. Zum einen müssen einzelne Komponententests durchgeführt werden, die nur einzelne Entwickler und Systemintegratoren betreffen, sowie Tests des Gesamtsystems bzw. Applikationstests, die jeweils einen Großteil der Entwickler involvieren.

2.3.3 Szenario 2: Servicerobotik-Teststand

Das Thema Benchmarking (sowohl einzelner Roboterkomponenten als auch gesamter Applikationen) gewinnt in den letzten Jahren mehr und mehr an Bedeutung innerhalb der Servicerobotik-Forschung, da die Auswertung und der Vergleich einer Vielzahl unterschiedlicher Algorithmen und Komponenten für den Fortschritt in vielen Anwendungsgebieten unerlässlich ist (s. [MLKN09] für einen Überblick über Benchmarking-Aktivitäten im Bereich Robotik). Oft werden die Ergebnisse neuer Ansätze jedoch bezüglich einer bestimmten Software- und Hardwareplattform veröffentlicht, so dass für einen Vergleich eine Portierung auf das lokal verfügbare Framework bzw. die verfügbare Hardware erforderlich ist. Aufgrund des hohen Aufwands für diese Portierungen werden solche Vergleichsmessungen aber nur selten durchgeführt. Benchmarking-Einrichtungen, die komplexe Serviceroboter-Plattformen sowie eine entsprechende Umgebung für Messungen und Tests zur Verfügung stellen, können den erforderlichen Aufwand reduzieren und haben das Potenzial, Referenzen zu schaffen (vgl. [BRPH09]). Ein Beispiel für eine solche Einrichtung (im Bereich Lokomotion) wird in [FMSOK08] gegeben.



(a) Verteilter Test einzelner Komponenten



(b) Verteilte Integrations-/Applikationstests des Gesamtsystems

Abbildung 2.2: Typische Verteilung der Akteure für ein Komponententests (a) sowie Integrations- und Applikationstests (b) innerhalb eines Verbund-Forschungsprojekts mit einer gemeinsamen Entwicklungsplattform.

Das Szenario Servicerobotik-Teststand knüpft an die Idee einer solchen Einrichtung an. Der Teststand soll einer breiteren Masse von Entwicklern Test und Verifikation von Komponenten und Applikationen ermöglichen, die keinen lokalen Zugang zu einem komplexen Serviceroboter inklusive einer aufwändigen Testumgebung haben. Im Gegensatz zum vorigen Szenario sind die beteiligten Akteure nicht in einem Projektkontext verbunden – deshalb sind *PMAT* und *PREC* nicht anwendbar – und es wird davon ausgegangen, dass geringe oder keine Vorkenntnisse über die bereitgestellte Hardware bestehen.

Dennoch sollen diese Entwickler aus der Ferne mit möglichst geringem Aufwand und mit möglichst geringer Unterstützung durch lokale Systemintegratoren in der Lage sein, systematische Tests auf der Hardware durchzuführen (s. Abbildung 2.4). Idealerweise sollen für die Portierung der zu testenden Komponenten möglichst wenig Rahmen- bzw. Zwangsbedingungen existieren, so dass im Extremfall sogar deren komplette Software-Umgebung

- $|E_l| \ll |E_r|$
- $|SI_l| \gg |SI_r|$
- $d_g > 0,7$ (typisch)
- $|E| < 10$
- $RF = nein$
- $PMAT$: n.a.
- $PREC$: n.a.
- räuml./zeitl. Kopplung der Integrationsaktivitäten: lokal/synchron (typisch)

Abbildung 2.3: Klassifikation des Servicerobotik-Teststand Szenarios

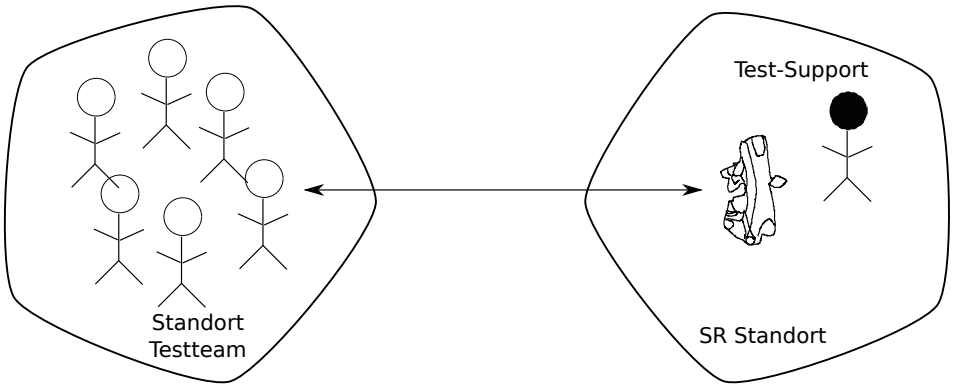


Abbildung 2.4: Typische Verteilung der Akteure im Szenario Servicerobotik-Teststand

(Architektur, Middleware, Framework) auf die Roboterhardware mitportiert werden kann. Voraussetzung dazu ist natürlich eine entsprechende Konfigurierbarkeit der Komponenten, so dass diese auf das Ziel-System angepasst werden können (z.B. hinsichtlich Kinematik, Kalibrierung, Sensorparameter etc.). Für die externen Rahmenbedingungen der durchzuführenden Tests können z.B. lokale Hilfskräfte sorgen (Einrichtung der Umgebung, Überwachung und Protokollierung der Tests, etc.).

Das Szenario ist weiter spezifiziert durch ein (örtlich konzentriertes) Entwicklerteam, das typischerweise im Rahmen eines (virtuellen) Integrationstreffens aus der Ferne oder auch direkt auf die Hardware zugreift.

Die Klassifikation des Szenarios ist in Abbildung 2.3 dargestellt.

2.3.4 Szenario 3: Diagnose und Fernwartung

In den letzten Jahren werden immer häufiger komplexe Serviceroboter-Plattformen zum Verkauf angeboten und anderen Forschungseinrichtungen zur Verfügung gestellt [WRPV10], z.B. PR2 von WillowGarage [Wil10] oder Care-O-bot 3 vom Fraunhofer IPA [Fra10]. Im

- $|E_l| \gg |E_r|$
- $|SI_l| \ll |SI_r|$
- $d_g > 0,5$ (typisch)
- $|E| < 10$
- $RF = nein$
- $PMAT$: n.a.
- $PREC$: n.a.
- räuml./zeitl. Kopplung der Integrationsaktivitäten: n.a.

Abbildung 2.5: Klassifikation des Szenarios Diagnose und Fernwartung

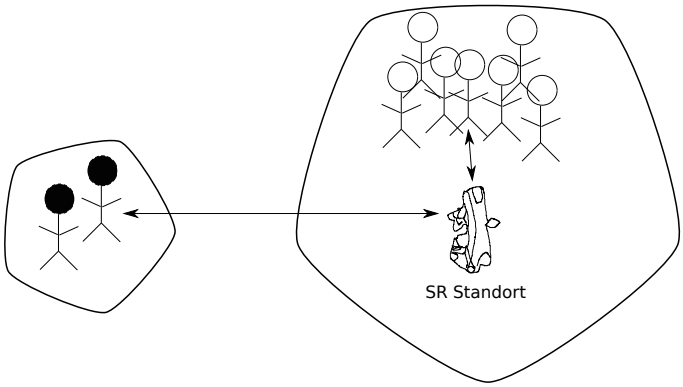


Abbildung 2.6: Typische Verteilung der Akteure im Szenario Diagnose und Fernwartung

Zuge dessen muss den Käufern solcher Plattformen eventuell Hilfestellung bei Funktionsstörungen oder Problemen geleistet werden. Dabei ist es wünschenswert, dass ein möglichst großer Anteil dieser Hilfestellung aus der Ferne erfolgen kann, um Kosten und Zeitaufwände zu minimieren.

Das im Folgenden beschriebene Fernwartungsszenario entspringt dieser Motivation und ist vor allem dadurch charakterisiert, dass sich die Systemintegratoren nicht am Standort des Serviceroboters befinden. Probleme mit bestehenden Hardware- oder Software-Komponenten sowie Schwierigkeiten bei der Integration von neuen Komponenten müssen soweit möglich durch die Integratoren aus der Ferne gelöst werden (s. Abbildung 2.6). Das Szenario umfasst auch den Fall, dass sich weitere Komponentenentwickler von anderen Standorten dazuschalten, um bei der Problemlösung bezüglich ihrer Komponente zu helfen. Oft ist sowohl die Problembeschreibung als auch die Fehlerdiagnose per E-Mail oder Telefon mühsam und im Vergleich zu einem Direktzugriff oft zeitintensiver. Dies trifft insbesondere bei abstrakteren Fehlerbeschreibungen und Verhaltensfehlern zu (z.B. “Roboter findet den Weg in die Küche nicht”). Es muss weiterhin davon ausgegangen werden, dass die “Forschungskunden” ein für die jeweilige Forschungsausrichtung spezifisches Framework auf der Serviceroboter Plattform verwenden. Da das Szenario nicht zwingend einen Projektkontext voraussetzt, sind $PMAT$ und $PREC$ nicht spezifiziert (vgl. Abbildung 2.5).

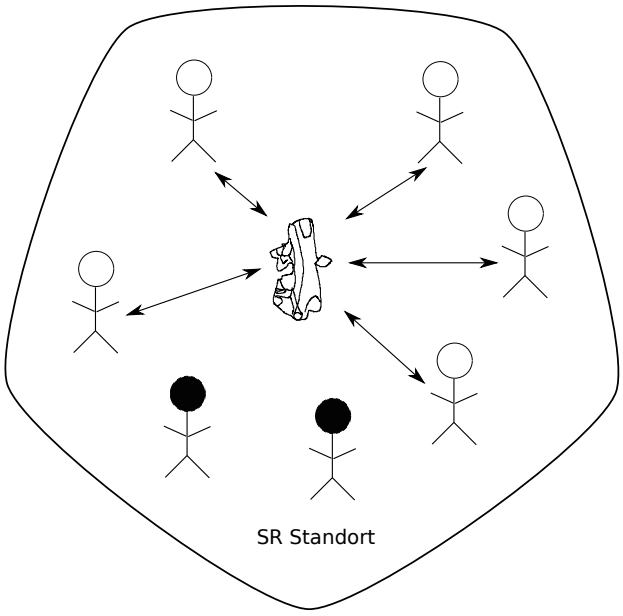


Abbildung 2.7: Verteilung der Akteure im Szenario projektübergreifende lokale Nutzung

2.3.5 Szenario 4: Projektübergreifende lokale Nutzung

Das letzte Szenario beschreibt den Fall, dass sich alle Entwickler und Systemintegratoren am Standort des Serviceroboters befinden bzw. am Standort verfügbar sind (s. Bild 2.7). Dies gilt zum Beispiel für ein lokales Projektteam innerhalb eines unternehmens- bzw. institutsinternen Entwicklungsprojektes. Die einzelnen Entwickler sind oft über verschiedene Gebäude verteilt und der Zugriff auf das Zielsystem während Tests muss ebenfalls koordiniert werden.

Oft arbeiten diese lokalen Teams zwar auf derselben Hardware, aber nicht notwendigerweise in demselben Projektkontext. Üblicherweise wird in jedem Projekt ein eigenes Software-Repository unterhalten, so dass sich die Integrationsarbeit für die Systemintegratoren oft auf mehrere Repositories bezieht. Auch wenn nur ein Repository verwendet wird, arbeiten die einzelnen Mitglieder lokaler Teams oft auf eigenen Softwarezweigen, um eine engere Zusammenarbeit an denselben Software-Komponenten bzw. Applikationen zu ermöglichen. Dadurch ergibt sich die Situation, dass Tests auf der Roboterhardware auf Basis unterschiedlicher Software-Repositories oder verschiedener Software Zweige desselben Repositories durchgeführt werden müssen. Die Vorbereitung der Hardware für die jeweiligen Tests kann daher recht aufwändig sein und schließt in der Regel das Aufspielen und Bauen des kompletten Softwarezweiges, der die zu testenden Komponenten enthält, auf die Rechner des Zielsystems mit ein. Die Integrationsaktivitäten werden in der Regel lokal in Form von Treffen durchgeführt, da alle involvierten Entwickler vor Ort sind. Eine zeitliche Entkopplung ist jedoch auch für dieses lokale Szenario relevant, da die verschiedenen Entwickler

- $|E_r| = 0$
- $|SI_r| = 0$
- $d_g > 0,5$ (typisch)
- $|E| > 10$
- $RF = ja$
- $PMAT$: n.a.
- $PREC$: n.a.
- räuml./zeitl. Kopplung der Integrationsaktivitäten: lokal/synchron und asynchron

Abbildung 2.8: Klassifikation des Szenarios projektübergreifende lokale Nutzung

oft in mehrere Projekte gleichzeitig eingebunden sind, so dass vollzählige Treffen terminlich häufig nicht zu realisieren sind. Formal lässt sich dieses Szenario wie in Abbildung 2.8 klassifizieren.

2.3.6 Zusammenfassung

Tabelle 2.3 gibt einen Überblick über die beschriebenen Szenarien im Bezug auf die in Abschnitt 2.3.1 definierten Kriterien (die Parameter $PMAT$, $PREC$ und die zeitliche und räumliche Entkopplung werden nicht aufgeführt, da sie nicht für alle Szenarien anwendbar sind).

Tabelle 2.3: Überblick über die verschiedenen Testszenarien

Kriterium / Szenario	Räumliche Verteilung der Entwickler E	Räumliche Verteilung der Integratoren SI	Anzahl der Entwickler $\|E\|$	Abhängigkeiten der Entwickler d_g	Referenzframework FR
Forschungspr.	$\|E_l\| < \|E_r\|$	$\|SI_l\| > \|SI_r\|$	20-30	>0,7	ja
Teststand	$\|E_l\| \ll \|E_r\|$	$\|SI_l\| \gg \|SI_r\|$	<10	>0,7	nein
Fernwartung	$\|E_l\| \gg \|E_r\|$	$\|SI_l\| \ll \|SI_r\|$	<10	>0,5	nein
Lokale Nutzung	$\|E_r\| = 0$	$\|SI_r\| = 0$	10-20	>0,5	ja

3 Analyse der Aufgabenstellung und Ableitung von Anforderungen

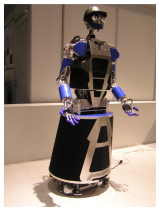
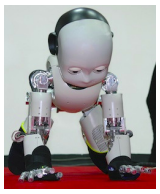
Um die Anforderungen zur Lösung der Problemstellung (s. Abschnitt 1.1) abzuleiten, wird in diesem Kapitel zunächst eine detaillierte Betrachtung der erforderlichen Arbeitsschritte für die Entwicklung einer typischen Servicerobotik-Anwendung durchgeführt. Daraufhin werden die speziellen Anforderungen eines verteilten Entwicklungsprozesses durchgeführt und die Referenzszenarien 2.3.2 - 2.3.5 auf weitere Anforderungen untersucht. Schließlich werden die Anforderungen strukturiert und einer Gesamtarchitektur zugeordnet.

3.1 Analyse der Integrations- und Test-Aktivitäten von komponenten-basierten Applikationen für die Entwicklungsdomäne Servicerobotik

Nach [ICO+09], [SHLS09], [SK08] ist die Entwicklungsdomäne Servicerobotik vor allem charakterisiert durch die Kombination eines komplexen mechatronischen Systems mit einer meist umfangreichen Software, bedingt durch die typischerweise unstrukturierten und nicht-abgeschlossenen Einsatzumgebungen, eine hohe Varianz bezüglich der zu erkennenden und zu handhabenden Objekte sowie die dynamische Mensch-Roboter-Interaktion.

In Tabelle 3.1 ist der Hardwareaufbau für typische Vertreter der Domäne komplexer Serviceroboter dargestellt. Charakteristisch ist die hohe Anzahl von Aktor- und Sensor-Komponenten, die in der Regel über verschiedene Feldbussysteme (z.B. CAN, Sercos, Ethernet, Ethercat) oder serielle Kommunikation (USB, RS232, RS422, RS485, etc.) an das Rechnernetz angebunden sind. Meist besteht das Rechnernetz aus mehreren pentium-basierten PCs, auf denen zum Teil unterschiedliche Betriebssysteme laufen (Linux, Windows, Echtzeitbetriebssysteme).

Tabelle 3.1: Hardware-Aufbau komplexer Serviceroboter, Beispiele Care-O-bot 3 [RCF+09], PR2 [Wil10], Armar-III [ARA+06], DESIRE TP [VM12], iCub [MSV+08]

		 Care-O-bot 3	 PR2	 Armar-III	 DESIRE TP	 iCub
Rechnereinheiten	Anz. int./ext.	3	2	5	8 / 1	1 / 2-3
	CPU	Core2Duo 2,26 GHz	2 quad-core Nehalem Prozessoren	1-2 GHz PC, PC104	Core2Duo 2,16 GHz	PC104 + external PCs
	RAM	4 GB	24 GB	n.a.	2 GB	n.a.
	Komm.-schnitt-stellen	Ethernet, USB, CAN, seriell	Ethernet	Ethernet, USB, CAN	Ethernet, USB, Firewire, CAN	Ethernet, USB, Firewire, CAN
	Betriebs-system	Linux, Windows	Linux	Linux	Linux, Windows	Linux, Windows
Aktoren	Anz. DOF	27	27	43	50	53
	Kompo-nententyp	1 Arm, 1 Hand, Torso, Basis	2 Arme, 2 Greifer, Torso, Kopf, Basis	2 Arme, 2 Hände, Kopf, Basis	2 Arme, 2 Hände, Kopf, Basis	2 Arme, 2 Hände, 2 Beine, Torso, Kopf
	Bussystem	CAN	Ethercat	CAN, SPI, RS232, etc.	CAN/Sercos/ RS232	CAN
Sensoren	Anzahl	7	10	13	8	10
	Sensor-Typ (ohne Mo-torsensoren)	Laserscanner, 3D-Kamera, Farbkamera, taktile Sensoren	Laserscanner, 3D-/Farbka-mera, IMU, etc.	Laserscanner, Farb-kameras, KMS, takt. Haut, etc.	Laserscanner, 3D-/Farbka-mera, takt. Haut, etc.	Stereo-kamera, IMU, KMS„ taktile Haut
	Bussystem	Seriell, USB, Ethernet, CAN	Ethercat, USB	USB, CAN, Firwire	Seriell, USB, Ethernet, CAN	CAN

Die zu entwickelnden Integrations- und Test-Werkzeuge müssen somit den folgenden Anforderungen genügen:

A1: Einsetzbarkeit der Werkzeuge für eine verteilte Rechnerarchitektur.

A2: Einsetzbarkeit der Werkzeuge für verschiedene Betriebssysteme.

A3: Einsetzbarkeit der Werkzeuge für eine Vielzahl von unterschiedlichen Sensor- und Aktortypen.

Typisch für Serviceroboter ist zudem die hohe Abhängigkeit zwischen Hardware- und Software-Komponenten: Der kinematische Aufbau des Serviceroboters (Hardware) hat z.B. direkten Einfluss auf die Bewegungsplanung der Manipulatoren (Software). Für die Ausführung typischer Aufgaben, z.B. das Greifen eines Objekts, ist im Allgemeinen eine häufige Interaktion zwischen den einzelnen Komponenten erforderlich, die teils synchron, teils asynchron oder unter Echtzeitbedingungen erfolgen muss. Die meisten komplexen Serviceroboter basieren auf einer Software-Architektur mit mehreren Abstraktionsebenen (Regelungsebene, Ausführungsebene, Deliberationsebene), die jeweils unterschiedliche Zykluszeiten aufweisen (in der Regelungsebene meist Millisekunden, in der Deliberationsebene von Minuten bis Stunden oder Tagen). Durch die hohen Abhängigkeiten der Entwickler untereinander und das enorme erforderliche Hintergrundwissen über die Hardwarearchitektur gestalten sich Integrationstests oft schwierig [RMP08]. Meist ist jedoch schon früh das Testen auf der Hardware unerlässlich, da sich der Serviceroboter auf wechselnde Umgebungen, Objekte und Personen einstellen können soll und in der Simulation nicht alle Tests durchgeführt werden können.

A4: Reduktion des notwendigen Wissens über die Hardwarearchitektur des Gesamtsystems.

3.1.1 Analyse der erforderlichen Arbeitsschritte für die Implementierung einer Servicerobotik-Applikation

In dieser Arbeit wird davon ausgegangen, dass ein Großteil der für die Zielapplikation relevanten Soft- und Hardware-Komponenten in unterschiedlicher Ausprägung (z.B. als Algorithmus, Bibliothek oder eingebettet in ein Framework) bereits existieren und in eine gemeinsame Soft- und Hardwarearchitektur integriert werden müssen.

Aus der Analyse verschiedener Forschungsprojekte aus dem Bereich Servicerobotik lassen sich – unter der Annahme, dass die für die Zielapplikation erforderlichen Komponenten

bereits existieren bzw. im Vorfeld entwickelt wurden – die folgenden Aktivitäten für die Integration und Applikationsentwicklung festhalten:

A) Binden der Komponenten in das Software-Framework des Zielsystems

B) Adaption der Komponenten an die Zielhardware

C) Adaption der Komponenten an die Zielapplikation

D) Durchführung von Komponenten- und Integrationstests

E) Applikationsentwicklung

F) Durchführung systemweiter Applikationstests

In den folgenden Unterkapiteln werden diese Arbeitsschritte erläutert.

A) Binden der Komponenten in das Software-Framework des Zielsystems

Zunächst muss der algorithmische Kern der ausgewählten Komponenten in das Framework bzw. die Kommunikationsstruktur des Zielsystems eingebettet werden. Diese Aktivität wird auch als die Erstellung von Integrationscode bezeichnet [ABB+00].

Die anfallenden Arbeitsschritte lassen sich wie folgt aufschlüsseln:

- Einarbeitung in das Zielframework
- Implementierung der entsprechenden Schnittstellen und sonstigen Anforderungen des Frameworks (Erstellung von “Integrationscode”). Der Aufwand hängt dabei sehr stark vom jeweiligen Zielframework ab.

Als Anforderung für die zu entwickelnden Werkzeuge lässt sich daher festhalten:

A5: Werkzeuggestützte Erstellung von Integrationscode.

B) Adaption der Komponenten an die Zielhardware

Generell muss für jede Komponente geprüft werden, ob Anpassungen an die Zielhardware notwendig sind. Dies könnte zum Beispiel der Fall sein, wenn die Komponente von einer bestimmten Sensorik bzw. einer bestimmten Datenkategorie (z.B. 3D-Daten), einer bestimmten Roboterkinematik (z.B. Manipulator mit 6 Freiheitsgraden) ausgeht oder für eine bestimmte Rechnerarchitektur oder ein bestimmtes Feldbussystem implementiert wurde. Je nachdem, wie die Komponente programmiert wurde, kann die Adaption entweder

durch reine Umkonfiguration der Komponente (d.h. Änderung von Parametern in Konfigurationsdateien) oder durch entsprechende Anpassung des Quellcodes erfolgen. Für die Überprüfung der Kompatibilität der Komponente ist eine formale Spezifikation der Hardwarearchitektur des Zielsystems sehr hilfreich.

A6: Formale Spezifikation der Hardewarearchitektur des Zielsystems.

C) Adaption der Komponenten an die Zielapplikation

Nach der einmaligen Bindung der Komponente an das Zielframework und die Adaption an die Zielhardware erfolgt die Anpassung der Komponente für die Zielapplikation. Je nach Applikationsanforderungen kann dies die Adaption von existierenden und die Generierung von neuen Schnittstellen bzw. Funktionalitäten erfordern.

Allgemein ist es für die Bewältigung der in den letzten beiden Abschnitten angeführten Arbeitsschritte hilfreich, wenn eine zentrale Dokumentation verfügbar ist, die z.B. Hinweise und Beispiele für die Erstellung des Integrationscodes und die formalen Schnittstellenspezifikationen aller Komponenten enthält.

A7: Zentrale Verfügbarkeit der Dokumentation für die Integration von Komponenten in das Zielsystem.

Weiterhin sollen die Anforderungen für das Deployment der Komponenten auf die Hardware, z.B. eine bestimmte Sensorik oder Mindestanforderungen an CPU oder Arbeitsspeicher, formal erfasst werden. Die Komposition der Komponenten für die Applikation, d.h. die erforderlichen Kommunikationsverbindungen zwischen den Komponenten auf dem verteilten System soll modelliert werden, so dass die Zuordnung der Komponenten-Architektur auf die Hardwarearchitektur algorithmisch durchgeführt werden kann.

A8: Formale Spezifikation der Hardwareanforderungen der Komponenten und der Komponenten-Architektur für die Applikation

D) Durchführung von Komponenten- und Integrationstests

Üblicherweise werden nach Fertigstellung der Komponente jeweils isolierte Unit- und Schnittstellentests durchgeführt. Für das Testen der Komponente im Zusammenspiel mit gegenseitig abhängigen Komponenten (Integrationstests) ist es jedoch meist notwendig, diese Komponenten auf das Zielsystem zu installieren.

Die anfallenden Arbeitsschritte des Entwicklers sind:

- Einarbeitung in die Rechner/Hardwarearchitektur des Zielsystems
- Installation und Inbetriebnahme der entwickelten Komponente
- ggf. Installation und Inbetriebnahme der abhängigen Komponenten

Middlewares wie ROS [QGC+09] ermöglichen zwar grundsätzlich den Betrieb von Komponenten auf einer verteilten Rechnerarchitektur, bieten aber selbst keine Unterstützung bei der Installation und Inbetriebnahme. Es werden deshalb Werkzeuge benötigt, die diese beiden Laufzeitaktivitäten unterstützen, und zwar unter Abstraktion der Implementierungsdetails der zugrunde liegenden Middleware und Hardwarearchitektur.

A9: Werkzeuggestützte Installation einzelner Komponenten aus dem Repository.

A10: Werkzeuggestützte Aktualisierung einzelner Komponenten unter Berücksichtigung von Abhängigkeiten.

A11: Werkzeuggestützte Inbetriebnahme/Aktivierung einzelner Komponenten auf dem Zielsystem.

Diese Anforderungen sind direkt gekoppelt an die Zielsetzung der Verbesserung der Bedienbarkeit von komplexen Servicerobotern und der Reduktion des erforderlichen Domänenwissens der einzelnen Entwickler.

Da die Komponenten für Serviceroboter-Applikationen in der Regel eng miteinander verflochten sind, erfordern meist auch einzelne Schnittstellentests die Aktivierung einer ganzen Gruppe von Komponenten auf dem System. Für die effiziente Durchführung dieser Komponententests ist es daher unerlässlich, dass der Zustand aller installierten Komponenten stabil ist und diese operabel sind. Die Qualität aller installierter Komponenten muss also durch geeignete Maßnahmen sichergestellt werden:

A12: Verankerung der Qualitätssicherung im Entwicklungsprozess.

A13: Gewährleistung eines jederzeit operablen Systemzustandes.

Insbesondere bei der Verwendung von Open-Source Komponenten aus nicht selbst-verwalteten Repositories ist eine systematische Durchführung von Tests auf Komponenten- und Systemebene erforderlich, um bei Komponentenaktualisierungen eventuelle Inkompatibilitäten mit der eigenen Software schnell zu bemerken. Automatisierte Tests, die bei einer Komponentenaktualisierung im Repository ausgelöst werden, sind dafür ein geeignetes Instrument.

A14: Automatisierte Durchführung von Komponenten- und Integrationstests.

Fehler, die während der Tests zu Tage treten, sollten sofort zentral erfasst und dokumentiert werden, so dass diese den zuständigen Entwicklern automatisch mitgeteilt werden können und in den jeweiligen Softwareständen der Komponenten als gelöst referenziert werden können. Dies ist notwendig, um eine räumliche und zeitliche Entkopplung der Integrationsarbeit zu ermöglichen.

A15: Zentrale Fehlerverwaltung zur räumlichen und zeitlichen Entkopplung der Integrationsarbeit.

E) Applikationsentwicklung

Sind die Komponententests erfolgreich abgeschlossen, kann die Applikation selbst unter Nutzung der jeweiligen Komponentenschnittstellen entwickelt werden. Dazu muss der Entwickler die zur Verfügung stehenden Funktionalitäten der einzelnen Komponenten sowie deren Fehlerbehandlung kennen, um diese für die Programmierung des Applikationsablaufs (z.B. in Form eines Zustandsautomaten) berücksichtigen zu können.

A16: Werkzeuggestützte Applikationsentwicklung (z.B. unterstützte Generierung von geskripteten Abläufen oder Zustandsautomaten).

F) Durchführung systemweiter Applikationstests

Nach der Entwicklung des Applikationscodes muss dieser unter verschiedenen Rahmenbedingungen (z.B. unterschiedliche Umgebungen, zu interagierende Objekte, verschiedene Nutzer, etc.) auf dem Zielsystem getestet werden. Dazu ist es erforderlich, dass alle Komponenten auf dem System installiert und in einem operablen Zustand sind. Die Applikation muss nun inklusive aller erforderlicher Komponenten, ggf. unter Berücksichtigung einer bestimmten Reihenfolge, geladen und gestartet werden. Damit ein Applikationsentwickler dies ohne Systemintegratoren und Komponentenentwickler durchführen kann – und somit zur Erfüllung der Ziele: Verbesserung der Rollentrennung, Entkopplung der Abhängigkeiten, bessere Bedienbarkeit des Zielsystems – sollen die zu konzipierenden Werkzeuge der folgenden Anforderung genügen:

A17: Werkzeuggestützte Inbetriebnahme von Serviceroboter-Applikationen.

Üblicherweise werden die beschriebenen Schritte A) bis F) mehrfach durchlaufen, bis die spezifizierten Anforderungen für die Applikation erfüllt sind. Dabei kann die Iteration je nach Ergebnis der Applikationstests zwischen Entwicklung und Test der Applikation erfolgen oder über die Adaption der Komponenten verlaufen.

Um die Einsatzfähigkeit einer erfolgreich getesteten Applikation auch unter der Annahme häufiger Komponentenaktualisierungen beurteilen zu können, sollten (teil-)automatisierte Applikationstests unterstützt werden.

A18: Automatisierte Durchführung von Applikationstests.

Damit die verschiedenen Werkzeuge aus den einzelnen Anforderungen nahtlos zusammenarbeiten und die Übergänge möglichst automatisiert unterstützt werden können, sollen weiterhin alle Werkzeuge in eine gemeinsame Umgebung integriert werden.

A19: Alle entwickelten Werkzeuge sollen in einer integrierten Umgebung zentral verfügbar sein.

3.1.2 Vorgehensmodell für die verteilte Integration

Um die in Abschnitt 3.1.1 analysierten typischen Arbeitsabläufe bei der Entwicklung von Serviceroboter-Applikationen formal als Prozesse modellieren zu können, soll ein dediziertes Vorgehensmodell entwickelt werden, das auch die speziellen Randbedingungen aus den Szenarien (vgl. Abschnitt 3.3) berücksichtigt. Grundsätzlich muss das Vorgehensmodell für räumlich verteilte Teams (z.B. im Szenario Verbund-Forschungsprojekt) einsetzbar sein:

A20: Anwendbarkeit des Vorgehensmodells für verteilte Projektteams.

Da Serviceroboter-Applikationen bisher gewöhnlich in einem sehr innovativen Umfeld entwickelt werden, muss das Vorgehensmodell berücksichtigen, dass im Allgemeinen wenig Erfahrungswerte für diese Applikationen existieren.

A21: Anwendbarkeit des Vorgehensmodells für hochgradig innovative Projekte mit wenig Erfahrungswerten.

Neben der Definition des konzeptuellen Ablaufs sind jedoch auch Werkzeuge erforderlich, die die Durchführung des verteilten Integrationsprozesses unterstützen. Dadurch kann auch die Qualität im Sinne von korrekt und zuverlässig durchgeführten Prozessen gesichert werden.

A22: Werkzeuggestützte Durchführung des Integrationsprozesses im Rahmen des Vorgehensmodells (Workflow-Management).

Insbesondere soll die Dokumentation der Testergebnisse und des Entwicklungsfortschritts werkzeuggestützt durchgeführt werden (Test-Management).

A23: Werkzeuggestützte Dokumentation erfolgter Komponenten- und Applikationstests.

3.1.3 Analyse der Entwicklerrollen

Für die Durchführung der in Abschnitt 3.1.1 analysierten Arbeitsschritte sind verschiedene Entwicklerrollen erforderlich, die jeweils vertiefte Kenntnisse in den einzelnen Komponenten bezüglich der Zielapplikation oder der Zielhardware haben.

[SSL12] definiert die folgenden Rollen für die Entwicklung von robotischen Softwaresystemen:

- Komponentenentwickler
- Systemintegratoren
- Applikationsentwickler
- Framework- und Werkzeugentwickler

Die Trennung der Entwicklerrollen ist eine notwendige Bedingung für eine effiziente Integration unterschiedlicher Komponenten in eine gemeinsame Hardwareplattform.

A24: Die Werkzeuge sollen die Rollentrennung für die verschiedenen Integrations- und Testaktivitäten unterstützen.

3.2 Analyse der Anforderungen durch die räumliche Verteilung der Integrationsarbeit

Für das erfolgreiche Bewältigen von Softwareprojekten allgemein spielt neben Kompetenzen in Projektmanagement, Risikomanagement und Softwareentwicklung auch die effiziente Zusammenarbeit im Team eine Rolle. Verteilte Projektteams werden dabei vor besondere Herausforderungen gestellt. Den Vorteilen von verteilter Projektarbeit, u.a. der Zusammenführung der jeweils besten Fachleute für eine Aufgabe und der Einsparung von Reisekosten, stellen sich zahlreiche Nachteile gegenüber [HO11]. Diese sind zum einen durch zwischenmenschliche Ursachen begründet. So existieren z.B. weniger Möglichkeiten zum informellen Kennenlernen der Teammitglieder und deren Kompetenzen und dadurch weniger Möglichkeiten zum Aufbau von Vertrauen und Identifikation mit dem Projekt, woraus sich tendenziell eine unklarere Aufteilung von Rollen und Kompetenzen innerhalb des Projekts und schwierigeres Konflikt- und Deeskalationsmanagement ergeben. Zum anderen gibt es in verteilten Projekten im Vergleich zu nicht-verteilten Projekten im Allgemeinen viel höhere Anforderungen an die Kooperation, z.B. durch elektronische Kommunikationsmedien,

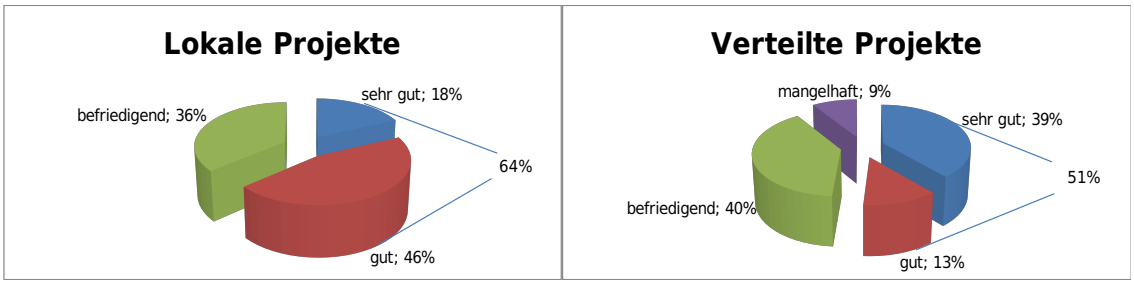


Abbildung 3.1: Vergleich des Projekterfolgs von verteilten und lokal durchgeführten Projekten (Zahlen aus einer Experten-Studie von [Gro08]). Die Studienteilnehmer wurden dabei aus verschiedenen Branchen gewählt, z.B. IT, Automotive, Chemie/Pharma, Bankwesen, Unternehmensberatung, etc. In lokalen Projekten schätzten 64% der Teilnehmer den Projekterfolg als gut oder sehr gut ein. In verteilten Projekten teilten jedoch nur 51% der Teilnehmer diese Einschätzung. Weiterhin bewerteten fast 10% der Teilnehmer den Projekterfolg als mangelhaft.

durch Werkzeuge für das Wissensmanagement und der Online-Dokumentation der Ergebnisse. Die Teammitglieder benötigen daher weitere nicht-fachliche Kompetenzen wie z.B. die Fähigkeit zur Kommunikation mit elektronischen Medien, Lernbereitschaft und Flexibilität, Eigenitiative und eine positive Einstellung zu kooperativer Arbeit [HO11].

Auch an das Projektmanagement werden neue Anforderungen gestellt, z.B. die Bereitschaft zu delegativen Führungsprinzipien, die den verteilten Projektmitgliedern eine höhere Selbstständigkeit und Eigenverantwortung einräumen.

Studien haben zwar gezeigt, dass die Kompensation fehlender direkter, persönlicher Kommunikation durch entsprechende computergestützte Kommunikation durchaus möglich ist [Fje04]. Nichtsdestotrotz ist die Erfolgsquote von verteilten im Vergleich zu nicht-verteilten Projekten deutlich geringer (s. Abbildung 3.1).

Eine weitere Aufgabe des Projektmanagements besteht darin, eine Ausgewogenheit zwischen Gruppen- und Einzelarbeiten zu erreichen [Lit95]. Für Sachbearbeitungsaufgaben, z.B. Implementierung und Test einer neuen Software-Komponente, ist Gruppenarbeit eher hinderlich. Die Ermöglichung von effizienten Einzelarbeiten stellt dabei besondere Herausforderungen an interdisziplinäre Projektteams mit hohen personellen Abhängigkeiten dar. Bei Servicerobotik-Projekten wird zusätzlich die Anforderung des schnellen Zugriffs und der einfachen Bedienbarkeit durch einzelne Personen auf die zu entwickelnde Hardware (vgl. Anforderungen A4 und A13) aufgestellt.

A25: Ermöglichung der Einzelarbeit am integrierten Gesamtsystem.

Zusammenfassend lässt sich demnach feststellen, dass bei verteilten Projekten vor allem Team-Werkzeuge für das Wissensmanagement und die Kommunikation eine wichtige Rolle spielen, die einfach bedienbar und auf das Projekt zugeschnitten sind. Für Servicerobotik-Projekte gilt dies umso mehr, als nicht nur an gemeinsamen Dokumenten oder Softwarebausteinen gearbeitet wird, sondern auf gemeinsamer Hardware.

A26: Werkzeuggestützte zentrale Organisation des Wissensmanagements.

A27: Bereitstellung einer geeigneten Infrastruktur für eine effiziente Kommunikation im verteilten Team.

A28: Unterstützung der gemeinsamen Arbeit an Dokumenten und Quellcode durch Kooperationswerkzeuge.

Weiterhin ist eine werkzeuggestützte Koordination der Aufgaben des verteilten Teams, insbesondere der Durchführung der verteilten Tests dazu geeignet, die dabei auftretenden Abhängigkeiten zu reduzieren [SSU01].

A29: Werkzeuggestützte Koordination des verteilten Entwicklungsteams.

3.3 Analyse der Szenarien für verteilte Integration

Im Folgenden werden die in Abschnitt 2.3 definierten Szenarien auf zusätzliche Anforderungen hin untersucht.

3.3.1 Szenario 1: Verbund-Forschungsprojekt

Da auch einfache Komponenten- und Schnittstellentests – wie in Abschnitt 2.3.2 erläutert – häufig das Dazuschalten weiterer Komponenten erfordern, müssen oft viele Entwickler an den Tests beteiligt werden. Die Integrationswerkzeuge sollen deshalb Mechanismen zur Reduktion der Abhängigkeiten bei Komponententests zur Verfügung stellen.

A30: Entkopplung der Abhängigkeiten zwischen den Entwicklern bei Tests.

Neben der werkzeuggestützten Inbetriebnahme von Applikationen und Komponenten auf dem Serviceroboter ist eine einheitliche Konfigurierbarkeit der Komponenten erforderlich, um den Komponenten- und Applikationsentwicklern die Parametrierung von fremden Komponenten zu ermöglichen. In der Regel kennen die Entwickler zwar die wesentlichen Konfi-

gurationsparameter einer Komponente, es ist aber häufig unklar, in welchen Dateien und an welcher Stelle sie diese Parameter modifizieren können.

A31: Einheitliche Konfigurierbarkeit aller Komponenten.

Eine Beschreibung der Parameter einschließlich deren Bedeutung und Wertebereiche sollte jedoch in der zentralen Dokumentation verfügbar sein, so dass die Konfiguration einer Komponente ohne Expertenkenntnisse durchgeführt werden kann. Um den Dokumentationsprozess zu vereinfachen, ist eine (teil-)automatische Dokumentation erforderlich:

A32: Werkzeuggestützte Dokumentierung von Komponentenparametern und deren Semantik.

Darüber hinaus sollte die Operabilität der Komponenten auch aus der Ferne zur Verfügung stehen, um z.B. einfache Schnittstellentests ohne Reisetätigkeit der örtlich verteilten Entwickler durchführen zu können.

A33: Ermöglichung des Fernzugriffs (anytime, anywhere) auf die Software- und ggf. Hardware-Komponenten des Roboters für die Durchführung von verteilten Tests.

Dazu ist es notwendig, dass ein zentrales Zugriffsmanagement auf die Roboterhardware existiert.

A34: Zugriffsmanagement auf die Roboterhardware.

Weiterhin ist bei der Arbeit des verteilten Projektteams an derselben Roboterhardware ein effizienter Wechsel von verschiedenen Softwareständen auf dem Zielsystem erforderlich. Der Aufwand für das Aufspielen und das Einrichten der jeweiligen entwicklerspezifischen Softwarestände auf der Roboterhardware soll minimiert werden.

A35: Werkzeuggestütztes Laden von bestimmten Softwareständen (Releases von Komponenten und Applikationen).

3.3.2 Szenario 2: Servicerobotik-Teststand

Aus dem Szenario “Servicerobotik-Teststand” leitet sich die Anforderung ab, dass die Integrationswerkzeuge nicht an eine bestimmte Middleware, an ein dezidiertes Komponentenmodell oder ein bestimmtes Framework gebunden sind. Andernfalls wäre der Teststand aufgrund der hohen Portierungsaufwände nur für eine eingeschränkte Anwendergruppe effektiv nutzbar.

A36: Unabhängigkeit aller Werkzeuge von Middleware und Software-Framework.

A37: Keine Voraussetzung eines bestimmten Komponentenmodells.

3.3.3 Szenario 3: Diagnose und Fernwartung

Im Wartungsszenario befinden sich die Systemintegratoren nicht am Standort des Serviceroboters, so dass möglichst umfangreiche Diagnoseinformationen aus der Ferne verfügbar sein sollten, um die Reisetätigkeit des Supportpersonals minimieren zu können.

A38: Fernüberwachung und -diagnose von Hardware- und Softewarekomponenten.

Weiterhin muss der Fall betrachtet werden, dass die Integrationswerkzeuge den Systemintegratoren nicht lokal zur Verfügung stehen, sondern sich ebenfalls am Standort des Roboters befinden. Daher müssen die Werkzeuge aus der Ferne eingerichtet und hinsichtlich eventueller Komponenten-Änderungen angepasst werden können.

A39: Einfache Konfiguration der Werkzeuge aus der Ferne.

3.3.4 Szenario 4: Projektübergreifende lokale Nutzung

Bei der Arbeit von lokalen Projekt-Teams an derselben Roboterhardware ist vor allem ein effizienter Wechsel von verschiedenen Projektkontexten und -softwareständen auf dem Zielsystem erforderlich. Der Aufwand für das Aufspielen und das Einrichten der jeweiligen projekt- oder entwicklerspezifischen Softwarestände auf der Roboterhardware soll minimiert werden. Dies wird bereits durch die Anforderungen A9 und A35 abgedeckt.

Weiterhin ist bei der intensiven lokalen Nutzung ein Ressourcen- und Zugriffsmanagement erforderlich (A34). Einzelarbeit am Zielsystem soll möglich sein (A25), so dass die Ressource Serviceroboter gleichmäßig und nicht nur während der Projekttreffen ausgelastet werden kann. Um die Vielzahl der projektbezogenen Applikationen und Komponenten bzw. Komponentenkonfigurationen für alle Entwickler beherrschbar zu halten, ist schließlich eine werkzeuggestützte Konfiguration und Aktivierung von Komponenten (A11, A31) und Applikationen (A17) wünschenswert.

3.4 Strukturierung der Anforderungen

Die Anforderungen aus den bisherigen Abschnitten lassen sich in die folgenden Kategorien einteilen: Allgemeine Anforderungen, Entwicklung eines Vorgehensmodells für die verteilte Integration, Entwicklung von Deployment-, Laufzeit- und Kooperationswerkzeugen. Diese verschiedenen Kategorien sollen als Funktionsblöcke in eine Integrations- und Testplattform integriert werden, in der alle Werkzeuge zentral verfügbar sind und konfiguriert werden können. In Abbildung 3.2 sind die einzelnen Funktionsblöcke schematisch dargestellt. Im Folgenden werden die einzelnen Funktionsblöcke kurz erläutert; weiterhin wird eine Zuordnung der einzelnen Anforderungen zu diesen Funktionsblöcken vorgenommen.

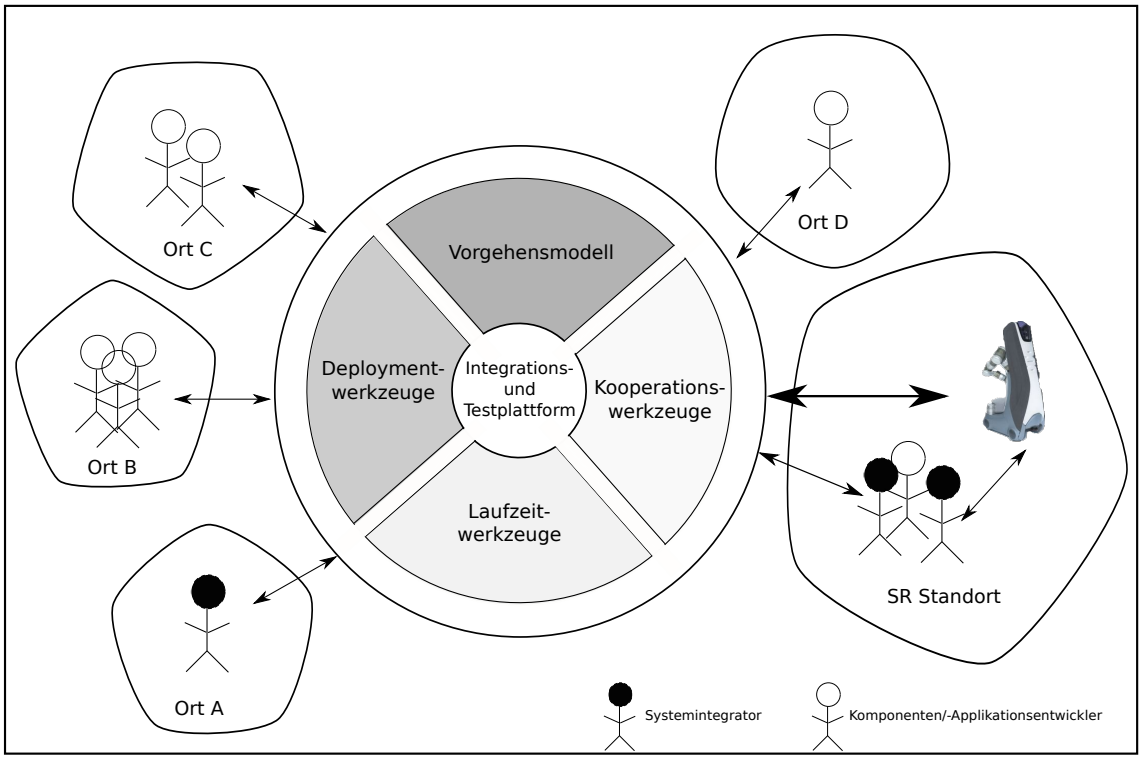


Abbildung 3.2: Schematische Darstellung der funktionalen Blöcke der zu entwickelnden Integrations- und Testplattform.

Allgemeine Anforderungen: Die Werkzeuge müssen anwendbar sein für die Entwicklungsdomäne “komplexer Serviceroboter”, jedoch nicht von einem bestimmten Framework oder von einer bestimmten Middleware ausgehen. Weiterhin soll auf die Kompatibilität der Integrations- und Testplattform zu existierenden Werkzeugketten geachtet werden. Diese allgemeinen Anforderungen, die für alle zu entwickelnden Werkzeuge gelten, sind in Abbildung 3.3 zusammengefasst.

A1:	Einsetzbarkeit der Werkzeuge für eine verteilte Rechnerarchitektur
A2:	Einsetzbarkeit der Werkzeuge für verschiedene Betriebssysteme
A3:	Einsetzbarkeit der Werkzeuge für eine Vielzahl von unterschiedlichen Sensor- und Aktortypen
A4:	Reduktion des notwendigen Wissens über die Hardwarearchitektur des Gesamtsystems
A13:	Gewährleistung eines jederzeit operablen Systemzustandes
A19:	Alle entwickelten Werkzeuge sollen in einer integrierten Umgebung zentral verfügbar sein
A24:	Die Werkzeuge sollen die Rollentrennung für die verschiedenen Integrations- und Testaktivitäten unterstützen
A30:	Entkopplung der Abhängigkeiten zwischen den Entwicklern bei Tests
A36:	Unabhängigkeit aller Werkzeuge von Middleware und Software-Framework
A37:	Keine Voraussetzung eines bestimmten Komponentenmodells
A39:	Einfache Konfiguration der Werkzeuge aus der Ferne

Abbildung 3.3: Allgemeine Anforderungen und Randbedingungen für die zu entwickelnden Werkzeuge

Vorgehensmodell für die verteilte Integration (VM): Vorgehensmodelle und darin enthaltene Prozesse sind unabdingbar für eine effiziente Entwicklung in verteilten Teams. Der damit verbundene Zusatzaufwand zur Durchführung und Kontrolle der Prozesse soll mit Hilfe von Werkzeugen durch die Integrations- und Testplattform reduziert werden. Dies schließt Prozesse für die Dokumentation sowie die automatisierte Durchführung von Unit- und Integrations- und Applikationstests mit ein. In Abbildung 3.4 sind diejenigen Anforderungen zusammengefasst, die für die Entwicklung des Vorgehensmodells relevant sind.

Deployment-Werkzeuge (DEPL): Dieser Funktionsblock soll die Bedienung und Wartung eines komplexen Serviceroboters während der Entwicklungsphase von neuen Applikationen erheblich vereinfachen. Ein Entwickler soll ohne detaillierte Kenntnis der einzelnen Komponenten und der Rechner- und Hardwarearchitektur des Serviceroboters in der Lage sein, einen bestimmten Softwarestand von Komponenten und Applikationen auf den Roboter zu laden sowie diese zu konfigurieren, zu aktivieren und zu deaktivieren. Dieses Werkzeug soll vor allem den Systemintegrator unterstützen. Die Anforderungen für die Entwicklung der Deployment-Werkzeuge sind in Abbildung 3.5 zusammengefasst.

Laufzeitunterstützung zur besseren Bedienbarkeit (RTE): Dieser Funktionsblock soll vor allem die Laufzeitaktivitäten bei der Applikationsentwicklung unterstützen: Kom-

A12:	Verankerung der Qualitätssicherung im Entwicklungsprozess
A14:	Automatisierte Durchführung von Komponenten- und Integrationstests
A18:	Automatisierte Durchführung von Applikationstests
A20:	Anwendbarkeit des Vorgehensmodells für verteilte Projektteams
A21:	Anwendbarkeit des Vorgehensmodells für hochgradig innovative Projekte mit wenig Erfahrungswerten
A22:	Werkzeuggestützte Durchführung des Integrationsprozesses im Rahmen des Vorgehensmodells (Workflow-Management)
A23:	Werkzeuggestützte Dokumentation erfolgter Komponenten- und Applikationstests
A32:	Werkzeuggestützte Dokumentierung von Komponentenparametern und deren Semantik

Abbildung 3.4: Anforderungen für das zu entwickelnde Vorgehensmodell zur Strukturierung des verteilten Entwicklungsprozesses von Serviceroboter-Applikationen und der Verbesserung der Rollentrennung.

A6:	Formale Spezifikation der Hardewarearchitektur des Zielsystems
A8:	Formale Spezifikation der Hardwareanforderungen der Komponenten und der Komponenten-Architektur für die Applikation
A9:	Werkzeuggestützte Installation von Komponenten aus dem Repository
A10:	Werkzeuggestützte Aktualisierung einzelner Komponenten unter Berücksichtigung von Abhängigkeiten
A35:	Werkzeuggestütztes Laden von bestimmten Softwareständen (Releases von Komponenten und Applikationen)

Abbildung 3.5: Anforderungen für die zu entwickelnden Deployment-Werkzeuge zur Unterstützung des Deployment-Prozesses

ponenten- und Applikationsentwickler sollen in der Lage sein, ohne genaue Kenntnis des Komponentendeployments (d.h. auf welchem Rechner welche Komponente installiert ist und wo die jeweiligen Konfigurations- und Startdateien liegen), Komponenten und Applikationen im Rahmen von Integrations- und Applikationstests zu konfigurieren und zu aktivieren. Dies soll unabhängig von der räumlichen Verteilung der Tester und dem Zielsystem erfolgen können.

Kooperationswerkzeuge (KOOP): Dieser Funktionsblock erfüllt die Anforderungen zur effizienten Kooperation in verteilten Entwicklungsteams: eine leistungsfähige Kommunikationsinfrastruktur, Kollaborationswerkzeuge sowie Wissensmanagement und Dokumentation (s. Abbildung 3.7).

Die Erstellung von Integrationscode (A5) und die werkzeuggestützte Applikationsentwicklung (A16) betreffen vielmehr Entwicklungswerkzeuge als Integrationswerkzeuge und pas-

A11:	Werkzeuggestützte Inbetriebnahme/Aktivierung einzelner Komponenten auf dem Zielsystem
A17:	Werkzeuggestützte Inbetriebnahme von Serviceroboter-Applikationen
A25:	Ermöglichung der Einzelarbeit am integrierten Gesamtsystem
A31:	Einheitliche Konfigurierbarkeit aller Komponenten
A33:	Ermöglichung des Fernzugriffs (anytime, anywhere) auf die Software- und ggf. Hardware-Komponenten des Roboters für die Durchführung von verteilten Tests
A34:	Zugriffsmanagement auf die Roboterhardware
A38:	Fernüberwachung und -diagnose von Hardware-Komponenten

Abbildung 3.6: Anforderungen für die zu entwickelnde Laufzeitunterstützung zur Verbesserung der Bedienbarkeit komplexer Serviceroboter (vor allem für Komponenten- und Applikationsentwickler).

A27:	Bereitstellung einer geeigneten Infrastruktur für eine effiziente Kommunikation im verteilten Team.
A28:	Unterstützung der gemeinsamen Arbeit an Dokumenten und Quellcode durch Kooperationswerkzeuge.
A29:	Werkzeuggestützte Koordination des verteilten Entwicklungsteams
A7:	Zentrale Verfügbarkeit der Dokumentation für die Integration von Komponenten in das Zielsystem
A26:	Werkzeuggestützte zentrale Organisation des Wissensmanagements
A15:	Zentrale Fehlerverwaltung zur räumlichen und zeitlichen Entkopplung der Integrationsarbeit

Abbildung 3.7: Anforderungen für die zu entwickelnden Kooperationswerkzeuge

sen daher nicht in die aufgestellten Kategorien. Diese Werkzeuge werden eher lokal von Komponenten- und Applikationsentwicklern benötigt und müssen nicht zentral verfügbar sein. Dennoch beinhalten diese wichtige Aktivitäten für die Entwicklung von komponentenbasierten Serviceroboter-Applikationen und werden daher unter “Sonstige Anforderungen” weitergeführt.

Abbildung 3.8 veranschaulicht den Beitrag der einzelnen Werkzeuge zur den proklamierten Zielen dieser Arbeit. Alle Werkzeuge unterstützen die Rollentrennung von Komponentenentwicklern, Systemintegratoren und Applikationsentwicklern. Die Verbesserung der Bedienbarkeit von komplexen Servicerobotern wird vor allem durch die Deployment- und Laufzeitwerkzeuge angestrebt. Das zu entwickelnde Vorgehensmodell soll den verteilten Entwicklungsprozess für die Entwicklung von Serviceroboter-Applikationen strukturieren. Die Steigerung der Effizienz im verteilten Entwicklungsteam durch räumliche und zeitliche Entkopplung der Integrationsaktivitäten schließlich soll durch die Kooperations- und Laufzeitwerkzeuge erreicht werden.

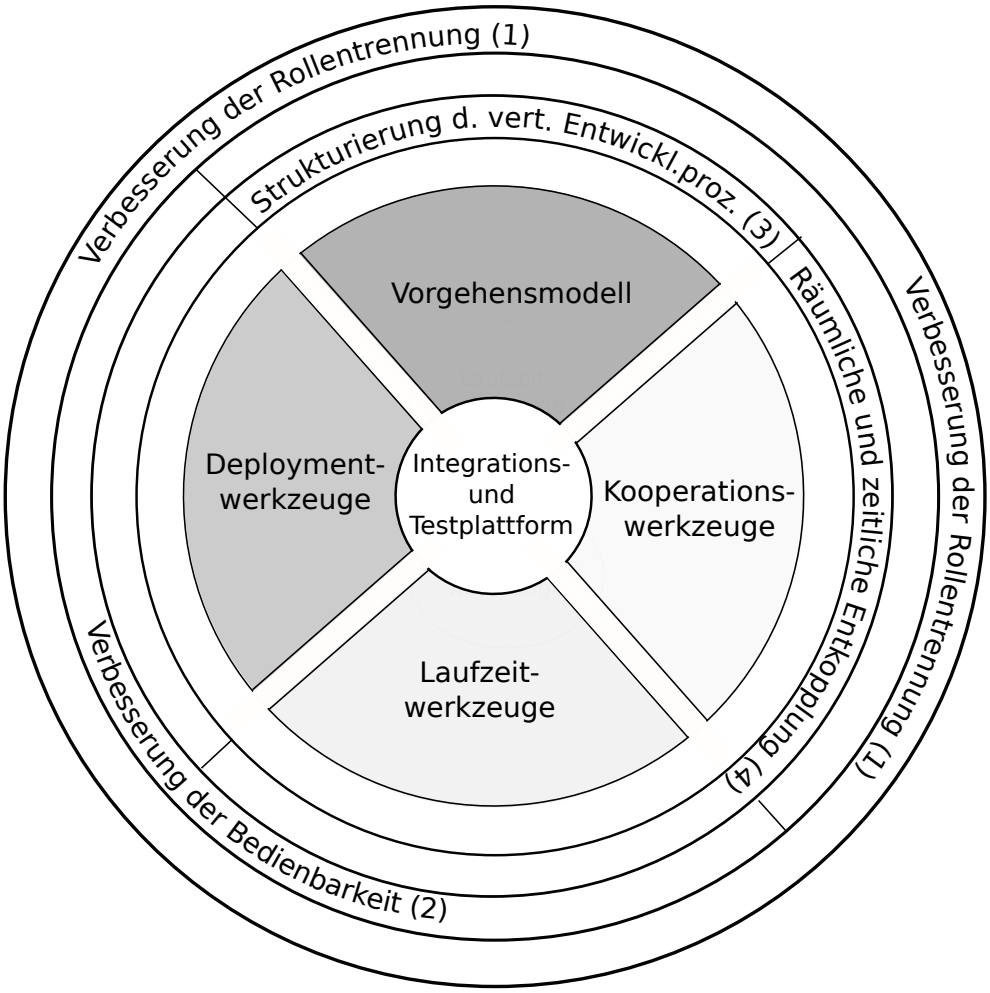


Abbildung 3.8: Zurodnung der zu erstellenden Werkzeuge zu den aufgestellten Zielen der Arbeit.

4 Stand der Technik

Im folgenden Kapitel werden existierende Arbeiten und Werkzeuge in den Bereichen Softwareprozesse bzw. Vorgehensmodelle, Deployment, Laufzeitunterstützung und Kooperation in verteilten Entwicklungsteams (entsprechend der Kategorien in Abschnitt 3.4) gesammelt und analysiert. In jedem Bereich wird ein Abgleich des Stands der Technik mit den korrespondierenden Anforderungen aus Kapitel 3 durchgeführt, dessen Ergebnisse in einem jeweils anschließenden Unterkapitel "Folgerungen" zusammengefasst werden.

4.1 Werkzeuge und Methoden für Integration und Test komponenten-basierter Applikationen

4.1.1 Unterstützung der Bindung von Komponenten an das Zielframework

Methoden und Werkzeuge der modellgetriebenen Softwareentwicklung (engl. model-driven engineering, MDE) [SVE+07] unterstützen bei der Integration von Komponenten in ein bestimmtes Zielframework vor allem durch Codegenerierung. Modellgetriebene Werkzeuge basieren in der Regel auf einem bestimmten Komponentenmodell, das die Grundlage für alle zu integrierenden Komponenten bildet und in der Regel auf eine bestimmte Middleware bezogen ist. Zwar gibt es Beispiele für den Austausch von Middlewares in der Literatur (z.B. Austausch von ACE und CORBA [LSS09]). Allerdings setzt dies wiederum voraus, dass diese Middlewares dieselben Kommunikationsmuster (z.B. Publish-Subscribe) nutzen, die in einem Meta-Modell abstrahiert werden können. In der Literatur werden zwar allgemeingültige Kommunikationsmuster für die Robotik proklamiert (z.B. [Sch06b]). Dennoch existiert bis zum heutigen Zeitpunkt kein standardisiertes Meta-Modell als Grundlage für die in der Robotik genutzten Middlewares.

MDE-Entwicklungsumgebungen umfassen meist grafische Nutzer-Oberflächen, in denen zunächst das Modell einer zu integrierenden Komponente inklusive der Schnittstellen und Parameter erstellt wird. Die MDE-Werkzeuge sind üblicherweise in der Lage, aus den erstellten

Modellen Quellcode zu generieren (Model-To-Text Transformation), der die im Modell abstrakt beschriebenen Kommunikations-Schnittstellen (meist auch inklusive der Datentypen) und Parameter konform zur zugrundeliegenden Middleware umsetzt. Die eigentlichen Funktionen und Algorithmen können dann in die vorgefertigten Methodenrümpfe eingebettet werden. Durch die Autogenerierung des Integrationscodes kann der Aufwand für die Bindung einer Komponente an eine bestimmte Middleware oder ein Framework also erheblich reduziert werden. Im Idealfall können die middleware-spezifischen Kommunikationsstrukturen vor dem Entwickler komplett verborgen bleiben, so dass auch die Einarbeitungszeit in das Zielframework bzw. in die Zielmiddleware erheblich reduziert werden kann.

Beispiele für Komponentenmodelle in der Servicerobotik-Domäne sind z.B. das OROCOS / RTT Komponentenmodell [Soe02], das BRICS Component Model (BCM) [KHG+13], das OPRoS Model [SJJK08], SmartMDSD für Smartsoft [LSS09] und das RT-Komponentenmodell [ASKK06] sowie das CORBA Component Model (CCM) [Obj06a]. Dedizierte MDE-Umgebungen für die Servicerobotik Anwendung sind z.B. SmartSoft [LSS09], BRIDE [BGH+12] und OpenRTM-AIST [ASK08]. Darüber hinaus existieren viele MDE-Werkzeuge für andere Domänen, z.B. TOPCASED [PC06] für die Luft- und Raumfahrtindustrie, und professionelle modellgetriebene Entwicklungsumgebungen wie objectIF [mici] und Matlab Simulink [Mat].

In der Literatur lassen sich auch **Codegeneratoren** finden, die kein Komponentenmodell voraussetzen. SWIG (Simplified Wrapper and Interface Generator) [Bea95] ist zum Beispiel ein Werkzeug zur Integration von C/C++ in skript-basierte Sprachen wie Python, Perl oder Lua [Bea96]. Dazu generiert SWIG z.B. aus der C++-Schnittstellendeklaration Integrationscode, der den Aufruf von C/C++ Routinen aus der Skriptsprache ermöglicht. Ähnliche Funktionalität bietet SIP [Riv]. Falls die Servicerobotik-Applikation in einer Skriptsprache entwickelt wird, können diese Werkzeuge den Integrationsaufwand erheblich reduzieren. Der Nachteil dieser Art von Codegeneratoren liegt jedoch in der limitierten Flexibilität bezüglich der Zielsprachen und Kommunikationsstrukturen.

4.1.2 Unterstützung der komponenten-basierten Applikationsentwicklung

Die komponenten-basierte Applikationsentwicklung besteht konzeptionell grundsätzlich aus der Konnektion und Koordination von Komponenten. Meist erfolgt die Koordination in Form von Zustandsautomaten, für deren Programmierung in vielen komponenten-basierten Frameworks bereits Werkzeuge mitgeliefert werden, z.B. ROS-SMACH [Boh10] oder OROCOS-rFSM [Klo12]. In SMACH können komplexere Roboterapplikationen in Form von

nebenläufigen hierarchischen Zustandsautomaten (engl. concurrent hierachical state machines) realisiert werden. Allerdings existiert kein Modell für die einzelnen SMACH Zustände, so dass die Wiederverwendbarkeit der Zustandsautomaten eher gering ist. rFSM basiert ebenfalls auf hierarchischen Zustandsautomaten und ist in der Skriptsprache Lua [IFF96] implementiert. In [KB11] wird die Echtzeitfähigkeit von rFSM experimentell belegt.

In [NFL10] werden hybride Zustandsautomaten (engl. Hybrid State Machines, HSM [Hen96]) verwendet, um Roboterverhalten im Sinne von Roboter-Applikationen zu modellieren. Diese HSM werden mit Hilfe der Skriptsprache Lua implementiert und repräsentieren die einzelnen Fähigkeiten oder "Skills" des Roboters, aus denen das Verhalten zusammengesetzt werden kann. Die Skills basieren dabei auf Komponenten des FAWKES Framework [Nie09].

Weiterhin werden in der Literatur Service-Komponenten Architekturen (engl. Service Component Architecture, SCA) für die Entwicklung von Applikationen in der Servicerobotik-Domäne vorgeschlagen [SSBK10, BGKB12]. SCA stellen eine Kombination von service-orientierten Architekturen (engl. service oriented architecture, SOA) und komponenten-basierten Architekturen dar [MR09]. Im Zusammenhang mit Abstract State Machines (ASM) [BS03] können SCAs nach [RSA11] zur Modellierung und Prototypisierung von service-basierten Applikationen verwendet werden. In [BGRS12] wird SCA-ASM für einen Anwendungsfall der Servicerobotik evaluiert. Ein wesentlicher Vorteil der SCA-ASM Kombination liegt in der Existenz einer breiten Werkzeugpalette zur Unterstützung der Entwicklung von service-basierten Applikationen, z.B. das SCA Eclipse Front-End [Ecl11], das Apache Tuscany Back-End [Apab] oder FraSCAti [OW209] zur Laufzeitunterstützung sowie ASMETA [RGS+08] zur Ausführung und Validierung von ASM.

Neben reinen Zustandsautomaten werden auch andere Konzepte zur Applikationsentwicklung genutzt. SmartTCL [SS10] z.B. organisiert die Koordination von Komponenten in Form von *Task Coordination Blocks (TCB)*, die zur Laufzeit anhand von Vorbedingungen in einer Lisp-basierten Sprache definiert werden. Die Definition umfasst neben den Vorbedingungen u.a. Ein- und Ausgangsvariablen sowie Verhaltensregeln. Die Selektion des nächsten aktiven TCBs erfolgt dann anhand der Auswertung der Vorbedingungen, der Eingangsvariablen und einer Priorisierung der TCBs. Weiterhin lassen sich TCBs zu komplexeren Blöcken komponieren und können zur Laufzeit modifiziert werden.

Beetz et al [BMT10] präsentieren einen KI-basierten Ansatz zur Realisierung komplexer Serviceroboter-Applikationen: CRAM (Cognitive Robot Abstract Machine). Entscheidungen werden nicht durch festprogrammierte Regeln, sondern durch Schlussfolgern auf Basis der jeweils zur Verfügung stehenden Informationen aus einer Wissensdatenbank, KnowRob [TB09], sowie dem aktuellen Weltmodell des Serviceroboters getroffen, das durch Daten aus den Perzeptions-, Navigations- und Manipulationsmodulen generiert und aktualisiert

wird. Weiterhin können Schlussfolgerungen aus dem Erfolg bzw. Misserfolg ausgeführter Aktionen getroffen werden, so dass CRAM die Ausführung im Applikations-Kontext zur Laufzeit optimieren kann. Im Unterschied zu anderen KI-basierten Ansätzen erlaubt CRAM Schlussfolgern nicht nur auf symbolischen, sondern auch berechenbaren Prädikaten. Applikationen werden in der Lisp-basierten CRAM Plan Language (CPL) auf Basis der verfügbaren Komponentenfunktionalitäten programmiert, die in der ROS Actionlib-Abstraktion [Pra09] erwartet werden.

Auch für industrielle Roboterzellen gibt es Unterstützung für die Applikationsentwicklung: Verl et al präsentieren z.B. einen Ansatz zur automatischen Generierung von Robotercode auf Basis der erweiterbaren Benutzerschnittstelle für Robotikanwendungen XIRP (engl. "eXtendable interface for robotic purposes") und dem Universal Plug And Play Protokoll (UPnP) [VN08].

Die Mehrheit der vorgestellten Ansätze und Werkzeuge zur komponenten-basierten Applikationsentwicklung basieren auf einem bestimmten Framework oder Komponentenmodell.

4.1.3 Unterstützung zur Durchführung von Komponenten-, Integrations- und Systemtests

Frameworks zur Durchführung von Unit- bzw. Komponententests existieren in großer Vielzahl für sehr viele verschiedene Programmiersprachen, z.B. [K+], [JUn], [Roz], [Goo]. Meist basieren diese auf der sogenannten xUnit-Architektur [Bec03], die Grundbausteine für Test-Frameworks definiert. Auch Software-Frameworks in der Robotik umfassen auf xUnit basierende Test-Suiten, z.B. [Con08], die Unittests, aber auch komponentenübergreifende Integrationstests ermöglichen. Die Testframeworks bieten strukturelle Funktionen zur Testvorbereitung, -durchführung und -auswertung, so dass dem Entwickler lediglich die Definition und Implementierung spezifischer Testfälle sowie die Bereitstellung geeigneter Testdaten obliegt.

Die Ausführung dieser generierten Testfälle kann durch sogenannte Continuous Integration (CI-)Werkzeuge [DMG07] automatisiert werden. Meist sind diese direkt an Software-Repositories angebunden, so dass die Ausführung der Tests z.B. durch eine Aktualisierung des Software-Stamms ausgelöst werden kann. Die Testergebnisse werden in der Regel dann in einer Datenbank abgelegt, so dass die Testhistorie für die definierten "Testjobs" verfügbar ist. Oft bieten diese Werkzeuge deshalb auch statistische Analysen bezüglich der Stabilität bestimmter Softwarepakete. Weiterhin zeichnen sich viele CI-Werkzeuge durch eine webbasierte Nutzerschnittstelle aus (z.B. Jenkins [Kaw11], Hudson [Oraa], Bitten [Edg], Apache Continuum [Apaa]), die ortsunabhängig einen schnellen Überblick über den Entwicklungs-

stand von Softwarepaketen ermöglicht. Automatisierte Tests, insbesondere Integrations- und Applikationstests können in der Regel jedoch nur auf der Basis von Simulationen und nicht direkt auf der Zielhardware durchgeführt werden. CI-Werkzeuge sind oft auch fester Bestandteil großer kommerzieller Softwareentwicklungswerkzeuge, z.B. Rational Team Concert [Man10].

4.1.4 Folgerungen

Für die Unterstützung von Integration und Test bieten MDE-Werkzeuge durch Model-To-Text Transformationen die automatische Erstellung von Integrationscode (A5), was den Aufwand zur Integration der Komponente in das Zielframework deutlich reduziert. d Zur Unterstützung der Applikationsentwicklung existieren einige vielversprechende Ansätze in der Literatur (vgl. Abschnitt 4.1). In der Regel sind diese jedoch gebunden an ein bestimmtes Framework oder Komponentenmodell. Zum Zeitpunkt dieser Arbeit zeichnet sich die Durchsetzung eines standardisierten Meta-Modells für Kommunikationsmuster in der Robotik noch nicht ab, von dem die Komponentenmodelle der aktuellen Robotik-Middlewares abgeleitet werden. Deshalb wird eine direkte Einbindung von MDE-Werkzeugen in die Integrations- und Testplattform (A16, A5) aufgrund der Anforderungen A37 und A36 in dieser Arbeit nicht weiter verfolgt. Auch für die automatisierte Durchführung von Komponenten- und Applikationstests existieren bereits viele Werkzeuge, die aufgrund ihrer Plattformunabhängigkeit in die Integrations- und Testplattform eingebettet werden können. Die Anforderungen A14 und A18 werden im Folgenden nicht weiter betrachtet.

4.2 Vorgehensmodelle zur Unterstützung des verteilten Entwicklungs- und Integrationsprozesses

Im Folgenden werden Vorgehensmodelle in der Softwaretechnik vorgestellt und auf die Anwendbarkeit für die verteilte Entwicklung von Serviceroboter-Applikationen hin untersucht.

4.2.1 Sequenzielle Phasenmodelle

Das **Wasserfallmodell** wurde in seiner ursprünglichen Form von Dr. Winston W. Royce 1970 eingeführt und gilt als der klassische Vertreter der Phasenmodelle [Roy70, Roy87]. Es

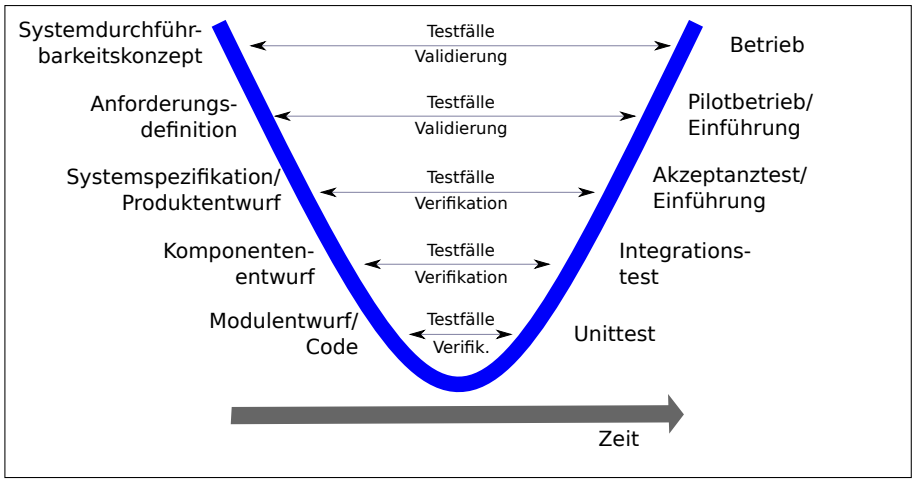


Abbildung 4.1: Das V-Modell

teilt den Softwareentwicklungsprozess in verschiedene Phasen ein, die nacheinander abzuarbeiten sind. Durch die streng sequenzielle Anordnung und die einfache Phasenstruktur bietet das Wasserfallmodell potenziell eine einfache Organisationsstruktur für Software-Projekte und eine direkte Kosten-, Termin- und Ressourcen-Planung sowie eine starke Regelung des Projektablaufs.

Das **V-Modell** [Brö93] basiert ebenso wie das Wasserfallmodell auf sequenziellen Entwicklungsphasen. Im Unterschied zum Wasserfallmodell steht anstatt der Entwicklungsaktivitäten jedoch das Entwicklungsprodukt sowie die das Produkt überprüfenden Tests im Zentrum (s. Abbildung 4.1). Das V-Modell beschreibt im Detail, „wer“ „wann“ „was“ in einem Projekt zu tun hat, gibt jedoch nicht vor „wie“ es zu tun ist, d.h. es ist unabhängig von bestimmten Methoden und Notationen (z.B. UML, ERM, etc.).

Als eine konkrete Implementierung des V-Modells ist das V-Modell XT zu nennen, das seit 2005 das Standardvorgehensmodell des Bundesverwaltungsamts [Bea10] ist. Vorteilhaft in diesem Modell ist die systematische Validierung und Verifikation für alle Projektphasen, so dass eine Qualitätssicherung sehr einfach gewährleistet werden kann.

Sequenzielle Phasenmodelle haben jedoch grundsätzlich die folgenden Nachteile: Die Anwender des zu entwickelnden Systems sind nur zu Beginn in den Analysephasen mit einbezogen, so dass auf eine Änderung von Anforderungen während der Entwicklung nur schwer reagiert werden kann. Da im Modell weitreichende Rückkopplungen über mehrere Phasen hinweg nicht vorgesehen sind, wiegen Fehler insbesondere in den frühen Phasen schwer. Im schlimmsten Fall werden z.B. Fehler in der Anforderungsanalyse erst am Ende des Projekts durch den Anwender offenbar. Da in vielen Softwareprojekten (und insbesondere bei Forschungsprojekten) die Anforderungen nicht zu Beginn des Projekts präzise bekannt sind

Tabelle 4.1: Gegenüberstellung der agilen Entwicklungsmethodik mit den klassischen Vorgehensmodellen

Agile Manifest		sequenzielle Vorgehensmodelle
Kontinuierliche Softwareauslieferung (im Rhythmus von 3-4 Wochen)	↔	Auslieferung am Ende bzw. nach großen Meilensteinen
Kontinuierliche Überprüfung der Anforderungen	↔	Änderungen der Anforderungen in einer späteren Phase nicht vorgesehen
Enge Zusammenarbeit zwischen Auftraggeber, Entwickler und Nutzer	↔	Kommunikation nur in Reviews
Kleine Teamgröße an einem Ort (Face-To-Face Kommunikation)	↔	Teamgröße nicht eingeschränkt
Selbstorganisierte Teams	↔	Planung und Organisation von außen
Vertrauen in die Leistungsfähigkeit des Teams	↔	Feingranulare Planung aller Workpackages für jeden Entwickler
Iterativer Entwicklungsprozess	↔	Möglichst langfristig durchgeplanter Prozess
Lauffähige Software ist das Hauptmaß des Erfolgs	↔	Definierte Metriken und Kriterien

und über den Projektverlauf selten stabil bleiben, wird dieses Modell meist nur noch bei sehr risikoarmen Projekten und bei der Umsetzung bekannter Verfahren angewandt.

4.2.2 Agile Entwicklungsmethoden

In Projekten mit großen Unsicherheiten oder häufigen Änderungen der Anforderungen (wie im Falle von Forschungsprojekten) zeigt sich der klassische gesamtheitlich und langfristig durchstrukturierte Softwareentwicklungsprozess nach einem sequenziellen Phasenmodell als zu unflexibel und schwerfällig. 2001 begründeten deshalb 17 Entwickler das Agile Manifest [BBB+01], in dem die Grundlagen der agilen Softwareentwicklung festgelegt sind. In Kernpunkten nimmt das Agile Manifest diametral entgegengesetzte Positionen zu sequenziellen Vorgehensmodellen ein (s. Tabelle 4.1)

Scrum ist ein prominentes Beispiel agiler Entwicklungsmethoden; es wurde von [SB01] entwickelt und ist heute eines der am häufigsten eingesetzten agilen Vorgehensmodelle (vgl. [Glo09]). Schwaber formuliert den agilen Kernsatz von Scrum folgendermaßen: “Scrum akzeptiert, dass der Entwicklungsprozess nicht vorherzusehen ist. Das Produkt ist die bestmögliche Software unter Berücksichtigung der Kosten, der Funktionalität, der Zeit und der Qualität” [SB01].

Extreme Programming (Kurzform XP) ist eine schlanke agile Entwicklungsmethodik, die sehr stark im nordamerikanischen Raum verbreitet ist (vgl. [BA04]). Neben den typischen Eigenschaften der agilen Methoden, u.a. kleine Entwicklungsteams und starke Ein-

bindung des Kunden legt XP einen besonderen Schwerpunkt auf den Software-Test durch das Paradigma der "testgetriebenen Entwicklung" (engl. Test-Driven Development, TDD) sowie die zentrale Nutzung von CI-Werkzeugen.

Der Vorteil agiler Methoden liegt in ihrer Flexibilität bezüglich häufiger Änderungen der Anforderungen, wie es in innovativen Projekten der Fall ist. Auf der anderen Seite lassen sie sich jedoch grundsätzlich nur schwer 1:1 für große und örtlich verteilte Projektteams umsetzen (vgl. [Cro04]).

4.2.3 Iterative und inkrementelle Vorgehensmodelle

Das **Spiralmodell** von Böhm [Boe88] ist streng formal nicht als Vorgehensmodell im Sinne der Definition in Abschnitt 2.1 zu bezeichnen, da es keine konkreten Handlungsanweisungen für die Systementwicklung in den einzelnen Phasen gibt, sondern eher ein Verfahren für das Risikomanagement von Projekten.

Durch die iterative Vorgehensweise und die wiederkehrenden Risikoanalysen, Evaluationen, Implementierungen und Reviews werden die Nachteile von sequentiellen Modellen vermieden. Das Spiralmodell zeichnet sich auch durch die fortwährende Erstellung von Prototypen aus, die eine kontinuierliche Prüfbarkeit des Systems erlauben [Bal08]. In dieser Hinsicht könnte man das Spiralmodell als einen Vorläufer der agilen Entwicklungsmethoden bezeichnen.

Der **Rational Unified Process (RUP)** [Kru04] ist ein objektorientiertes, aktivitätsgetriebenes inkrementelles Vorgehensmodell, das seit 1999 als konkrete Implementierung des Unified Process [JBR99] von Rational (später IBM Rational) gepflegt und als kommerzielles Produkt vertrieben wird. Der RUP ist sehr stark auf die Unified Modelling Language (UML) ausgerichtet und liefert eine Methode zur Softwareentwicklung auf Basis der UML. Der RUP ist unter anderem durch eine komponenten-basierte Entwicklung charakterisiert und zeichnet sich durch eine breite Palette an verfügbaren Werkzeugen aus.

Ein Projekt wird nach dem RUP in die folgenden vier Phasen gegliedert: Konzeption (*Inception*), Entwurf (*Elaboration*), Konstruktion bzw. Implementierung (*Construction*) und Übergabe (*Transition*). Jede dieser Phasen verläuft inkrementell und jede Iteration ist mit einem Meilenstein abgeschlossen. Quer zu den einzelnen Phasen werden Disziplinen definiert, die in den Phasen als Arbeitsschritte durchgeführt werden, z.B. Anforderungsanalyse, Implementierung, Test aber auch Projektmanagement, Konfigurations- und Änderungsmanagement sowie die Zusammenstellung von Werkzeugen, Prozessen und Methoden.

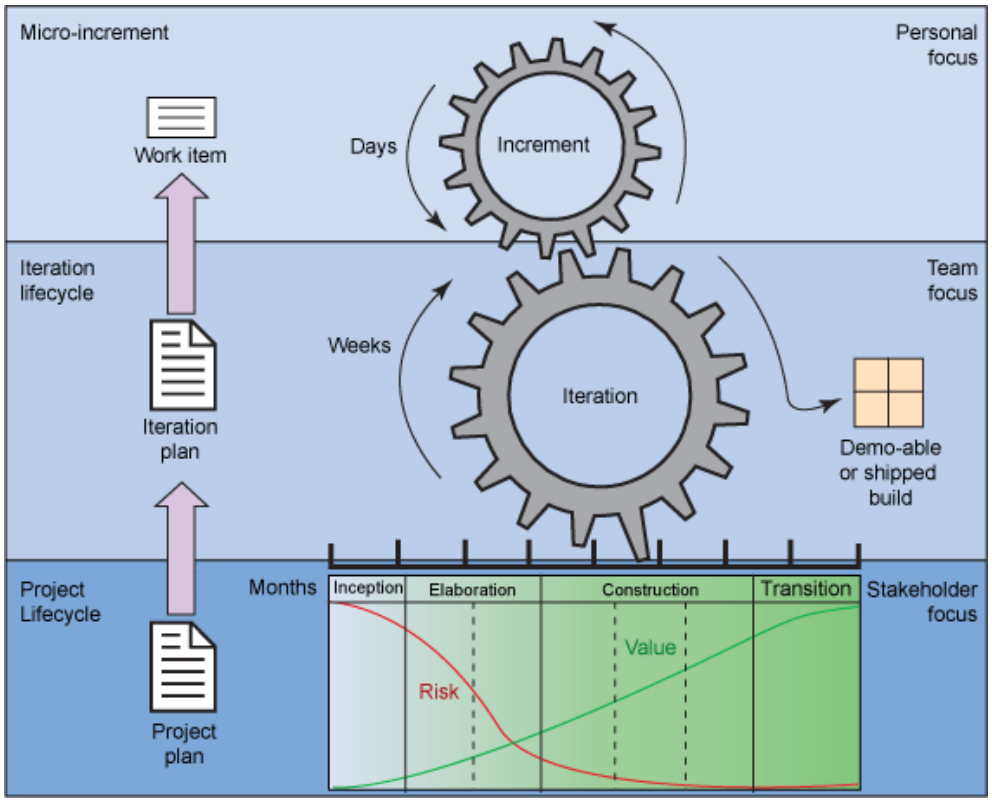


Abbildung 4.2: OpenUP Schichten: Projektebene, Teamebene und Entwicklerebene [Kro07]

Im Kontrast zum V-Modell XT wird der RUP vor allem für die Entwicklung freistehender Software-Applikationen eingesetzt und weniger zur Entwicklung kompletter Systeme inklusive Hardwareeinbettung. Neben dem V-Modell XT gehört der RUP zu den meistverwendeten Vorgehensmodellen in Deutschland [Rei01].

Der **Open Unified Process (OpenUP)** ist Teil des Eclipse Process Framework (EPF) und versucht, die Vorteile des RUP (Prozesse, Struktur, konkrete Methoden und Handlungsanweisungen) und agilen Methoden (Flexibilität, inkrementelle Entwicklung) zu verbinden [Gus08]. Agile Vorgehensmodelle bieten im Allgemeinen wenig konkrete Anleitung, sind oft kaum dokumentiert und ihre Anpassung auf ein spezifisches Problem erfordert ein hohes Maß an Erfahrung. Der RUP hingegen ist weniger flexibel und bringt durch seine Komplexität und Größe einen hohen Anpassungsaufwand für ein bestimmtes Projekt mit sich.

Die Kombination des RUP mit agilen Methoden ist in Abbildung 4.2 dargestellt: Die einzelnen RUP-Phasen sind durch die inkrementelle Entwicklung in Form von Iterationen überlagert. Die Forderung agiler Praktiken, regelmäßige stabile Software-Versionen auszuliefern, wird ebenso beibehalten. Auf der anderen Seite werden die Strukturelemente aus dem RUP übernommen, z.B. Dokumente und Meilensteine zwischen den einzelnen Phasen.

Besondere Eigenschaften des OpenUP sind vor allem die Bereitstellung als Open Source Framework und die Verwendung eines Prozess-Meta-Modells (SPEM [Obj08]). Insbesondere durch SPEM ist der OpenUP sehr einfach erweiterbar und kann auf verschiedene, insbesondere auch kleinere Projekte angepasst werden.

4.2.4 Folgerungen

Im Abgleich der verschiedenen Vorgehensmodelle mit den Anforderungen filtert das Kriterium "Anwendbarkeit für innovative Projekte" (Anforderung A21) schon viele der in der Industrie gängigen Vorgehensmodelle aus. Iterative und inkrementelle Vorgehensmodelle sind z.B. für Forschungsprojekte besser geeignet, da in den Iterationen auf sich ändernde Anforderungen reagiert werden kann. Weiterhin können detaillierte Pläne für einen langen Zeitraum insbesondere im forschungsintensiven Feld der Servicerobotik in den meisten Fällen nicht eingehalten werden. Einige inkrementelle Modelle wie das Spiralmodell oder RUP haben andererseits allerdings den Nachteil, dass sie sehr komplex sind und einen hohen Zusatzaufwand für das Projektmanagement und das Projektteam bedeuten. Die Einführung dieser Vorgehensmodelle für Forschungsprojekte mit kurzen Laufzeiten und Teams ohne Erfahrung in der Anwendung dieser Vorgehensmodelle ist daher eher unrealistisch. Agile Methoden wie XP und Scrum kommen wiederum mit sehr wenig Overhead aus, da sie von einer großen Leistungsfähigkeit des Teams ausgehen und daher keine formalen Prozesse für das Projektmanagement vorsehen. Oft entspricht aber dieses Idealbild eines Projektteams nicht der Wirklichkeit, so dass die agilen Modelle insbesondere für verteilte und suboptimal vernetzte Teams nicht implementierbar sind (Widerspruch zu A20).

Die werkzeuggestützte Steuerung von Prozessen (A22) inklusive werkzeuggestützter Dokumentation (A23) kann durch kommerzielle oder Open-Source Workflow-Management Systeme abgedeckt werden und wird daher in dieser Arbeit nicht weiter verfolgt.

In Tabelle 4.2 werden die verschiedenen Vorgehensmodelle hinsichtlich des Erfüllungsgrads der Anforderungen gegenübergestellt. Keines der vorgestellten Modelle erfüllt alle Anforderungen, so dass ein bestehendes Modell als Basis ausgewählt und zur Erfüllung der fehlenden Anforderungen erweitert werden muss.

Tabelle 4.2: Bewertung existierender Vorgehensmodelle anhand der in Kapitel 3 definierten Anforderungen

Kriterium Anforderung	Anwendbarkeit für verteilte Teams A20	Anwendbarkeit für innovative Projekte A21	Prozesse für Qualitätsmanagement A12
Wasserfallmodell	○ anwendbar	− für Projekte mit variierenden Anforderungen nicht geeignet	○ realisierbar
Spiralmodell	○ anwendbar	○ inkrementell, aber aufwändige Begleitmaßnahmen (z.B. Risikoanalysen)	○ realisierbar
V-Modell	• wird in großen und verteilten Projekten eingesetzt	− für Projekte mit variierenden Anforderungen nicht geeignet	• Qualitätssicherung verankert im Modell
RUP	• Auslegung für große Projekte	○ inkrementell, jedoch sehr komplexes Modell	○ Qualitätssicherung nicht im Modell verankert
Agile Methoden (Scrum, XP)	− Modell basiert auf enger räumliche Zusammenarbeit kleiner Teams (z.B. tägliche Meetings)	• kontinuierliche Änderungen der Anforderungen im Modell verankert	○ einige Tools, z.B. continuous Integration, sonst eher informelle Konzepte (z.B: XP)
OpenUP	○ agile Methoden erfordern räumliche Zusammenarbeit	• schlanker als RUP, inkrementelle Entwicklung	○ nicht explizit im Modell verankert

Legende:

• gut geeignet
○ neutral
− nicht geeignet

4.3 Deployment-Technologien

Im Folgenden werden diejenigen Deployment-Techniken aus Industrie und Forschung untersucht, die für die Installation, Konfiguration und Inbetriebnahme von Software-Komponenten auf komplexen Servicerobotern relevant sind.

4.3.1 Deployment-Technologien aus der Industrie

In der Industrie werden vielfältige Technologien für das Deployment von Software eingesetzt. Eine weit verbreitete Kategorie stellen die nutzergetriebenen Installationsprogramme wie Installshield [Flea] oder InstallAnyWhere [Fleb] dar, die meist im Wesentlichen aus einem Kompressionswerkzeug mit einer nutzerfreundlichen graphischen Oberfläche bestehen. Eine weitere Kategorie bilden die Paketmanagementsysteme, die oft mit Linux-Betriebssystemen mitgeliefert werden (z.B. [Red97], [JWSM93] oder Yum [V$^+$]). Paketmanager basieren auf dem Konzept von Paketen, die aus auf dem Zielsystem zu installierenden Dateien und Metadaten (z.B. Versionsinformationen, Abhängigkeiten) bestehen, und einem Repository, das diese Pakete enthält. Sie bilden mehrere Aktivitäten des generischen Deployment-Prozesses ab, von der Unterstützung bei der Paketerstellung (Release) über die Installation, Deinstallation und Aktualisierung der Pakete unter Berücksichtigung ihrer Abhängigkeiten sowie teilweise sogar der Konfiguration von Paketen. Die Ausführung wird jedoch nicht abgedeckt (in den Paketen müssen sich keine ausführbaren Dateien befinden), und das Deployment wird nicht für verteilte Systeme unterstützt.

Auch sogenannte Systems Management Tools wie Microsoft System Center [Micc], IBM Tivoli Management Environment [IBMb] oder Altiris Deployment Solution [Sym] bieten viele Deployment-Aktivitäten; allerdings sind diese Systeme für mittlere bis große Unternehmen mit Tausenden von Rechnern ausgelegt und aufgrund ihrer Komplexität nicht für so spezielle Zielsysteme wie Serviceroboter geeignet. Webbasierte Deployment-Werkzeuge wie Windows Update [Mice] oder Java Webstart [Orac] laden automatisch neue Versionen einer Applikation von einem Web-Server und installieren diese auf dem Zielsystem. Dazu benötigen sie jedoch permanente Internetkonnektivität. Remote Session Werkzeuge wie ssh [Ope99] oder PowerTCP [Dar] haben den Vorteil, dass Applikationen von Client-Rechnern gestartet werden können, ohne dass sie lokal installiert werden müssen. Die Applikationen müssen lediglich auf dem Server vorgehalten werden, so dass Inkonsistenzen bezüglich unterschiedlicher Softwareversionen unter den Clients vermieden werden können. Ein Nachteil besteht in der potenziell hohen Netzwerkauslastung bei vielen gleichzeitigen Client-Zugriffen. Zu dieser Kategorie zählt auch Virtual Network Computing (VNC) Software, z.B. RealVNC [Rea].

Schließlich sind eventbasierte Werkzeuge für die Einrichtung von verteilten Applikationen zu nennen, z.B. TIBCO Rendezvous [TIB] oder Sun Java Message Service [Orab]. Sie bieten meist eine Software-Schnittstelle zur Integration in die verteilte Applikation an und werden daher in der Regel als message-basierte Middleware eingesetzt. In Tabelle 4.3 werden die verschiedenen Technologien zusammengefasst und bezüglich ihrer Abdeckung des generischen Deployment-Prozesses vergleichend dargestellt.

Tabelle 4.3: Vergleich von verschiedenen Deployment-Technologien bezüglich der abgedeckten Aktivitäten des generischen Deploymentprozesses [Hey08].

Deployment-Technologie	Beispiel	Release	Acquire	Planning	Install	Configure	Activate	Deactivate	Update	Uninstall	Retire
Installations-programme	Install-Shield, Install Anywhere	•			•	○			○	•	
Paketmanagement-systeme	dpkg, yum, RPM	•		○	•	○			•	•	
Webbasierte Deploymenttools	Windows Update, Java Webstart	•	•		•	○			•	•	
System Management Tools	Microsoft System Center, IBM TME, Altiris			•	•	•	•	•	•	•	
Remote Session/ VNC Werkzeuge	ssh, PowerTCP, RealVNC						•	•			
Publish/Subscribe Tools	TIBCO Rendezvous, IBM Gryphon, JMS	•	•		•				•		

Legende:

- • unterstützt
- ○ teilweise unterstützt

4.3.2 Deployment-Methoden aus der Forschung

Die im Folgenden dargestellten Deployment-Methoden haben ihren Ursprung meist in den verschiedenen Disziplinen des Software-Engineering, allerdings sind auch Konzepte aus der künstlichen Intelligenz vertreten (s. Tabelle 4.4).

In architektur-getriebenen Deployment-Verfahren kommen abstrakte Architekturbeschreibungssprachen (ADLs) zum Einsatz, um valide Deployment-Konfigurationen zu spezifizieren. Als Beispiele für architektur-basierte Deploymentverfahren sind Prism-DE [MRM02]

und Olan [BBB+98] zu nennen. Prism-DE unterstützt das initiale Deployment und Aktualisierungen für komponenten-basierte Applikationen für verteilte, heterogene und mobile Zielsysteme und basiert auf dem architekturalen Konzept von Komponenten, Konnektoren und (Deployment-)Konfigurationen. Komponenten enthalten die Algorithmik und halten ihren Zustand vor; Konnektoren regeln die Interaktion zwischen Komponenten, und Konfigurationen beschreiben die entstehende Komponenten-Topologie der Applikation bzw. des Systems. Obwohl die architekturalen Konzepte grundsätzlich plattformunabhängig sind, werden sie in konkreten Implementierungen meist auf spezifische Komponentenmodelle, Middlewares oder Architekturen angepasst. So basiert Prism-DE z.B. auf der Architektur PitM [MMR01]. Diese Implementierungen sind also in der Regel nicht mehr plattformunabhängig.

Andere Ansätze basieren auf mobilen Software-Agenten, die auf verschiedene Hosts im verteilten System migrieren und Aktivitäten des Deployment-Prozesses unterstützen können. Software Dock [HHHW97] z.B. unterstützt die Kooperation von Software-Produzenten und Software-Konsumenten: Auf Produzenten-Seite hält das "Release Dock" Softwarepakete sowie Informationen bezüglich deren Funktionalität, Version, etc. vor, auf der Konsumentenseite verwaltet das "Field Dock" Informationen zum Zielsystem (z.B. aktuelle Konfiguration, Randbedingungen wie Hardware und Betriebssystem, etc.), auf dem die Applikation installiert werden soll. Mobile Agenten docken sich auf beiden Seiten an und führen dort bestimmte Deploymentaktivitäten aus (Install, Update, Remove, etc.). Die Informationen zu Software-Produkt und Zielsystem werden formal durch eine Beschreibungssprache spezifiziert, die "Deployable Software Description" (DSD) und von den mobilen Agenten interpretiert. Allerdings beschränkt sich das Framework im Wesentlichen auf die Installation, Konfiguration und Aktualisierung von Software-Komponenten.

Weiterhin existieren spezielle Ansätze für Grid Computing (z.B. Globus Toolkit [LPP04]) oder Systeme, die eine dynamische Rekonfiguration während der Laufzeit erfordern (z.B. MagicBeans [CEM04]). Aus dem Bereich der künstlichen Intelligenz wurden Deployment-Planer für resourcen-beschränkte Einsatzumgebungen entwickelt (z.B. Sekitei [KIK03] oder CANS [FK03]). Caspian [Hey06] beschreibt einen graph-basierten Algorithmus zur Deploymentplanung von komponenten-basierten Applikationen auf einer verteilten Zielumgebung unter Berücksichtigung von Gütekriterien wie Effizienz, Zuverlässigkeit, Verfügbarkeit oder Robustheit. Formale Deployment Frameworks wie LTS [LS06] definieren plattformunabhängige Formalismen für die Aktivitäten des Deployment-Prozesses und liefern für den Entwurf von Deployment-Werkzeugen eine theoretische Basis.

Als Beispiele für modellgetriebene Deployment-Verfahren sind z.B. DAnCE [DBO+05] und der OMG Deployment und Configuration Standard [Obj06b] zu nennen (vgl. auch Abschnitt 4.3.2). Für modellgetriebene Verfahren spielen Ontologien und domänenspezifische

Sprachen (engl. Domain Specific Languages, DSL) eine große Rolle, die neben der Code-Generierung zur Erstellung von Ressourcenmodellen für Software- und Hardwarearchitekturen und dadurch für die algorithmische Deployment-Planung genutzt werden können. Für die Entwicklungsdomäne "Embedded Software und Echtzeitsysteme" existieren bereits UML Profile und Modellierungssprachen, z.B. AADL [FLVC05] und UML Marte [MAR09]. Auch für die Robotik werden Ontologien und Domänen-Modellierungssprachen in der Literatur vorgeschlagen, z.B. durch Lortal et al. [LDG11] oder Dhouib et al. (RobotML) [DKS+12]. Ontologien können somit die Integration vereinfachen und die Wiederverwendbarkeit erhöhen, nach Nilsson et al. [NMNF09] allerdings nur, falls die Spezifikation und die Schnittstellen der Hardware-Komponenten standardisiert genug sind.

In Tabelle 4.4 werden die unterschiedlichen Verfahren gegenübergestellt und hinsichtlich ihrer Abdeckung des gesamten Deployment-Prozesses verglichen.

Tabelle 4.4: Vergleich von verschiedenen Deployment-Ansätzen aus der Forschung bezüglich der abgedeckten Aktivitäten des generischen Deploymentprozesses [Hey08].

Deployment-Technologie	Beispiel	Release	Acquire	Planning	Install	Configure	Activate	Deactivate	Update	Uninstall	Retire
Modell-getr. Deployment	OMG D&C		●	●	●	●	●		○		
	DAnCE		●	●	●	●	●	●	○	●	
Architektur-getr. Deployment	Prism-DE		○	●	●		●		●		
	Olan	●	●		●	●	●				
Agentenbas. Deployment	Software Dock	●	●		●	●			●		
	TACOMA	●	●		●						
Grid Deployment	Globus Toolkit			●	●	●	●				
KI-Planer-bas. Deployment	Sekitei			●							
	CANS		●	●	●	○			○	○	
Dienstgütebas. Deployment	Caspian			●							
	DeSi			●							

Legende:

- ● unterstützt
- ○ teilweise unterstützt

Der OMG Deployment and Configuration Standard

Der OMG Deployment and Configuration Standard (OMG D&C) spezifiziert Modelle und Schnittstellen, die das Deployment und die Konfiguration von komponenten-basierten Applikationen auf heterogene verteilte Zielsysteme erleichtern sollen. Die Spezifikation umfasst dabei ein plattformunabhängiges Modell (PIM), das wiederum Modelle für Komponenten,

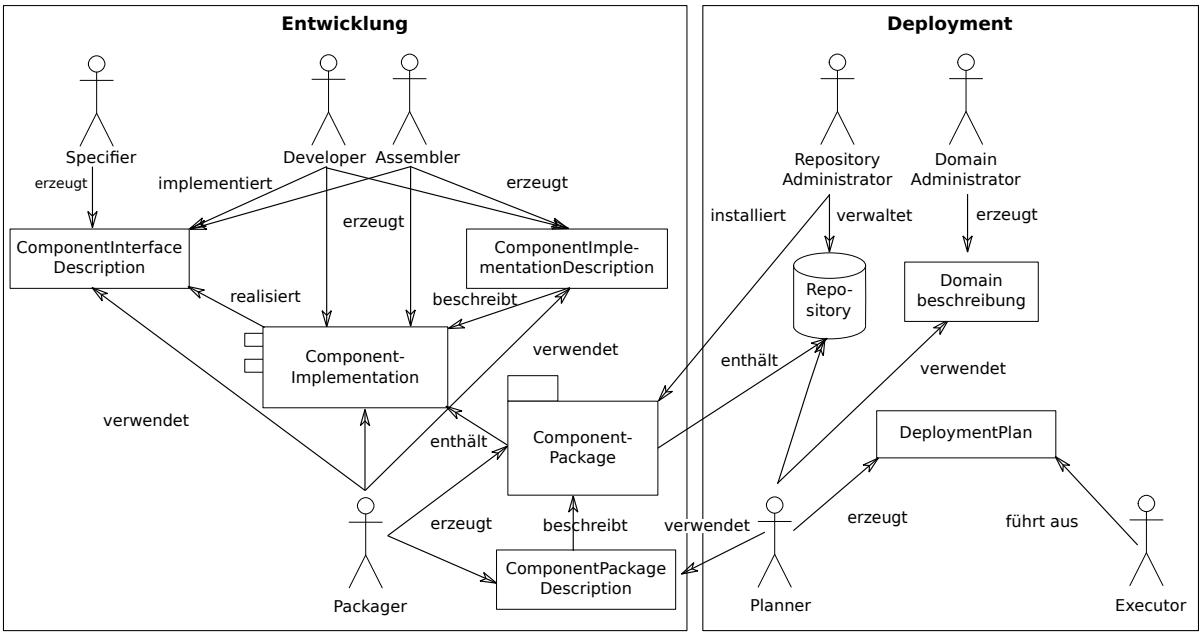


Abbildung 4.3: Vereinfachte Darstellung der Rollenverteilung im OMG Deployment and Configuration Standard [Obj06b]

das Zielsystem und die Ausführung der Applikation beinhaltet, und eine plattformspezifische Umsetzung für das CORBA Komponentenmodell (CCM). Diese Modelle sind jeweils eingeteilt in Datenmodelle, die Schnittstellen und Abhängigkeiten zwischen den einzelnen Artefakten beschreiben, und Laufzeit- bzw. Managementmodelle, die Schnittstellen und Aktionen für Werkzeuge zur Unterstützung des Deployment-Prozesses vorschlagen. Darüberhinaus definiert der OMG D&C Standard die folgenden Entwicklerrollen, deren Zusammenspiel in Abbildung 4.3 dargestellt wird.

Der *Specifier* erstellt die Schnittstellenspezifikationen für die einzelnen Komponenten, die vom *Developer* oder *Assembler* implementiert werden. Der Assembler baut hierbei auf Teilen bereits existierender Komponenten auf, während der Developer eine neue Implementierung erzeugt. Der *Packager* fasst verschiedene Implementierungen nebst ihren Spezifikationen in einem Komponentenpaket zusammen. Auf der Deploymentseite installiert der *Repository Administrator* die Komponentenpakete zunächst in ein Komponenten-Repository. Von dort werden sie entsprechend einem Deploymentplan auf das Zielsystem transferiert und konfiguriert, so dass der *Executor* die Komponenten und die Zielapplikation schließlich in Betrieb nehmen kann. Der Deploymentplan wird vom *Planner* auf Basis der Komponentenbeschreibungen und der vom *Domain Administrator* erstellten Spezifikation des Zielsystems ("Domain") erstellt. Er definiert, welche Implementierungen der Komponenten auf welcher Ressource und mit welchen Kommunikationsverbindungen auf dem System instanziiert werden. Die Erstellung des Deployment-Plans stellt somit einen wesentlichen Teil

des Deployment-Prozesses dar; der OMG D&C gibt hierbei jedoch lediglich Empfehlungen für die Vorgehensweise.

In der Literatur existieren bereits einige Implementierungen bzw. Erweiterungen des OMG D&C. Als Beispiel ist an dieser Stelle DAnCE [DBO+05] zu nennen, das den OMG D&C für die Middleware CORBA implementiert und eine Erweiterung für Echtzeitsysteme mit einschließt.

4.3.3 Folgerungen

Trotz der Vielzahl an existierenden Deployment-Werkzeugen in Industrie und Forschung kann kein Werkzeug alle Anforderungen abdecken: manche sind an ein bestimmtes Komponentenmodell gebunden (↯ A37), andere nicht für verteilte Architekturen (↯ A1) geeignet oder unterstützen nicht die gesamte Deploymentkette (↯ A9, A10, A11, A31, A17, vgl. Tabelle 4.5). Die Anforderungen müssen deshalb durch eine Kombination verschiedener Ansätze erfüllt werden.

Nach Auswertung der Tabelle 4.5 bieten sich für die Aktivitäten *Release*, *Install*, *Update* und *Uninstall* Paketmanager an, da diese Technologie unabhängig von den zu installierenden Komponenten ist. Allerdings müssen die Paketmanager für den Einsatz in einem verteilten Zielsystem erweitert werden. Für die Planung des Deployments wird der Ansatz aus Caspian [Hey06] gewählt, da die Methodik im Gegensatz zu den meisten Alternativen unabhängig vom Komponentenmodell anwendbar ist. Dazu müssen die vorgestellten Modelle aus Caspian jedoch für die Zieldomäne Serviceroboter adaptiert und erweitert werden. Die plattformunabhängigen Modelle des OMG D&C Standards, insbesondere die Rollenverteilung, können für die Konzeption herangezogen werden.

4.4 Laufzeitunterstützung zur Bedienung und Wartung von Robotersystemen

Im Folgenden werden Ausführungsumgebungen für verteilte Systeme in der Literatur untersucht, die für die Laufzeitunterstützung komplexer Serviceroboter genutzt werden. Diese umfassen im einfachsten Fall Kommandozeileninterpreter und gehen über Web-Schnittstellen zu Teleoperations- und Diagnoseumgebungen bis hin zu graphischen Bedienoberflächen.

Tabelle 4.5: Bewertung existierender Deployment-Werkzeuge anhand einiger der in Kapitel 3 definierten Anforderungen

Kriterium	Anwendbarkeit für Domäne SR (A1)	Plattformunabhängigkeit (A36)	Abdeckung Deploymentprozess (A9-11, 17, 31)
Paketmanagement-systeme	◦ unterstützen keine verteilten Systeme	• plattformunabhängig	◦ keine Unterstützung der Laufzeitaktivitäten
Web-basierte Deploymenttools	◦ Technologie benötigt permanente Internetkonnektivität	◦ teilweise abhängig von Betriebssystem und Programmiersprache	◦ keine Unterstützung für Planungs- und Laufzeitaktivitäten
System Management Tools	– komplexe Software-Systeme, ausgelegt auf eine sehr große Anzahl von Rechnern	◦ meist zugeschnitten auf Unternehmen	◦ keine Unterstützung der Entwicklungsseite (Release, Acquire, etc.)
Remote Session Tools	• meist PC-basierte Rechnerarchitekturen bei komplexen SR	• Werkzeuge existieren für die meisten Betriebssysteme	◦ nur Unterstützung bei Laufzeitaktivitäten, keine Deployment-Planung
Deployment-Planer, z.B. Caspian	◦ Modelle für Applikation und Zielumgebung müssen für SR Domäne angepasst werden	• Plattformunabhängigkeit	◦ nur Unterstützung der Deployment-Planung
Modell-basierte Methoden, z.B. OMG D&C	• geeignet	– plattformunabhängige Modelle müssen für spezielle Komponentenmodelle implementiert werden	◦ weitreichende Abdeckung des Deployment-Prozess
Architektur-basierte Methoden, z.B. Prism-DE	• geeignet für verteilte, mobile Systeme	– basiert auf dedizierter Architektur	◦ gute Abdeckung des Deployment-Prozess
Agenten-basierte Methoden, z.B. Software Dock	◦ geeignet lediglich für lose gekoppelte verteilte Komponenten, Overhead durch Agenten	• plattformunabhängige formale Beschreibungssprache DSD	◦ gute Abdeckung des Deployment-Prozesses, keine Deployment-Planung

Legende:

• gut geeignet
◦ neutral
– nicht geeignet

4.4.1 Kommandozeileninterpreter

Für die Durchführung von Entwicklertests auf komplexen Servicerobotern kommen häufig Kommandozeileninterpreter (z.B. cmd [Mica], csh [Joy78], Bash [Fox89], etc.) zum Einsatz, mit deren Hilfe die einzelnen Komponenten – sofern sie in ausführbarer Form im Dateisystem vorliegen – per Befehlszeile gestartet und abgebrochen werden können. Die Konfiguration erfolgt durch direktes Editieren entsprechender Parameterdateien im Dateisystem. Die Lokation dieser Dateien muss dem Entwickler dazu bekannt sein. Die Konsolen können wiederum eingebettet in eine integrierte Entwicklungsumgebung (engl. integrated development environment, IDE) sein, so dass der Entwickler direkt nach der Übersetzung des Quellcodes die Komponenten aus dieser Umgebung heraus starten kann, ohne Befehlszeilen verwenden zu müssen. Ein Beispiel für eine solche IDEs ist eclipse [Ecl]. Skriptsprachen wie Python bieten ebenfalls Kommandozeileninterpreter an. Die Kommandozeileninterpreter können über Remote Session Werkzeuge wie ssh (vgl. 4.3.1) auch auf einem entfernten Rechner über Netzwerk genutzt werden.

4.4.2 Webbasierte Ausführungsumgebungen

Webbasierte Ausführungsmechanismen sind im Vergleich zu Kommandozeileninterpreter abstrakter vom zugrundeliegenden Betriebsystem und der verwendeten Programmiersprache und können ortstransparent zur Kommunikation – meist über xml-basierte Nachrichtenformate – mit Komponenten oder Anwendungen eingesetzt werden.

Für das Robotik Framework ROS existiert z.B. eine Webanbindung der verfügbaren Schnittstellen über Javascript (rosjs, [OJC+11]). Diese Web-Schnittstellen werden in der ROS-Community für den Fernzugriff zur verteilten Entwicklung auf den PR2-Roboter [Wil10] genutzt. Dazu wurde 2011 von einem Konsortium der Firmen Bosch und WillowGarage sowie der Brown University das PR2-RemoteLab [Rob11] entwickelt. Die PR2-Remotelab Umgebung erlaubt die Assoziation von Testdaten zu Quellcode im Repository, bietet eine effiziente WebGL 3D Visualisierung, die Darstellung mehrerer unterschiedlicher (sowohl roboter-interner als auch externer) Kameraansichten und bietet über rosjs eine Reihe von Fernsteuerungs- und Interaktionsmechanismen (z.B. Tastatursteuerung, Starten von Komponenten, Auswahl von Objekten in der Visualisierung). Das RemoteLab ist eine der elaboriertesten Ferntestumgebungen, allerdings ist sie auf eine bestimmte Hardware (PR2) optimiert und auf eine dedizierte Middleware beschränkt (rosjs). Das Deployment neuer Komponenten auf das Testsystem ist nicht vorgesehen.

Web-Schnittstellen für Serviceroboter werden auch von Blake et al. [BRWH11] propagiert, allerdings liegt hier der Schwerpunkt im bidirektionalen Austausch von verschiedenen Infor-

mationen (z.B. Telemetriedaten, Karteninformationen, Daten zur Greifpunktbestimmung, etc.) zwischen Internet und Roboter. Kim et al [KMO+05] verwenden Web-Services, um dem Roboter die verteilte Informationsbeschaffung z.B. über zu interagierende Objekte zu ermöglichen.

Ausführungsumgebungen für service-orientierte Architekturen

Service-orientierte Architekturen sind unter anderem charakterisiert durch einen zentralen Verzeichnisdienst, an dem sich alle im System verfügbaren Komponenten registrieren und Informationen über ihre verfügbaren Services bereitstellen. Darüber hinaus überwacht der Verzeichnisdienst, ob die registrierten Komponenten online oder offline sind. Werkzeuge zur Laufzeitunterstützung für service-orientierte Architekturen können sich somit direkt mit dem Verzeichnisdienst verbinden, um weitreichende Dienste anzubieten. Die ADE Entwicklungsumgebung für agenten-basierte Komponentenarchitekturen [Sch06a] stellt z.B. eine Ausführungsumgebung bereit, die neben der Konfiguration, Ausführung und Inspektion der Komponenten (Laufzeitansicht, *virtual machine view*) auch die Erstellung und Modifikation der Komponentenarchitektur (Architekturansicht, *architecture view*) ermöglichen soll. Für Service-Komponenten-Architekturen (SCA, vgl. Abschnitt 4.1) existieren ebenfalls graphische Werkzeuge zur Laufzeitunterstützung, z.B. Apache Tuscany [Apab] oder FraSCAti [OW209].

Service-orientierte Ausführungsumgebungen bieten eine einfache Möglichkeit zur Abstraktion von der Rechnerarchitektur, allerdings müssen die Software-Komponenten ihre Schnittstellen und Funktionalitäten als Services bereitstellen. Die aktuellen komponenten-basierten Software-Frameworks in der Robotik sind jedoch in der Regel datenfluss-orientiert [BGKB12].

4.4.3 Teleoperationsumgebungen

Bei Teleoperationssystemen liegt der Schwerpunkt neben der Entwicklung von Softwarestrukturen und Protokollen, die eine adäquate Übertragung multimodaler (visueller, auditiver, haptischer, etc.) Daten an den Nutzer gewährleisten sollen, häufig auf der Entwicklung von geeigneten Eingabegeräten, die die Bewegungen des Teleoperators intuitiv in den Konfigurationsraum des Roboters übersetzen können sowie teilautonomen Verfahren zur Erleichterung der Ausführung komplizierter robotischer Greif- und Handhabungsfähigkeiten (vgl. [NMH+10]). In [BTC99, BCS05] wird z.B. ein Java3D-basiertes Framework für die Teleoperation von Robotern in dynamischen Umgebungen präsentiert.

Weiterhin sind Telepräsenzsysteme (vgl. [TVRI+12], [CLK+11]) als Untergruppe der Teleoperationssysteme zu nennen, die im Wesentlichen aus mobilen Kommunikationsterminals bestehen und vor allem für die Anwendungsfelder Telemedizin und Telecare entwickelt und evaluiert werden. Beispiele sind der CompanionAble-Roboter [BEH+09] oder das kommerziell verfügbare Giraff-System [Gir11], das im Rahmen des Forschungsprojektes ExCITe evaluiert wird (vgl. [CCC+10]).

Die Rolle der mit dem Roboter interagierenden Person ist in den Teleoperations- und Telepräsenzsystemen vorrangig die des Anwenders. Die Interaktionsmechanismen sind somit kaum für verteilte Hardware- und Integrationstests ausgelegt. Das Deployment von Softwarepaketen aus Repositories sowie die Generierung von Testdaten ist bei diesen Fernzugriffsmechanismen in der Regel nicht vorgesehen.

4.4.4 Ferndiagnosesysteme

Ferndiagnosesysteme bieten vorrangig Konfigurations- und Introspektionsmöglichkeiten, die speziell auf eine bestimmte Komponente zugeschnitten sind. Sie werden vor allem für industrielle Systeme und Produkte entwickelt, die über längere Zeit hinweg möglichst effizient gewartet werden müssen. Das Fanuc Ferndiagnosesystem [FAN] z.B. ermöglicht den vollständigen Zugriff über Internet auf Fanuc Industrieroboter. Hierbei können Techniker und Kunde sogar gemeinsam durch die Support-Session geführt werden. KUKA Remote Control [KUK] ermöglicht darüber hinaus auch die Programmierung, die Inbetriebnahme und das Einspielen von Software-Updates aus der Ferne. ABB Remote Service [ABB] stellt die Verbindung zum Servicepersonal nicht nur im Störfall her, sondern überwacht kontinuierlich Schlüsselparameter, die die Lebenszeit des Roboters beeinflussen (z.B. Drehmomente in den Achsen) und meldet Auffälligkeiten proaktiv an den Nutzer. So soll neben der Reduktion der mittleren Reparaturzeit auch eine Verbesserung des mittleren Fehlerabstands erreicht werden. Der ABB Remote Service nutzt dazu den VPN-basierten Internetdienst Talk2M [eWO], der Konfigurations-Schnittstellen zu verbreiteten Automatisierungslösungen (z.B. Siemens SPS) anbietet.

Oft werden zur Ferndiagnose auch Remote-Session Werkzeuge (vgl. Abschnitt 4.3.1) wie ssh und VNC eingesetzt. Im Unterschied zu Werkzeugen wie ssh, das auch ohne Oberfläche nur auf Basis einer Eingabeaufforderung bedient werden kann, verbindet VNC Software immer zu einer Desktop-Oberfläche (und benötigt eine entsprechend höhere Netzwerkbandbreite). Auch die Anzahl möglicher gleichzeitiger Zugriffe ist bei VNC-Produkten meist geringer. 3D-Oberflächen können dabei mit Hilfe zusätzlicher Werkzeuge wie VirtualGL [Virt] durch die sequenzielle Übertragung von auf der lokalen Graphikkarte des Quellrechners gerenderten Bildern auf einem entfernten Rechner verfügbar gemacht werden. Neben VNC existieren

weitere kommerzielle Varianten von Remote Desktop Software, z.B. [Tea] und [Mich]. Remote Desktop Software ist meist an ein bestimmtes Betriebssystem gebunden, wohingegen VNC-basierte Systeme für mehrere Betriebssysteme einsetzbar sind.

Wörn et. al stellen ein Teleservice und Diagnosesystem für industrielle Produktionszellen auf Basis von CORBA vor [WLH99]. Im Forschungsbereich lassen sich hingegen kaum ausgereifte Diagnosesysteme für Serviceroboterplattformen finden. Manche Robotik-Frameworks sehen jedoch bereits Mechanismen und Datenstrukturen zur Entwicklung von Diagnosefunktionen vor (z.B. [Wat09]).

4.4.5 Folgerungen

Als Ausführungsumgebung wird in der Servicerobotik-Entwicklung meist die Eingabe von Befehlszeilen in Konsolen verwendet, die sehr viel Flexibilität bieten, jedoch detaillierte Kenntnisse der Komponentenstruktur auf dem Dateisystem der Zielplattform erfordern (Speicherort der Binär-, Konfigurations- und ggf. Logdateien). Bei verteilten Systemen ist jeweils noch die Kenntnis des entsprechenden Hostrechners der einzelnen Komponenten notwendig. Abstraktere Umgebungen, die z.B. Webschnittstellen verwenden, existieren für Robotersysteme kaum; die wenigen Ausnahmen sind in der Regel auf ein bestimmtes Zielframework bzw. eine bestimmte Middleware beschränkt. Teleoperationssysteme fokussieren meist nur einen Teilaspekt eines Systems zur Realisierung einer bestimmten Funktion und können daher für die Durchführung von Integrationstests, bei denen alle Software-Komponenten individuell bedient werden müssen, nur eingeschränkt verwendet werden.

4.5 Kooperationswerkzeuge

Im Laufe der letzten Jahre und Jahrzehnte haben sich viele Software-Engineering Werkzeuge zur Unterstützung der Kooperation in verteilten und lokalen Softwareentwicklungsprojekten etabliert. Das folgende Kapitel gibt einen Überblick über die wichtigsten Werkzeuge im Hinblick auf die Entwicklungsdomäne Serviceroboter. Zunächst werden allgemeine Werkzeuge zur Unterstützung der Zusammenarbeit in verteilten Projekten behandelt.

4.5.1 Unterstützung der Kollaboration in verteilten Projekten

In verteilten Projekten ist eine effiziente Kollaborationsstruktur von besonderer Bedeutung. Viele verschiedene Programme und Werkzeuge sind verfügbar, die jeweils mehr oder weni-

Tabelle 4.6: Raum-Zeit-Taxonomie nach Johansen [Joh91] zur Klassifikation von Kooperationswerkzeugen

	synchron	asynchron
gleicher Ort	Gruppenmoderationssysteme, Brainstormingunterstützung, Abstimmungswerkzeuge	Schwarzes Brett, Gruppenarbeitsraum
verschiedener Ort	Videokonferenzen Application Sharing, Virtuelle Sitzungsräume	E-Mails, Nachrichtensysteme, -blogs, Wissensmanagementsysteme, Gruppen-Portale, Whiteboards

ger für bestimmte Anwendungsfälle geeignet sind. In der Literatur werden Methoden und Werkzeuge für die Kollaboration oft unter den Begriffen "computer-supported collaborative work" (CSCW) oder "Group Ware" zusammengefasst. Die unterschiedlichen Groupware-Werkzeuge und Methoden lassen sich nach Johansen [Joh91] in einer Raum-Zeit-Taxonomie darstellen (vgl.Tabelle 4.6).

Im Folgenden werden existierende Werkzeuge in den Kategorien Kommunikation, Koordination und Kooperation bzw. Kollaboration vorgestellt. Es sei jedoch angemerkt, dass viele kommerzielle Groupware-Lösungen ein Bündel von Einzelwerkzeugen aus den verschiedenen Kategorien enthalten, z.B. Outlook E-Mail und Kalender. Somit ist eine klare Einteilung oft nicht möglich.

Kommunikationswerkzeuge können entsprechend der Taxonomie in Tabelle 4.6 in synchrone und asynchrone Werkzeuge eingeteilt werden. Zu den synchronen Werkzeugen gehören unter anderem Konferenzsysteme, die textbasiert (Chatsysteme oder Instant Messenger, z.B. Skype [Micg], ICQ [ICQ98], AIM [AOL], XMPP-basierte Clients [XMP]), audiobasiert (VOIP, Telefon) oder videobasiert (Skype) sein können [Kai01]. Kommunikationsorientierte asynchrone Werkzeuge [PB01] umfassen E-Mail, Bulletin-Board-Systeme (z.B. Lotus-Notes), Newsgroups (z.B. Google Groups) und verteilte Hypertext-Systeme (z.B. Wikis). Wikis (z.B. Trac Wiki [Edg07], MoinMoinWiki [JH00] oder MediaWiki [Wik03]) werden neben Dokumentenmanagementsystemen (DMS) (z.B. BSCW [Orb] oder agorum [ago]) häufig zur internen Dokumentation und zum Wissensmanagement genutzt.

Koordinationswerkzeuge erleichtern die Handhabung von Abhängigkeiten, insbesondere bei räumlicher Verteilung der beteiligten Personen [SSU01]. In diese Kategorie fallen z.B. Gruppenterminkalender (z.B. Google Calendar, Office Calendar, MS Exchange, Zimbra

Web Calendar) und Workflow-Management-Systeme (WfMS) [Mat12]. WfMS vereinfachen nicht nur die Koordination in verteilten Projektteams, sondern können auch zur Qualitätssicherung von Prozessen eingesetzt werden. Als Beispiele für WfMS ist Imixs [Imi] zu nennen, das teilweise als Open-Source zur Verfügung steht und mit dem sich Arbeitsabläufe modellieren, steuern und pflegen lassen. Weitere kommerzielle WfMS-Werkzeuge sind z.B. der PAVONE Espresso Workflow [GBS], Intalio BPM [Inta] und JIRA [Atl].

Kollaborationswerkzeuge können ebenfalls in synchrone und asynchrone Varianten eingeteilt werden. Synchrone kollaborationsorientierte Werkzeuge sind charakterisiert durch die gemeinsame Arbeit an einem Objekt (z.B. Dokument, Präsentation, Grafik) und umfassen meist auch Funktionen der Kommunikation und Koordination [HHS01]. Zu dieser Kategorie gehören beispielsweise MS NetMeeting [Micb] und Cisco WebEx [Cis]. MS NetMeeting wird in neueren Windows-Versionen allerdings ersetzt durch Windows Meeting Space und Microsoft Shared View. Kollaborationsorientierte asynchrone Werkzeuge [ABK01] unterstützen im Allgemeinen den Gruppenarbeitsprozess durch die Zugriffsmöglichkeiten auf gemeinsam genutzte Arbeitsmittel bzw. gemeinsam bearbeitete Artefakte. Als Realisierungen dieses “gemeinsamer Arbeitsbereich”-Konzeptes sind z.B. LotusNotes, Implementierungen des WebDAV-Standards [G. 99], BSCW [Orb] oder die Google Werkzeuge Google Docs [Goo10] und Google Cloud Connect [Goo11] sowie DropBox [Dro] zu nennen. Die meisten dieser Werkzeuge unterstützen Awareness-Dienste, z.B. das Senden einer Mail bei Änderung eines Dokuments auf dem gemeinsamen Arbeitsbereich. Eine Versionsverwaltung der Objekte im gemeinsamen Arbeitsbereich ist bei LotusNotes, BSCW, Google Docs und WebDAV-Implementierungen enthalten. Manche Werkzeuge sind auf bestimmte Dokumentarten beschränkt (z.B. Google Docs). Für die gemeinsame Bearbeitung von Quellcode werden jedoch ausschließlich dedizierte Versionierungssysteme verwendet.

4.5.2 Werkzeuge zur kollaborativen Softwareerstellung

In den letzten Jahrzehnten entstand eine große Bandbreite von Werkzeugen zur Unterstützung verteilter Softwareentwicklung. Im Folgenden werden die wichtigsten Werkzeugkategorien zur Unterstützung der kollaborativen Softwareerstellung vorgestellt. Werkzeuge für Integration und Test wurden bereits in Abschnitt 4.2 untersucht.

Versionsverwaltungssysteme sind heute die Grundlage für die kollaborative Softwareerstellung. Sie werden zur Versionierung von Quelltexten eingesetzt, um Änderungen zu erfassen und verschiedene Softwarestände archivieren zu können. Die einzelnen Versionen sind dabei mit Zeitstempel und ggf. einer Kennzeichnung versehen, so dass sie jederzeit wiederhergestellt werden können. Die meisten Versionsverwaltungssysteme unterstützen dabei die parallele Entwicklung an denselben Quelltexten (engl. “Branching”) sowie die nachträg-

liche, teils interaktive Zusammenführung unter Berücksichtigung aller Änderungen (engl. "Merging"). Versionsmanagementsysteme für die Softwareentwicklung können je nach Lokation der Repositories in verteilte und zentrale Systeme eingeteilt werden. Bei zentralen Systemen existiert nur ein gemeinsames Repository, in das alle Entwickler ihre Änderungen über Netzwerk einpflegen müssen. Als Beispiele sind die Open-Source Systeme Subversion [Pil04] und CVS [Ber90] zu nennen und proprietäre Systeme wie Microsoft Visual Source Safe [Micd] oder Rational ClearCase [IBMc]. Verteilte Systeme sind dadurch charakterisiert, dass jeder Entwickler sein lokales Repository vorhalten und dieses dann mit anderen Repositories zusammenführen kann. Beispiele hierfür sind Git [HT+05], Bazaar [Can], Mercurial [M+] oder BitKeeper [Bit].

Issue-Tracking Systeme werden dazu eingesetzt, um einzelne Aufgaben wie Fehlerbehebungen bestimmten Funktionsstellen oder Personen zuzuweisen (in Form eines sogenannten "Tickets") und deren Bearbeitungsstand zu verfolgen. Issue-Tracking Systeme bieten somit an zentraler Stelle einen Gesamtüberblick über die bearbeiteten Vorgänge in einem Projekt ohne Notwendigkeit für personelle Kommunikation. Beispiele für freie Issue-Tracking Systeme sind Trac [Edg07], Mantis [KI00] oder Bugzilla [Moz].

4.5.3 Integrierte kommerzielle Kollaborations-Plattformen

Große Softwareunternehmen wie Microsoft oder IBM kompilieren oft ihre gesammelten Entwicklungs- und Kollaborations-Werkzeuge zu umfangreichen integrierten Plattformen für die effiziente Zusammenarbeit in großen verteilten Teams. Der Microsoft Team Foundation Server (TFS) [Min06] z.B. unterstützt das verteilte Entwicklungsteam u.a. durch Workflows verschiedener Vorgehensmodelle, Werkzeuge für Versionskontrolle, Issue-Tracking, etc. sowie Strukturen für Koordination und Projektmanagement. Obwohl TFS auch eine Work-Group Edition für kleinere Projektteams anbietet, ist es vor allem für die Unterstützung von Großprojekten geeignet. Die IBM Jazz Suite [MTW09] wurde 2007 von IBM Rational entwickelt und zielt ebenfalls auf die Unterstützung von verteilter Entwicklung in Großprojekten. Kern der Suite stellt Rational Team Concert [IBMa] dar, das vor allem Kommunikationsfunktionen sowie verschiedene Versionierungswerkzeuge (u.a. Subversion, ClearCase, ClearQuest) und ein Build- und Releasemanagement umfasst.

Diese Kollaborationssuiten bestehen in der Regel aus hausinternen Werkzeugen, so dass die einzelnen enthaltenen Werkzeuge nicht für beliebige Entwicklungsumgebungen eingesetzt werden können. Oft sind die Kollaborationsfunktionen direkt als Erweiterungen in die Entwicklungsumgebung integriert, vgl. Microsoft Visual Studio Team System [GP06].

Für Forschungsprojekte, in denen oft viele verschiedene Entwicklungsumgebungen bei den verschiedenden Projektpartnern eingesetzt werden, eignen sich solche Plattformen deshalb kaum.

4.5.4 Folgerungen

Kooperationswerkzeuge für Wissensmanagement und Zusammenarbeit auf Code-Ebene existieren in großer Anzahl; die Anforderungen A28, A15, A29 und A27 werden durch den heutigen Stand der Technik abgedeckt. Kommerziell verfügbare Koordinationswerkzeuge wie Gruppenkalender und WfMS etc. decken die Anforderung A29 ab. Für die zentrale Dokumentation (A7) können Wikis verwendet werden. Große kommerzielle Kollaborationsplattformen, die diese Werkzeuge ebenfalls enthalten, können aufgrund des starken Vendor Lock-Ins kaum verwendet werden, da in der Forschung oft unterschiedliche Entwicklungsumgebungen genutzt werden. Stattdessen werden existierende Kollaborationswerkzeuge in die Integrations- und Testplattform eingebettet.

5 Lösungskonzeption

Im Folgenden werden die Konzepte für die einzelnen Funktionsblöcke der Integrations- und Testplattform entwickelt und am Ende des Kapitels zur Gesamtarchitektur zusammengeführt.

5.1 Entwicklung des Vorgehensmodells für die verteilte Entwicklung von komplexen Servicerobotern

5.1.1 Variantenbildung

Anhand der Folgerungen in Abschnitt 4.2.4 werden nun Modifikationen und Erweiterungen der bestehenden Vorgehensmodelle auf ihre Eignung für den Einsatz in Servicerobotik-Projekten untersucht. In Abbildung 5.1 sind verschiedene Vorgehensmodelle eingeordnet in Bezug auf den Verteilungsgrad des Projektteams und den Innovationsgrad des Projekts. Anhand dieser beiden Kriterien können jeweils auch verschiedene Projekttypen klassifiziert werden, von klassischen lokalen Industrieprojekten mit geringer Innovation über industrielle Forschung bis hin zu großen Verbundforschungsprojekten. Aus der Abbildung können so geeignete Vorgehensmodelle für einen bestimmten Projekttyp entnommen werden. Im Folgenden werden Varianten der Vorgehensmodelle gebildet, die die Eignung in Richtung Einsatzfähigkeit für verteilte, innovative Forschungsprojekte verbessern. Die Variationen der einzelnen Vorgehensmodelle sind in der Abbildung jeweils durch entsprechende Pfeile markiert.

Für Servicerobotik-Projekte mit geringer Innovationshöhe kann ein klassisches sequenzielles Vorgehensmodell wie das V-Modell XT interessant sein. Das im V-Modell sehr gut verankerte Qualitätsmanagement bietet sich vor allem für sehr konkrete und abgegrenzte Problemstellungen im kommerziellen Umfeld an, bei denen die Risiken bei der Entwicklung im Vorfeld sehr gut abgeschätzt werden können und die Robustheit und Zuverlässigkeit des

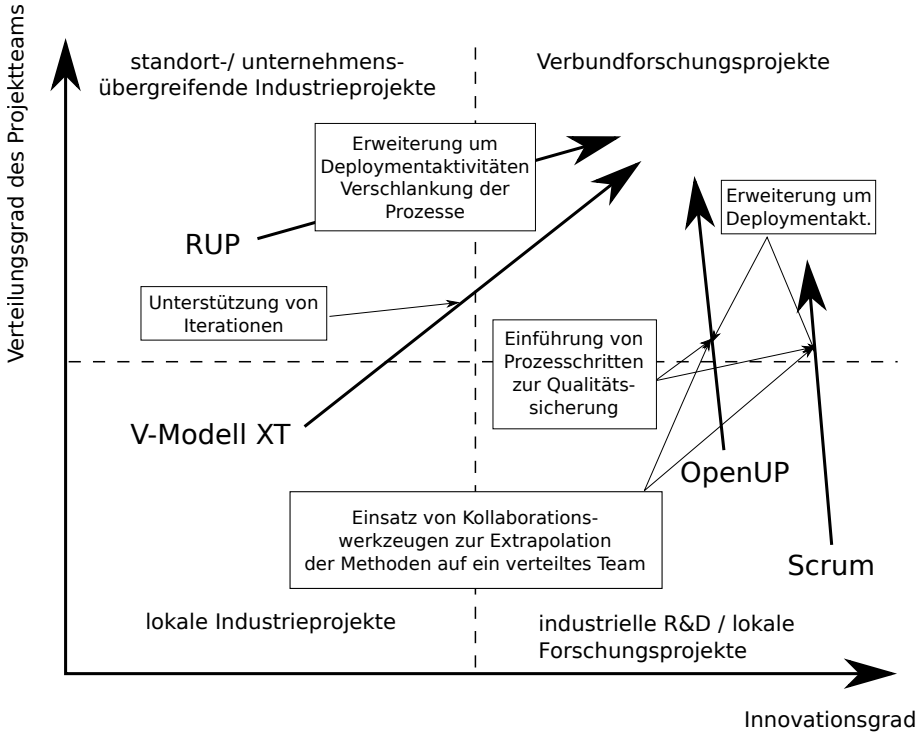


Abbildung 5.1: Modifikationsmöglichkeiten verschiedenener Vorgehensmodelle im Hinblick auf ihre Eignung für innovative, verteilte Forschungsprojekte.

Systems im Vordergrund steht. Je komplexer das Zielsystem hinsichtlich Hardwareaufbau und Funktionalität, desto höher jedoch das zu erwartende Risiko und desto eher ist eine zyklische Anwendung des Vorgehensmodells erforderlich. Dazu müssen einige Prozessschritte des V-Modells hinsichtlich Iterierbarkeit angepasst werden.

RUP wird vor allem für große unternehmensübergreifende Softwareprojekte genutzt und erfordert im Allgemeinen einen großen Einarbeitungsaufwand für das Projektteam. Soll RUP für Servicerobotik-Projekte eingesetzt werden, muss es zunächst um Prozesschritte für die Hardwareintegration bzw. das Deployment erweitert werden. Für Projekte im Bereich der angewandten (oder industriellen Vorlauf-)Forschung, die eine längere Laufzeit besitzen und ein sehr großes, verteiltes Team involvieren, das idealerweise schon einige Erfahrung mit RUP hat, kann dieses Vorgehensmodell passend sein.

Agile Methoden wie Scrum sind vor allem für kleine lokale, fokussierte und gut zusammenarbeitende Teams mit viel Erfahrung in der Zieldomäne geeignet, so z.B. in der industriellen Vorlaufforschung oder an angewandten Forschungsinstituten mit einer stabilen Mitarbeiterbasis. Soll z.B. Scrum für Verbundforschungsprojekte mit einem heterogenen, verteilten Projektteam eingesetzt werden, müssen administrative Vorkehrungen und Vereinbarungen getroffen sowie Kooperationswerkzeuge eingesetzt werden, die jederzeit eine Ad-Hoc Kommunikation der einzelnen Teammitglieder erlauben. Idealerweise werden zudem formale

Prozessschritte zur Qualitätssicherung eingeführt, die eher informelle agile Praktiken wie Code Reviews unterstützen. Je größer und verteilter und heterogener das Team ist, desto schwieriger sind die Kernmethoden der agilen Vorgehensmodelle jedoch in der Regel umzusetzen.

Einen Kompromiss zwischen der agilen und formalen prozessorientierten Welt bietet OpenUP an, das die Strukturen aus RUP mit agilen Methoden vereinigt. Der OpenUP genügt als inkrementelles und iteratives Vorgehensmodell mit wenig Prozessoverhead am ehesten der Anforderung A21 und enthält keine grundlegenden Hemmnisse hinsichtlich der Implementierung der übrigen Anforderungen. Die immanente agile Methodik ist ebenfalls durch geeignete Strukturen zur Kooperation zu unterstützen, so dass das Modell auch für ein verteiltes Team einsetzbar ist (A20). Weiterhin muss das Modell um die Deployment-Aktivitäten A9, A10, A11, A31, A17 und das Qualitätsmanagement A12 erweitert werden.

Im nächsten Abschnitt wird das Vorgehensmodell für verteilte Integration und Test, im Folgenden bezeichnet als DDT-Modell, (*D*istributed *D*evelopment and *T*est) deshalb auf Basis von OpenUP entwickelt.

5.1.2 Konzeption des DDT-Vorgehensmodells

Das DDT-Vorgehensmodell stellt Handlungsanweisungen und Abläufe für die Integration einer komponenten-basierten Applikation auf ein gemeinsames Zielsystem zur Verfügung. Die grundlegende Entwicklung von Komponenten und einer Architektur liegt somit nicht mehr im Anwendungsfeld des DDT-Modells, so dass die in OpenUP definierte Entwurfsphase (*Elaboration*) daher nicht mehr notwendig ist. Auf der anderen Seite soll aber die Implementierungsphase (*Construction*) zu einer Integrationsphase (*Integration*) um- bzw. ausgebaut werden, die den speziellen Anforderungen der verteilten Integration gerecht wird.

Die *Integration*-Phase erweitert OpenUP konkret in den folgenden Punkten:

- Das Entwicklungsinkrement wird aufgeteilt in ein Komponenten- und ein Applikationsinkrement
- Formale Qualitätskontrolle und Abnahme der einzelnen Komponenten
- Deployment-Prozess für abgenommene Komponenten
- Durchführung des Applikationsinkrements nach erfolgreichem Deployment der Komponenten
- Neue OpenUP-Rollen: Komponentenentwickler, Applikationsentwickler, Systemintegrator

Die wichtigste DDT-Erweiterung zum OpenUP besteht darin, dass für eine effiziente Durchführung zeitlich und räumlich entkoppelter Integrationstests eine einfache Form der Qualitätssicherung eingeführt wird. Diese wird durch das Einfügen der Aktivität "Abnahme und Deployment der Komponenten" nach dem Komponenteninkrement (s. Abbildung 5.2). Die Abnahmetests bestehen in der Regel aus den von den Komponentenentwicklern bereitgestellten Unittests, sollten aber durch weitere von unabhängigen Personen entsprechend der Komponentenspezifikation erstellten Tests ergänzt werden.

Die korrekte und gewissenhafte Durchführung der Abnahmetests ist für den weiteren Verlauf der Integrationsphase von erheblicher Bedeutung: falls stark fehlerbehaftete oder wenig robuste Komponenten als Pakete bereitgestellt werden, kann die weitere Entwicklung der Applikation oder auch die Weiterentwicklung der anderen Komponenten im nächsten Zyklus stark behindert oder verzögert werden. Nach erfolgreicher Abnahme werden die Komponenten als Pakete in einer neuen Revision zur Verfügung gestellt (Release).

Diese Releases werden auf dem System daraufhin werkzeuggestützt installiert und von den anderen Entwicklern konfiguriert und für Tests genutzt (vgl. Deployment-Werkzeuge, Abschnitt 5.2). Dadurch soll das notwendige Experten-Wissen des einzelnen Entwicklers trotz der hohen Abhängigkeiten in der Servicerobotik auf seine Komponente begrenzt werden. Applikationstests werden erst dann durchgeführt, wenn die Komponenten in einer stabilen und auf Fehler geprüften Version sowie in ausführbarer Form auf dem Zielsystem verfügbar sind. Durch diese Maßnahme wird gewährleistet, dass die Applikationstests nicht durch instabile Komponenten erschwert und in die Länge gezogen, sondern effizient durchgeführt werden können. Im DDT-Modell wird dies durch die Einführung einer Aktivität "Abnahme und Deployment aller Komponenten" zwischen den Aktivitäten der Komponentenentwicklung und der Applikationsentwicklung implementiert (s. Abbildung 5.2). Aus den Applikationstests können neue Anforderungen an die Komponenten entstehen, deren Implementierung für das Komponenteninkrement der nächsten Iteration vorgesehen werden kann.

5.1.3 Definition der DDT-Rollen

Die Trennung der Applikations- und Komponentenentwicklung wird auch in der Definition der Rollen des Vorgehensmodells abgebildet. Die Rolle des Entwicklers wird auf den Applikationsentwickler und den Komponentenentwickler aufgeteilt. Zusätzlich wird die Rolle des Systemintegrators eingeführt, der die Rolle des Testers um das Packaging und Deployment der einzelnen Komponenten auf dem Zielsystem erweitert. Die Rolle des Architekten entfällt, da das DDT-Modell nicht für die Architekturentwicklung konzipiert wird. Falls keine Analysten im Projekt verfügbar sein sollten, kann die Rolle des Analysten zusätzlich vom

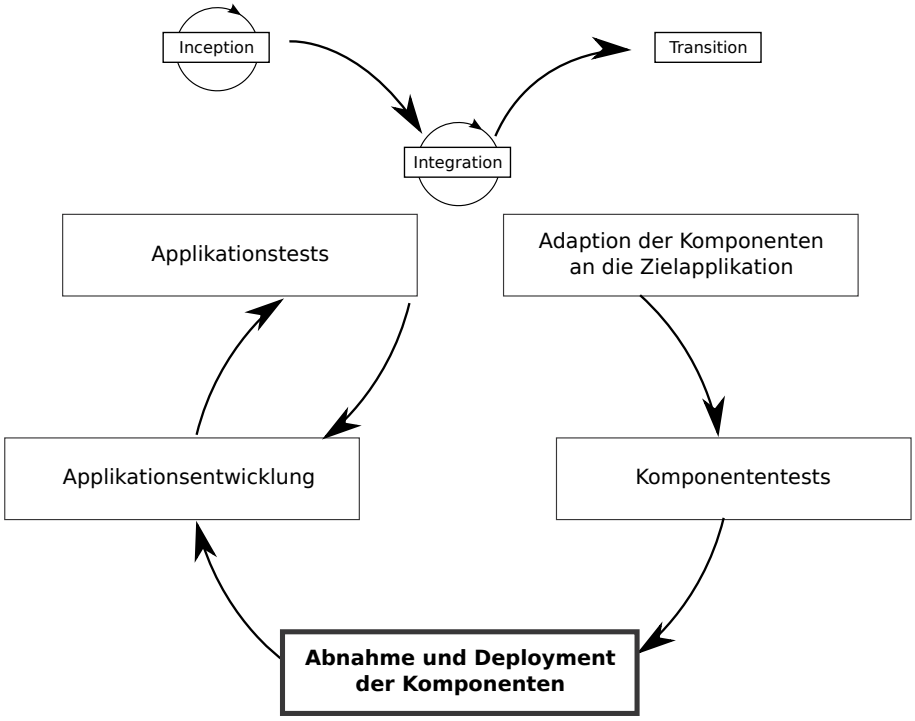


Abbildung 5.2: Integrationsphase des DDT-Vorgehensmodells.

Projektmanager ausgefüllt werden. Diese umfassen entsprechend dem OpenUP neben der Aufstellung der Systemanforderungen und der Detaillierung der Anwendungs- und Testfälle, die Analyse der Testergebnisse sowie die Planung und das Management von Iterationen. Abbildung 5.3 veranschaulicht die Gegenüberstellung der Rollen von OpenUP und DDT.

Die DDT-Akteure werden im Folgenden im Detail definiert und den Rollen des OMG Deployment und Configuration Standards gegenübergestellt.

Auftraggeber / Stakeholder

Der Auftraggeber ist für die Definition und Überprüfung der Anforderungen sowie für die Abnahme der ausgelieferten Lösung bzw. Applikation verantwortlich.

Projektleiter (PL)

Der Projektleiter oder Projektmanager ist zuständig für die operative Planung und Steuerung der im Vorgehensmodell verankerten Prozesse. Er trägt die Verantwortung für das Erreichen der definierten Ziele und verteilt die verfügbaren Ressourcen.

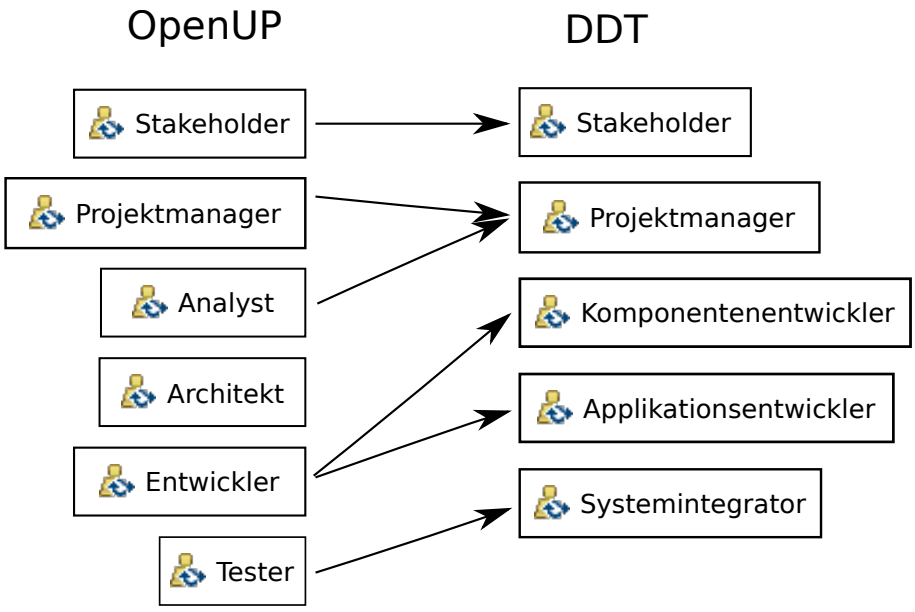


Abbildung 5.3: Rollen von DDT im Vergleich zu OpenUP.

Komponentenentwickler (KE)

Der Komponentenentwickler ist jeweils verantwortlich für die Entwicklung der ihm zugeordneten Komponente. Zu seinen Aufgaben gehört die Entwicklung und Einbettung der Algorithmen in die Kommunikationsstruktur und Architektur des Zielsystems sowie die Implementierung von entsprechenden Unittests. Sofern die Spezifikation für das Zielsystem es erfordert, hat der KE die Funktionalität der Komponente entsprechend zu erweitern. Diese Definition korrespondiert mit den Rollen *(ComponentInterface)Specifier*, *Developer*, *Assembler* des OMG Deployment und

Systemintegrator (SI)

Der Systemintegrator kennt den genauen Aufbau und die Schnittstellen der Hardwareplattform inklusive Aktorik, Sensorik und Rechnerarchitektur. Er ist in der Lage, bestehende Komponenten in das System zu integrieren, diese zu testen und in ausführbare Einheiten zu verpacken. Die Aufgaben des SI entsprechen somit den Rollen *Packager*, *Domain Administrator* und *Planner* des OMG D&C-Standards.

Applikationsentwickler (AE)

Die Applikationsentwickler implementieren die letztendliche Zielapplikation des Serviceroboters. Ihnen sind die Anforderungen an die Applikation bekannt. Entsprechend diesen

Anforderungen wählen sie geeignete Komponenten für die Applikationen aus und bauen auf deren Funktionalität, nehmen an diesen jedoch keine Änderungen vor. Die AE implementieren komponentenübergreifende Funktionen und insbesondere höherwertige Abläufe. Sie müssen für die Gesamtsystemtests in der Lage sein, alle Komponenten entsprechend bedienen und konfigurieren zu können. Sie verwenden dazu die vom Systemintegrator erstellten Pakete (entsprechend der Rolle des *Executors* nach OMG D&C). In kleinen Projekten mit einem kleinen lokalen Team können auch Komponentenentwickler teilweise die Aufgaben der Applikationsentwickler übernehmen.

5.1.4 Entwicklung der Modell-Phasen

DDT besteht aus den zwei Phasen *Initiierung (Inception)* und *Integration*, wobei die letzte Iteration der Integrationsphase der Transition in OpenUP entspricht. Durch kontinuierliche Deployments soll gewährleistet werden, dass zu einem definierten Zeitpunkt im Projekt alle Komponenten in einer integrierten Form mit einer Basisfunktionalität auf dem Zielsystem zur Verfügung stehen. Im Folgenden werden die einzelnen Phasen jeweils mit Bezug auf OpenUP beschrieben.

Die Initiierungs-Phase

Zu Beginn des Projektes werden analog zur OpenUP-Inception-Phase neben der Durchführung der Projektplanung die Anforderungen der geplanten Applikation spezifiziert. Neben den Anforderungen an die Funktionalität der Zielapplikation sollen an dieser Stelle in Erweiterung zu der Spezifikation der Initiierungs-Arbeitselemente im OpenUP explizit die Schnittstellen der einzelnen Komponenten spezifiziert werden. Auch das Zielsystem soll nach den Vorgaben des OMG D&C Standards spezifiziert werden (Rechnerkapazität, Kommunikationsstruktur, Ressourcen, Software-Architektur, etc.). Schließlich sollen für die jeweiligen Komponenten und die Zielapplikation auf Basis der Schnittstellendefinition getrennte Testcases entworfen werden. In einem finalen Schritt wird die Spezifikation formal auf Vollständigkeit bzw. Redundanz überprüft. Tabelle 5.1 veranschaulicht die zusätzlichen Aktivitäten und führt die jeweils verantwortlichen Rollen mit auf.

Die Integrations-Phase

Die Integrationsphase baut auf der *Construction*-Phase von OpenUP auf und erweitert diese um spezielle Aktivitäten für die Integration komponenten-basierter Applikationen. Das Entwicklungsinkrement wird aufgeteilt in die Komponenten- und Applikationsinkremente,

Tabelle 5.1: Aktivitäten in der Inception-Phase des DDT-Vorgehensmodells mit verantwortlichen Rollen: Projektmanager (PM), Komponentenentwickler (KE), Applikationsentwickler (AE), Systemintegrator (SI).

Aktivität	Beschreibung	Akteure
A1.1	**Initiierung des Projekts**	PM
	– Aktivitäten analog zu OpenUP	
	– Projektplanung	
A1.2	**Planung und Management der Iteration**	PM
	Aktivitäten analog zu OpenUP	
A1.3	**Spezifikation der Anforderungen**	PM KE AE SI
	– Aktivitäten analog zu OpenUP	
	– Spezifikation der Komponentenschnittstellen	KE
	– Spezifikation der Zieldomäne	SI
	– Definition von Testcases für Komponenten und Applikation	SI
	– Überprüfung der Spezifikation	AE SI

die durch eine formale Abnahme und das Deployment der Komponenten voneinander getrennt werden. Durch diese Maßnahme wird bereits im Vorgehensmodell methodisch die Erfüllung der Anforderungen bezüglich Deployment und Qualitätssicherung angelegt.

In Tabelle 5.2 werden die neuen Aktivitäten während einer Integrationsiteration im Detail beschrieben. Der Unterprozess "Abnahme der Komponenten" umfasst zum einen die OpenUP Test-Aktivitäten und schließt mit der formalen Abnahme durch den Systemintegrator. Zum anderen werden Pakete erstellt, die verschiedene Implementierungen der Komponente enthält und die Abhängigkeitsverhältnisse zu anderen Komponenten abbildet (vgl. z.B. das Debian-Paketmanagementsystem). Die Pakete bilden die Grundlage für den Deployment-Schritt, in dem zunächst auf Basis der Schnittstellenspezifikationen der Komponenten sowie der Domänenspezifikation ein Deployment-Plan erstellt wird. Dieser Plan enthält unter anderem, welche Komponente in welcher Implementierung auf welcher Ressource des Zielsystems laufen soll. Bei der Deployment-Planung müssen die Ressourcenanforderungen der einzelnen Komponenten berücksichtigt werden (s. folgender Abschnitt 5.2.2).

Die Instanziierung dieses Deployment-Plans vereinfacht schließlich das Starten von Komponenten und Applikationen erheblich (Anforderung A17), da der Operator zur Ausführung des Deployment-Plans keine Kenntnisse der Ressourcen des Zielsystems mehr benötigt. Weiterhin wird durch die Pakete das Laden von bestimmten Softwareständen (A35) technisch einfach umsetzbar. OpenUP fordert die Erstellung und Dokumentation von Testlogs, so dass Anforderung A23 bereits implizit erfüllt ist.

Die Transition im DDT-Zyklus wird nicht mehr als eigene Phase benötigt, da während der Integrationsphase kontinuierliche Deployments stattfinden. Die letzte Iteration der Integrationsphase entspricht somit der Transition, in der der finale Entwicklungsstand der Applikation an den Kunden ausgeliefert wird.

5.1.5 Entwicklung von Werkzeugen zur Unterstützung der Prozesse im Vorgehensmodell

Das in Abschnitt 5.1 entwickelte Vorgehensmodell ist abhängig von den übrigen Funktionsblöcken der Integrationsplattform. Insbesondere der Block "Laufzeitunterstützung" sowie der Einsatz von geeigneten Kooperationswerkzeugen ist erforderlich, um die Durchführbarkeit der agilen Methoden mit häufigen Iterationen durchzuführen. Diese Werkzeuge werden in den Abschnitten 5.3 bzw. 5.4 entwickelt. Ebenso sind Werkzeuge erforderlich, um die häufigen Abnahme- und Deployment-Aktivitäten des Systemintegrators sowie die Durchführung der Komponententests der Komponentenentwickler zu unterstützen. Diese

Tabelle 5.2: Aktivitäten in der Integrationsphase des DDT-Vorgehensmodells mit verantwortlichen Rollen: Projektmanager (PM), Komponentenentwickler (KE), Applikationsentwickler (AE), Systemintegrator (SI).

Aktivität	Beschreibung	Akteure
A2.1	**Planung und Management der Iteration**	PM
	Aktivitäten analog zu OpenUP, entspricht A1.2	
A2.2	**Aktualisierung der Anforderungen**	PM KE AE SI
	Aktivitäten entsprechen A1.3	
A2.3	**Komponenteninkrement**	KE
	Aktivitäten analog zum Entwicklungsinkrement in OpenUP bezogen auf die einzelnen Komponenten	
A2.4	**Abnahme der Komponenten**	SI
	– Implementierung der Komponenten-Testcases	
	– Durchführung der Komponenten Testcases	
	– Abnahme der Komponente und Erstellung eines Pakets	
A2.5	**Deployment der Komponenten**	SI
	– Erstellung eines Deployment-Plans	SI
	– Implementierung des Deployment-Plans	SI
A2.6	**Entwicklung des Applikationsinkrements**	AE
	Aktivitäten analog zum Entwicklungsinkrement in OpenUP bezogen auf die Zielapplikation	
A2.7	**Applikationstest**	SI
	Aktivitäten analog zum Test in OpenUP bezogen auf die Zielapplikation	
A2.8	**Fortlaufende Aktivitäten**	ALLE
	Aufnahme von Änderungsanfragen (z.B. bzgl. Anforderungen, Schnittstellen, Implementierungen, etc.)	

Werkzeuge werden im Abschnitt 5.2 entwickelt. Die in den Tabellen 5.1 und 5.2 definierten Prozesse können mit Hilfe von Workflow-Managementsystemen so unterstützt werden, dass die Arbeitsschritte bei örtlicher Verteilung der verschiedenen Entwicklerrollen effizient durchgeführt werden können.

5.2 Entwicklung von Deployment-Werkzeugen für komplexe Serviceroboter

Die folgenden Abschnitte befassen sich entsprechend der Analyse des Stands der Technik mit der Erstellung eines Deployment-Plans und dessen Umsetzung. Vor der Konzeption der Modelle für Servicerobotik-Applikationen und die Servicerobotik-Zieldomäne wird eine Gegenüberstellung verschiedener Modellierungsmethoden durchgeführt. Danach werden Algorithmen zur Erstellung und Umsetzung des Deployment-Plans hergeleitet. Schließlich werden die Werkzeuge für die einzelnen Deployment-Aktivitäten konzipiert. Zunächst folgt eine Diskussion bezüglich der Verwendung von Ontologien.

5.2.1 Diskussion: Modellierung von Applikation und Zieldomäne mit Hilfe von Ontologien

Wie in Abschnitt 4.3.2 dargestellt, können Ontologien zur Modellierung von Ressourcenmodellen für Applikation und Zieldomäne verwendet werden. Die Abstraktionsebene der Ontologie und damit die Spezifikationstiefe der Modelle hat Auswirkungen auf den Modellierungsaufwand, den erzielbaren Automatisierungsgrad der Deploymentplanung, die mögliche Ausprägung der Rollentrennung und die Anwendungsbreite der Ontologie für beliebige Servicerobotik Zielsysteme. Tabelle 5.3 stellt diese Auswirkungen jeweils für die Verwendung einer Ontologie auf einer hohen und niedrigen Abstraktionsebene gegenüber.

Eine höhere Abstraktionsebene bedeutet grundsätzlich einen geringeren Modellierungsaufwand für den Systemintegrator zur Modellierung der Ressourcen der Zieldomäne bzw. für den Applikationsentwickler zur Modellierung der Applikationsanforderungen. Die abstraktere Ontologie kann weiterhin für die Spezifikation einer größeren Bandbreite von unterschiedlichen Servicerobotik-Plattformen verwendet werden. Je detaillierter eine Ontologie, desto höher in der Regel die Einschränkung auf eine bestimmte Hardware-Klasse (z.B. MarteUML $\rightarrow$ embedded Systeme).

Durch die Verwendung der abstrakten Modellierung muss der Systemintegrator das notwendige spezifische Domänenwissen aufbringen, um z.B. entscheiden zu können, ob die

Tabelle 5.3: Gegenüberstellung der Vor- und Nachteile von Ontologien hinsichtlich deren Abstraktionsebene

Vergleichskriterien	hohe Abstraktionsebene	tiefe Abstraktionsebene
Modellierungsaufwand	↓	↑
Mögl iche Ausprägung der Rollentrennung	↓	↑
Mögliche Automatisierungsgrad	↓	↑
Anwendungsbreite	↑	↓

Legende:

↑ eher hoch
↓ eher gering

Hardwarearchitektur im Detail für die Anforderungen der Applikation geeignet ist (z.B. hinsichtlich Positionierung und Sichtfelder der Sensorik, Kinematik und Arbeitsraum der Manipulatoren, etc.). Je tiefer die Abstraktionsebene der Ontologie, desto mehr Entscheidungen kann ein Werkzeug zur Deploymentplanung treffen (z.B. ob der Arbeitsraum eines mobilen Manipulators für eine bestimmte Anwendung ausreichend ist, oder eine andere Hardware gewählt werden muss). Liegen Modelle auf einer tieferen Abstraktionsebene vor, enthalten sie mehr gekapseltes Domänen- bzw. Applikationswissen, so dass das werkzeuggestützte Deployment weniger Hintergrundwissen erfordert. Dadurch kann eine stärkere Ausprägung der Rollentrennung erreicht werden.

Soll das Deployment-Werkzeug jedoch nur eine Zuordnung der einzelnen Software-Komponenten zu bestimmten Rechnern des Zielsystems entsprechend ihrer Ressourcenanforderungen leisten (z.B. da die Hardware bereits durch das Projektteam festgelegt wurde), kann eine abstrakte Ontologie bereits ausreichend sein.

5.2.2 Entwicklung eines Algorithmus für die Deployment-Planung von Servicerobotik-Applikationen

Die Modellierung der Anforderungen der Applikation und der Ressourcen der Zieldomäne stellen die Basis für den zu entwickelnden Algorithmus dar, der für das Deployment einer bestimmten Komponente die jeweils in Frage kommenden Rechner des Zielsystems ermitteln soll. Analog zu Caspian [Hey06] werden Applikations- und Domänenmodell für die Servicerobotik durch Graphen auf einer eher hohen Abstraktionsebene erstellt, so dass die Modelle für eine große Bandbreite von verschiedenen Servicerobotern verwendet werden können (vgl. Diskussion in Abschnitt 5.2.1).

Abbildung 5.4 veranschaulicht das konzipierte generische Ressourcenmodell. Die schwarzen Blöcke bilden den abstrakten Kern des Modells, durch das sowohl die Zieldomäne

als auch die Applikation modelliert werden können. Komponenten werden als Knoten des Graphen modelliert, die durch Kommunikationskanäle als Kanten des Graphen verbunden werden. Knoten und Kanten können dabei beliebig viele Ressourcen besitzen, die durch einen eindeutigen Bezeichner und einen Typ spezifiziert werden. Basis für den generischen Ressourcentyp sind die im OMG Deployment-Standard spezifizierten Typen. Knoten werden ebenfalls noch durch einen Bezeichner und einen Typen spezifiziert. Knoten, Kanten und Ressourcen können dabei durch spezifischere Modelle erweitert werden, die die Domäne entsprechend auf einer tieferen Abstraktionsebene beschreiben können. In Abbildung 5.4 wird als Beispiel ein Manipulationsknoten abgeleitet, der eine speziellere Beschreibung der Kinematik erlaubt, die mit den generischen Ressourcentypen nicht spezifiziert werden kann.

Aufgrund eines fehlenden Standards für die Spezifikation der Hardware-Komponenten für die Servicerobotik-Domäne (vgl. Abschnitt 4.3.2) werden die Algorithmen im Folgenden primär für das generischen Ressourcenmodell entwickelt.

Durch den hierarchischen Aufbau des Modells können die Werkzeuge jedoch grundsätzlich auch für Modelle auf einer tieferen Abstraktionsebene verwendet werden. Dazu sind entsprechende Erweiterungen der Werkzeuge notwendig, die jedoch keine konzeptionellen Änderungen erfordern. Im Folgenden wird das Applikations- und das Domänenmodell formal definiert.

Definition des Applikationsmodells

Eine Applikation A sei definiert durch

$$A = (V^A, E^A) \tag{5.1}$$

$$V^A = C_{\text{sw}} \cup C_{\text{hw}} \tag{5.2}$$

$$E^A = \{k_1, \ldots, k_{\text{n}}\} \subseteq (V^A \times V^A). \tag{5.3}$$

Dabei ist V^A die Menge der verschiedenen (Soft- und Hardware-)Komponenten und E^A die Menge der Kommunikationskanäle, die die einzelnen Komponenten miteinander verbinden. Die Menge C_{sw} umfasst alle Software-Komponenten, die auf den verteilten Rechnern des Zielsystems installiert werden müssen und C_{hw} ist die Menge aller Hardware-Komponenten, die auf dem Zielsystem entweder schon verfügbar oder mechatronisch integriert werden müssen. Der Kommunikationskanal zwischen zwei Komponenten $c_i, c_j \in V^A$ wird durch $k_{c_i,c_j} = (c_i, c_j)$ bezeichnet.

Definition der Anforderungen der Applikation

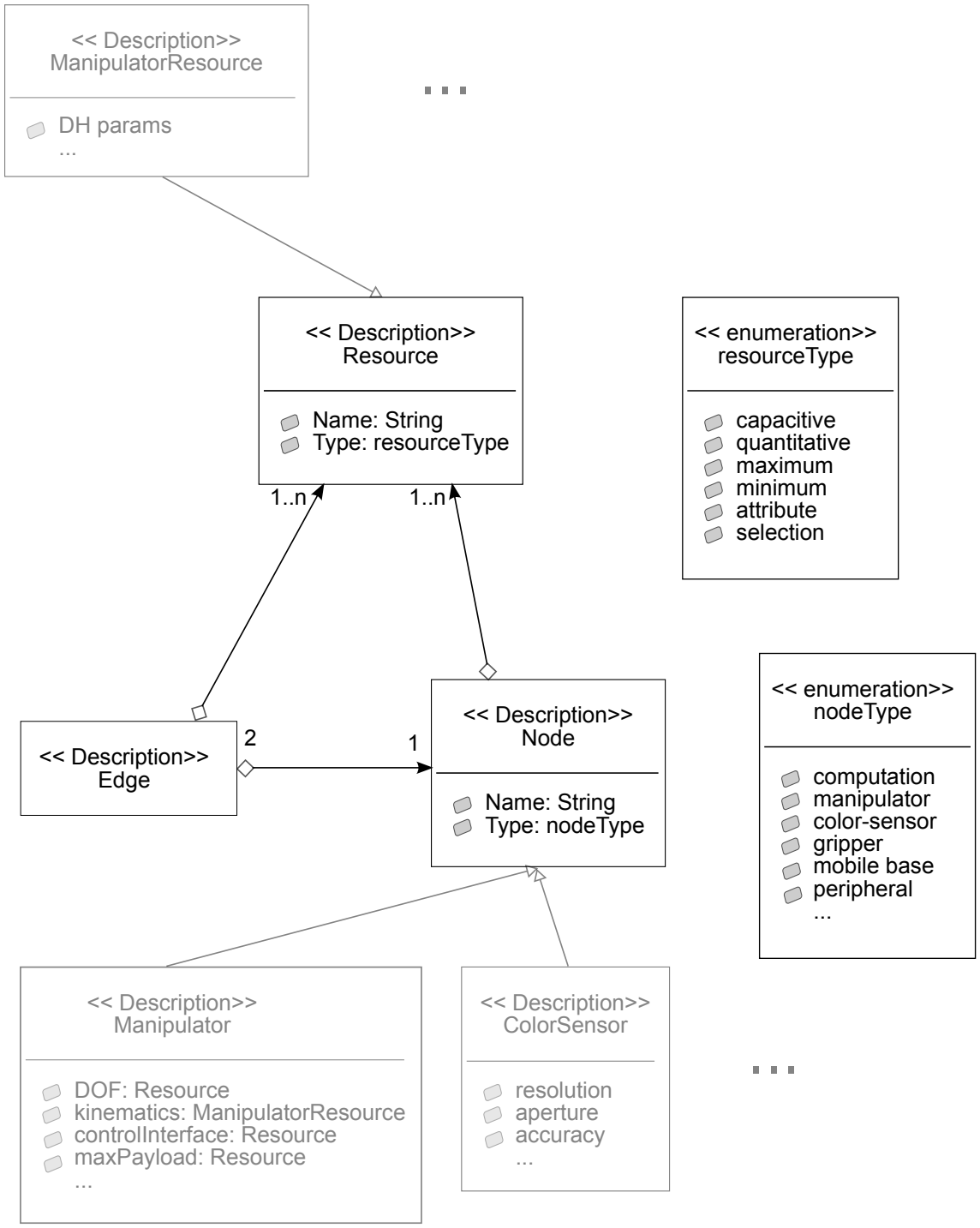


Abbildung 5.4: Generisches Ressourcenmodell für Servicerobotik-Hardware-Komponenten. Die schwarzen Blöcke umfassen den abstrakten Kern des Modells; die grauen Blöcke stellen beispielhaft mögliche Erweiterungen des Modells dar.

Die Anforderungen $R(x)$ eines Elements $x \in V^A \cup E^A$ setzen sich konform zu OMG D&C zusammen aus jeweils einer Menge von quantitativen Anforderungen $R_{\text{quant}}(x)$ (z.B. Anzahl der CPU Kerne), Anforderungen betreffend der Kapazität $R_{\text{cap}}(x)$ (z.B. Arbeitsspeicher), Minimalwertanforderungen $R_{\text{min}}(x)$ (z.B. CPU-Takt), Maximalwertanforderungen $R_{\text{max}}(x)$ (z.B. Latenzzeit), Attributen $R_{\text{att}}(x)$ (z.B. Betriebssystem) und Menge an alternativen Attributen $R_{\text{sel}}(x)$. Hier können mehrere Optionen angegeben werden, die die Anforderung erfüllen.

Die Anforderungen der gesamten Applikation ergeben sich dann zu

$$R(A) = \bigcup_{x \in V^A \cup E^A} R(x). \tag{5.4}$$

Definition des Modells für die verteilte Zielumgebung (Domäne)

Die Zieldomäne wird beschrieben durch die Gleichungen

$$D = (V^D, E^D) \tag{5.5}$$

$$V^D = C_{\text{hw}} \cup H \tag{5.6}$$

$$E^D \subseteq (V^D \times V^D) \tag{5.7}$$

Dabei ist V^D die Menge aller Komponenten des Zielsystems, die sowohl die Menge der für das Deployment verfügbaren Rechner bzw. Hosts $H = \{h_1, \ldots, h_{\text{p}}\}$ als auch die Menge der verfügbaren Hardware-Komponenten C_{hw} (d.h. Sensoren, Aktoren) umfassen. Die Menge E^D umfasst die verfügbaren Kommunikationskanäle zwischen den Komponenten auf dem Zielsystem. Diese schließen neben Kommunikationsverbindungen zwischen den Hosts auch die Kommunikationsverbindungen zwischen Hosts und Hardware-Komponenten mit ein (z.B. CAN-Bus zur Motorsteuerung).

Definition der Eigenschaften der Zieldomäne

Die Eigenschaften $P(x)$ eines Elements $x \in V_D, E_D$ bestehen analog zu den Anforderungen in Abschnitt 5.2.2 aus jeweils einer Menge von quantitativen Eigenschaften $P_{\text{quant}}(x)$ (z.B. Anzahl der CPU Kerne), Eigenschaften betreffend der Kapazität $P_{\text{cap}}(x)$ (z.B. RAM), Minimalwerteigenschaften $P_{\text{min}}(x)$ (z.B. Latenzzeit), Maximalwerteigenschaften $P_{\text{max}}(x)$ (z.B. CPU-Takt), Attribute $P_{\text{att}}(x)$ (z.B. Betriebssystem), und einer Menge $P_{\text{sel}}(x)$ an gleichzeitig verfügbaren Attributen (z.B. Kanaltypen). Die Eigenschaften der Zieldomäne ergeben sich

dann zu

$$P(D) = \bigcup_{x \in V^D \cup E^D} P(x). \tag{5.8}$$

Aufstellung und Validierung eines Deployment-Plans

Das Planungsproblem ist charakterisiert durch die Abbildung des Applikationsgraphen auf den Domänengraphen $\iota : A \mapsto D$: alle Knoten der Menge V^A müssen in solcher Weise Knoten der Menge V^D zugeordnet werden, dass die Anforderungen der Knoten und Kanten von A durch die Eigenschaften der Knoten und Kanten von D erfüllt sind.

Die Funktion $M : X^A \times X^B \mapsto \{\text{wahr}, \text{falsch}\}, X \in \{V, E\}$ gebe zurück, ob die Anforderungen $R(x^A)$ eines Elements $x^A \in \{V^A, E^A\}$ durch die Eigenschaften $P(x^D)$ eines Elements $x^D \in \{V^D, E^D\}$ erfüllt werden, d.h.

$$M(x^A, x^D) = \begin{cases} \text{wahr}, & \text{falls } \forall r \in R,\ \forall p \in P :\ \mu(r(x^A), p(x^D)) = \text{wahr}, \\ \text{falsch}, & \text{andernfalls} \end{cases} \tag{5.9}$$

wobei die Funktion $\mu : R \times P \mapsto \{\text{wahr}, \text{falsch}\}$ bzw. definiert ist zu:

$$\mu(r(x^A), p(x^D)) = \begin{cases} v(p) \geq v(r) \wedge l(r) = l(p) & \text{falls } r \in R_{\text{quant}}(x^A) \wedge p \in P_{\text{quant}}(x^D) \\ v(p) \geq v(r) \wedge l(r) = l(p) & \text{falls } r \in R_{\text{cap}}(x^A) \wedge p \in P_{\text{cap}}(x^D) \\ v(p) \geq v(r) \wedge l(r) = l(p) & \text{falls } r \in R_{\text{min}}(x^A) \wedge p \in P_{\text{min}}(x^D) \\ v(p) \leq v(r) \wedge l(r) = l(p) & \text{falls } r \in R_{\text{max}}(x^A) \wedge p \in P_{\text{max}}(x^D) \\ l(r) = l(p) & \text{falls } r \in R_{\text{attr}}(x^A) \wedge p \in P_{\text{attr}}(x^D) \\ \exists p_i \in v(p) \wedge & \\ \exists r_j \in v(r) : l(p_i) = l(r_j) & \text{falls } r \in R_{\text{sel}}(x^A) \wedge p \in P_{\text{sel}}(x^D) \end{cases} \tag{5.10}$$

und $l(y)$ das Label und $v(y)$ den Wert des Eintrags y beschreibt.

Sei zum Beispiel $k^A = (h_1, h_2)$ ein Kommunikationskanal zwischen zwei Hosts h_1 und h_2 der Zieldomäne mit den Eigenschaften $P_{\text{att}}(k^A) = \{tp\}$ mit $v(tp) = \{\text{“wireless LAN”}\}$ und $k^D = (c^A_{\text{sw}_1}, c^A_{\text{sw}_2})$ ein Kanal zwischen zwei Software-Komponenten $c^A_{\text{sw}_1}$ und $c^A_{\text{sw}_2}$ der Applikation A mit der Anforderung $R_{\text{att}}(k^D) = \{tp\}$, mit $v(tp) = \{\text{“ethernet”}\}$, dann kann durch die Matching-Funktion μ überprüft werden, ob die Anforderungen des Kanals k^A durch Kanal k^D erfüllt sind: $\mu(p_{\text{sel}}(k^A), r_{\text{sel}}(k^D)) = \text{falsch}$.

Sei E_c die Menge aller mit einer Komponente c verbundenen Kanäle

$$E_c = \{(c_x, c_y) | (c_x, c_y) \in E, c_x = c \vee c_y = c\} \tag{5.11}$$

Eine Komponente des Applikationsgraphen $c^A \in V^A$ kann einer Komponente des Domänengraphen $c^D \in V^D$ dann und nur dann zum Deployment zugewiesen werden, falls gilt:

1. alle Anforderungen $R(c^A)$ werden durch Eigenschaften des Host $P(c^D)$ erfüllt.
2. für alle mit c^A verbundenen Kommunikationskanäle existieren entsprechende Kanäle der Komponente c^D, die deren Anforderungen erfüllen;

Dazu sei die Funktion $\delta : V^A \times V^D \mapsto \{\text{wahr}, \text{falsch}\}$ definiert zu:

$$\begin{aligned} \delta(c^A, c^D) = & M(c^A, c^D) \wedge \\ & \forall (c^A_m, c^A_n) \in E_{c^A} : \\ & \exists (c^D_k, c^D_l) \in E(c^D) : \ M((c^A_m, c^A_n), (c^D_k, c^D_l)) \end{aligned} \tag{5.12}$$

Die Menge $Z_D(c^A)$ aller möglichen Kandidaten für das Deployment einer Komponente c^A ist gegeben durch

$$Z_D(c^A) = \{c^D | \ c^D \in V^D \wedge \ \delta(c^A, c^D)\}. \tag{5.13}$$

Ein konkretes Deployment einer Applikation lässt sich als Relation $i^{A \mapsto D} \subseteq V^A \times V^D$ definieren:

$$i^{A \mapsto D} = \{(c^A, c^D) | \ c^D \in Z(c^A)\}. \tag{5.14}$$

Ein Deployment $i^{A \mapsto D}$ ist vollständig, wenn gilt: die Relation i ist linkstotal, d.h. die Projektion $\rho_{V^A}(i)$ der Relation i auf V^A ergibt V_A. Dies ist gleichbedeutend mit der Bedingung, dass jeder Komponente $c^A \in V^A$ eine Komponente $c^D \in V^D$ zugeordnet werden kann:

$$\begin{aligned} i^{A \mapsto D} \text{ ist vollständig} & \Leftrightarrow \forall c^A \in V^A : Z(c^A) \neq \emptyset \\ & \Leftrightarrow \rho_{V^A}(i) = V^A \end{aligned} \tag{5.15}$$

Die Relation i ist jedoch nicht linkseindeutig, d.h. mehrere $c_{\text{sw}}^A \in V^A$ können demselben Host $h \in V^D$ zugeordnet werden. Allerdings kann jede Hardware-Komponente c_{hw}^D der Zieldomäne innerhalb einem Deployment i nur genau eine Komponente c_{hw}^A aus dem Applikationsgraphen als Zuordnung haben.

Die Menge $I^{A \mapsto D}$ aller möglichen vollständigen Deployments ergibt sich über alle möglichen Zuordnungenvariationen über Z. Alle $i^{A \mapsto D} \in I^{A \mapsto D}$ müssen vollständig sein. Die Anzahl aller möglichen vollständigen Deployments lässt sich berechnen durch

$$|I^{A \mapsto D}| = \prod_{c_n \in V^A} |Z_D(c_n)| \tag{5.16}$$

Für ein bestimmtes Deployment i lässt sich die Relation $j^{E^A \mapsto E^D}$ für die entsprechenden Zuordnungen der Kommunikationskanäle von Applikation A auf die Kommunikationskanäle der Zieldomäne D definieren:

$$\begin{aligned} j^{E^A \mapsto E^D}(i^{A \mapsto D}) = \{((c_m^A, c_n^A), (c_k^D, c_l^D)) | (c_m^A, c_k^D) \in i^{A \mapsto D} \wedge (c_n^A, c_l^D) \in i^{A \mapsto D} \vee \\ (c_n^A, c_k^D) \in i^{A \mapsto D} \wedge (c_m^A, c_l^D) \in i^{A \mapsto D}\} \end{aligned} \tag{5.17}$$

Die Funktion $validateCandidateDeployment(i^{A \mapsto D})$ führt die Zuordnungen der paarweisen Tupel sequenziell durch und passt jeweils die verfügbaren Ressourcen der Hosts an. Vor jeder Zuordnung wird wieder geprüft, ob die Anforderungen der Komponenten $R(c_{\text{sw},i})$ durch die Ressourcen der Hosts $P(h_j)$noch gedeckt sind, da innerhalb eines Deployments mehrere Software-Komponenten auf denselben Host gemappt werden können (s.o.). Analog werden nach der erfolgreichen Zuordnung aller Komponenten die Kommunikationskanäle sequenziell zugeordnet und auf die Erfüllung ihrer Anforderungen $R((c_m, c_n))$ durch die Ressourcen $P((c_k, c_l))$ hin überprüft (vgl. Algorithmen 5.2.1 und 5.2.2).

Algorithmus 5.2.1 Aktualisierung der verfügbaren Ressourcen

```
updateResources(R, P):
for all r ∈ R and p ∈ P do
  if l(r) = l(p) then
    if (r ∈ R_quant and p ∈ P_quant) or (r ∈ R_cap and p ∈ P_cap) then
      v(p) ← v(p) − v(r)
    end if
  end if
end for
```

Algorithmus 5.2.2 Überprüfung der Validität eines Deployments

```
validateCandidateDeployment(i^{A↦D}):
for all (c^A_sw, h^D) ∈ i^{A↦D} do
  if M(c^A_sw, h^D) then
    updateResources(R(c^A_{sw,i}), P(h^D))
  else
    return false
  end if
end for
Derive j^{E^A↦E^D}(i^{A↦D}).
for all (k^A, k^D) ∈ j do
  if M(k^A, k^D) then
    updateResources(R(k^A), P(k^D))
  else
    return false
  end if
end for
```

Entwicklung eines Algorithmus für die Deployment-Planung

Mit Hilfe der Definitionen aus Abschnitt 5.2.2 und den Algorithmen 5.2.1 und 5.2.2 wird Algorithmus 5.2.3 zur Ermittlung aller gültigen Deployments $\tilde{I}^{A\mapsto D} \subseteq I^{A\mapsto D}$ aufgestellt.

Bei Bedarf können in diesem Algorithmus mit der Funktion $userContraints(i^{A\mapsto D})$ noch bestimmte Randbedingungen des Deployment-Planers berücksichtigt werden (z.B. Gütekriterien wie gleichmäßige Auslastung, maximale Verfügbarkeit, Kostenminimierung, etc.). Der Algorithmus kann so sehr einfach für die Verwendung mit Modellierungen und Ontologien angepasst werden, die spezielle Robotersysteme fokussieren und detaillierter beschreiben. Für die im Folgenden verwendete einfache Modellierung kann der Algorithmus bereits ohne weitere Anpassung verwendet werden.

5.2.3 Entwicklung von Werkzeugen zur Unterstützung der Deployment-Aktivitäten

Um die Aktivitäten des Deployment-Prozess für die beteiligten Akteure (s. Abschnitt 5.1) zu vereinfachen, werden im Folgenden Werkzeuge zur Unterstützung konzipiert. Der Fokus liegt dabei auf den Aktivitäten Deployment-Planung, Installation/Aktualisierung, Aktivierung/Deaktivierung sowie Konfiguration, da für die übrigen Aktivitäten schon entsprechende Werkzeuge existieren. Eine beispielhafte Realisierung der Werkzeuge einer konkreten Modellierungssprache zur Spezifikation der Modelle wird in Abschnitt 6.2.1 präsentiert.

Algorithmus 5.2.3 Ermittlung aller gültigen Deployments

```
planDeployment(A, D):
Ĩ^{A↦D} := ∅
for all c_i^A ∈ V_A do
  find set of candidate hosts Z(c_i^A)
  if Z(c_i^A) == NULL then
    return ∅
  end if
end for
derive set of all candidate deployments I^{A↦D} through variation of all possible relations
of c_i^A and Z(c_i^A)
for all i^{A↦D} ∈ I^{A↦D} do
  if validateCandidateDeployment(i^{A↦D}) == true then
    if userContraints(i^{A↦D}) == true then
      add i^{A↦D} to Ĩ^{A↦D}
    end if
  end if
end for
return Ĩ^{A↦D}
```

Abbildung 5.5 veranschaulicht den Zusammenhang der Deployment-Werkzeuge mit dem entwickelten Vorgehensmodell.

Unterstützung von Deployment-Planung und Installation

Die aufwändigsten Tätigkeiten für die Deployment-Planung bestehen in der Generierung und Unterhaltung des Applikations- und Domänenmodells. Dazu wird ein graphisches Modellierungswerkzeug konzipiert, das einen einfachen Aufbau der Modelle durch den Benutzer ermöglicht und die Modelle dann in eine formale Spezifikation mit allen erforderlichen Informationen für den Deployment-Planer (vgl. 5.2.2) übersetzen kann. Weiterhin wird ein Werkzeug für den Deployment-Planer implementiert, das die Algorithmen aus Abschnitt 5.2.2 umsetzt und die Zuordnung Applikationsgraph nach Domänengraph generiert. Schließlich sorgt am Ende das Installationswerkzeug dafür, dass diese Zuordnung auf dem Zielsystem umgesetzt wird. Dazu greift dieses Werkzeug direkt auf das Komponenten- bzw. Applikations-Repository zu und installiert die Komponenten auf das Zielsystem unter Verwendung eines Paketmanagers und Remote Session Werkzeugen entsprechend des Deployment-Plans . Zur werkzeuggestützten Installation ist in der Regel ein plattformabhängiger Installationscode erforderlich, den der Systemintegrator z.B. in Form von Skripten dem Installationswerkzeug zur Verfügung stellen muss. Das Installationswerkzeug selbst ist jedoch plattformunabhängig aufgebaut.

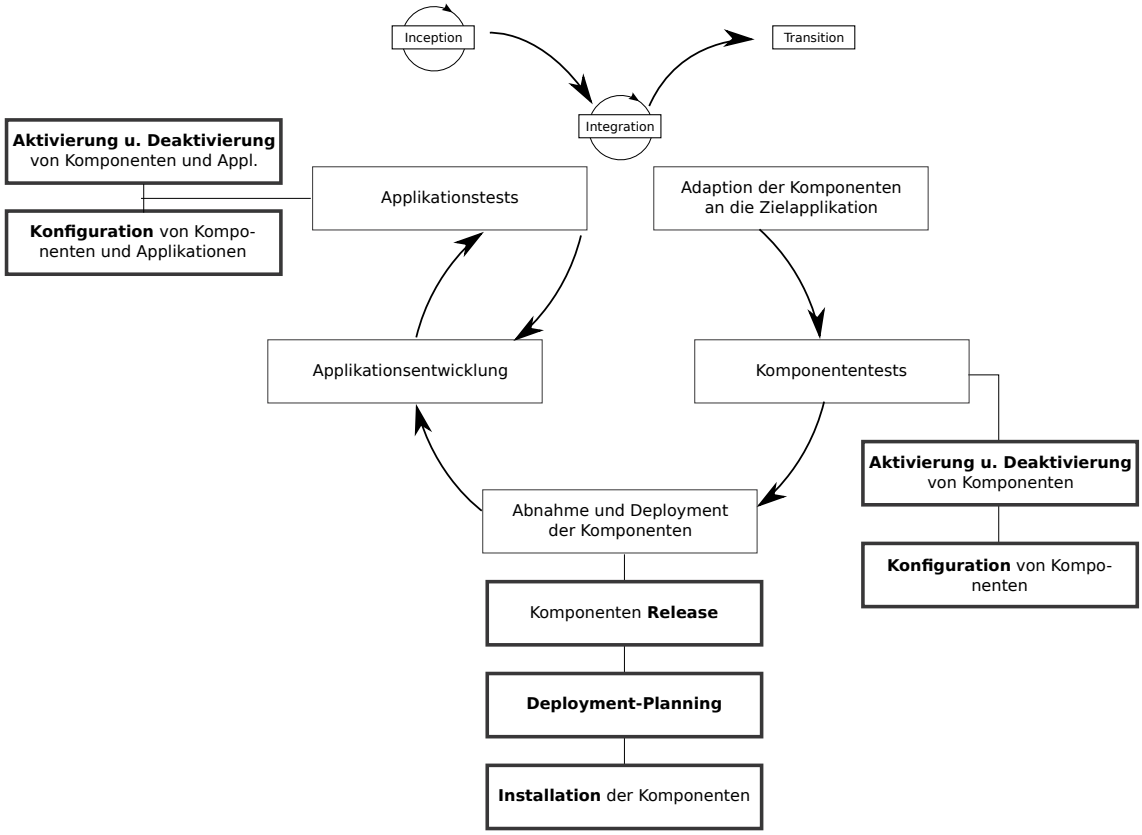


Abbildung 5.5: Aktivitäten des Deployment-Prozesses für die verteilte Entwicklung und Integration im Zusammenspiel mit den Aktivitäten in der Integrationsphase des DDT-Vorgehensmodells

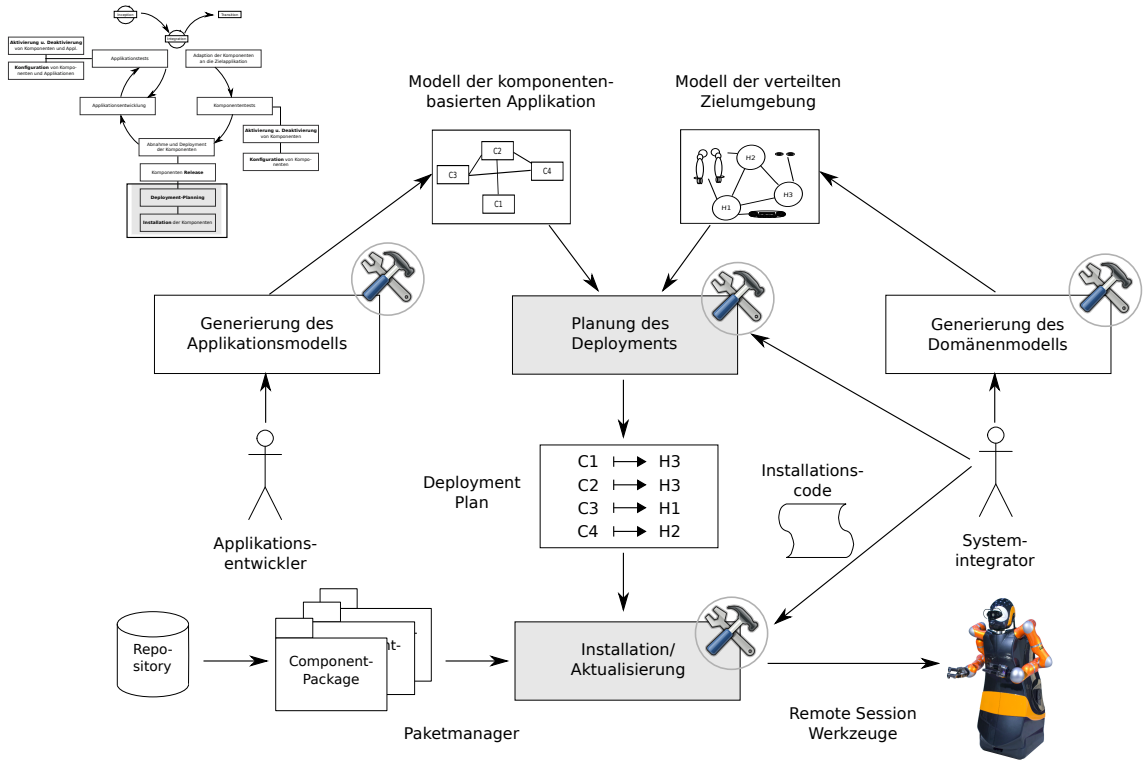


Abbildung 5.6: Werkzeuge für die Aktivitäten Deployment-Planung und Installation

Aktualisierung

Mit Hilfe des Paketmanagers kann analog zur Installation auch die Aktualisierung der Komponentenversion durchgeführt werden. Die Auswahl bestimmter Releases ist in der Regel im Funktionsumfang der Paketmanager enthalten, so dass Anforderung A35 implizit erfüllt werden kann.

Abbildung 5.6 veranschaulicht die konzipierte Werkzeugkette für die Deployment-Planung, Installation und Aktualisierung von Komponenten bzw. Applikationen auf dem Zielsystem.

Aktualisierung kann sich sowohl auf die Konfiguration als auch die Releaseversion von Komponenten beziehen, so dass sich diese Aktivität aus den beiden Aktivitäten “Installation” und “Konfiguration” zusammensetzen lässt und somit keines eigenen Werkzeuges bedarf.

Es wird jedoch davon ausgegangen, dass durch die Aktualisierung keine weiteren Komponenten hinzukommen bzw. Komponenten wegfallen. Im Falle von Änderungen an der Auswahl der Komponenten im Laufe der Applikationsentwicklung müssen die Aktivitäten Deployment-Planung und Installation erneut durchlaufen werden.

5.3 Laufzeitwerkzeuge zur besseren Bedienbarkeit für verteilte Integration und Test

Im folgenden Kapitel wird zunächst das Werkzeug zur Laufzeitunterstützung konzipiert, das Komponenten- und Applikationsentwicklern ohne detaillierte Kenntnisse der Hardware- und Software-Architektur die einfache Konfiguration und Aktivierung von Komponenten und Applikationen ermöglichen soll. Daraufhin folgt eine Diskussion der Zugriffsvarianten der Entwickler auf die Zielplattform, insbesondere ob der Zugriff (über das Laufzeitwerkzeug) zentral mittels eines Servers oder verteilt über die Rechnern der einzelnen Entwickler direkt erfolgen soll. Abschließend werden auf Basis der in Abschnitt 4.4 vorgestellten Laufzeitumgebungen verschiedene Abstraktionsebenen für den Zugriff auf die Komponenten des Zielsystems erörtert.

Aktivierung/Deaktivierung und Konfiguration von Komponenten und Applikationen

Für die Laufzeitwerkzeuge wird eine graphische Oberfläche konzipiert, die die verfügbaren Komponenten und Applikationen auf dem System sowie deren aktuellen Laufzeitstatus (offline, online, running) visualisieren. Dazu benötigt dieses Werkzeug für die Anzahl und Lokation der Komponenten zum einen die Deployment-Zuordnung, zum anderen analog zum Installationscode in Abschnitt 5.2.3 den Aktivierungs- bzw. Deaktivierungscode für die einzelnen Komponenten und Applikationen. Diese Skripte müssen jeweils durch die entsprechenden Komponenten- und Applikationsentwickler dem Werkzeug zur Verfügung gestellt werden, da das Kriterium der Plattformunabhängigkeit (vgl. A36) durch Annahmen oder Bedingungen bezüglich der Aktivierung und Deaktivierung verletzt würde. Weiterhin sind im Allgemeinen Monitoring-Skripte nötig, um den Zustand der Komponenten abzufragen. Sind diese Skripte einmalig dem Werkzeug bekannt gemacht worden, kann das Werkzeug jedem Entwickler durch ein entsprechendes Interface die Aktivierung und Deaktivierung der Komponenten ermöglichen, ohne dass diesem die Skripte bekannt sein müssen. Die Skripte können grundsätzlich beliebig komplex werden und z.B. eine Überprüfung von Vorbedingungen zur Aktivierung und Deaktivierung von Komponenten enthalten (z.B. Deaktivierung der Kollisionsüberwachung nur, wenn die Aktoren nicht in Betrieb sind). Auch hier bringt die Flexibilität der Skripte einen höheren Aufwand für Komponentenentwickler mit sich (vgl. Diskussion Abschnitt 5.2.1). Eine Spezialisierung kann jedoch jederzeit durchgeführt werden (z.B. automatisierte Generierung von Launch-Files bei der Verwendung mit ROS für die Aktivierungsskripte).

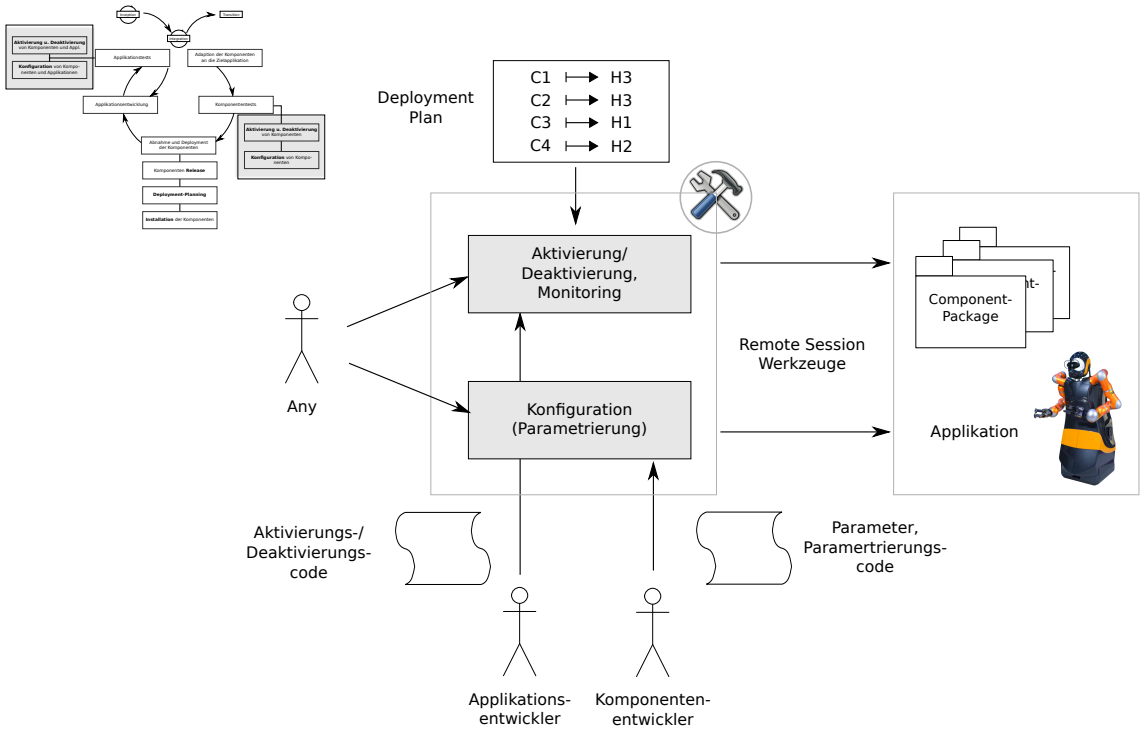


Abbildung 5.7: Werkzeugkette für die Aktivierung/Deaktivierung und Konfiguration

Für die Konfiguration bzw. Parametrierung der Komponenten und Applikationen werden ebenfalls entsprechende Skripte benötigt. Zusätzlich müssen jedoch die öffentlich zugänglichen Parameter dem Werkzeug explizit bekannt gemacht werden.

Abbildung 5.7 veranschaulicht die Werkzeugkette der Laufzeitaktivitäten.

5.3.1 Diskussion: Zentraler vs. dezentraler Zugriff

Der Zugriff auf die verteilten Rechner der Zielplattform kann grundsätzlich zentral und indirekt, z.B. über einen Server, erfolgen oder dezentral und direkt. Im dezentralen Fall kann jeder Entwickler jederzeit eigenständig eine Verbindung zum Zielsystem aufbauen (unter der Voraussetzung, dass das Zielsystem sich im selben Subnetz befindet). Dadurch kann z.B. ein Komponentenentwickler seine Komponente ohne großen Vorbereitungsaufwand sehr schnell auf bzw. im Zusammenspiel mit dem Zielsystem testen. Durch die direkte Verbindung kann die Komponente wahlweise auf dem Host des Entwicklers ausgeführt oder auf das Zielsystem aufgespielt werden, sofern der Entwickler ein Nutzerkonto auf den Rechnern des Zielsystems besitzt. Er kann so alle auf dem System installierten Software-Komponenten aktivieren und deaktivieren. Auf der anderen Seite kann jedoch - selbst bei lokalen Teams - schwieriger kontrolliert werden, wer gerade auf das Zielsystem zugreift und Tests durchführt.

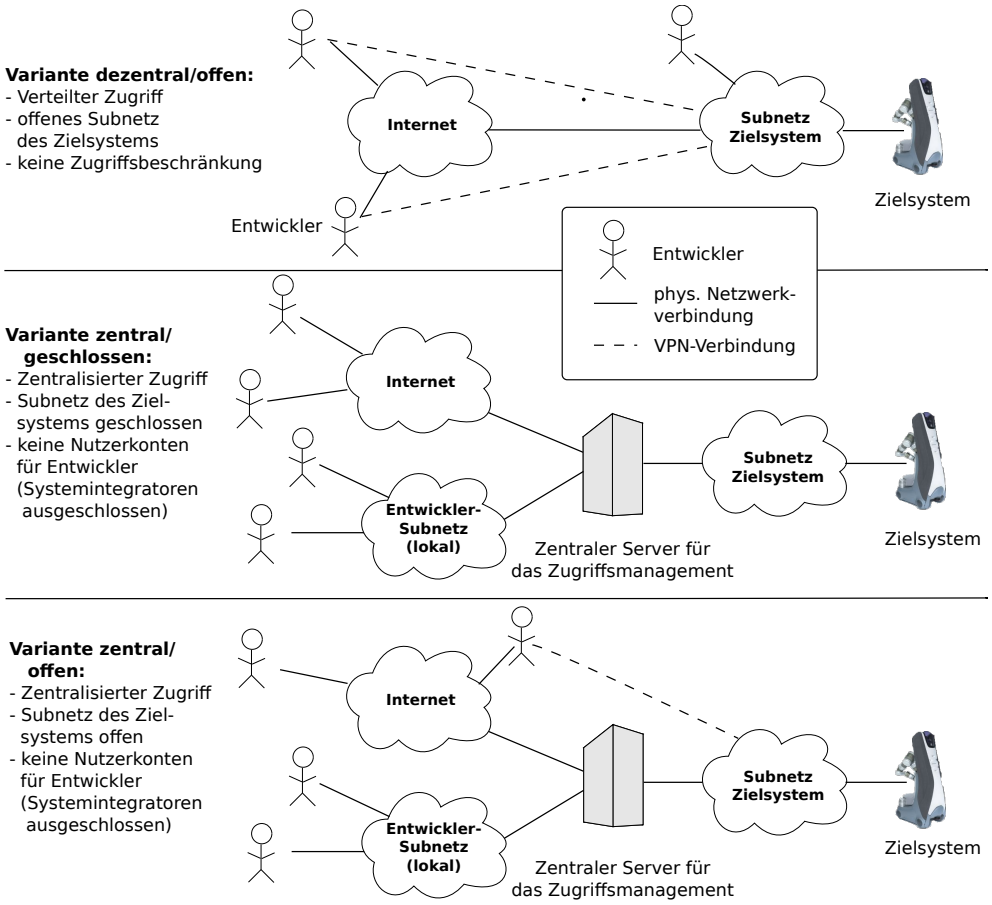


Abbildung 5.8: Darstellung der verschiedenen Zugriffsvarianten mit beispielhaften Nutzungsszenarien und entsprechenden Netzwerktopologien.

Ein Zugriffsmanagement erfordert bei dezentralem Zugriff in der Regel Änderungen am Zielsystem.

Bei einem zentralisierten Zugriff kann das Zugriffsmanagement sehr einfach durch ein zentrales Reservierungssystem (z.B. auf einem entsprechenden Server) durchgeführt werden. Die Entwickler verbinden sich dann nicht mehr direkt auf das Zielsystem, sondern indirekt über den zentralen Server. Dies hat auch zur Folge, dass die zu testenden Komponenten, auf das System aufgespielt werden müssen. Entsprechend dem Vorgehensmodell (vgl. Tabelle 5.2) wäre dazu das Deployment und unter Umständen die Einbindung des Systemintegrators erforderlich. Dieser Aufwand ist für kleine Tests “zwischendurch” ggf. nicht praktikabel. Als Untervariante des zentralisierten Zugriffs wäre es jedoch denkbar, dass der Entwickler zwar keinen Zugriff in Form eines Nutzerkontos direkt auf den Rechnern erhält, aber über Netzwerk Daten mit den laufenden Komponenten auf dem Zielsystem austauschen kann. Über den zentralen Server erhält der Entwickler dann die Rechte zur Aktivierung/Deaktivierung der installierten Komponenten und Informationen über deren

Tabelle 5.4: Gegenüberstellung des zentralisierten und verteilten Zugriffs auf die Zielplattform. Beim zentralisierten Zugriff wird zudem unterschieden, ob das Subnetz des Zielsystems abgetrennt oder offen zugänglich ist.

Vergleichskriterium		verteilt offenes Subnetz		zentralisiert abgetrenntes Subnetz		zentralisiert offenes Subnetz
Zugriffsmanagement (A34)	–	schwer realisierbar, Eingriffe am Zielsystem notwendig	•	einfach realisierbar	○	realisierbar (Zugriff über zentralen Server ersichtlich), technisch ist jedoch die Kommunikation mit dem Zielsystem ohne Reservierung möglich
Vorbereitungsaufw. für einen Test	•	lauffähige zu testende Komponente auf entferntem Host	○	vollständiger Durchlauf des Deployment-prozesses erforderlich	•	lauffähige zu testende Komponente auf entferntem Host
Betriebsvorauss.	○	Entwicklungs-umgebung muss auf entferntem Host verfügbar sein	•	lauffähiger Browser	•	beide Varianten (Browser, lokale Entw.umg.) sind möglich

aktuellen Ausführungszustand. Für diese Variante ist jedoch eine offene Netzwerktopologie erforderlich, die das Zielsystem nicht vom Entwicklernetz trennt (s. Abbildung 5.8).

In Tabelle 5.4 werden die Vor- und Nachteile der zentralen und dezentralen Zugriffsvarianten zusammengefasst gegenübergestellt. Für die weitere Konzeption der Integrations- und Testplattform wird der Zugriffsmechanismus des zentralen Servers mit offenem Subnetz übernommen, da das Zugriffsmanagement insbesondere für ein verteiltes Team und eine zeitliche und räumliche Entkopplung von Integration und Test wichtig ist.

5.3.2 Abstraktionsgrade der Laufzeitumgebung

Die in Abschnitt 4.4 vorgestellten Umgebungen zur Unterstützung der Laufzeitaktivitäten lassen sich einordnen bezüglich deren Abstraktionsgrad und deren Modifikationsmöglichkeiten in Bezug auf die Konfiguration der Komponenten und deren Konnektion sowie Inspektionsmöglichkeiten.

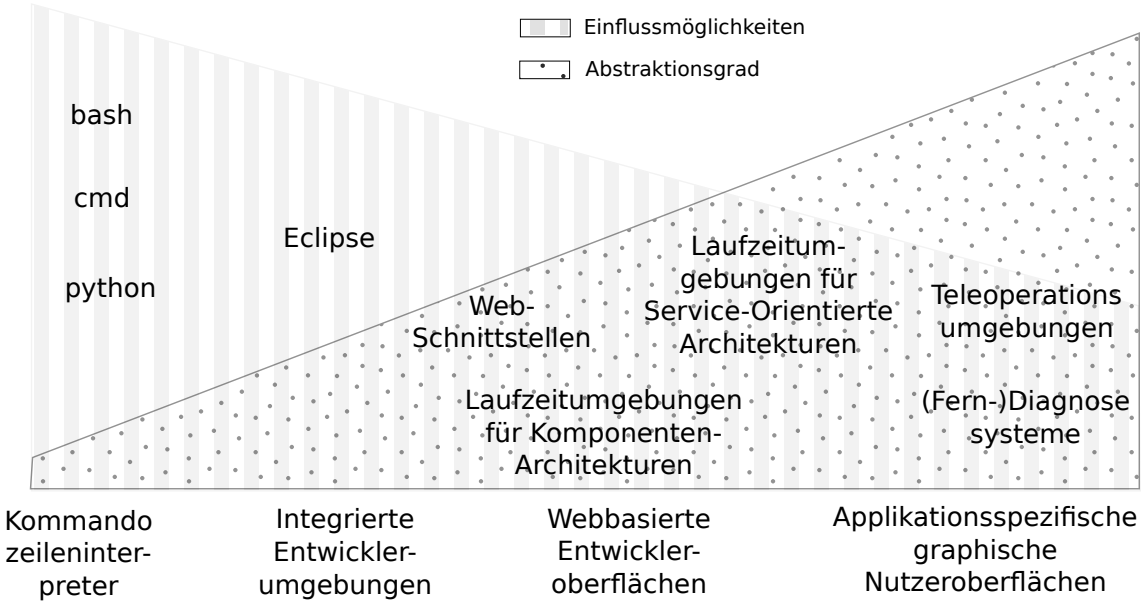


Abbildung 5.9: Vor- und Nachteile verschiedener Laufzeitumgebungen

Je höher der Abstraktionsgrad der Umgebung, desto mehr wird vor dem Entwickler verborgen und desto geringer ist das erforderliche Hintergrundwissen über die einzelnen Komponenten und die Zielplattform für die Durchführung der Laufzeitaktivitäten. Ein höherer Abstraktionsgrad geht in der Regel mit einem Verlust an Eingriffs- und Einsichtsmöglichkeiten einher (vgl. Abbildung 5.9). Kommandozeileninterpreter sind z.B. näher auf Betriebssystemebene und bieten deshalb Zugang zu einem umfangreichen Repertoire an Low-Level Werkzeugen, z.B. für die Inspektion von Prozessen. Auf der anderen Seite sind genaue Informationen bezüglich der Installation der Komponenten auf dem verteilten System erforderlich (d.h. Lokation der Ausführungs- und Parameterdateien, etc.).

Je nach Entwicklerrolle ist in der Regel ein unterschiedlicher Abstraktionsgrad erforderlich bzw. gewünscht; Komponentenentwickler benötigen meist eine Ausführungsumgebung, die einen schnellen Zyklus für die Modifikation und Test der Implementierung erlaubt und ziehen deshalb oft Kommandozeileninterpreter vor, ggf. integriert in die Entwicklerumgebung. Applikationsentwickler nehmen in der Regel keine Änderung der Implementierung von Komponenten vor, sondern müssen diese während der Applikationstests lediglich konfigurieren. Endnutzer wiederum schätzen eine möglichst einfache Ausführung einer bestimmten Applikation ohne eine große Konfigurationsvielfalt. Die Auswahl der Ausführungsumgebung muss somit entsprechend der involvierten Nutzerrollen abgewogen werden. Weiterhin bieten sich Umgebungen in einer hohen Abstraktionsebene vor allem für sehr umfangreiche Applikationen mit sehr vielen Komponenten an.

Die bisher konzipierten Deployment- und Laufzeitwerkzeuge werden im Rahmen dieser Arbeit in webbasierte Laufzeitumgebungen eingebettet, die einen mittleren Abstraktions-

grad besitzen und sowohl für Komponentenentwickler als auch Applikationsentwickler für Integrations- und Applikationstests eingesetzt werden können. Da service-basierte Architekturen in der Robotik bis dato nicht verfügbar sind, ist diese Laufzeitumgebung für komponenten-basierte Applikationen konzipiert.

5.4 Zusammenführung der einzelnen Module

Die Realisierung der Laufzeitwerkzeuge mittels eines zentralen Webservers ermöglicht eine elegante Zusammenführung der einzelnen Module zur Integrations- und Testplattform. Die Werkzeuge sind somit zentral verfügbar (A19) sowie einfach zugänglich und konfigurierbar (A39) über einen Webbrowser. Abbildung 5.10 veranschaulicht die Gesamtarchitektur der Integrations- und Testplattform: Die Standorte des Entwicklers, des zentralen Webservers für die Integrations- und Testplattform sowie des Zielsystems sind durch drei große Blöcke dargestellt. Grundsätzlich können sich alle diese Standorte unterscheiden; aus Performance-Gründen sollte sich der Webserver allerdings in demselben Netz mit geringen Latenzzeiten zum Zielsystem befinden, da zwischen den einzelnen Werkzeugen und den Rechnern des Zielsystems Remote-Session Verbindungen (ssh, VNC) aufgebaut werden. Die Verbindung zwischen Webserver und Browser des entfernten Entwicklers kann dann wiederum mit den üblichen Latenzzeiten des Internets ($\gg$ 20 ms) erfolgen.

Für die Implementierung der einzelnen Module der Integrations- und Testplattform kann grundsätzlich ein beliebiger Webserver als Basis verwendet werden. Für die Umsetzung wird in dieser Arbeit die Open-Source Kollaborationsplattform Trac (vgl. Abschnitt 4.5) ausgewählt, da in Trac bereits viele Kooperationswerkzeuge integriert sind (Continuous Integration, Issue-Tracking, Versionsverwaltung, Wiki, etc.) und weitere Module einfach durch die plugin-basierte Struktur integriert werden können.

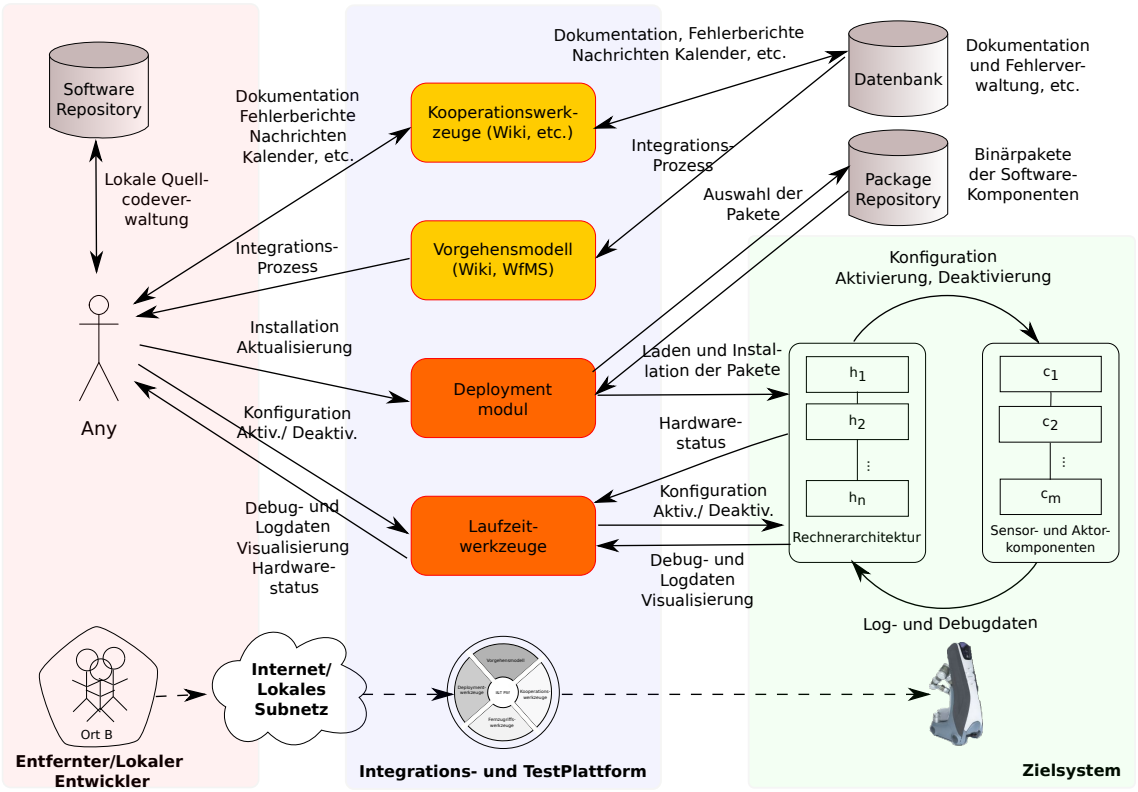


Abbildung 5.10: Darstellung der räumlichen Verteilung der Integrations-und Testplattform, des Zielsystems und eines entfernten Entwicklers sowie der verschiedenen Interaktionsmöglichkeiten mit den einzelnen Werkzeugen. Die Datenbank und das Paketrepository können sich an einem beliebigen (vernetzten) Ort befinden.

6 Realisierung der Integrations- und Testplattform

Die in Kapitel 5 entwickelten Konzepte wurden implementiert und für unterschiedliche Zielsysteme (z.B. DESIRE TP und Care-O-bot 3, Beschreibung s. Abschnitt 3.1) umgesetzt.

Abbildung 6.1 illustriert die Gesamtarchitektur der Integrations- und Testplattform in einem typischen Anwendungsfall: Ein Entwickler an Standort A (linker Block) will auf der Hardware (rechter Block) an Standort B Tests durchführen, wobei Standort A und Standort B über ein beliebiges Netzwerk (z.B. Internet) verbunden sind. Im mittleren Block ist die Integrations- und Testplattform in Form eines zentralen Webportals angesiedelt, das zwar grundsätzlich ortsunabhängig ist, sich jedoch tendenziell ebenfalls am Standort der Hardware befinden sollte, um eine möglichst hohe Bandbreite für die zahlreichen ssh-Verbindungen zwischen Webportal und Rechnern der Zielplattform zu gewährleisten.

Der Webportal-Server wurde auf Basis des Apache-Webservers mit mod-python Modul und der Kollaborationsplattform Trac implementiert. Die entwickelten Werkzeuge sind gemäß Abschnitt 5.4 als Trac-Plugins realisiert, die ihre Daten (z.B. Nutzerdaten, Rechte, etc.) jeweils in einem zentralen SQL-Datenserver ablegen. Für die einzelnen Plugins stehen weitere spezielle Server und Datenbanken zur Verfügung (Komponenten-Package Repository, Software-Repository, Build-Server). Der Webportal-Server kommuniziert via der remote-session Werkzeuge ssh [Ope99] und screen [O. 87] mit den Host Rechner des Zielsystems. Der Entwickler-Client benötigt somit keine direkte Verbindung zum Zielsystem.

Der Browser kommuniziert mit der Integrations- und Testplattform über XML zur Übertragung dynamischer Daten, z.B. der aktuellen Komponentenzustände sowie deren Logdaten und HTTP-Requests zur Übertragung statischer Inhalte wie dem Aufbau der HTML-Seiten. Visualisierungsoberflächen der Komponenten werden vom VNC-Server als Java-Applet bereitgestellt und können so in jedem Java-fähigen Browser angezeigt werden. Abbildung 6.2 zeigt die Browser-Anicht der Integrations- und Testplattform. Alle Plugins sind in Form von Schaltflächen zentral zugänglich.

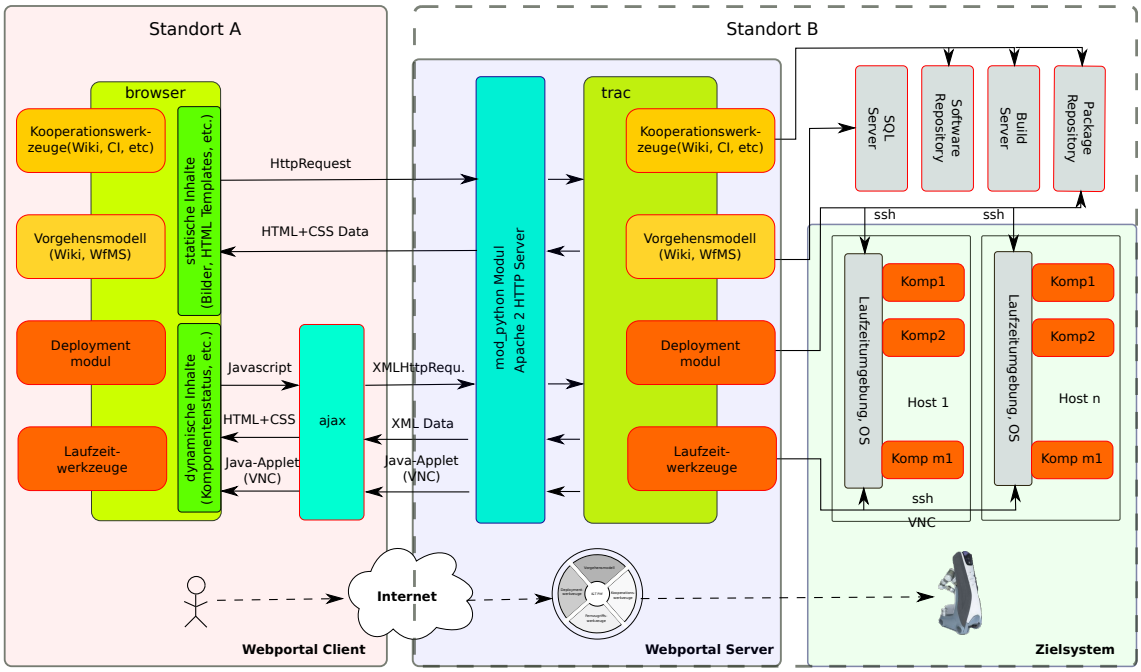


Abbildung 6.1: Gesamtarchitektur der Integrations- und Testplattform und Zusammenspiel zwischen Webportal-Server, Entwickler-Client und Zielsystem Hosts.

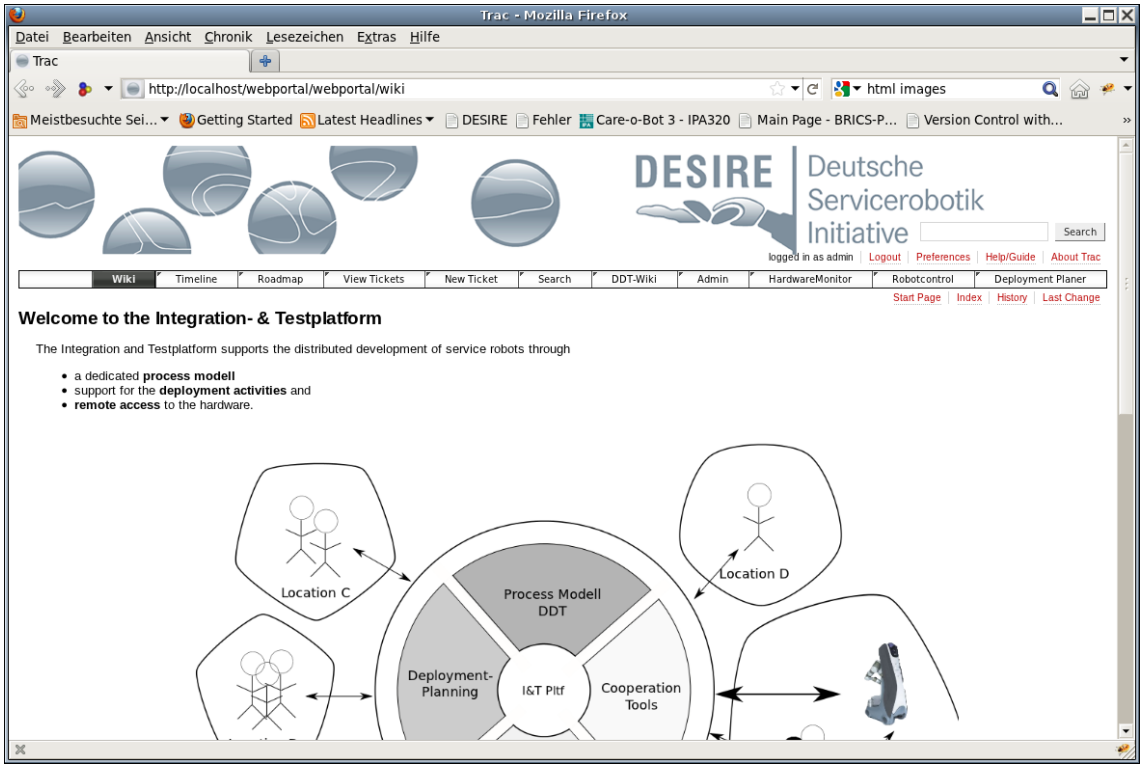


Abbildung 6.2: Browser-Ansicht der Integrations- und Testplattform.

In den folgenden Abschnitten wird die Realisierung der Trac-Plugins für die konzipierten Werkeuge präsentiert.

6.1 Einbettung des Vorgehensmodells

Das in Abschnitt 5.1 entwickelte Vorgehensmodell für die verteilte Integration und den Test von Serviceroboterapplikationen, DDT, wird mit Hilfe des Eclipse Process Frameworks [Kro05] modelliert. Der Vorteil der Nutzung von EPF besteht zum einen in der impliziten Konformität mit dem OMG Prozess Meta-Modell Standard (SPEM, vgl. Abschnitt 4.2.3), und zum anderen in der Möglichkeit der Publizierung des Vorgehensmodells in Form einer Webseite. Dadurch kann das Modell sehr gut in die webbasierte Integrationsplattform (vgl Abschnitt 5.4) eingefügt werden.

6.1.1 Implementierung des Vorgehensmodells in EPF

Die Iterationen der einzelnen Phasen wurden entsprechend der Tabellen 5.1 und 5.2 im Abschnitt 5.1.4 in EPF modelliert. Die einzelnen Phasen, Aktivitäten, Rollen und Aufgaben des OpenUP Prozessmodells wurden dabei als Basis übernommen und entsprechend des DDT Konzepts modifiziert. Der Vorteil dieser Vorgehensweise besteht darin, dass so allgemeine Best-Practice Handlungsanweisungen, Leitfäden und Checklisten aus OpenUP übernommen werden konnten und im DDT Modell als *Guidelines* zur Verfügung stehen.

Das Vorgehensmodell wurde mit Hilfe von EPF als Webseite publiziert und als DDT-Wiki-Plugin in die Integrations- und Testplattform eingebunden. Es unterstützt den Integrationsprozess durch die Definition der verschiedenen Aktivitäten und Aufgaben und deren Zuordnung zu den verschiedenen Rollen sowie konkrete Handlungsanweisungen zur Durchführung der einzelnen Aufgaben. Das Vorgehensmodell kann für konkrete Projekte weiter spezialisiert werden.

Abbildung 6.3 illustriert das DDT-Wiki Plugin eingebettet in der Trac-Oberfläche. Im Plugin ist das gesamte Vorgehensmodell als Wiki enthalten, so dass Informationen und Handlungsanweisungen zum Integrationsprozess schnell auffindbar sind. In der Abbildung sind z.B. die Aktivitäten der DDT-Integrationsphase dargestellt. Die konzeptuelle Trennung des Komponenten- und des Applikationsinkrements wird durch einen entsprechenden Synchronisierungspunkt vor der Aktivität (“Applikationstest”) realisiert. Die Abhängigkeitsbeziehungen der einzelnen Aktivitäten sind in den Abbildungen durch Pfeile dargestellt. Aktivitäten, die nicht über einen Pfeil oder eine Synchronisierungsleiste miteinander in

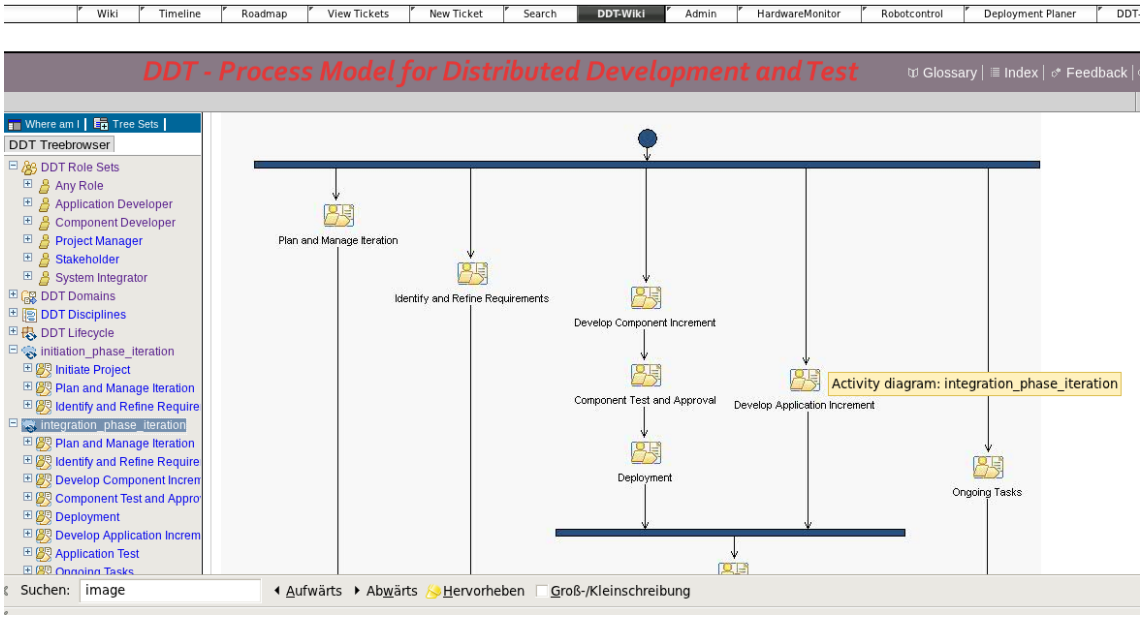


Abbildung 6.3: DDT Plugin - Darstellung der Aktivitäten der Integrationsphase.

Verbindung stehen, können nebenläufig durchgeführt werden, sofern es die Teamressourcen erlauben.

Abbildung 6.4 veranschaulicht die Aktivität “Abnahme und Test”. Ziel dieser Aktivität ist die Bereitstellung der abgenommenen Pakete in ausführbarer Form auf dem Zielsystem. Dazu muss zunächst auf Basis der Spezifikationen des Zielsystems und der modellierten Komponenten- und Applikationsgraphen sowie eventuellen Randbedingungen aus den Anforderungen ein Deployment-Plan erstellt werden, der die Zuordnungen der einzelnen Komponenten auf die Recheneinheiten des Zielsystems enthält (vgl. Abschn. 5.2.2 und 6.2). Falls sich die Zielsystem- und Applikations-Spezifikationen nach den ersten Iterationen nicht mehr ändern sollten, kann die Planungsaktivität entfallen. Der Deployment-Plan wird daraufhin für die abgenommenen Komponenten umgesetzt (s. auch Abschn. 6.2.2).

Auch die einzelnen Rollen sind im Wiki nebst ihren Aufgaben und den Arbeitsprodukten, die sie jeweils verantworten, beschrieben. Abbildung 6.5 zeigt z.B. die Tätigkeiten des Komponentenentwicklers für die Weiterentwicklung der Komponente entsprechend der aktualisierten Anforderungen. Der Komponentenentwickler ist angehalten, die aktualisierte Komponente durch selbst konzipierte Entwickler-Tests, z.B. in Form von Unittests oder auch komponentenübergreifenden Tests zu überprüfen, bevor er sie dem Systemintegrator zur Abnahme übergibt. Komponentenübergreifende Tests können vom Entwickler (falls erforderlich) mit Hilfe der Laufzeitwerkzeuge (s. Abschn. 5.3 und 6.3) durchgeführt werden.

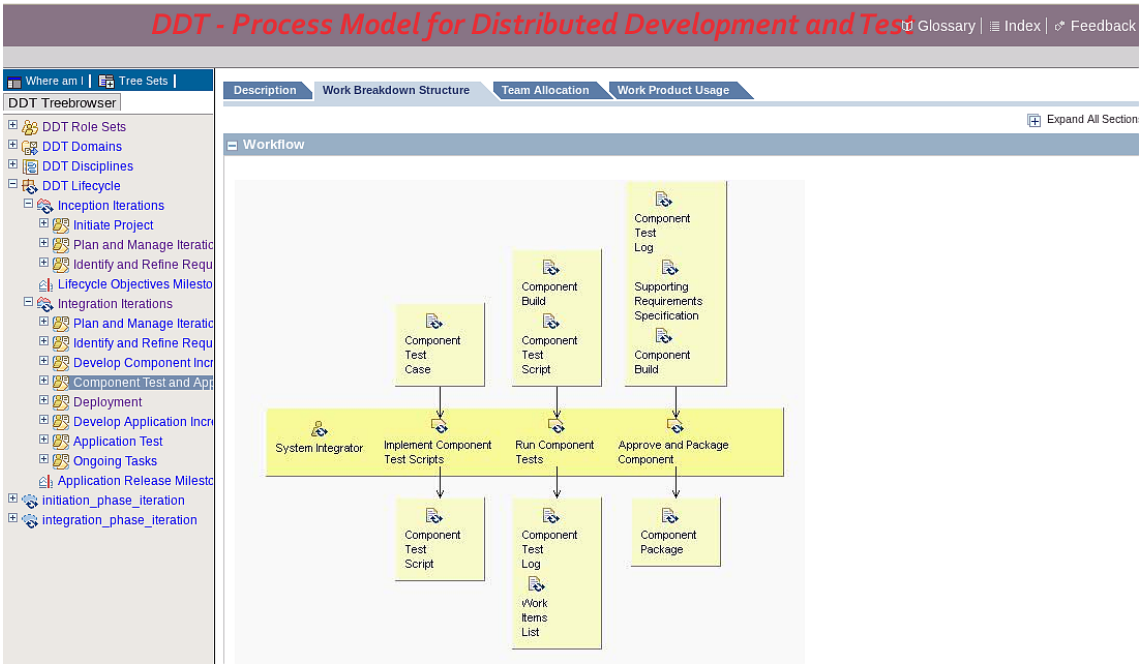


Abbildung 6.4: DDT Plugin - Darstellung der Aktivität “Abnahme und Test”.

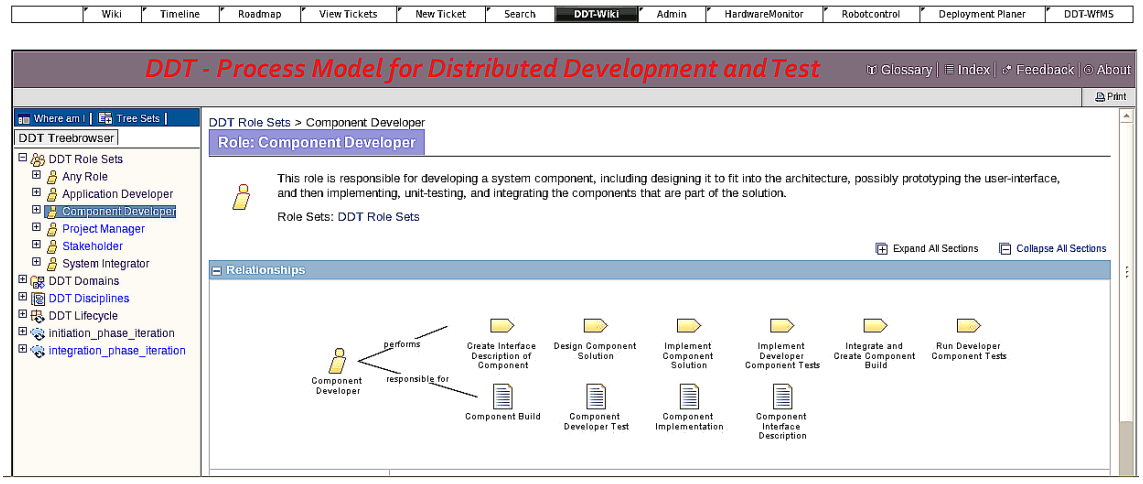


Abbildung 6.5: DDT Plugin - Aufgaben und Arbeitsprodukte des Komponentenentwicklers.

6.2 Realisierung des Deployment-Moduls

Das Deployment-Modul bildet zusammen mit den Laufzeitwerkzeugen direkt die Deploymentaktivitäten des DDT Vorgehensmodells ab. In den folgenden Abschnitten wird die Realisierung der Aktivitäten Deployment-Planung, Installation und Aktualisierung präsentiert. Die Aktivitäten Aktivierung/Deaktivierung und Konfiguration werden durch die Laufzeitwerkzeuge abgedeckt (s. Abschnitt 6.3).

6.2.1 Deployment-Planung

Das Werkzeug zur Deployment-Planung setzt den in Abschnitt 5.2.2 konzipierten Algorithmus zur Planung des Deployments einer Serviceroboter-Applikation auf das Zielsystem in Form eines Trac-Plugin um. Der Algorithmus benötigt als Eingangsinformationen sowohl ein Modell der komponenten-basierten Applikation als auch ein Modell der Zieldomäne Serviceroboter, jeweils in Form eines Graphen. Um die Erstellung der Modelle zu vereinfachen und diese visualisieren zu können, wird zunächst ein graphisches Werkzeug realisiert.

Graphisches Werkzeug zur Modellierung des Applikations- und Domänengraphen

Das graphische Werkzeug erlaubt die Modellierung eines Graphen sowie die Zuordnung von Ressourcen zu den einzelnen Knoten und Kanten. Die so modellierten Graphen können mit Hilfe dieses Editors als XML-Strukturen abgespeichert und in dieser Form dem Deployment-Planer als Eingangsinformation übergeben werden. Der Editor ist auch in der Lage, Graphen aus vorhandenen XML-Strukturen einzulesen und zu visualisieren. So kann z.B. mit Hilfe der Informationen aus der Hardware-Überwachung (vgl. Abschnitt 6.3.2) die aktuelle Auslastung des Systems im Domänen-Graphen (z.B. durch unterschiedliche Farbgebung) dargestellt werden.

Mit Hilfe der XML-Dokumenttypdefinitionssprache DTD (document type definition) wurde eine formelle Spezifikation des Datenformats für die Graphmodelle erstellt. Somit können die XML-Strukturen jeweils validiert werden, bevor sie dem Algorithmus bzw. dem Graph-Editor übergeben werden. In Abbildung 6.6 ist die DTD-Struktur in der ENBF-Repräsentation [Intd] dargestellt. Das graphische Werkzeug wurde in Javascript implementiert, so dass es über einen Webbrowser ausführbar ist und damit einfach in die webbasierte Architektur der Integrations- und Testplattform eingebettet werden kann.

Abbildung 6.6: Dokumenttypdefinition (DTD) für die Beschreibung von Applikations- und Domänengraphen

```
GRAPH = <graph name=GRAPH_IDENTIFIER>
[NODE NODE (NODE | EDGE)* ];
</graph>;

NODE = <node identifier=NODE_IDENTIFIER type=NODE_TYPE>
(RESOURCE)*
(PROPERTY)*
</node>;

EDGE = <edge start=START_NODE end=END_NODE >
(RESOURCE)*
</edge>;

RESOURCE = <resource name=RESOURCE_IDENTIFIER type=RESOURCE_TYPE>
VALUE
</resource>;

PROPERTY = <property name=PROPERTY_IDENTIFIER>
VALUE
</property>;

NODE_TYPE =( comp|color-sensor|3D-sensor|manipulator|gripper|peripheral);
RESOURCE_TYPE =(cap|quant|max|min|attr|sel);
START_NODE = NODE_IDENTIFIER;
END_NODE = NODE_IDENTIFIER;
GRAPH_IDENTIFIER = (CHARACTER)+;
NODE_IDENTIFIER = (CHARACTER)+;
RESOURCE_IDENTIFIER = (CHARACTER)+;
PROPERTY_IDENTIFIER = (CHARACTER)+;
CHARACTER = "A" | ... | "z"| "0"| ..."9"
VALUE = ("0"| ..."9")+
```

Die Knoten und Kanten des Applikationsgraphen repräsentieren Komponenten und deren Kommunikationsverbindungen, wobei jeder Knoten und jede Kante mit beliebig vielen Ressourcen-Anforderungen verknüpft werden kann. Die Knoten werden jeweils durch einen eineindeutigen Bezeichner und einen Knotentyp definiert, die Kanten jeweils durch einen Start- und Endknoten. Für die Domäne Servicerobotik sind in Abbildung 6.6 generische Komponenten-Kategorien zur Spezifikation des Knotentyps angegeben. Diese Menge erhebt nicht den Anspruch auf Vollständigkeit und kann einfach erweitert bzw. auf einer tieferen Abstraktionsebene spezifiziert werden (vgl. Diskussion Ontologien in Abschnitt 5.2.1 und Abschnitt 5.2.2). Die Kanten werden eindeutig beschrieben durch einen Start- und einen Endknoten. Die Ressourcen werden durch einen Bezeichner, einen Eigenschaftstyp entsprechend der OMG D&C Norm und einen Zahlenwert beschrieben. Der Bezeichner der Software-Komponenten im Applikationsgraphen sollte dabei mit dem entsprechenden Paketnamen übereinstimmen, unter dem die Komponente im Repository abgelegt ist. Sind Installationsskripte in den Software-Komponenten des Applikationsgraphen als Eigenschaft (property-tag) hinterlegt, kann die Installation automatisiert durch ein Werkzeug erfolgen (vgl. Abschnitt 6.2.2).

Abbildung 6.7 zeigt einfache Beispiele für einen Applikations-und einen Domänengraphen, die jeweils mit Hilfe des Editors modelliert wurden. Für die Applikation "Greifen eines Objekts“ sind Treiber-Komponenten für die relevanten Aktoren und Sensoren erforderlich sowie Komponenten zur Objekterkennung und Greifplanung. Ein Spracherkennungsmodul ermöglicht die Interaktion mit dem Nutzer, und eine weitere Komponente sorgt für die Koordination zwischen den einzelnen Softwarebausteinen (*Sequencer*). Hardware-Komponenten können durch entsprechende Knote-Attribute näher spezifiziert werden, (z.B. Hersteller, Baureihe, Firmware-Version, etc.). Falls eine Hardwaregruppe standardisierte Schnittstellen besitzt (z.B. GiGE Vision oder IEEE1394 bei Kameras) , kann es auch ausreichen, nur die relevanten Parameter der Komponente zu spezifizieren, z.B. die Bildwiederholrate oder Auflösung einer Farbkamera. Dadurch kann der Planer je nach Abstraktionsebene der verwendeten Spezifikationssprache bis zu einem gewissen Grad formal überprüfen, ob die Hardware-Ausstattung des Zielsystems den Anforderungen der Zielapplikation genügt. Der Domänengraph enthält eine Teilmenge der Hardware-Komponenten der Care-O-bot 3 Plattform (Rechner, Farbkameras, Arm, Greifer). Die Information, welche HW-Komponenten an welche Rechner angeschlossen sind, ist implizit über die Kommunikationskanäle gegeben.

Der Domänengraph wird typischerweise einmal vom Systemintegrator für eine bestimmte Zielplattform erstellt und lediglich bei Änderungen der Hardware aktualisiert. Der Applikationsentwickler greift dann zur Modellierung des Applikationsgraphen auf den erstellten Domänengraphen zurück, durch den er in abstrakter Form alle erforderlichen Informationen über die Hardwarearchitektur erhält. In der Ansicht für die Modellierung des Applikations-

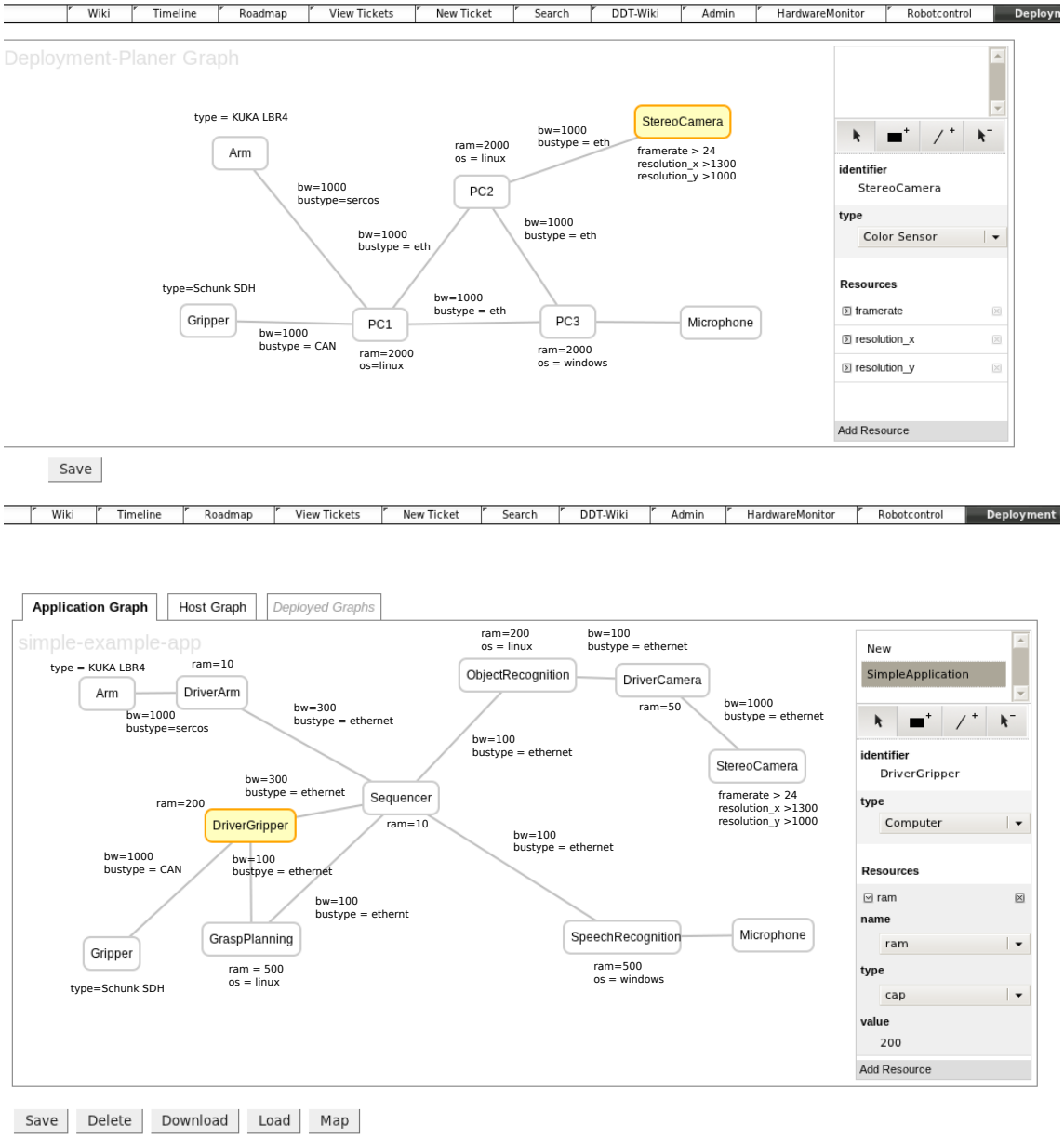


Abbildung 6.7: Oben: Vereinfachter Domänengraph von Care-O-bot 3, unten: Applikationsgraph für eine Greifanwendung

Tabelle 6.1: Deployment Kandidaten für die Komponenten der Beispielapplikation in Abbildung 6.7

Komponente C_i^A	**Menge der Kandidaten** $Z_D(C_i^A)$
Arm	Arm
Gripper	Gripper
Microphone	Microphone
StereoCamera	StereoCamera
DriverArm	PC1
DriverGripper	PC1
DriverCamera	PC2
SpeechRecognition	PC3
ObjectRecognition	PC1, PC2
GraspPlanning	PC1, PC2
Sequencer	PC1, PC2, PC3

graphen wird neben der Lade- und Speicherfunktionen für Graphen die Durchführung der Deployment-Planung angeboten, so dass der Applikationsentwickler sich eine direkte Rückmeldung einholen kann, ob das Zielsystem alle Anforderungen der Applikation erfüllt und diese installiert werden kann.

Anhand der Beispielgraphen in Abbildung 6.7 wird im Folgenden die Umsetzung des Deployment-Planers dargestellt.

Umsetzung des Deployment-Planers

Nach Algorithmus 5.2.3 besteht der erste Planungsschritt in der Ermittlung der jeweiligen Deployment-Kandidaten für jede Komponente der Applikation nach Gleichung 5.13. In Tabelle 6.1 sind die Deployment-Kandidaten für die Beispielgraphen dargestellt. Nach Gleichung 5.16 ergeben sich also $|I_{A,D}| = 12$ mögliche Deployments.

Allerdings sind nicht alle diese Deployments auch automatisch gültige Deployments im Sinne der Ressourcenbelegung. Die Valididät eines möglichen Deploymentkandidaten $i^{A \mapsto D}$ wird durch durch sequenzielle Zuordnungen unter Aktualisierung der verbleibenden Ressourcen des Zielsystems ermittelt (vgl. 5.2.1). Würde zum Beispiel die Komponente *DriverGripper* die (modellierten) Ressourcen von *PC1* völlig ausschöpfen, könnten auf diesem Rechner keine weiteren Komponenten betrieben werden.

Nach Anwendung des Algorithmus 5.2.1 sind für das Beispiel alle 12 Deployments gültig. Benötigt die Komponente *DriverCamera* jedoch z.B. 2000 MB Arbeitsspeicher, schrumpft die Anzahl gültiger Deployments auf 2. In dem reduzierten Beispiel kann das Berechnungsergebnis des Algorithmus einfach nachvollzogen werden: Da das Softwaremodul *DriverCa-*

mera durch die Abhängigkeit von der Komponente *StereoCamera* an *PC2* gebunden ist und dessen Ressourcen vollständig belegt, fällt dieser Rechner als Deploymentoption für alle anderen Komponenten weg, so dass nur noch für die Komponente *Sequencer* eine Deploymentoption (PC1 oder PC3) bleibt:

1. { ('Gripper' ↦ 'Gripper'), ('DriverArm'↦ 'PC1'), ('Arm'↦ 'Arm'), ('GraspPlanning'↦ 'PC1'), ('StereoCamera'↦ 'StereoCamera'), ('SpeechRecognition'↦ 'PC3'), (Object-Recognition'↦ 'PC1'), ('Microphone'↦ 'Microphone'), ('Sequencer'↦ 'PC1'), ('Driver-Camera'↦ 'PC2'), ('DriverGripper'↦ 'PC1') }
2. { ('Gripper'↦ 'Gripper'), ('DriverArm'↦ 'PC1'), ('Arm'↦ 'Arm'), ('GraspPlanning'↦ 'PC1'), ('StereoCamera'↦ 'StereoCamera'), ('SpeechRecognition'↦ 'PC3'), ('Object-Recognition'↦ 'PC1'), ('Microphone'↦ 'Microphone'), ('Sequencer'↦ 'PC3'), ('Driver-Camera'↦ 'PC2'), ('DriverGripper'↦ 'PC1')] }.

Benötigte auch die Komponente *DriverGripper* 2000 MB Arbeitsspeicher, würde kein gültiges Deployment mehr existieren.

Die verschiedenen gültigen Deployments können nun entsprechend weiterer nutzerspezifischer Kriterien bewertet werden, z.B. gleichmäßige Auslastung der Ressourcen, möglichst geringe Netzwerkauslastung, etc. In Abbildung 6.8 sind die Deploymentvarianten für die Beispielgraphen dargestellt. Während die Auslastung der Rechner kaum Unterschiede aufweist, ergibt sich für Deploymentvariante 2 eine deutlich höhere Auslastung des Kommunikationskanals zwischen PC1 und PC3, so dass die Deploymentvariante 1 unter dem Optimierungskriterium "gleichmäßige Auslastung der Ressourcen" vorzuziehen wäre.

Das Deployment Planer Modul wurde in der Sprache Python umgesetzt, da die Trac-Umgebung ebenfalls auf Python basiert und somit eine Integration als Plugin einfach erfolgen kann.

6.2.2 Installation und Aktualisierung

Wenn das Deployment nach dem Planungsschritt feststeht, können die Komponenten gemäß der Deployment-Zuordnung auf dem Zielsystem installiert werden. Diese Funktionalität wurde in das graphische Werkzeug zur Deployment-Planung in Form einer entsprechenden Befehlsschaltfläche eingebunden. Um die Plattformunabhängigkeit zu wahren, muss einmalig ein host-spezifischer Installationscode duch den Systemintegrator bereitgestellt werden, der in den Hostmodellen als Installationsskript hinterlegt werden kann. Dieser Installationscode besteht im einfachsten Fall aus dem Befehl zur Installation eines Pakets auf dem Betriebssystem, kann aber durch weitere Skripte vor und nach der eigentlichen Installati-

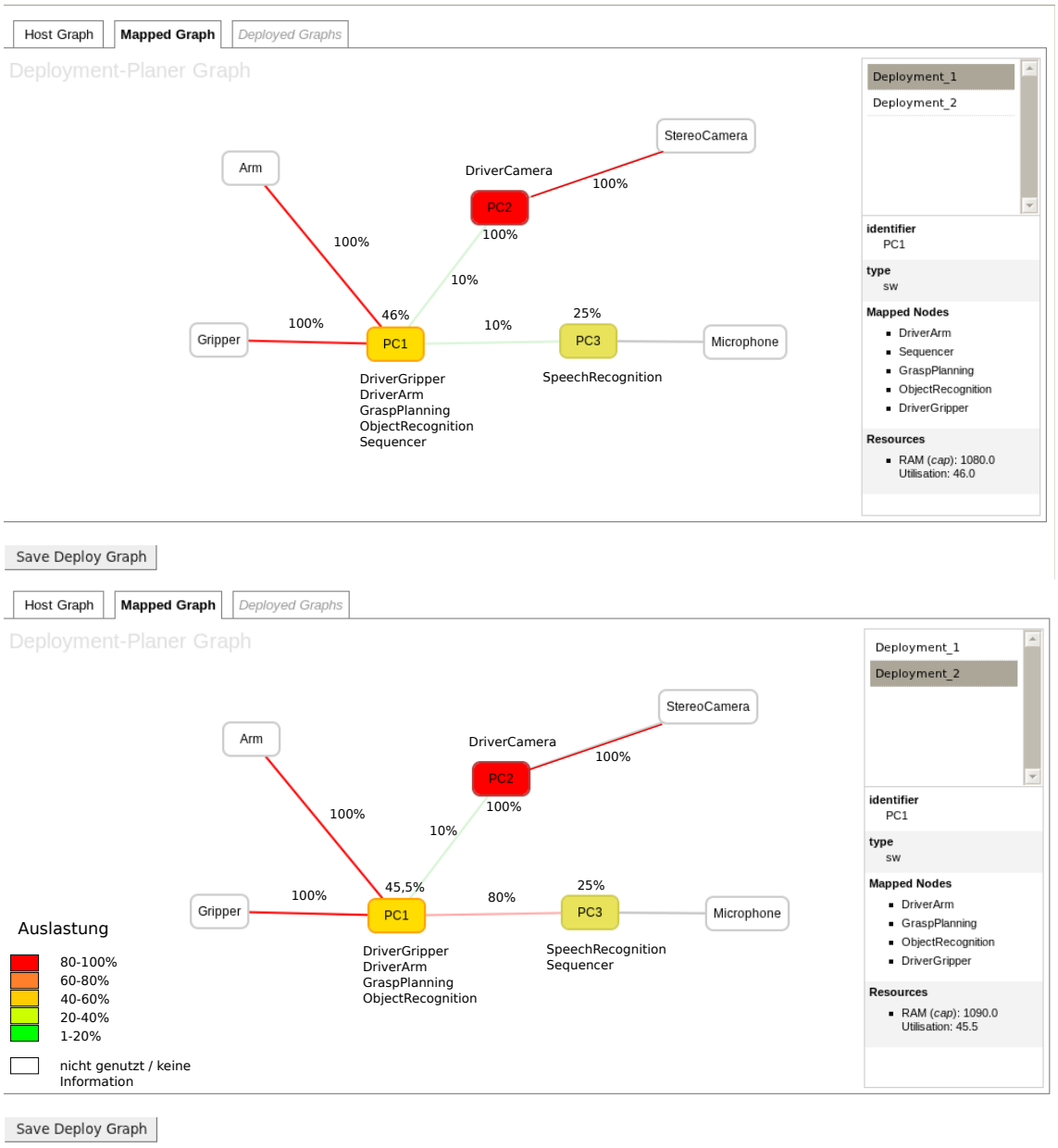


Abbildung 6.8: Vergleich der beiden Deployments bezüglich der Auslastung der modellierten Ressourcen.

```
#preinstallation instructions
#the following script must be provided by component developer
preinstallation_instructions.sh

# install component $1 (command line argument 1)
# using the packet management system dpkg that
# in addition checks all (version) dependencies
# and installs the missing packages if they are available
# in the package repository
apt-get install $1

# post installation instructions
# the following script must be provided by component developer
postinstallation_instructions.sh
```

Abbildung 6.9: Beispiel für ein Installationsskript für einen Linux-Host.

on, z.B. zur Einrichtung von Umgebungsvariablen auf dem System o.ä. ergänzt werden. In Abbildung 6.9 ist ein Beispiel für ein Installationskript für einen Linux-Host abgebildet.

Die Installations-Aktivität kann ebenfalls ohne Kenntnisse der Hardwarearchitektur und somit grundsätzlich von allen Rollen durchgeführt werden. Allerdings kann durch die Anwendung fehlerhafter Installationsskripte Schaden auf dem Zielsystem entstehen. Deshalb sollte diese Aktivität durch den Systemintegrator durchgeführt werden, der die Installationsskripte vorab auf unerwünschte Nebeneffekte überprüft. Um sicherheitskritische Aktivitäten wie die Installation nur für einen eingeschränkten Nutzerkreis zuzulassen, wurden entsprechende Nutzerrechte für die Werkzeuge eingeführt (vgl. Abschnitt 6.3).

Neben der Überprüfung der Installationsskripte bleibt dem Systemintegrator noch ein manueller Einrichtungsschritt, damit die Installation von Komponenten über das Deployment-Werkzeug durchgeführt werden kann: das Paketrepository, in dem sich die zu installierenden Komponenten befinden, muss den Paketmanagementsystemen auf den Zielrechnern bekanntgemacht werden. Für Betriebssysteme, in denen kein Paketmanagementsystem mitgeliefert wird, muss ggf. ein Paketmanagementsystem nachinstalliert werden (z.B. [Leb] für Windows). Eine weitere Voraussetzung ist die korrekte Konfiguration der Laufzeitumgebung (insbesondere die Rechnerkonfiguration, s. Abschnitt 6.3).

Zusammenfassend werden durch das Installationswerkzeug die folgenden Schritte durchgeführt:

- Herunterladen der Komponentenpakete aus dem Repository (dpkg-basierte Skripte)
- Entpacken der Pakete

- Vor-Konfiguration der Pakete
- Durchführung von Post-Install-Code
- Übertragung des Deployment-Plans in Form eines Komponenten/Host-Mappings in die Trac-Datenbank, so dass sie dem Laufzeitwerkzeug zur Verfügung steht.

Mit dem letzten Schritt erfolgt die nahtlose Anbindung der Werkzeugkette Deployment-Planung - Installation (vgl. Abb. 5.6) an die Werkzeugkette Konfiguration - Aktivierung (vgl. Abb. 5.7).

Die Aktivität *Aktualisierung* wird durch diese Schritte implizit mit abgedeckt, da Paketmanagementsysteme so konfiguriert werden können, dass automatisch die aktuelle Version eines Paketes unter Berücksichtigung der Abhängigkeiten heruntergeladen wird.

6.3 Realisierung der Werkzeuge zur Laufzeitunterstützung

Neben der reinen Unterstützung der Laufzeitaktivitäten realisiert das Laufzeitwerkzeug die Möglichkeit des Fernzugriffs für alle Aktivitäten des Deployment-Prozesses, die direkten Zugang zur Hardware benötigen: Installation/Aktualisierung, Konfiguration, Aktivierung/Deaktivierung. Es setzt sich aus zwei Trac-Modulen zusammen: dem *Robotcontrol*-Plugin, das den eigentlichen Zugang zur Rechnerarchitektur des Zielsystems realisiert, und das *HardwareMonitor*-Plugin, das die Überwachung relevanter Messgrößen der Hardware erlaubt. In den folgenden Abschnitten wird die Umsetzung beider Module sowie deren Konfiguration präsentiert. Zum Abschluss folgt eine Analyse der Eignung verschiedener Übertragungsprotokolle für die effiziente Übertragung von 3D-Anwendungen, die in Form von Visualisierungen in der Servicerobotik sehr häufig Anwendung finden.

6.3.1 Das Robotcontrol-Plugin

Das Robotcontrol-Plugin stellt das Herzstück des Laufzeitwerkzeugs dar. Es realisiert den ortstransparenten und architektur-agnostischen Zugriff auf das verteilte System der Zielplattform über Remote Session Werkzeuge. Die Zugriffskontrolle bezüglich der Deployment-Aktivitäten erfolgt dabei über zwei Mechanismen: die Zuteilung von unterschiedlichen Nutzerrechten, die gewährleisten dass die einzelnen Aktivitäten nur von einer bestimmten

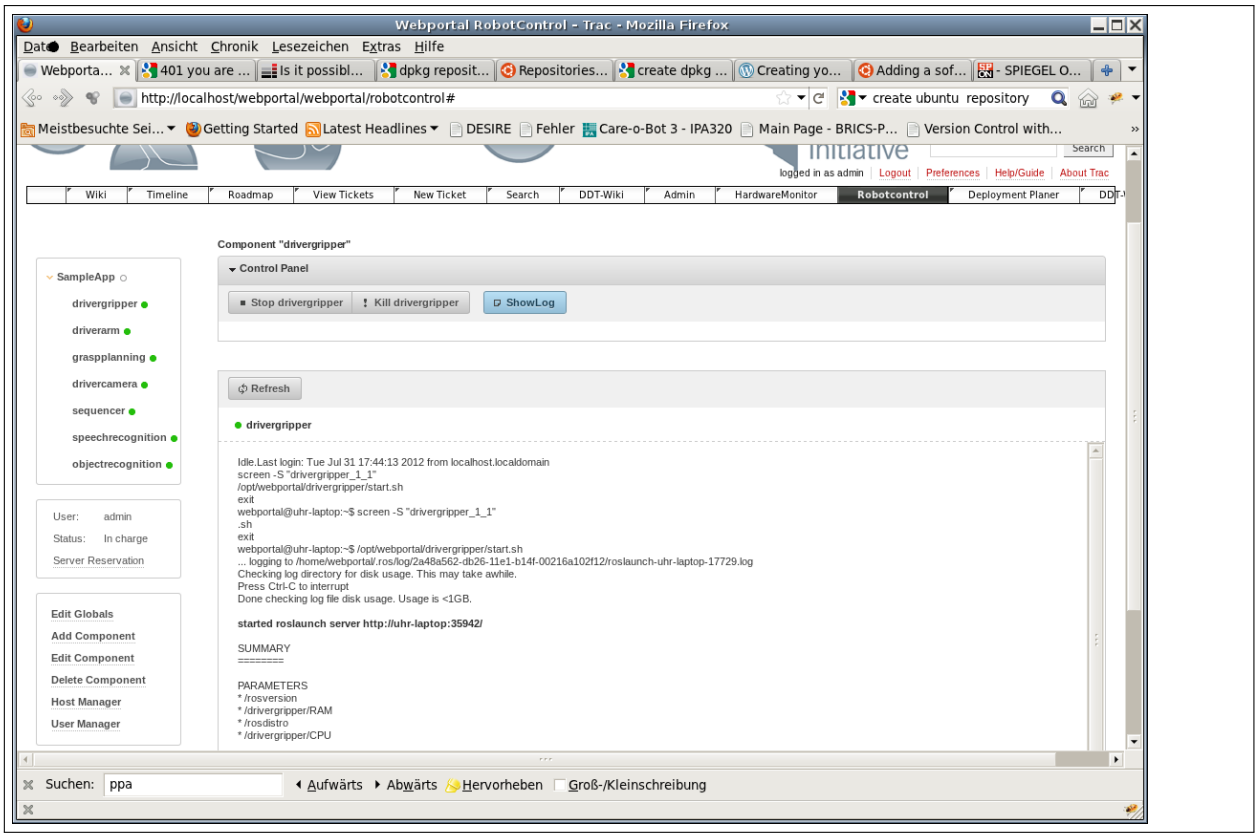


Abbildung 6.10: Robotcontrol Plugin. Aktivierung und Deakivierung der Applikation bzw. der einzelnen Komponenten der Applikation

Nutzergruppe (z.B. Systemintegratoren) durchgeführt werden können, und einem Reservierungssystem, das die Nutzungshoheit für eine bestimmte Person und einen bestimmten Zeitraum gewährleistet.

Abbildung 6.10 zeigt die Oberfläche des Robotcontrol-Plugins. In der Mitte befindet sich die Hauptansicht, in der z.B. Informationen zur Ausführung in Form von Log-Ausgaben angezeigt werden und die Konfiguration von Komponenten erfolgt. Auf der linken Seite befinden sich verschiedene Untermenüs zur Steuerung des Plugins. Oben links befindet sich das Menü zur Aktivierung und Deaktivierung von Komponenten, in der Mitte das Reservierungsmenü und unten das Konfigurationsmenü. Die einzelnen Menüs sowie deren Funktionen werden in den folgenden Abschnitten genauer beschrieben.

Aktivierung und Deaktivierung von Komponenten und Applikationen

Alle Komponenten können im Robotcontrol-Plugin grundsätzlich entweder einzeln oder gruppiert aktiviert und deaktiviert werden. Dazu können hierarchische Gruppen von Komponenten angelegt werden. In Abbildung 6.10 sind z.B. alle Komponenten der Gruppe ”SampleApp“ zugeordnet, so dass zur Aktivierung und Deaktivierung dieser Applikation

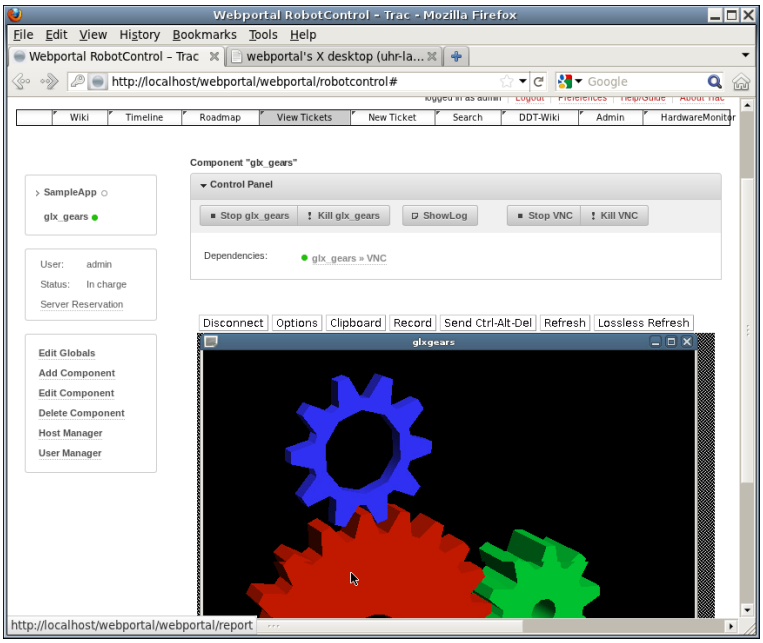


Abbildung 6.11: Robotcontrol Plugin - VNC

lediglich die Gruppe gestartet bzw. gestoppt werden muss. Aktivierung und Deaktivierung von Komponenten erfolgen über entsprechende Skripte des Komponentenentwicklers, die den Zustandsautomaten der Komponente berücksichtigen. Der Ausführungsstatus der Gruppen sowie der einzelnen Komponenten wird jeweils durch ein Statuslämpchen angezeigt. Die Zustandsabfrage erfolgt grundsätzlich über ein Skript des Komponentenentwicklers, das in regelmäßigen Abständen vom Robotcontrol-Plugin auf dem entsprechenden Rechner ausgeführt wird (z.B. Abfrage von "Alive-Messages"). Falls kein Skript vorhanden ist, wird der Ausführungszustand einer Komponente aus dem Status des entsprechenden Prozesses auf Betriebssystemebene abgeleitet. Der Zustand der Gruppe ergibt sich als Und-Verknüpfung aus den Komponentenzuständen. In der Hauptansicht kann nun die Log-Ausgabe einer ausgewählten Komponente angezeigt werden. Falls eine Komponente eine Oberfläche zur Visualisierung oder Steuerung besitzt, kann diese über VNC vom Rechner auf das Robotcontrol-Plugin übertragen werden. Dazu kann für jede Komponente ein VNC-Server eingerichtet werden, der die Oberfläche der Komponente auf einer bestimmten URL überträgt. Wahlweise kann diese URL in einem eigenen Fenster oder in der Hauptansicht des Robotcontrol Plugins angezeigt werden. Abbildung 6.11 zeigt als Beispiel die 3D-Anwendung "glxgears."

Die einzelnen Komponenten können grundsätzlich über das Konfigurationsmenü (s.u.) hinzugefügt und modifiziert werden. Sofern die Installation der einzelnen Komponenten über das Deployment-Plugin erfolgreich war, werden die Komponenten automatisch entsprechend des Deployment-Planes im Robotcontrol-Plugin konfiguriert, so dass die Applikation nach der Installation in der Regel direkt ausführbar ist.

```
CONFIGURATION = <Configuration>
(PARAM )*;
</Configuration>

PARAM = <param name=PARAM_IDENTIFIER  value=PARAM_VALUE >

PARAM_IDENTIFIER = (CHARACTER)+;
PARAM_VALUE = (CHARACTER)+;
```

Abbildung 6.12: Dokumenttypdefinition (DTD) für die Beschreibung von Konfigurationsdateien

Konfiguration

Der werkzeuggestützte Parameterzugriff muss über ein komponentenspezifisches Nutzer-Skript erfolgen, da das Deployment-Werkzeug unabhängig vom Komponentenmodell sein soll und daher keine Annahmen über die Zugriffsweise auf die Komponentenparameter treffen kann. Dazu wurde das in Abbildung 6.12 spezifizierte XML-Format definiert, welches vom Trac-Plugin interpretiert werden kann. Das Nutzer-Skript, dessen Bereitstellung dem Komponentenentwickler obliegt (vgl. Abbildung 5.7), muss also die Hin- und Rückkonvertierung aller komponentenspezifischen Parameter in dieses XML-Format gewährleisten. Es muss für eine Komponente jedoch nur einmal erstellt bzw. lediglich bei Änderungen der Anzahl oder Zugriffsweise der Parameter angepasst werden.

Über das Werkzeug können die Komponentenparameter dann eingesehen und (bei entsprechenden Rechten) geändert und zunächst in Form der XML-Datei zurückgeschrieben werden. Als zweiter Schritt wird vom Trac-Plugin dann wieder die Konvertierungsroutine aufgerufen, die die Parameter aus der XML-Datei wieder entsprechend in der installierten Komponente aktualisiert. Im Trac Plugin ist grundsätzlich auch das Hinzufügen neuer Parameter möglich - es hängt allerdings von der Konvertierungsroutine ab, ob diese neuen Parameter aus dem XML-File korrekt auf die Komponente übertragen werden.

Abbildung 6.13 zeigt ein Beispiel für die Konvertierung einer komponentenspezifischen Parameterdatei in das vom Trac-Plugin lesbare XML Format. Die Komponentenparameter können dabei über mehrere Dateien vertreut sein, allerdings müssen alle Parameter durch die Konvertierungsroutine in eine gemeinsame XML-Datei überführt werden.

Nutzerrechte und Reservierungssystem

Um verschiedene Aktivitäten - insbesondere die Konfiguration der Laufzeitumgebung und der verfügbaren Komponenten und Applikationen - nur einer eingeschränkten Nutzergruppe zugänglich zu machen, wurden Nutzerrechte mit unterschiedlichen Privilegien erstellt (s.

cob_sdh.yaml

```
sdhdevicetype:     PCAN
d sdhdevicestring: devpcan0
dsadevicestring:   devttyS0
baudrate:          1000000
JointNames:        ['sdh_knuckle_joint',
                    'sdh_thumb_2_joint',
                    'sdh_thumb_3_joint',
                    'sdh_finger_12_joint',
                    'sdh_finger_13_joint',
                    'sdh_finger_22_joint',
                    'sdh_finger_23_joint']
OperationMode:     position
frequency:         5
```

cob_sdh.xml

```
<Configuration>
  <Param name=sdhdevicetype value=PCAN/>
  <Param name=sdhdevicestring value=devpcan0/>
  <Param name=dsadevicestring value=devttyS0/>
  <Param name=baudrate value=1000000
  <Param name=JointNames value=
    ['sdh_knuckle_joint',
    'sdh_thumb_2_joint',
    'sdh_thumb_3_joint',
    'sdh_finger_12_joint',
    'sdh_finger_13_joint',
    'sdh_finger_22_joint',
    'sdh_finger_2der Rechner3_joint']/>
  <Param name=OperationMode value=position/>
  <Param name=frequency value=5/>
</Configuration>
```

Abbildung 6.13: Beispiel für die Umwandlung komponentenspezifischer Parameterfiles in das DTD XML-Format durch ein Konfigurationsskript. (Links: ROS yaml Konfigration für die cob_sdh Komponente, Rechts: durch die Konvertierungsroutine generierte XML-Version)

Tabelle 6.2: Zugriffsrechte

Zugriffsrecht	**Privileg**	**typ. Rolle**
DEPLOYMENT_GRAP	Erstellung und Modifikation des Domänengraphen	SI
APPLICATION_GRAPH	Erstellung und Modifikation des Applikationsgraphen	AE
ACTION_RUN	Aktivierung von Komponenten	KE, AE
ACTION_STOP	Deaktivierung von Komponenten	KE, AE
COMP_ADMIN	Erstellung und Modifikation von Komponenten	KE, AE
HOST_ADMIN	Erstellung und Modifikation von Rechnerverbindungsdaten	SI
HWMONITOR_VIEW	Ansicht des Hardwaremonitors	AE, SI
HWMONITOR_ADMIN	Modifikation der Konfiguration des Hardware-Monitor-Plugins	SI

Tabelle 6.2). Der Nutzer ist allen Trac-Plugins durch den Authentifizierungsmechanismus von Apache bekannt, so dass lediglich eine Zuordnung von Nutzer zu den entsprechenden Rechten erforderlich ist. Diese Zuordnung kann im *User Manager* (s. folgender Abschnitt) vorgenommen werden.

Alle Aktivitäten, die eine Modifikation beinhalten, setzen darüber hinaus eine Reservierung im zentralen Reservierungssystem voraus. Ohne Reservierung können grundsätzlich nur die Ansichtsrechte (sofern der Nutzer diese besitzt) genutzt werden. So können z.B. mehrere verteilte Entwickler einen entfernten Test in der gemeinsamen Entwicklungsumgebung des Robotcontrol-Plugins verfolgen. Das Reservierungssystem ist durch einen Kalender realisiert, der in der Trac-Datenbank vorgehalten wird.

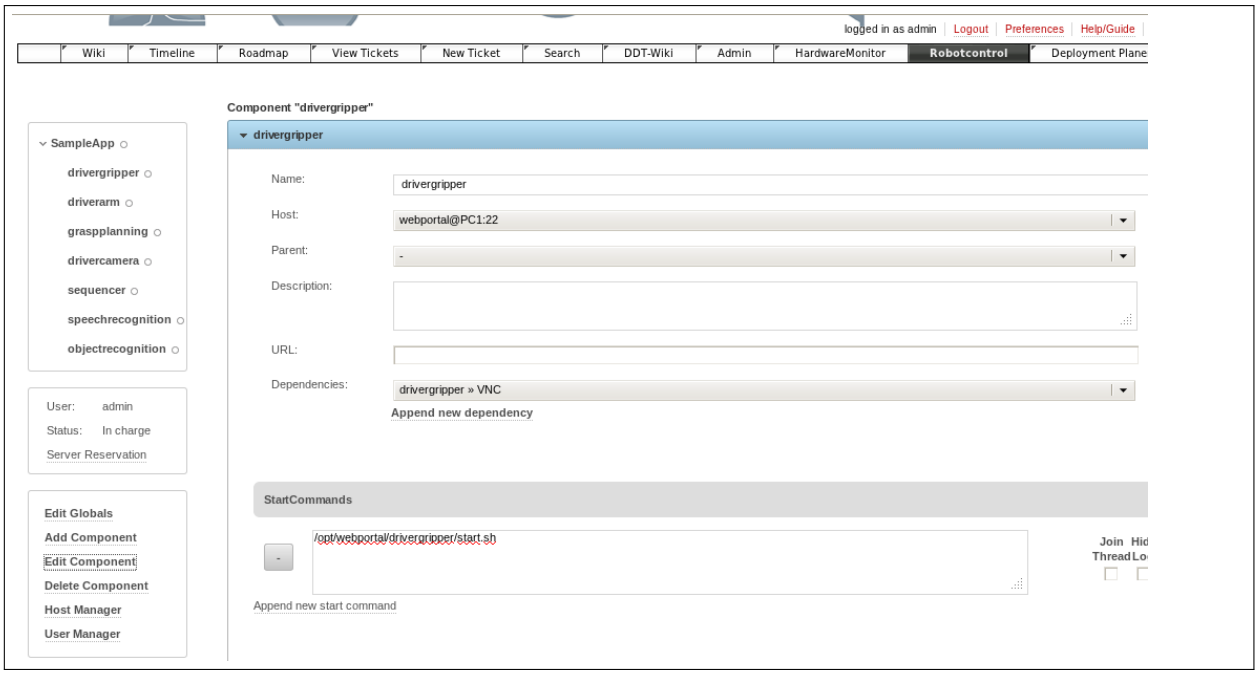


Abbildung 6.14: Robotcontrol-Plugin - Komponenten-Manager. Konfiguration der Komponenten

Konfiguration des Robotcontrol-Plugins

Die Konfiguration des Robotcontrol-Plugins erfolgt im wesentlichen über drei verschiedene Oberflächen: im *Host Manager* können die Verbindungsdaten der Rechner im Zielsystem konfiguriert werden, der *Komponenten Manager* (s. Abbildung 6.14) erlaubt die Konfiguration der einzelnen Komponenten und Komponentenengruppen und im *User Manager* können die Nutzerrechte vergeben werden. Zusätzlich können über die Schaltfläche *Edit Globals* Umgebungsvariablen definiert werden, die während des Zugriffs auf den einzelnen Rechnern gesetzt werden. Alle Konfigurationsdaten legt das Plugin nutzerspezifisch in der Trac-Datenbank ab, so dass jeder Nutzer bei Bedarf seine eigene Konfiguration vorhalten kann.

In Tabelle 6.3 sind die Parameter zur Konfiguration der Rechner-Verbindungsdaten aufgelistet. Da die Verbindung zwischen dem Webserver der Integrations- und Testplattform und den einzelnen Rechnern des Zielsystems über ssh erfolgt, müssen entsprechende Account-Daten eingetragen werden. Die ssh-Portadresse kann modifiziert werden, falls die Standardadresse nicht gewünscht ist. Der Host-Manager Oberfläche ist aufgrund der sensiblen Konfigurationsdaten nur Nutzern mit den HOST_ADMIN Rechten zugänglich.

In Tabelle 6.4 sind die Konfigurationsmöglichkeiten des Komponenten-Managers dargestellt. Neben der Konfiguration des Hauptprozesses einer Komponente besteht die Möglichkeit, eine beliebige Anzahl von Begleitprozessen (im Robotcontrol-Plugin *Aktionen* genannt) ein-

Tabelle 6.3: Konfigurationsparameter im Host Manager

Parameter	**Beispielwert**
Betriebssystem	Linux
IP-Adresse	10.0.1.113
Account Name	webportal
Account Passwort	secret
SSH-Port	22

Tabelle 6.4: Konfigurationsparameter im Komponenten Manager

Parameter	**Beispielwert**
Name	drivercamera
Host	10.0.1.113
Parent	SampleApp
Description	Camera driver for stereo cameras
URL	http://www.ros.org/wiki/cob_camera_sensors
StartCommands	/opt/webportal/drivercamera/start.sh
StopCommands	/opt/webportal/drivercamera/stop.sh

zurichten. So kann z.B. das Starten eines VNC Servers als Aktion eingerichtet werden. Unter den einzelnen Aktionen können Abhängigkeiten definiert werden, so dass z.B. eine Aktion erst dann ausgeführt wird, wenn alle abhängigen Aktionen bereits laufen. Die Konfiguration der Aktionen erfolgt ebenfalls im Komponenten-Manager, wobei alle Parameter außer der Rechneradresse zur Verfügung stehen (Begleitprozesse müssen auf demselben Rechner wie der Hauptprozess der Komponente angesiedelt sein).

6.3.2 Das HardwareMonitor-Plugin

Anforderung A38 beschreibt die Notwendigkeit, bestimmte Informationen über den Zustand des Zielsystems am entfernten Standort des testenden Entwicklers zur Verfügung zu haben. Bei Komponenten-Abhängigkeiten zu Hardware-Komponenten (z.B. Arm, mobile Basis, etc.) muss zum Beispiel sichergestellt sein, dass diese Komponenten betriebsbereit und sich in einem einwandfreien Zustand befinden. Zur Erfüllung dieser Anforderungen wurde das HardwareMonitor-Plugin (s. Abbildung 6.15) entwickelt, das in der Lage ist, in einem vorgegebenen Takt relevante physikalische Größen wie die Spannungsversorgung, ggf. unterteilt in Logik- und Motorspannungen, aktuelle Stromstärken, die Temperatur sowie Minimal- und Maximalwerte dieser Größen zu messen und über den Webserver bereitzustellen.

Es greift auf eine speziell entwickelte mikrocontrollerbasierte Hardwaremonitor-Platine [GÏ0] zurück, die Spannungen, Ströme und Temperaturen in konfigurierbaren Abständen regelmäßig misst. Diese analogen Messwerte werden im Mikrocontroller in digitale Signale gewandelt und für die Visualisierung entsprechend aufbereitet. Leitungen, für die Spannung

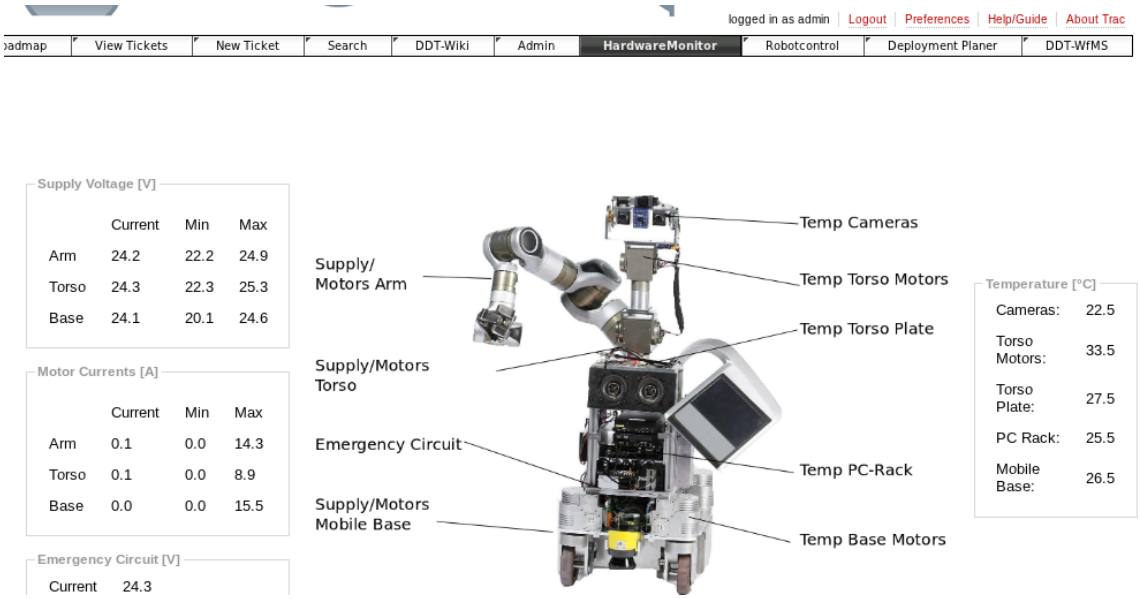


Abbildung 6.15: Hardware Monitor Plugin (Konfiguration für Care-O-bot 3)

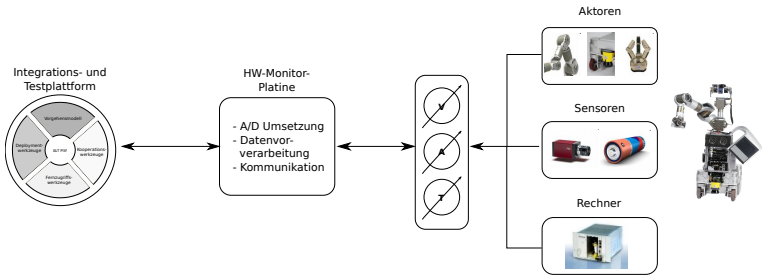


Abbildung 6.16: Konzept Hardware Monitor zur Überwachung von physikalischen Größen (z.B. Spannung, Temperatur) des Zielsystems

bzw. Strom gemessen werden sollen (z.B. Spannungsversorgungsleitungen, Motorzuleitungen, etc.) werden dabei durch entsprechende Ein- und Ausgänge der Platine durchgeschleift. Die Temperatursensoren können an kritischen Punkten im System verteilt und an die entsprechenden Eingänge der Platine angeschlossen werden. Die Platine ist durch einen USB-Bus an einen Rechner des Zielsystems angeschlossen. Das HardwareMonitor Plugin verbindet sich auf den entsprechenden Rechner, liest die in der Platine vorgehaltenen Werte aus und überträgt sie zur Darstellung an den Webserver. Zielrechner, Messabstände, etc. können über das Plugin konfiguriert werden. Das Blockschaltbild in Abbildung 6.16 veranschaulicht die Interaktion mit der Integrations- und Testplattform.

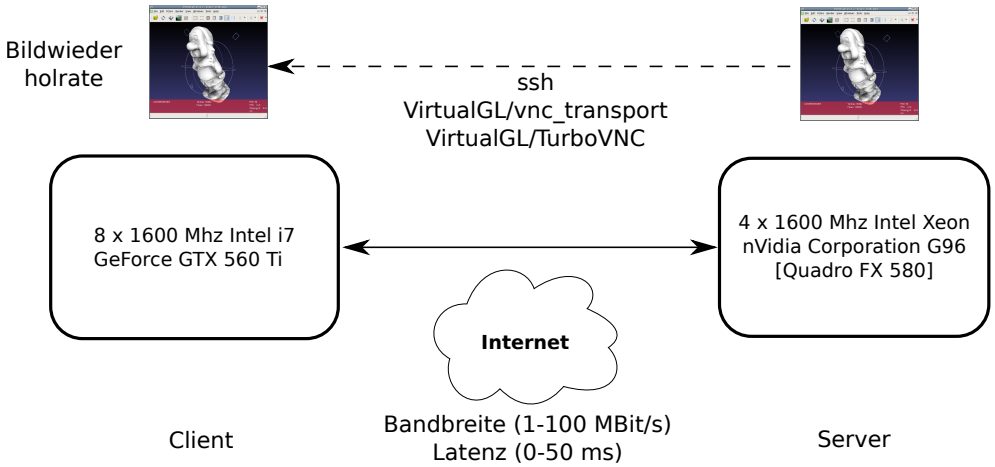


Abbildung 6.17: Aufbau zur Messung der Bildwiederholrate bei den Übertragungstechnologien ssh, vgl_transport und TurboVNC.

6.4 Auswahl eines geeigneten Übertragungsprotokolls für 3D-Applikationen

Die Übertragung von 3D-Oberflächen stellt die höchsten Anforderungen an die Effizienz der eingesetzten Remote Session Werkzeuge. Deshalb wird im Folgenden untersucht, welches Protokoll für welche Randbedingungen hinsichtlich des verfügbaren Übertragungskanals geeignet ist. Dazu wird eine Messung der effektiven Bildwiederholrate einer 3D-Anwendung für unterschiedliche Ressourcenverfügbarkeiten (Bandbreite, Latenz) der zugrundeliegenden Übertragungskanäle durchgeführt. Der Messaufbau ist in Abbildung 6.17 veranschaulicht. Als Beispielanwendung wird Meshlab [IST] verwendet, da mit dieser 3D-Applikation die Rendergeschwindigkeit einer unterschiedlichen Anzahl von Polygonen gemessen und unterschiedliche Interaktionsmöglichkeiten des Nutzers mit der Oberfläche getestet werden können. Zur Messung der Bildwiederholrate der 3D-Anwendung wurde das Programm “tcbench” verwendet, das in der VirtualGL-Suite mitgeliefert wird [Virb].

Grundsätzlich in Frage kommen zur Übertragung der 3D-Daten ssh, eine Kombination von VNC-basierten Diensten und VirtualGL (vgl. Abschnitt 4.4.4), oder das VirtualGL vgl_transport Protokoll [Virc]. Bei ssh erfolgt sowohl das 2D als auch 3D Rendering auf der Grafikkarte des Client-Rechners, so dass alle Renderingbefehle über das Netzwerk übertragen werden müssen. Bei VirtualGL erfolgt das Rendering auf der Grafikkarte des Servers, so dass lediglich gerenderte Bilder und die Daten der Nutzerinteraktion (z.B. Mausklicks) über das Netzwerk übertragen werden müssen. Im Unterschied zu TurboVNC erfolgt beim vgl_transport das 2D Rendering auf dem Client-Rechner, so dass alle 2D Rendering (X11) Befehle auf den Client übertragen werden müssen. Dies hat einerseits eine Entlastung der CPU des Servers zur Folge, andererseits wirkt sich die notwendige Übertragung des

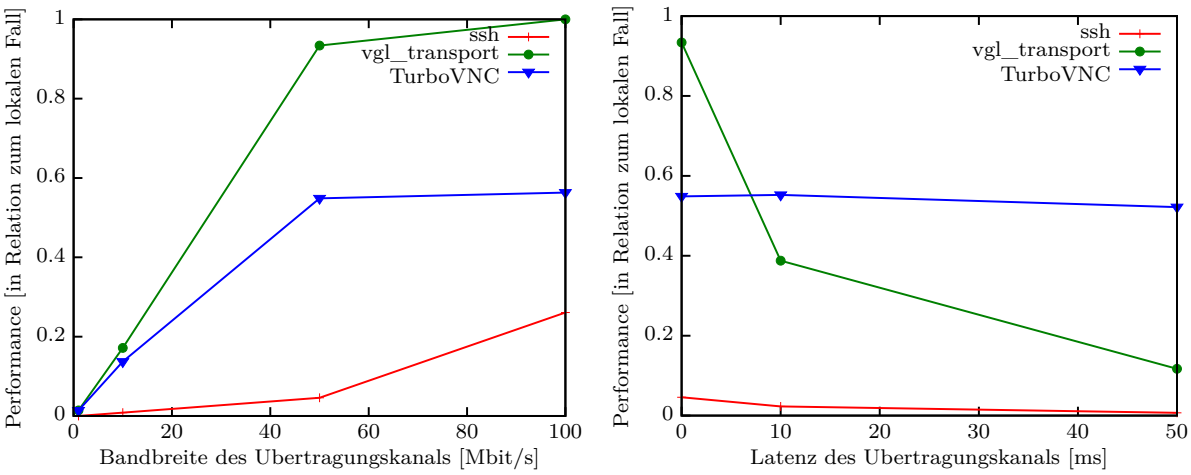


Abbildung 6.18: Bildwiederholraten bei der Netzwerkübertragung der 3D-Anwendung Meshlab bezogen auf die Bildwiederholrate im lokalen Fall. Dabei wird die Leistung von OpenSSH, TurboVNC und vgl_transport miteinander verglichen. Auf der linken Seite wir die Leistung bei unterschiedlichen Netzwerkbandbreiten untersucht (Latenz 0ms), auf der rechten Seite wird die Netzwerklatenz variiert (Bandbreite 50 Mbit/s). Es wurde jeweils die JPEG Kompression 95 verwendet (Die absolute Bildwiederholrate im lokalen Fall beträgt 47 Frames/sec.

X11-Protokolls bei Netzwerken mit hoher Latenz negativ auf die Bildwiederholrate aus. Bei vgl_transport wird nicht der komplette Desktop übertragen, da jede Applikation in einem separaten Fenster dargestellt werden kann. Sowohl bei TurboVNC als auch bei vgl_transport besteht die Möglichkeit, die gerenderten Bilder vor der Übertragung (mit unterschiedlichen Kompressionsraten) zu komprimieren. Dadurch wird zwar weniger Bandbreite für die Übertragung an den Client benötigt, auf der Serverseite jedoch zusätzliche Zeit für die Komprimierung beansprucht, so dass die maximal übertragbare Bildrate reduziert wird. Für eine konkrete Server/Netzwerk-Kombination kann die Komprimierungsoption also je nach Ressourcenverfügbarkeit (schneller Server-Rechner, geringe Netzwerkbandbreite → Komprimierung, langsamer Server-Rechner, große Netzwerkbandbreite → ohne Komprimierung) gewählt werden.

Zur Gegenüberstellung werden die Messungen konkret für OpenSSH [Ope99], TurboVNC 1.0 (in Verbindung mit VirtualGL 2.2) [Vira] und VirtualGL 2.2/vgl_transport durchgeführt. Abbildung 6.18 veranschaulicht den Leistungsvergleich der verschiedenen Übertragungsprotokolle bei unterschiedlichen Netzwerkressourcen. Erwartungsgemäß liegen die Bildwiederholraten bei der Übertragung über OpenSSH deutlich unter TurboVNC und vgl_transport, da bei OpenSSH nicht komprimierte Bilder sondern alle Renderbefehle übertragen werden. Selbst wenn die Grafikkarte des Clients viel schneller als die des Servers sein sollte, ist mit der Applikation bei geringen Bandbreiten durch die hohe Verzögerung durch das Netzwerk auf dem Client praktisch keine Nutzerinteraktion mehr möglich. Während

Bedienungstests ist eine Wiederholrate von 5 Frames/sec als Toleranzschwelle für die Bedienbarkeit ermittelt worden, was in der Abbildung etwa 10% der Leistung im lokalen Fall entspricht.

Nach den vorliegenden Messungen liegt die Bildwiederholrate für OpenSSH bei einer Bandbreite von 50 Mbit/s schon deutlich unter 5 Frames/sec. Bei sehr geringen Latenzzeiten und hohen Netzwerkbandbreiten liegen die Bildwiederholraten für vgl_transport über denen von TurboVNC. Bei Latenzzeiten von über 20ms knickt die Leistung von vgl_transport aber erwartungsgemäß deutlich ein, da die 2D Renderbefehle mit übertragen werden müssen. Unter typischen Netzwerkbedingungen des Internets (Bandbreite: 10 - 50 MBit/s, Latenz: 50 - 100 ms) weist die Kombination VirtualGL/TurboVNC somit im Vergleich die höchste Bildwiederholrate auf. Die Verwendung von OpenSSH für Ferntests ist nur für sehr hochperformante Netzwerke ratsam oder für 2D-Applikationen, für die keine hohe Bildwiederholrate erforderlich ist.

Standardmäßig verwendet das Laufzeitwerkzeug deshalb TurboVNC zur Übertragung von 3D-Anwendungen. Der TurboVNC-Client kann als Applet in die Webseite des Robotcontrol-Plugins eingebunden werden (vgl. Abschnitt 6.3.1). Grundsätzlich können die Werkzeuge der Integrations- und Testplattform jedoch auch mit dem vgl_transport Protokoll verwendet werden, falls ein Netzwerk mit geringer Latenz vorliegt. Ssh wird zur Übertragung von 2D-Anwendungen und sonstigen Daten verwendet.

7 Auswertung

Zur Bewertung der entwickelten Integrations- und Testplattform wird zunächst die Umsetzung aller Anforderungen überprüft. Danach folgt die Evaluation der umgesetzten Werkzeuge im Rahmen eines großen Verbund-Forschungsprojekts.

7.1 Abgleich mit den Anforderungen

Im folgenden Abschnitt werden die umgesetzten Werkzeuge den Anforderungen aus Kapitel 3 gegenübergestellt und hinsichtlich deren Erfüllungsgrad bewertet.

7.1.1 Zielerfüllung der allgemeinen Anforderungen

Bei der Auswahl existierender Werkzeuge zur Integration wurde auf Plattformunabhängigkeit geachtet sowie die Existenz von Varianten für mehrere Betriebssysteme. Die in Kapitel 6 beschriebene Umsetzung ist vollständig kompatibel zu Linux, kann jedoch einfach durch die Verwendung der entsprechenden Varianten auf andere Betriebssysteme erweitert werden. Die neuen Werkzeuge in Form der Trac-Plugins wurden in Python und Webprogrammiersprachen implementiert, so dass auch hier eine breite Unterstützung verschiedener PC-basierter Betriebssysteme gewährleistet ist. Für die Laufzeitunterstützung und die Deployment-Werkzeuge wurden Remote Session Werkzeuge eingesetzt, so dass die Einsetzbarkeit für ein verteiltes System gewährleistet ist. Die harte Forderung der Unabhängigkeit aller Werkzeuge von einer bestimmten Middleware und einem bestimmten Komponentenmodell wurde strikt durchgehalten, so dass die Werkzeuge für sehr viele in der Robotik genutzte Frameworks eingesetzt werden können. Die werkzeuggestützte Erstellung von Integrationscode und die Unterstützung der Applikationsentwicklung ist ohne Festlegung auf ein bestimmtes Komponentenmodell jedoch nicht möglich. Eine Spezialisierung der Werkzeuge auf bestimmte Frameworks und spezifische Komponentenmodelle kann auf der middlewareunabhängigen Basis sehr einfach aufgesetzt werden, so dass weitergehende Unterstützungsfunktionen hinsichtlich Applikationsentwicklung und Generierung von Integrationscode implementiert werden können. Die Verwendung der Kollaborationsplattform

Tabelle 7.1: Erfüllung der allgemeinen Anforderungen

Anforderung		erfüllt
A1:	Einsetzbarkeit der Werkzeuge für eine verteilte Rechnerarchitektur	•
A2:	Einsetzbarkeit der Werkzeuge für verschiedene Betriebssysteme	•
A3:	Einsatzbarkeit der Werkzeuge für eine Vielzahl von unterschiedlichen Sensor- und Aktortypen	■
A4:	Reduktion des notwendigen Wissens über die Hardwarearchitektur des Gesamtsystems	■
A13:	Gewährleistung eines jederzeit operablen Systemzustandes	■
A19:	Alle entwickelten Werkzeuge sollen in einer integrierten Umgebung zentral verfügbar sein	•
A24:	Die Werkzeuge sollen die Rollentrennung für die verschiedenen Integrations- und Testaktivitäten unterstützen	■
A30:	Entkopplung der Abhängigkeiten zwischen den Entwicklern bei Tests	■
A36:	Unabhängigkeit aller Werkzeuge von Middleware und Software-Framework	■
A37:	Keine Voraussetzung eines bestimmten Komponentenmodells	•
A39:	Einfache Konfiguration der Werkzeuge aus der Ferne	■

Legende:

■ erfüllt durch eigene Implementierung
• erfüllt durch Integration des Stands der Technik
○ erfüllbar durch Stand der Technik, jedoch nicht integriert
– nicht erfüllt

Trac bietet den zentralisierten Zugriff aller Werzeuge der Integrations- und Testplattform, wobei eine Integration weiterer Werkzeuge durch die plugin-basierte Struktur sehr einfach möglich ist. Durch die Vergabe bestimmter Zugriffsrechte des Laufzeitwerkzeugs, die für die Durchführung bestimmter Aktionen erforderlich sind, kann die Rollentrennung durch die Integrations- und Testplattform implizit forciert werden. Tabelle 7.1 veranschaulicht die Erfüllung der allgemeinen Anforderungen an die Integrations- und Testplattform.

7.1.2 Zielerfüllung des Vorgehensmodells

Das DDT Vorgehensmodell setzt auf dem OpenUP auf, der im Gegensatz zu anderen verbreiteten Modellen für verteilte Projektteams und Forschungsprojekte geeignet ist (vgl. Abschnitt 4.2.4). Durch die Auftrennung der Entwicklungsphase in Komponenten- und Applikationsinkremente wird methodisch die Erfüllung der Anforderungen bezüglich Deployment und Qualitätssicherung (A12) angelegt (vgl. Abschnitt 5.1.4). Durch die Aufteilung von Komponenten- und Applikationsentwicklung wird die Abhängigkeit der verschiedenen Akteure so weit wie möglich durch den Prozess reduziert (A30). Weiterhin kann durch die formale Abnahme der Komponenten einfacher ein allzeit operabler Systemzustand ge-

Tabelle 7.2: Erfüllung der Anforderungen an das Vorgehensmodell

Anforderung		erfüllt
A12:	Verankerung der Qualitätssicherung im Entwicklungsprozess	■
A14:	Automatisierte Durchführung von Komponenten- und Integrationstests	○
A18:	Automatisierte Durchführung von Applikationstests	–
A20:	Anwendbarkeit des Vorgehensmodells für verteilte Projektteams	●
A21:	Anwendbarkeit des Vorgehensmodells für hochgradig innovative Projekte mit wenig Erfahrungswerten	●
A22:	Werkzeuggestützte Durchführung des Integrationsprozesses im Rahmen des Vorgehensmodells (Workflow-Management)	○
A23:	Werkzeuggestützte Dokumentation erfolgter Komponenten und Applikationstests	○
A32:	Werkzeuggestützte Dokumentierung von Komponentenparametern und deren Semantik	○

Legende:

■ erfüllt durch eigene Implementierung
● erfüllt durch Integration des Stands der Technik
○ erfüllbar durch Stand der Technik, jedoch nicht integriert
– nicht erfüllbar durch aktuellen Stand der Technik

währleistet werden (A13). Das Deployment der Komponenten wird vom Systemintegrator übernommen, so dass die einzelnen Komponentenentwickler keine tiefgehenden Kenntnisse des Zielsystems mehr benötigen (A4). Die Durchführung von Integrations- und Komponententests lässt z.B. durch das CI-Plugin Bitten (vgl. Abschnitt 4.1) automatisieren, das auch eine entsprechende Testdokumentation generiert und in der zentralen SQL-Datenbank ablegt. Die automatisierte Durchführung von Applikationstests ist im Bereich Servicerobotik aufgrund der komplexen, unstrukturierten und nicht abgeschlossenen Umgebungsbedingungen noch Gegenstand der Forschung. Tabelle 7.2 fasst den Abgleich mit den Anforderungen zusammen.

7.1.3 Zielerfüllung der Deployment-Werkzeuge

Die Deployment-Werkzeugkette ermöglicht die Deployment-Planung und Installation von Komponenten und Applikationen auf einem Serviceroboter mit einem verteilten Rechnersystem unter Berücksichtigung der Hardwarearchitektur bestehend aus den verschiedenen Aktorik- und Sensorikkomponenten. Auch hier hat die Rollentrennung eine große Bedeutung - der Systemintegrator, der Detailkenntnisse über das Zielsystem besitzt, kann mit dem Deployment-Planer ein Modell der Hardwarearchitektur erstellen (A6). Der Applikationsentwickler hat wiederum die Möglichkeit, seinerseits ein Modell der Applikation zu erstellen, das die Anforderungen an die Hardware reflektiert (A8). Mit diesen Modellen ist der Appli-

Tabelle 7.3: Erfüllung der Anforderungen an die Deployment-Werkzeuge

Anforderung		erfüllt
A6:	Formale Spezifikation der Hardewarearchitektur des Zielsystems	■
A8:	Formale Spezifikation der Hardwareanforderungen der Komponenten und der Komponenten-Architektur für die Applikation	■
A9:	Werkzeuggestützte Installation von Komponenten aus dem Repository	■
A10:	Werkzeuggestützte Aktualisierung einzelner Komponenten unter Berücksichtigung von Abhängigkeiten	•
A35:	Werkzeuggestütztes Laden von bestimmten Softwareständen (Releases von Komponenten und Applikationen)	○

Legende:

■ erfüllt durch eigene Implementierung
• erfüllt durch Integration des Stands der Technik
○ erfüllbar durch Stand der Technik, jedoch nicht integriert
− nicht erfüllt

kationsentwickler in der Lage, alle Software-Komponenten seiner Applikation auf entsprechende Rechner des Zielsystems so zuzuordnen, dass sowohl die Abhängigkeiten zwischen den Software-Komponenten als auch die Abhängigkeiten zu Hardware-Komponenten erfüllt werden (sofern eine Lösung existiert). Diese Aktion kann vom Applikationsentwickler ebenso wie die nachfolgende Installation ohne detaillierte Kenntnisse der Hardwarearchitektur (A4) und der Software-Komponenten (A30) durchgeführt werden. Die Umsetzung ermöglicht eine Aktualisierung aller Komponenten über das Paketmanagement-System. Für das Laden bestimmter Softwarestände bzw. Releases unter Erhaltung aller Versionsabhängigkeiten bietet sich die Entwicklung eines weiteren Plugins an, die auf dem Paketmanagement-System aufsetzen kann. Alle Werkzeuge der Ketten gehen nahtlos ineinander über, so dass keine manuellen Zwischenoperationen erforderlich sind. In Tabelle 7.3 ist der Abgleich mit den Anforderungen dargestellt.

7.1.4 Zielerfüllung der Laufzeitwerkzeuge

Das Robotcontrol-Plugin als Kernstück der Integrations- und Testplattform setzt die Anforderungen bezüglich der Laufzeitunterstützung um (s. Tabelle 7.4). Die Konfiguration und Ausführung der einzelnen Komponten bzw. der Gesamtapplikation kann von jeder Rolle durch das Robotcontrol-Plugin durchgeführt werden, so dass die Bedienbarkeit des komplexen Systems wesentlich vereinfacht ist (A13). Das Reservierungssystem gewährleistet, dass keine Konflikte durch Parallelzugriff auf das Zielsystem enstehen (A34), so dass ein einzelner Entwickler das Gesamtsystem allein nutzen kann (A25). Andere (ggf. an anderen Standorten sitzende) Entwickler haben jedoch die Möglichkeit, z.B. im Rahmen eines Integrationstests

Tabelle 7.4: Erfüllung der Anforderungen an die Laufzeitunterstützung

Anforderung		erfüllt
A11:	Werkzeuggestützte Inbetriebnahme/Aktivierung einzelner Komponenten auf dem Zielsystem	■
A17:	Werkzeuggestützte Inbetriebnahme von Serviceroboter-Applikationen	■
A25:	Ermöglichung der Einzelarbeit am integrierten Gesamtsystem	•
A31:	Einheitliche Konfigurierbarkeit aller Komponenten	■
A33:	Ermöglichung des Fernzugriffs (anytime, anywhere) auf die Software- und ggf. Hardware-Komponenten des Roboters für die Durchführung von verteilten Tests.	■
A34:	Zugriffsmanagement auf die Roboterhardware	■
A38:	Fernüberwachung und -diagnose von Hardware-Komponenten	■

Legende:

■ erfüllt durch eigene Implementierung
• erfüllt durch Integration des Stands der Technik
○ erfüllbar durch Stand der Technik, jedoch nicht integriert
– nicht erfüllt

die Aktionen des aktiven Entwicklers z.B. über das Log-Fenster oder die VNC-Übertragung zu verfolgen. Auch bei höheren Latenzzeiten, die für den Übertragungskanal Internet typisch sind, hat ein Entwickler eine leistungsfähige Ferntestumgebung für Komponenten- und Integrationstests. Für Applikationstests, bei denen Roboterbewegungen erforderlich sind, sollte selbstverständlich ein Testbegleiter vor Ort sein. Das Hardwaremonitor-Plugin bietet darüber hinaus die Überwachung der Hardware-Komponenten auf Schaltkreisebene an, so dass kritische Spannungswerte (z.B. Notauskreis) oder Maximalströme überwacht werden können (A38).

7.1.5 Zielerfüllung der Kooperationswerkzeuge

Diese Komponente des Gesamtkonzepts erfüllt die Anforderungen zur effizienten Kooperation in verteilten Entwicklungsteams: eine leistungsfähige Kommunikationsinfrastruktur, Kollaborationswerkzeuge sowie Wissensmanagement und Dokumentation (s. Tabelle 7.5). Alle Anforderungen konnten durch die Integration existierender Trac-Plugins erfüllt werden: den Trac Repository Browser (Unterstützung für Subversion und Git, weitere Plugins für Bazaar und Mercurial) (A28), das Roadmap-Plugin (A29), das Trac Ticket Plugin (A15) und das Trac-interne Wiki (A26, A7).

Tabelle 7.5: Erfüllung der Anforderungen an die Kooperationswerkzeuge

Anforderung		erfüllt
A7:	Zentrale Verfügbarkeit der Dokumentation für die Integration von Komponenten in das Zielsystem	•
A15:	Zentrale Fehlerverwaltung zur räumlichen und zeitlichen Entkopplung der Integrationsarbeit	•
A26:	Werkzeuggestützte zentrale Organisation des Wissensmanagements	•
A27:	Bereitstellung einer geeigneten Infrastruktur für eine effiziente Kommunikation im verteilten Team	•
A28:	Unterstützung der gemeinsamen Arbeit an Dokumenten und Quellcode durch Kooperationswerkzeuge	•
A29:	Werkzeuggestützte Koordination des verteilten Entwicklungsteams	•

Legende:

■ erfüllt durch eigene Implementierung
• erfüllt durch Integration des Stands der Technik
○ erfüllbar durch Stand der Technik, jedoch nicht integriert
– nicht erfüllt

7.2 Evaluation des Webportals im Kontext eines Forschungsprojekts

Das Webportal wurde im Rahmen des Verbundforschungsprojektes DESIRE [DES09] (s. Anhang 10) über mehr als sechs Monate hinweg als Plattform für die Integration und den verteilten Test genutzt. Im Folgenden wird zunächst die Methodik zur Messung und Bewertung der Auswirkungen des Webportaleinsatzes auf die Projektarbeit dargestellt. Anschließend wird die Evaluation auf Basis dieser Methodik und der im Projekt gesammelten Messdaten präsentiert.

Evaluationsprinzip

In diesem Abschnitt wird die Auswirkung der Webportalnutzung auf die Lokalität und die Effizienz der geleisteten Integrationsarbeit analysiert. Als grundlegende Messgrößen der Integrationsarbeit dienen dabei zum einen Anzahl und Teilnehmerstärke der Integrationstreffen und zum anderen die Nutzungscharakteristik des gemeinsamen Projektrepositories.

Zur Integrationsarbeit werden dabei die Anpassung der eigenen Komponenten an die spezifizierten DESIRE-Datentypen und die DESIRE-Architektur, die Entwicklung von Integrationscode (“glue code”) (z.B. Umrechnung von Koordinatensystemen der Zielhardware), sowie systemweite Integrationstests gerechnet. Die Ergebnisse dieser Arbeiten wurden dann in das gemeinsame Repository übertragen (eingecheckt) - im Gegensatz zur Entwicklung

der Basisfunktionalität der einzelnen Komponenten, die meist in eigenen lokalen Softwarerepositories organisiert war. Deshalb werden der Umfang der Integrationstreffen sowie die Anzahl der Eincheckvorgänge in das gemeinsame Projektrepository im Folgenden als Maß für die in DESIRE geleistete Integrationsarbeit verwendet.

Der Gesamtaufwand der Integration wird nach der "NDI Software Integration" Kostenschätzung [AB97] (vgl. Abschnitt 2.2.3) berechnet, da in DESIRE bereits existierende (NDI) Komponenten der Partner in das Projekt gebracht wurden und zu integrieren waren. Da der Quellcode der Komponenten den Entwicklern vorliegt, ist nicht das COCOTS Modell zu verwenden. Die Schätzung der Integrationsarbeit in Personenmonaten PM_{NDI} erfolgt nach Abts et al. [AB97] durch die Formel :

$$PM_{\mathrm{NDI}} = 2{,}94 \cdot Size^{K} \cdot M, \tag{7.1}$$

$$Size = (KNSLOC + KASLOC \cdot AAM) \cdot \left[1 + \frac{BRAK}{100}\right], \tag{7.2}$$

$$K = 0{,}91 + 0{,}01 \cdot \sum_{j=1}^{5} SF_j, \tag{7.3}$$

$$M = \prod_{i=1}^{17} EM_i, \tag{7.4}$$

$$AAF = 0{,}4 \cdot DM + 0{,}3 \cdot CM + 0{,}3 \cdot IM \tag{7.5}$$

$$AAM = \begin{cases} \frac{AAF \cdot (1+0{,}02 \cdot SU \cdot UNFM)}{100} & AAF \leq 50 \\ \frac{AAF + SU \cdot UNFM}{100} & AAF > 50 \end{cases} \tag{7.6}$$

Dabei bezeichnet *Size* die Größe der zu entwickelnden Software bzw. Applikation, der Exponent K Skalierungsfaktoren für die Komplexität des Projekts, M einen Faktor, der Produkt-, Personal-, Hardware- und Projektattribute berücksichtigt, AAF den Modifikationsaufwand für eine Komponente und AAM einen Code-Anpassungsmultiplikator. Die übrigen Parameter sind in den Tabellen 7.6, 7.7 und 7.8 erläutert.

Im COCOMO II Modellierungs-Handbuch [BCC+00] sind Werte für die Konstanten und Einstufungsklassen der Parameter enthalten, die mit Hilfe von Daten aus über 160 Projekten kalibriert wurden. Die konkreten Werte der Parameter aus dem DESIRE Projekt sind in Abschnitt 10 gegeben, so dass durch Einsetzen dieser Werte für den geschätzten Aufwand für die Integration der DESIRE Komponenten gilt:

$$PM_{\mathrm{NDI}} = 237$$

Tabelle 7.6: Beschreibung der Parameter für die Kostenschätzungsgleichungen 7.1 - 7.6.

Symbol	**Beschreibung**
BRAK	Anteil des Integrationscodes für die Komponente, der aufgrund von Anforderungsänderungen neugeschrieben werden muss
CM	Anteil des zu verändernden Komponenten-Codes
DM	Anteil des zu verändernden Designs
IM	Anteil des zu verändernden Gesamttest- und Integrationscodes durch die Integration der neuen Komponente
KASLOC	Größe der zu integrierenden Komponente in Tausenden von Quellcodezeilen
KNSLOC	erforderlicher Integrationscode ("glue code") für die Integration der Komponente in Tausenden von Quellcodezeilen
SU	erforderliches Verständnis der zu integrierenden Komponente ($SU = 0$, falls $CM = 0$ und $DM = 0$)
UNFM	Erfahrung des Programmierers mit der zu integrierenden Komponenten

Tabelle 7.7: Skalierungsfaktoren des COCOMO II Modells [BCH+95]. Bewertet werden diese in einer 6-Punkte Skala von jeweils sehr niedrig bis extrem hoch.

SF_j	**Formelz.**	**Bedeutung**
SF_1	*PREC*	Erfahrungsgrad einer Organisation mit dem zu schätzenden Projekttyp. Die Skala reicht von *sehr niedrig* - keine Erfahrung - bis *sehr hoch* - vollständig vertraut mit der Anwendungsdomäne (Precedentness).
SF_2	*FLEX*	Vom Auftraggeber zugestandene Flexibilität bezüglich des Entwicklungsprozesses (Development Flexibility).
SF_3	*RESL*	Ausmaß der durchgeführten Risikoanalyse und Architekturkonzeption (Architecture/risk resolution).
SF_4	*TEAM*	Gibt an, wie gut die einzelnen Mitglieder der Projekt-Akteure (Auftraggeber, Entwickler, Nutzer, etc.) kommunizieren und zusammenarbeiten. Die Bandbreite reicht von einem Team mit schwieriger Zusammenarbeit bis hin zum perfekt integrierten Team ohne Kommunikationsprobleme (Team cohesion).
SF_5	*PMAT*	Reglementierungsgrad der Prozesse der Organisation basierend auf der CMM Skala [PCCW95] (Process maturity).

Tabelle 7.8: Kostentreiberfaktoren des COCOMO II Modells [BCH+95]. Bewertet werden diese in einer 6-Punkte Skala von jeweils sehr niedrig bis extrem hoch.

EM_i	**Formelz.**	**Typ**	**Bedeutung**
EM_1	*RELY*	Produkt	Erforderliche Zuverlässigkeit des Gesamtsystems
EM_2	*CPLX*	Produkt	Komplexität der Systemmodule
EM_3	*DOCU*	Produkt	Erforderlicher Dokumentationsgrad
EM_4	*DATA*	Produkt	Größe der verwendeten Datenbank
EM_5	*RUSE*	Produkt	Erforderlicher Anteil von wiederverwendbaren Komponenten
EM_6	*TIME*	Entwickl.plattf.	Randbedingungen hinsichtlich Ausführungszeit
EM_7	*PVOL*	Entwickl.plattf.	Volatilität der Entwicklungsplattform
EM_8	*STOR*	Entwickl.plattf.	Speichereinschränkungen
EM_9	*ACAP*	Personal	Leistungsfähigkeit der System-Architekten
EM_{10}	*PCON*	Personal	Kontinuität des Personals
EM_{11}	*PCAP*	Personal	Leistungsfähigkeit der Programmierer
EM_{12}	*APEX*	Personal	Vorerfahrung mit der Applikation
EM_{13}	*PLEX*	Personal	Vorerfahrung mit der Zielplattform / in der Zieldomäne
EM_{14}	*LTEX*	Personal	Erfahrung des Personals mit den einzusetzenden Werkzeugen und Programmiersprachen
EM_{15}	*TOOL*	Projekt	Verwendung von Software-Werkzeugen
EM_{16}	*SCED*	Projekt	Zeitliche Rahmenbedingungen
EM_{17}	*SITE*	Projekt	Verteilungsgrad des Entwicklungsteams und verfügbare Kommunikationswerkzeuge

Dabei lässt sich die gesamte Integrationsarbeit W_{I} aufteilen in einen lokalen Arbeitsanteil $W_{\mathrm{I,l}}$, in dem die Integrationsarbeit direkt am Zielsystem stattfindet und einen entfernten Arbeitsanteil $W_{\mathrm{I,r}}$, in dem die Integrationsarbeit räumlich getrennt vom Zielsystem stattfindet. Es gilt also:

$$W_{\mathrm{I}} = W_{\mathrm{I,l}} + W_{\mathrm{I,r}}. \tag{7.7}$$

Der lokale Anteil der Integrationsarbeit (in PM) lässt sich folgendermaßen bestimmen:

$$W_{\mathrm{I,l}}(t) = \frac{MT_{\mathrm{I,l}}(t) - V_{\mathrm{I,l}}(t)}{d(t)} \tag{7.8}$$

$$MT_{\mathrm{I,l}}(t) = \sum_{m_i \in M_{\mathrm{it}}(t)} n_{\mathrm{it}}(m_i) \cdot d(m_i) \tag{7.9}$$

$$V_{\mathrm{I,l}}(t) = \bar{f_v} \cdot MT_{\mathrm{I,l}}(t). \tag{7.10}$$

Dabei bezeichnet $M_{\mathrm{it}}(t) = \{m_1, \ldots, m_n(t)\}$ die Menge der Integrationstreffen während eines Projektmonats t, $n_{\mathrm{it}}(m_i)$ die Anzahl von Teilnehmern an einem Integrationstreffen m_i, $d(m_i)$ die Dauer eines Integrationstreffens m_i in Tagen und $MT_{\mathrm{I,l}}(t)$ die reine Anwesenheitszeit der Projektpartner während der Integrationstreffen in Manntagen. Der Verlustterm $V_{\mathrm{I,l}}$ berücksichtigt, dass die gezählten Anwesenheitsstunden während der Integrationstreffen nicht vollständig in die tatsächlich erfolgte lokale Integrationsarbeit übergehen, sondern teilweise aus Essenspausen, Privatgesprächen und sonstigen Ablenkungen durch die Teamarbeit fließen. Der durchschnittliche Verlustfaktor $\bar{f_v}$ spiegelt somit die Effizienz der Integrationstreffen wieder. Für die entfernt geleistete Integrationsarbeit sind Teamverluste hingegen in der Regel vernachlässigbar, da davon ausgegangen werden kann, dass die Arbeit einzeln am Arbeitsplatz erfolgt. Der Umrechnungsfaktor $d(t)$ bezeichnet die zugrundegelegte Anzahl von Arbeitstagen pro Monat.

Geht man davon aus, dass die geleistete Integrationsarbeit proportional zu der Gesamtanzahl der Eincheckvorgänge $n_{\mathrm{c}}(t)$ in das gemeinsame Repository innerhalb des Projektmonats t ist, lässt sich der zeitliche Ablauf der gesamten Integrationsarbeit $W_I(t)$ in Projektmonat t definieren:

$$W_{\mathrm{I}}(t) = k_{\mathrm{c}} \cdot n_{\mathrm{c}}(t), \text{ wobei} \tag{7.11}$$

$$k_{\mathrm{c}} = \frac{PM_{\mathrm{NDI}}}{\sum_0^{PL} n_{\mathrm{c}}(t)}. \tag{7.12}$$

Dabei bezeichnet k_{c} den Proportionalitätsfaktor, PL die Gesamtlaufzeit des Projekts in Monaten.

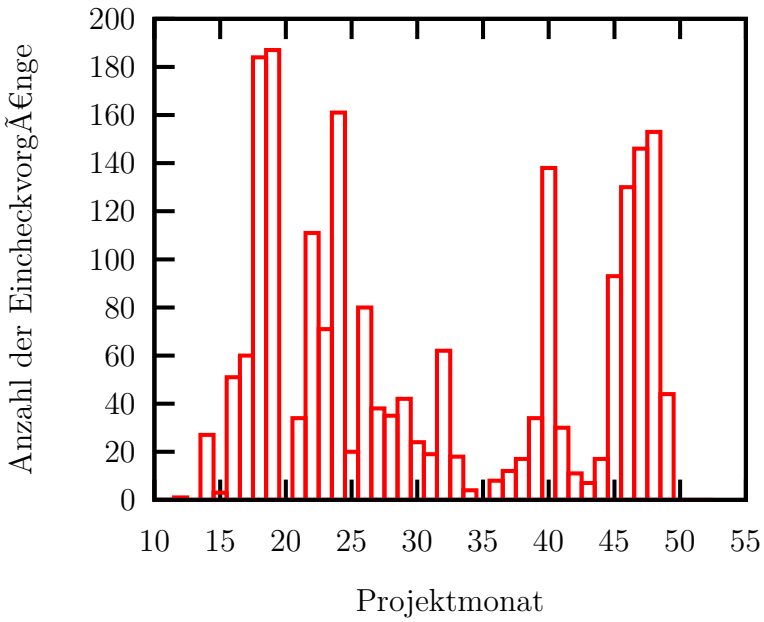


Abbildung 7.1: Anzahl der Eincheckvorgänge in das gemeinsame DESIRE Repository, jeweils aufsummiert pro Monat. Im Diagramm sind die drei letzten Jahre des Projektes dargestellt, da im ersten Jahr vornehmlich die Spezifikation erstellt wurde. Markant sind die Spitzen um die jeweiligen Projektmeilensteine M18, M24, M42 und M48 (vgl. auch Kapitel 10).

Normiert man den lokalen Integrationsanteil $W_{\mathrm{I,l}}$ durch die Anzahl der Eincheckvorgänge n_{c}, erhält man den relativen Meetingaufwand $W_{\mathrm{I,l_{rel}}}$:

$$W_{\mathrm{I,l_{rel}}}(t) = \frac{W_{\mathrm{I,l}}}{n_{\mathrm{c}}(t)}, n_{\mathrm{c}}(t) > 0. \tag{7.13}$$

Je höher der Wert des relativen Meetingaufwands ist, desto mehr lokale Integrationstreffen sind erforderlich, um einen bestimmten Teil der Integrationsarbeit zu leisten (gemessen in Eincheckvorgängen). Der relative Meetingaufwand ist nur für Werte $n_{\mathrm{c}}(t) > 0$ definiert.

In Abbildung 7.1 ist die Anzahl der Repository Eincheckvorgänge pro Monat über die Projektlaufzeit aufgetragen. Erwartungsgemäß befinden sich Maxima um die Projektmeilensteine (vgl. Kapitel 10). Eine ähnliche Kurve ergibt die Aufzeichnung der lokalen Integrationsarbeit $W_{\mathrm{I,l}}(t)$ über die Projektlaufzeit t (s. Abbildung 7.2).

Die starke Korrelation dieser beiden Größen wird noch deutlicher, wenn beide Kurven überlagert werden: Abbildung 7.4 (unten) zeigt die aufsummierten Checkins pro Monat sowie die Teilnehmerzahlen der Integrationstreffen über der Projektlaufzeit. Insbesondere zu Beginn des Projektes, als noch keine Laufzeitwerkzeuge eingesetzt wurden, ist der Integrationsfortschritt sehr deutlich auf die Integrationstreffen begrenzt. Dieser Zusammenhang wird auch durch Abbildung 7.3 belegt. Im Schaubild ist erkennbar, dass für eine vergleichbare Integra-

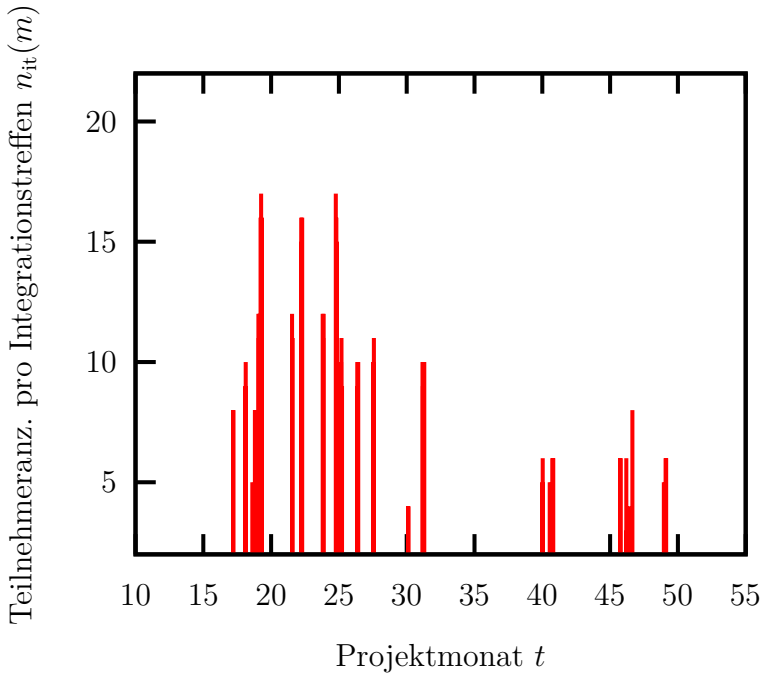


Abbildung 7.2: Zeitpunkte und Teilnehmeranzahl von Integrationstreffen. Jeder Impuls entspricht jeweils einem Integrationstreffen.

tionsarbeit nach der Einführung des Webportals in Monat 36 der relative Meetingaufwand kleiner ist, d.h. weniger Integrationstreffen erforderlich waren.

Diese These wird weiter gestützt durch das obere Diagramm in Abbildung 7.4, in dem exemplarisch für den Projektmonat 47 die Nutzungstatistik des Webportals dargestellt ist. In Projektmonat 47 wurden zwar Integrationsarbeiten durchgeführt (Anzahl der Checkins 146), es erfolgten aber keine Integrationstreffen. Aus der Nutzungsstatistik des Webportals lässt sich nun ein Zusammenhang zwischen den Checkins und den Zugriffszeiten auf das Webportal erkennen. In der Abbildung bezeichnet $n_{\mathrm{wp}}(t_{\mathrm{d}})$ die Anzahl der Webportalnutzer an einem Tag t_{d}. Die Integrationsarbeit (z.B. die Durchführung von Komponenten- und Systemtests) konnte also offensichtlich erfolgreich mit dem Webportal statt wie bisher über Integrationstreffen (Durchführung von lokalen Tests) durchgeführt werden. Somit konnte sowohl eine räumliche Entkopplung (Integrationsarbeit über das Webportal statt über lokale Integrationstreffen) als auch eine zeitliche Entkopplung (statt konzentrierter Treffen mit vielen Personen finden häufige Webportal-Zugriffe von wenigen Personen statt) der Entwickler realisiert werden.

In Abbildung 7.5 wird die Zusammensetzung der gesamten Integrationsarbeit W_{I} dargestellt. Zum einen ist wieder erkennbar, dass die Anteile der lokalen Integrationsarbeit $W_{\mathrm{I,l}}(t)$ ab Einführung des Webportals geringer sind. Weiterhin fallen auch die Zeitverluste während der Integrationstreffen geringer aus, da diese mit einem reduzierten und effektiven Kernteam durchgeführt werden konnten. Komponentenentwickler, die nur kurzzeitig, z.B. für

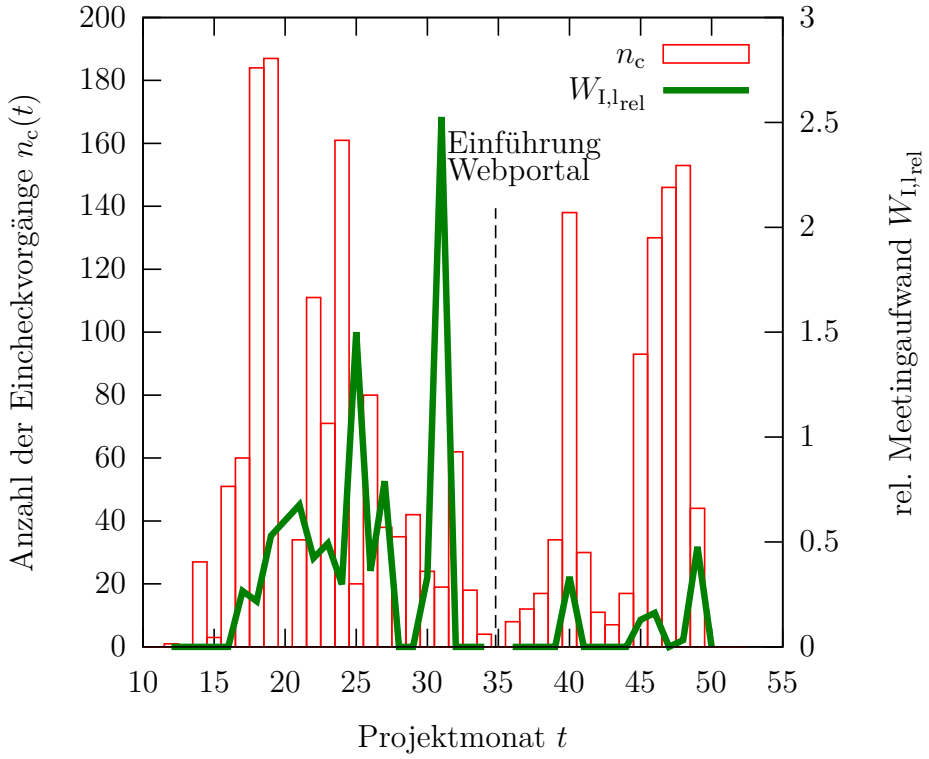


Abbildung 7.3: Relativer Meetingaufwand: Teilnehmerzahl für lokale Integrationstreffen bezogen auf die erfolgten Eincheckvorgänge.

Schnittstellentests benötigt wurden, konnten sich über das Webportal dazuschalten. Summiert man die lokale Integrationsarbeit jeweils für die Zeit ohne Webportal und die Zeit mit Webportal auf und setzt diese Summe in Bezug zum Gesamtaufwand, so lässt sich folgendes feststellen: ohne Webportal wurden rund 18% der Integrationsarbeit lokal geleistet. Unter Nutzung des Webportals ging dieser lokale Anteil der aufgewandten Integrationsarbeit auf etwa ein Drittel (ca. 6%) zurück.. Da die lokal geleistete Integrationsarbeit durch die damit verbundenen Reisekosten teurer ist, konnte somit auch eine Kostenreduktion der Integrationsarbeit erreicht werden.

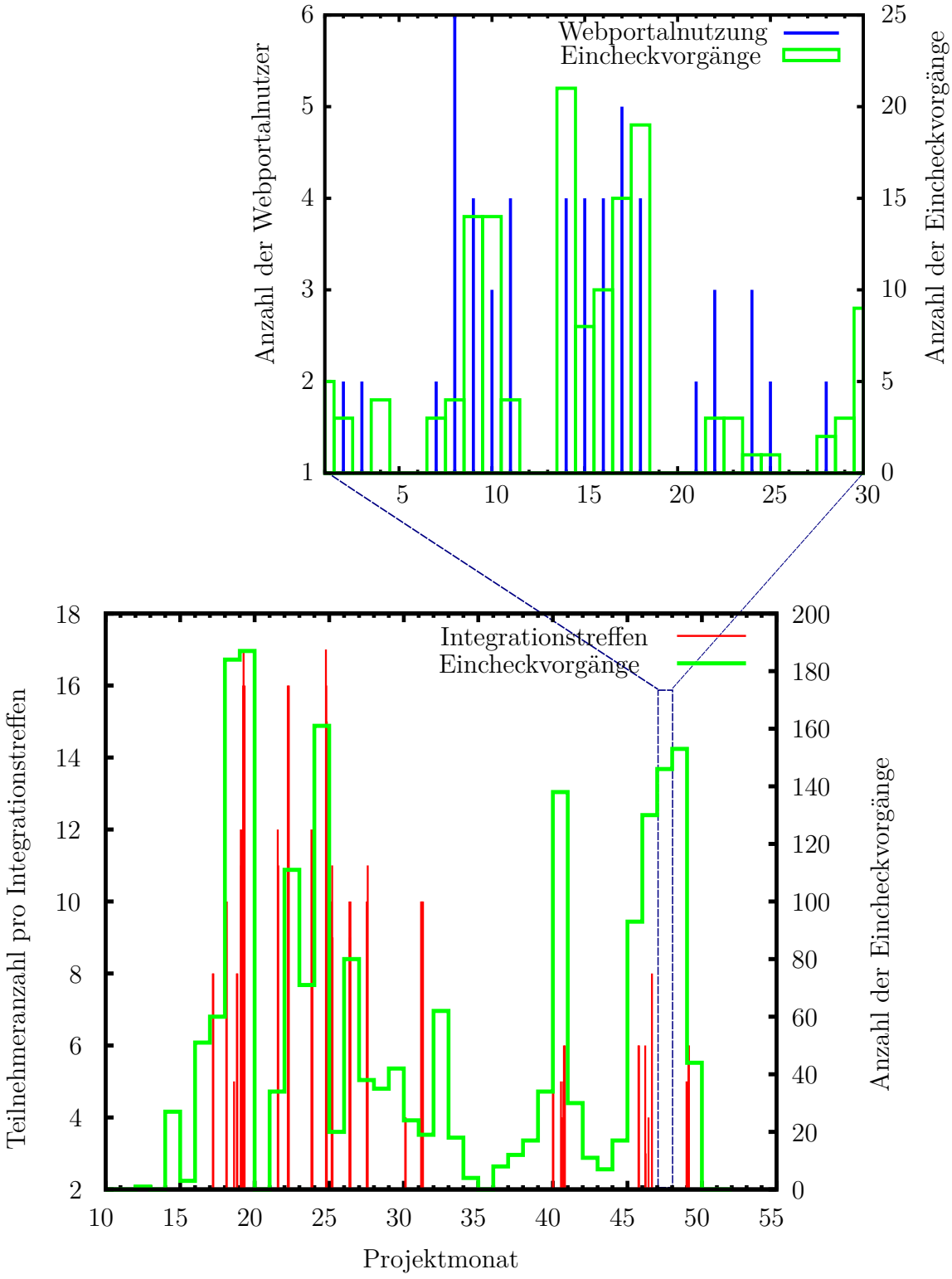


Abbildung 7.4: Unteres Diagramm: Teilnehmerzahl der Integrationstreffen sowie aufsummierte Checkin-Vorgänge pro Monat über die Projektlaufzeit. Oberes Diagramm: Nutzung des Webportals (Anzahl der Zugriffe verschiedener Personen) in Projektmonat 47. Die Grafik veranschaulicht die räumliche und zeitliche Entkopplung der Integrationsarbeiten [Rei12].

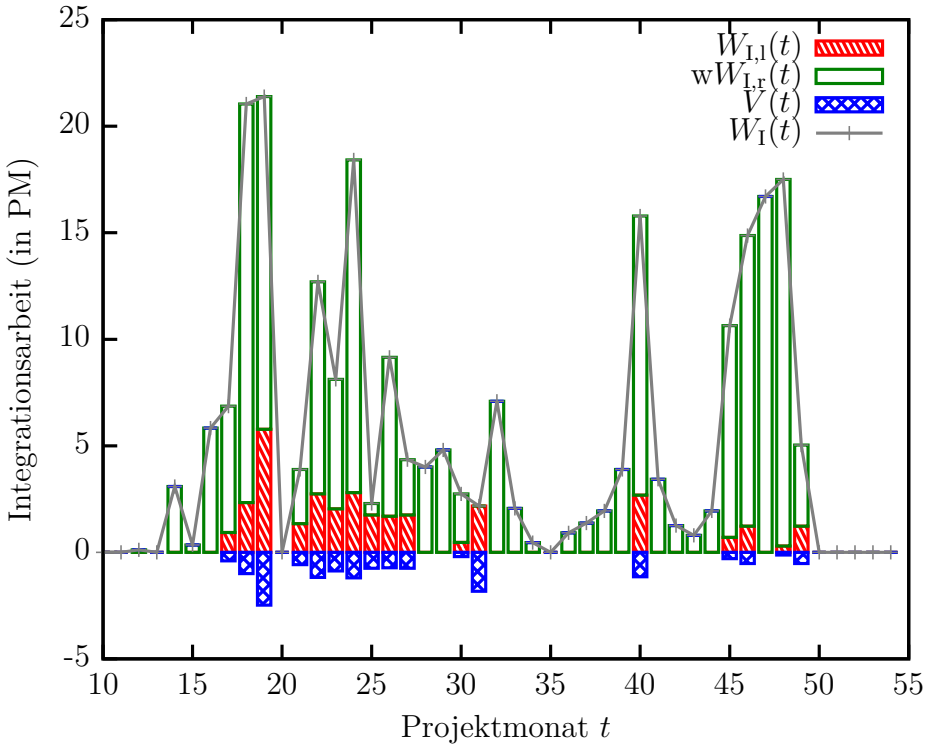


Abbildung 7.5: Anteile der entfernt ($W_{\mathrm{I,r}}(t)$) und lokal ($W_{\mathrm{I,l}}(t)$) geleisteten Integrationsarbeit. Die gesamte Integrationsarbeit $W_{\mathrm{I}}(t)$ wurde hierbei nach Gleichung 7.1 geschätzt. Schließlich werden im Diagramm noch die geschätzten Teamverluste $V(t)$ während der Integrationstreffen dargestellt.

8 Zusammenfassung und Ausblick

Die räumlich verteilte Entwicklung auf einer gemeinsamen, komplexen Zielhardware in einem interdisziplinären Expertenteam ist oft verbunden mit immensen Aufwänden für die Integration (bestehend aus Arbeitszeit und Reisekosten) der Komponenten in das Gesamtsystem sowie für die Durchführung von Komponenten- und Applikationstests. Im Unterschied zu ähnlich gelagerten Projekten in der Industrie (z.B. Automotive, Luftfahrt), haben sich für in der (Service-)Robotik bis dato jedoch noch keine Standards hinsichtlich Komponentenschnittstellen und Entwicklungsmethoden durchgesetzt, so dass auch kaum Werkzeuge für die effiziente Applikationsentwicklung verfügbar sind. Die Werkzeuge aus der Industrie werden dem hohen Innovations- und Forschungscharakter der Serviceroboterprojekte in der Regel nicht gerecht. Weiterhin ist die hohe Abhängigkeit der einzelnen Hardware- und Software-Komponenten charakteristisch für die Servicerobotik-Domäne, so dass die Integration von Komponenten in das Gesamtsystem und insbesondere die Durchführung von einzelnen Komponententests erschwert wird.

Zur Unterstützung des Softwareentwicklungsprozesses existieren viele Methoden und Werkzeuge aus dem Software-Engineering (Vorgehensmodelle, IDEs, Libraries, Versionierungswerkzeuge, Bugtracker, Continuous Integration, etc.). Für den Post-Entwicklungsprozess im Sinne des Deployments von Komponenten und Applikationen auf komplexe Serviceroboter sind entsprechende Hilfsmittel kaum verfügbar. Die Zuordnung der einzelnen Software-Komponenten einer Applikation zu den Rechnern auf der Zielplattform unter Berücksichtigung aller Hardware- und Softwareabhängigkeiten sowie die Installation, Konfiguration und Inbetriebnahme derselben, erfordert aufgrund der starken Verwobenheit der Robotikkomponenten jedoch sehr viel Detailwissen bezüglich der Hardwarearchitektur und der einzelnen Software-Komponenten der Applikation. Daher kann eine Rollentrennung in Komponentenentwickler, Applikationsentwickler und Systemintegratoren in der Entwicklungsdomäne Servicerobotik, die in anderen Domänen die Applikationsentwicklung enorm beschleunigt (vgl. z.B. Apps für Smartphones), bisher kaum erreicht werden. Durch die hohen Abhängigkeiten zwischen den einzelnen Komponenten ist der Integrations- und Testaufwand sehr hoch, was insbesondere in Forschungsprojekten mit verteilten Experten oft den geplanten Fortschritt behindert. Da die Zielplattformen oft sehr teuer und nur an einem Ort zur Verfügung stehen, erfolgt die Integration in der Regel in aufwändigen lokalen Integrationstreffen.

Webbasierte Werkzeuge für den netzwerkbasierten Zugriff auf Serviceroboter (networked robots) sind in der Regel nicht für die Kooperation und die Durchführung von Ferntests ausgelegt sondern fokussieren meist Teleoperationsanwendungen. Die wenigen Beispiele, die die verteilte Entwicklung und Kooperation auf gemeinsamer Hardware unterstützen (z.B. PR2-remotelab) sind jedoch meist auf eine bestimmte Hardwareplattform und Middlewares beschränkt.

In der vorliegenden Arbeit wurden deshalb Methoden und Werkzeuge zur Unterstützung von verteilten, komponenten-basierten Softwareprojekten für komplexe Serviceroboter entwickelt, die nicht auf eine bestimmte Software-Architektur oder Hardwareplattform beschränkt sind.

Die Beiträge dieser Arbeit kristallisieren sich in den folgenden drei Kernpunkten:

- der Definition eines Vorgehensmodells für die verteilte Entwicklung und Integration von Servicerobotik-Applikationen
- der Entwicklung einer Werkzeugkette für das Deployment einer komponenten-basierten Applikation auf die Zielhardware “Serviceroboter”
- der Schaffung von leistungsfähigen Werkzeugen zur Laufzeitunterstützung für Test und Betrieb der Zielhardware

Das in dieser Arbeit entwickelte **Vorgehensmodell** für die verteilte Entwicklung und Test, *DDT*, basiert auf dem OpenUP-Vorgehensmodell, das bereits sehr gut auf die Anforderungen der Zielszenarien zugeschnitten ist. DDT erweitert den OpenUP um Elemente, die für die verteilte Entwicklung essentiell sind, z.B. die Qualitätssicherung abgelieferter Komponenten und fördert strukturell die Rollentrennung von Komponentenentwicklern, Applikationsentwicklern und Systemintegratoren.

Eine **Deployment-Werkzeugkette** wurde konzipiert und umgesetzt, die unabhängig von einem bestimmten Komponentenmodell oder einer bestimmten Middleware eingesetzt werden kann. Der Deployment-Planer ordnet die verschiedenen Software-Komponenten einer Applikation bestimmten Zielrechnern auf dem Serviceroboter auf der Basis eines Applikations- und Domänenmodells zu, das jeweils vom Applikationsentwickler bzw. dem Systemintegrator generiert wird. Die Zuordnung erfolgt dabei durch die Analyse der jeweils erforderlichen bzw. verfügbaren Ressourcen und Abhängigkeiten zwischen den einzelnen Komponenten. Nach erfolgreicher Zuordnung erfolgt durch das Werkzeug die automatische Installation der Komponenten, sofern diese in Form von Paketen in einem Paketmanagementsystem organisiert sind. Durch die Werkzeuge für die **Laufzeitunterstützung** können die verschiedenen Komponenten dann konfiguriert und entweder einzeln oder im Applikationszusammenhang gestartet werden. Dabei steht dem Entwickler eine leistungsfähige

Ausführungsumgebung zur Verfügung, bestehend aus eingebetteten Konsolen, in denen die Logdaten angezeigt werden, VNC-Fenstern für Visualisierungs- oder Nutzeroberflächen und einem Hardwaremonitor, der Diagnoseinformationen bis auf die Schaltkreisebene bietet. Der verteilte Zugriff wird durch ein Reservierungssystem verwaltet, so dass keine unkoordinierten Tests auf der Hardware erfolgen können. Durch entsprechende Nutzerrechte kann der Nutzerkreis für die jeweiligen Werkzeuge bei Bedarf eingeschränkt werden. Die Zugangsbeschränkung kann somit implizit die Aufgabentrennung der einzelnen Entwicklerrollen regeln.

Insbesondere durch den Einsatz der Deployment- und Laufzeitwerkzeuge konnte im Verbund-Forschungsprojekt DESIRE eine **stärkere räumliche und zeitliche Entkopplung** der Integrationsaktivitäten und somit eine höhere Effizienz des verteilten Entwicklungsprozesses erreicht werden. Die **Bedienbarkeit** der komplexen Projektplattform konnte wesentlich verbessert werden - mit den Werkzeugen war jeder Entwickler in der Lage, Komponenten und Applikationen auf dem System auszuführen. Unterschiedliche Nutzerrechte der einzelnen Werkzeuge **forcieren die Rollentrennung**: Komponentenentwickler stellen ihre Komponenten als Paket im Komponentenrepository zur Verfügung und dokumentieren die Funktionalität und die Schnittstellen, Applikationsentwickler erstellen das Applikationsmodell im Deployment-Werkzeug auf Basis der verfügbaren Komponenteninformation, Systemintegratoren verwalten das Domänenmodell und führen Deployment-Planung sowie Installation und Aktualisierung der Komponenten auf dem Zielsystem durch. Alle Entwicklerrollen können für die Durchführung von Komponenten- und Applikationstests sowie deren Konfiguration die Laufzeitwerkzeuge nutzen und unabhängiger von Systemintegratoren und Komponentenentwicklern eine neue Applikation auf dem System einrichten und testen.

Die einzelnen Werkzeuge wurden in eine webbasierte Integrations- und Testplattform eingebettet. Somit ist der verteilte Zugriff auf die gemeinsame Zielhardware ermöglicht und alle entwickelten Werkzeuge sind über eine zentrale Weboberfläche von überall zugänglich. Im Rahmen des großen nationalen Verbundprojektes DESIRE wurde die Integrations- und Testplattform erfolgreich eingesetzt und evaluiert.

Durch den offenen und modularen Aufbau der Integrations- und Testplattform bestehen viele Erweiterungs- und Anpassungsmöglichkeiten. Die grundsätzliche middelware-unabhängige Implementierung kann bei Bedarf auf ein bestimmtes Framework spezialisiert werden, um einige Arbeiten der bereits automatisiert in den einzelnen Plugins abzubilden. Für das ROS Framework können z.B. dedizierte Aktivierungs, Deaktivierungs- und Konfigurationsskripte eingebettet werden. Falls sich in Zukunft eine Modellierungssprache für die Servicerobotik-Domäne durchsetzen sollte oder eine Festlegung aus anderweitigen Gründen getroffen werden kann, besteht die Möglichkeit den Deployment-Planer für die detaillierten Modellie-

rungselemente zu erweitern. Dadurch könnte das erforderliche Experten-Wissen für das Deployment von Applikationen auf eine beliebige Serviceroboter-Hardware weiter reduziert werden. Durch die plugin-basierte Architektur können schließlich bei Bedarf sehr einfach weitere Module (z.B. Workflow-Management) in die Integrations- und Testplattform integriert werden.

9 Literaturverzeichnis

[AB97] Abts, Christopher M. ; Boehm, Barry W.: COTS/NDI Software Integration Cost Estimation & USC-CSE COTS Integration Cost Calculator V2.0 User Guide / Center for Software Engineering, University of Southern California. 1997. – Forschungsbericht

[ABB] ABB: *Technische Unterstützung und Remote Service.* – `http://www.abb.de/product/seitp327/00cda896644bb168c125722c0035783f.aspx` (23.12.2012)

[ABB+00] Abts, C. ; Barry, M. S. ; Boehm, Barry W. ; Elizabeth, Ph. D. ; Clark, E. B.: COCOTS: A COTS Software Integration Lifecycle Cost Model - Model Overview and Preliminary Data Collection Findings. In: *Proc. European Software Control and Metrics (ESCOM-SCOPE) Conference.* München : Shaker, April 2000, S. 325–333

[ABK01] Appelt, W. ; Busbach, U. ; Koch, T.: Kollaborationsorientierte asynchrone Werkzeuge. In: Unland, R. (Hrsg.): *CSCW-Kompendium. Lehr- und Handbuch zum computerunterstützten kooperativen Arbeiten.* Berlin u.a. : Springer, 2001, S. 194–203

[ACC11] *ACCOMPANY - Acceptable robotiCs COMPanions for AgeiNg Years.* Oct 2011. – 7th Framework Programme (FP7), Objective ICT-2011.5.4 ICT for Ageing and Wellbeing

[ACF+98] Alami, R. ; Chatila, R. ; Fleury, S. ; Ghallab, M. ; Ingrand, F.: An Architecture for Autonomy. In: *International Journal of Robotics Research* 17 (1998), S. 315–337

[ago] agorum: *agorum core.* – `http://www.agorum.com/` (23.12.2012)

[AOL] AOL: *AIM - AOL Instand Messenger.* – `http://www.aim.com/` (23.12.2012)

[Apaa] Apache Software Foundation: *Apache Continuum.* – `http://continuum.apache.org/` (23.12.2012)

[Apab] Apache Software Foundation: *Apache Tuscany.* – `http://tuscany.apache.org/` (23.12.2012)

[ARA+06] Asfour, T. ; Regenstein, K. ; Azad, P. ; Schröder, J. ; Vahrenkamp, N. ; Dillmann, R.: ARMAR-III: An Integrated Humanoid Platform for Sensory-Motor Control. In: *6th IEEE-RAS International Conference on Humanoid Robots, December 4-6, 2006, Genova, Italy.* Piscataway, NJ : IEEE, 2006, S. 169–175

[AS05] Andronache, Virgil ; Scheutz, Matthias: ADE - An Architecture Development Environment for Virtual and Robotic Agents. In: *International Journal of Artificial Intelligence Tools* 15 (2005), Nr. 2, S. 251–286

[ASK08] Ando, N. ; Suehiro, T. ; Kotoku, T.: A Software Platform for Component Based RT-System Development: OpenRTM-Aist. In: Carpin, Stefano (Hrsg.) ; Noda, Itsuki (Hrsg.) ; Pagello, Enrico (Hrsg.) ; Reggiani, Monica (Hrsg.) u. a.: *Simulation, Modeling, and Programming for Autonomous Robots* Bd. 5325. Berlin u.a. : Springer, 2008, S. 87–98

[ASKK06] Ando, N. ; Suehiro, T. ; Kitagaki, K. ; Kotoku, T.: RT (Robot Technology) - Component and its Standardization - Towards Component Based Networked Robot Systems Development. In: *SICE-ICASE, 2006. International Joint Conference, 18-21 October 2006, Busan, Korea.* Piscataway, NJ : IEEE, 2006, S. 2633 –2638

[Atl] Atlassian: *JIRA: Your Team, Your Process, Your Rules.* – `http://www.atlassian.com/software/jira/overview` (23.12.2012)

[AUT04] AUTOSAR: *AUTomotive Open System ARchiteture.* (2004). – `http://www.autosar.org/` (23.12.2012)

[BA04] Beck, Kent ; Andres, Cynthia: *Extreme Programming Explained: Embrace Change.* 2. Aufl. Boston : Addison-Wesley Professional, 2004

[BAB+00] Boehm, B. W. ; Abts, C. ; Brown, A. W. ; Chulani, S. ; Clark, B. K. ; Horowitz, E. ; Madachy, R. ; Reifer, D. J. ; Steece, B.: *Software Cost Estimation with Cocomo II.* 1. Aufl. Upper Saddle River, NJ, USA : Prentice Hall PTR, 2000

[BAC00] Boehm, Barry ; Abts, Chris ; Chulani, Sunita: Software development cost estimation approaches – A survey. In: *Annals of Software Engineering* 10 (2000), Nr. 1, S. 177–205

[Bal08] Balzert, H.: *Lehrbuch der Softwaretechnik: Softwaremanagement.* 2. Aufl. Heidelberg : Spektrum, Akad. Verl., 2008 (Lehrbücher der Informatik)

[BB09] Bernhard, M. ; Breiteneder, R.: Phase Integration/Test - wie wird zusammengefügt/geprüft? In: Grechenig, T. (Hrsg.) u. a.: *Softwaretechnik - Mit Fallbeispielen aus realen Entwicklungsprojekten.* München : Pearson Studium, 2009, S. 297–348

[BBB+98] Balter, R. ; Bellissard, L. ; Boyer, F. ; Riveill, M. ; Vion-Dury, J.-Y.: Architecturing and configuring distributed application with Olan. In: *Proceedings of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing.* London, UK : Springer, 1998, S. 241–256

[BBB+01] Beck, K. ; Beedle, M. ; Bennekum, A. v. ; Cockburn, A. u. a.: *Manifesto for Agile Software Development.* 2001. – `http://agilemanifesto.org` (23.12.2012)

[BCC+00] Boehm, B. ; Clark, B. ; Chulani, S. ; Horowitz, E. ; Madachy, R. ; Reifer, D. ; Selby, R. ; Steece, B.: *COCOMO II Model Definition Manual, Version 2.1.* (2000). – `http://csse.usc.edu/csse/research/COCOMOII/cocomo2000.0/CII_modelman2000.0.pdf` (25.12.2012)

[BCH+95] Boehm, Barry ; Clark, Bradford ; Horowitz, Ellis ; Westland, Chris ; Madachy, Ray ; Selby, Richard: Cost Models for Future Software Life Cycle Processes: COCOMO 2.0. In: *Annals of Software Engineering* 1 (1995), S. 57–94

[BCS05] Belousov, Igor R. ; Chebukov, Sviatoslav ; Sazonov, Victor V.: Web-based Teleoperation of the Robot Interacting with Fast Moving Objects. In: *Proceedings of the IEEE International Conference on Robotics & Automation (ICRA).* Piscataway, NJ : IEEE, 2005, S. 673–678

[BDA+04] Bourque, P. ; Dupuis, R. ; Abran, A. ; Moore, J. W. ; Tripp, L. L.: *The Guide to the Software Engineering Body of Knowledge (SWEBOK).* Los Alamitos, CA, USA : IEEE Computer Society, 2004

[Bea95] Beazley, D.: *SWIG (Simplified Wrapper and Interface Generator).* (1995). – `http://www.swig.org/` (23.12.2012)

[Bea96] Beazley, David M.: SWIG: an easy to use tool for integrating scripting languages with C and C++. In: *Proceedings of the 4th conference on USENIX Tcl/Tk Workshop, July 10-13, 1996 Monterey, California.* Berkeley, CA, USA : USENIX Association, 1996, S. 15

[Bea10] Beauftragte der Bundesregierung für Informationstechnik: *Das V-Modell XT Webportal.* (2010). – `http://www.v-modell-xt.de` (23.12.2012)

[Bec03] Beck, Kent: *Test-Driven Development by Example.* Boston : Addison-Wesley, 2003 (The Addison-Wesley Signature Series)

[BEH+09] Badii, A. ; Etxeberria, I. ; Huijnen, C. ; Maseda, M. ; Dittenberger, S. ; Hochgatterer, A ; Thiemert, D. ; Rigaud, A-S.: CompanionAble-Graceful integration of mobile robot companion with a smart home environment. In: *Gerontology* 8 (2009), Nr. 3, S. 185

[Ber90] Berliner, B.: CVS II: Parallelizing software development. In: *Proceedings of the USENIX Conference.* Berkeley, CA, USA : USENIX Association, 1990, S. 341–352

[BGH+12] Bubeck, A. ; Garcia, H. ; Hochgschwendner, N. ; Gherardi, L. u. a.: *BRIDE: A Model-driven Engineering IDE for robotics based on Eclipse.* (2012). – `http://www.best-of-robotics.org/bride/` (23.12.2012)

[BGKB12] Brugali, D. ; Gherardi, L. ; Klotzbücher, M. ; Bruyninckx, H.: Service Component Architectures in Robotics: The SCA-Orocos Integration. In: Hähnle, R. (Hrsg.) ; Knoop, J. (Hrsg.) ; Margaria, T. (Hrsg.) u. a.: *Leveraging Applications of Formal Methods, Verification, and Validation.* Berlin u.a. : Springer, 2012, S. 46–60

[BGRS12] Brugali, D. ; Gherardi, L. ; Riccobene, E. ; Scandurra, P.: Coordinated Execution of Heterogeneous Service-Oriented Components by Abstract State Machines. In: Arbab, F. (Hrsg.) ; Ölveczky, P. (Hrsg.): *Formal Aspects of Component Software* Bd. 7253. Berlin u.a. : Springer, 2012, S. 331–349

[Bit] BitMover, Inc.: *BitKeeper.* – `http://www.bitkeeper.com/` (23.12.2012)

[BKM+05] Brooks, A. ; Kaupp, T. ; Makarenko, A. ; Williams, S. ; Oreback, A.: Towards component-based robotics. In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Edmonton, Alberta, Canada, August 2 - 6, 2005.* Piscataway, NJ : IEEE, 2005, S. 163 – 168

[BMT10] Beetz, M. ; Mösenlechner, L. ; Tenorth, M.: CRAM – A Cognitive Robot Abstract Machine for Everyday Manipulation in Human Environments. In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Ro-*

bots and Systems (IROS), Taipei, Taiwan, 18-22 Oct. 2010. Piscataway, NJ : IEEE, 2010, S. 1012–1017

[Boe81] Boehm, B. W.: *Software Engineering Economics*. Upper Saddle River, NJ, USA : Prentice Hall PTR, 1981

[Boe88] Boehm, B. W.: A spiral model of software development and enhancement. In: *Computer Journal* 21 (1988), Nr. 5, S. 61 –72

[Boh10] Bohren, J.: *SMACH*. (2010). – `http://www.ros.org/wiki/smach` (23.12.2012)

[Bra02] Bradski, G. R.: *OpenCV: Examples of use and new applications in stereo, recognition and tracking*. (2002). – `http://opencvlibrary.sourceforge.net/` (23.12.2012)

[BRPH09] Bubeck, A. ; Reiser, U. ; Pfeiffer, K. ; Hägele, M.: Distributed Access to a Centralized Testing and Benchmarking Facility via the DESIRE Webportal. In: *IEEE/RSJ International Conference on Robots and Intelligent Systems (IROS), Workshop on Performance Evaluation and Benchmarking for Next Intelligent Robots and Systems, Oct. 11-15, 2009, St. Louis, MO, USA*. Piscataway, NJ : IEEE, 2009, S. 5

[Bru01] Bruyninckx, H.: Open robot control software: The OROCOS project. In: *Proceedings of the IEEE International Conference on Robotics & Automation (ICRA), Seoul, Korea, 21-26 May 2001*. New York, USA : IEEE, 2001, S. 2523 – 2528

[BRWH11] Blake, M.B. ; Remy, S.L. ; Wei, Yi ; Howard, A.M.: Robots on the Web. In: *Robotics Automation Magazine, IEEE* 18 (2011), Nr. 2, S. 33 –43

[Brö93] Bröhl, Adolf P. (Hrsg.): *Das V-Modell*. München [u.a.] : Oldenbourg, 1993 (Software - Anwendungsentwicklung - Informationssysteme)

[BS03] Börger, E. ; Stärk, R.: *Abstract State Machines: A Method for High-Level System Design and Analysis*. Berlin : Springer, 2003

[BTC99] Belousov, Igor R. ; Tan, JiaCheng ; Clapworthy, Gordon J.: Teleoperation and Java3D Visualization of a Robot Manipulator over the World Wide Web. In: *International Conference on Information Visualisation*. Los Alamitos, CA, USA : IEEE Computer Society, 1999, S. 543–548

[Can] Canonical Ltd.: *bazaar*. – `http://bazaar.canonical.com/en/` (23.12.2012)

[CCC+10] Cesta, A. ; Coradeschi, S. ; Cortellessa, G. ; Gonzalez, J. ; Tiberio, L. ; Rump, S. von: *Enabling Social Interaction Through Embodiment in ExCITE*. ForItAAL 2010. Second Italian Forum on Ambient Assisted Living, October 6-8, 2010, Trento, Italy, 2010

[CEM04] Chatley, Robert ; Eisenbach, Susan ; Magee, Jeff: MagicBeans: a Platform for Deploying Plugin Components. In: *Proc. of the Second Int. Working Conf. on Component Development, volume 3083 of LNCS*. Berlin : Springer, 2004, S. 97–112

[Cis] Cisco: *Cisco WebEx*. – `http://www.webex.com/` (23.12.2012)

[Cle01] Clements, Paul C.: From subroutines to subsystems: component-based software development. In: Heineman, G. T. (Hrsg.) ; Councill, W. T. (Hrsg.): *Component-based software engineering*. Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc., 2001, S. 189–198

[CLK+11] Coradeschi, S. ; Loutfi, A. ; Kristoffersson, A. ; Cortellessa, G. ; Severinson Eklundh, K.: Social robotic telepresence. In: *Proceedings of the 6th international conference on Human-robot interaction (HRI), March 6-11, 2011, Lausanne, Switzerland*. New York, NY, USA : ACM, 2011, S. 5–6

[CLMB07] Côté, C. ; Létourneau, D. ; Michaud, F. ; Brosseau, Y.: Using MARIE for Mobile Robot Component Development and Integration. In: *Software Engineering for Experimental Robotics* Bd. 30. Heidelberg : Springer, 2007, S. 211–230

[Con08] Conley, K.: *rostest - Integration test suite based compatible with xUnit frameworks*. (2008). – `http://ros.org/wiki/rostest` (23.12.2012)

[Cro04] Crocker, R.: Large Scale Agile Software Development. In: Zannier, C. (Hrsg.) ; Erdogmus, H. (Hrsg.) ; Lindstrom, L. (Hrsg.): *Extreme Programming and Agile Methods - XP/Agile Universe 2004* Bd. 3134. Berlin u.a. : Springer, 2004, S. 199–216

[D+] Danial, A. u. a.: *CLOC - Count Lines of Code.*. – `http://cloc.sourceforge.net/` (23.12.2012)

[Dar] Dart Communications: *PowerTCP*. – `http://www.dart.com/` (23.12.2012)

[DBO+05] Deng, G. ; Balasubramanian, J. ; Otte, W. ; Schmidt, D.. ; Gokhale, A.: DAnCE: A QoS-Enabled Component Deployment and Configuration Engi-

ne. In: Dearle, A. (Hrsg.) ; Eisenbach, S. (Hrsg.): *Component Deployment*. Berlin u.a. : Springer, 2005, S. 67–82

[DES09] *Deutsche Service-Robotik Initiative (DESIRE)*. 2005-2009. – Gefördert durch das Bundesministerium für Bildung und Forschung

[DKS+12] Dhouib, Saadia ; Kchir, Selma ; Stinckwich, Serge ; Ziadi, Tewfik ; Ziane, Mikal: RobotML, a Domain-Specific Language to Design, Simulate and Deploy Robotic Applications. In: *3nd International Conference on Simulation, Modeling and Programming for Autonomous Robots, SIMPAR 2012, Tsukuba, Japan, Nov 5-8, 2012*. Berlin : Springer, 2012 (Lecture Notes in Computer Science), S. 149–160

[DMG07] Duvall, Paul ; Matyas, Stephen M. ; Glover, Andrew: *Continuous Integration: Improving Software Quality and Reducing Risk*. Amsterdam : Addison-Wesley Longman, 2007 (The Addison-Wesley Signature Series)

[Dro] Dropbox Inc.: *Dropbox*. – `http://www.dropbox.com` (23.12.2012)

[DS91] Denert, E. ; Siedersleben, J.: *Software-Engineering: methodische Projektabwicklung*. 1. Aufl. Berlin : Springer, 1991

[Ecl] Eclipse Foundation: *Eclipse*. – `http://www.eclipse.org` (23.12.2012)

[Ecl11] Eclipse Foundation: *Eclipse SCA Tools*. (2011). – `http://www.eclipse.org/soa/sca/` (23.12.2012)

[Edg] Edgewall Software: *Bitten - A Continuous Integration plugin for Trac*. – `http://bitten.edgewall.org/` (23.12.2012)

[Edg07] Edgewall Software: *trac - Integrated SCM & Project Management*. (2007). – `http://trac.edgewall.org/` (23.12.2012)

[eWO] eWON: *Talk2M*. – `http://www.talk2m.com` (23.12.2012)

[FAN] FANUC Robotics: *Fanuc Robotics Remote Diagnose Center*. – `http://www.fanucrobotics.de/~/media/Global/Files/Downloads/Magazines/After%20Sales/DE/Remote_Diagnos_DE.ashx` (23.12.2012)

[Fje04] Fjermestad, J.: An analysis of communication mode in group support systems research. In: *Decision Support Systems* 37 (2004), S. 239–263

[FK03] Fu, X. ; Karamcheti, V.: Planning for Network-Aware Paths. In: Stefani, Jean-Bernard (Hrsg.) ; Demeure, Isabelle M. (Hrsg.) ; Hagimont, Da-

niel (Hrsg.): *Distributed Applications and Interoperable Systems*. Berlin u.a. : Springer, 2003 (Lecture Notes in Computer Science, Bd. 2893), S. 187–199

[Flea] Flexera Software LLC: *InstallShield*. – http://www.flexerasoftware.com/products/installshield.htm (23.12.2012)

[Fleb] Flexera Software LLC: *Zero G software deployment and lifecycle management solutions.*. – http://www.flexerasoftware.com/products/installanywhere.htm (23.12.2012)

[FLVC05] Feiler, P. H. ; Lewis, B. ; Vestal, S. ; Colbert, E.: An Overview of the SAE Architecture Analysis & Design Language (AADL) Standard: A Basis for Model-Based Architecture-Driven Embedded Systems Engineering Architecture Description Languages. In: Dissaux, P. (Hrsg.) ; Filali-Amine, M. (Hrsg.) ; Michel, P. (Hrsg.) u. a.: *Architecture Description Languages*. Boston : Springer, 2005 (IFIP International Federation for Information Processing Vol. 176), S. 3–15

[FMSOK08] Fjerdingen, S. ; Mathiassen, J. ; Schumann-Olsen, H. ; Kyrkjebø, E.: Adaptive Snake Robot Locomotion: A Benchmarking Facility for Experiments. In: Bruyninckx, Herman (Hrsg.) ; Preucil, Libor (Hrsg.) ; Kulich, Miroslav (Hrsg.): *European Robotics Symposium 2008*. Berlin u.a. : Springer, 2008 (Springer Tracts in Advanced Robotics Vol. 44), S. 13–22

[Fox89] Fox, B.: *Bash Reference Manual*. (1989). – http://www.gnu.org/software/bash/manual/bashref.html (23.12.2012)

[Fra10] Fraunhofer IPA: *Care-O-bot Research*. (2010). – http://care-o-bot-research.org (23.12.2012)

[G. 99] G. Stein and J. Whitehead : *WebDAV - Web-based Distributed Authoring and Versioning*. 1999. – http://www.webdav.org (23.12.2012)

[GÏ10] Gümgümcü, Ö.: *Konzept und Implementierung einer Hardware-Architektur zur Überwachung und Steuerung von elektronischen Bauteilen eines Service-Roboters.*, Institut für Automatisierungstechnik/Mechatronik Hochschule Esslingen, Bachelorthesis, 2010

[Gat98] Gat, E.: Three-layer architectures. In: Kortenkamp, D. (Hrsg.) ; Bonasso, P. (Hrsg.) ; Murphy, R. R. (Hrsg.): *Artificial Intelligence and Mobile Robots*. Cambridge, Ma, USA : MIT Press, 1998, S. 195–211

[GBS] GBS PAVONE GROUPWARE GMBH: *Pavone Espresso Workflow.* – `http://www.pavonelive.de/` (23.12.2012)

[Gir11] GIRAFF TECHNOLOGIES AB: *Giraff.* (2011). – `http://www.giraff.org/` (23.12.2012)

[Glo09] GLOGER, B.: *Scrum: Produkte zuverlässig und schnell entwickeln.* 2. Aufl. München : Hanser, 2009

[Goo] GOOGLE: *googletest - Google C++ Testing Framework .* – `http://code.google.com/p/googletest/` (23.12.2012)

[Goo10] GOOGLE: *Google Docs.* (2010). – `http://www.google.com/apps/intl/de/business/docs.html` (23.12.2012)

[Goo11] GOOGLE: *Google Cloud Connect for Microsoft Office.* (2011). – `http://tools.google.com/dlpage/cloudconnect` (23.12.2012)

[GOS03] GOSTAI: *Urbi Open Source.* (2003). – `http://www.gostai.com/products/urbi/` (23.12.2012)

[GP06] GUCKENHEIMER, S. ; PEREZ, J. J.: *Software Engineering with Microsoft Visual Studio Team System.* Amsterdam : Addison-Wesley Longman, 2006 (Microsoft .NET Development Series)

[Gro08] GROHMANN, S.: *Projektmanagement in verteilten Projektteams*, Trigonum GmbH/ Universität Leipzig, Expertenumfrage, 2008

[Gru09] GRUBER, Tom: Ontology. In: LIU, Ling (Hrsg.) ; ÖZSU, M. T. (Hrsg.): *Encyclopedia of Database Systems.* Berlin : Springer, 2009, S. 1963–1965

[Gus08] GUSTAFSSON, B.: OpenUP -The Best of Two Worlds. In: *METHODS and TOOLS - Practical knowledge for the software developer, tester and project manager* 16 (2008), Nr. 1, S. 21–32

[GVS+01] GERKEY, B. ; VAUGHAN, R. ; STY, K. ; HOWARD, A. ; SUKHATME, G. ; MATARIC, M.: Most valuable player: A robot device server for distributed control. In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, Oct 29 - Nov 3, 2001, Outrigger Wailea Resort, Maui, Hawaii, USA.* Piscataway, NJ : IEEE, 2001

[Hay85] HAYES, P. J.: The Second Naive Physics Manifesto. In: HOBBS, J. R. (Hrsg.) ; MOORE, R. C. (Hrsg.): *Formal Theories of the Commonsense World.* Norwood, NJ : Ablex, 1985, S. 1–36

[HC01] Heineman, George T. (Hrsg.) ; Councill, William T. (Hrsg.): *Component-based software engineering: putting the pieces together*. Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc., 2001

[Hen96] Henzinger, T. A.: The theory of hybrid automata. In: *Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science, New Brunswick, NJ, 27-30 Jul 1996*. Washington, DC, USA : IEEE Computer Society, 1996 (LICS '96), 278–292

[Hey06] Heydarnoori, A.: Caspian: A QoS-Aware Deployment Approach for Channel-based Componentbased Applications / David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, ON, N2L 3G1, Canada. 2006. – Forschungsbericht

[Hey08] Heydarnoori, A.: Deploying Component–Based Applications: Tools and Techniques. In: Lee, R. (Hrsg.): *Software Engineering Research, Management and Applications, Aug. 20-22, 2008, Prague, Czech Republic*. Berlin : Springer, 2008 (Studies in Computational Intelligence, Vol. 150), S. 29–42

[HHHW97] Hall, Richard S. ; Heimbigner, Dennis ; Hoek, Andre van d. ; Wolf, Alexander L.: An Architecture for Post-Development Configuration Management in a Wide-Area Network. In: *Proceedings of the 17th International Conference on Distributed Computing Systems (ICDCS '97), Baltimore, MD ,27-30 May 1997*. Washington, DC, USA : IEEE Computer Society, 1997, S. 269

[HHS01] Holmer, T. ; Haake, J. M. ; Streitz, N.: Kollaborationsorientierte synchrone Werkzeuge. In: Unland, R. (Hrsg.): *CSCW-Kompendium. Lehr- und Handbuch zum computerunterstützten kooperativen Arbeiten*. Berlin : Springer, 2001, S. 180–193

[HHW99] Heimbigner, D. ; Hall, R. S. ; Wolf, A. L.: A Framework for Analyzing Configurations of Deployable Software Systems. In: *Proceedings of the Fifth IEEE International Conference on Engineering of Complex Computer Systems*. Washington, DC : IEEE Computer Society, 1999, S. 32–42

[HO11] Hertel, G. ; Orlikowski, B.: Projektmanagement in ortsverteilten "virtuellen" Teams. In: Wastian, M. (Hrsg.) ; Braumandl, I. (Hrsg.) ; Rosenstiel, L. v. (Hrsg.): *Angewandte Psychologie für Projektmanager*. 2. Aufl. Berlin : Springer, 2011, S. 327–346

[Hon04] Honda: *ASIMO*. (2004). – `http://world.honda.com/ASIMO/` (23.12.2012)

[HT+05] Hamano, J. ; Torvalds, L. u. a.: *git - the fast version control system.* (2005). – http://git-scm.com/ (23.12.2012)

[IBMa] IBM: *IBM Rational Team Concert.* – https://jazz.net/projects/rational-team-concert/ (23.12.2012)

[IBMb] IBM: *IBM Tivoli software..* – http://www-01.ibm.com/software/tivoli/ (23.12.2012)

[IBMc] IBM: *Rational ClearCase.* – http://www-01.ibm.com/software/awdtools/clearcase/ (23.12.2012)

[ICO+09] Iborra, A. ; Caceres, D. ; Ortiz, F. ; Franco, J. ; Palma, P. ; Alvarez, B.: Design of service robots. In: *IEEE Robotics & Automation Magazine* 16 (2009), Nr. 1, S. 24–33

[ICQ98] ICQ LLC: *ICQ.* (1998). – http://www.icq.com/ (23.12.2012)

[IFF96] Ierusalimschy, R. ; Figueiredo, L. H. ; Filho, W. C.: Lua - an extensible extension language. In: *Software: Practice and Experience* 26 (1996), Nr. 6, S. 635–652

[Imi] Imixs Software Solutions GmbH: *Imixs Office Workflow - Software für Geschäftsprozessmanagement.* – http://www.office-workflow.de (23.12.2012)

[Inta] Intalio, Inc: *Intalio BPMS.* – http://www.intalio.com/bpms (23.12.2012)

[Intb] International Standard IEEE 610.12-1990: *IEEE Standard Glossary of Software Engineering Terminology*

[Intc] International Standard ISO 8373-2012: *Robots and robotic devices – Vocabulary*

[Intd] International Standard ISO/IEC 14977-1996: *Extended BNF*

[Int12] International Federation of Robotics (IFR) Statistical Department: *WORLD ROBOTICS 2011 - Servicerobots.* (2012). – http://www.worldrobotics.org (23.12.2012)

[IST] ISTI-CNR: *MeshLab - an open source, portable, and extensible system for the processing and editing of unstructured 3D triangular meshes..* – http://meshlab.sourceforge.net/ (23.12.2012)

[JBR99] Jacobson, I. ; Booch, G. ; Rumbaugh, J.: *The unified software development process*. Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc., 1999

[JH00] J. Hermann, T. W.: *The MoinMoin Wiki Engine*. (2000). – http://moinmo.in/ (23.12.2012)

[Joh91] Johansen, R.: Teams for tomorrow [groupware]. In: *Proceedings of the Twenty-Fourth Annual Hawaii International Conference on System Sciences, Jan 8-11, 1991, Kauai,HI*. Piscataway, NJ : IEEE, 1991, S. 521 –534 vol.3

[Joy78] Joy, W.: *C-Shell*. (1978). – http://linux.die.net/man/1/csh (23.12.2012)

[JUn] *JUnit*. – http://junit.sourceforge.net/ (23.12.2012)

[JWSM93] Jackson, I. ; Welsh, M. ; Streeter, C. ; Murdock, I.: *Package maintenance system for Debian.* (1993). – http://alioth.debian.org/projects/dpkg (23.12.2012)

[K+] Kumar, A. u. a.: *CUnit - A Unit Testing Framework for C*. – http://cunit.sourceforge.net/ (23.12.2012)

[Kai01] Kaiser, S.: Kommunikationsorientierte synchrone Werkzeuge. In: Unland, R. (Hrsg.): *CSCW-Kompendium. Lehr- und Handbuch zum computerunterstüzten kooperativen Arbeiten.* Berlin : Springer, 2001, S. 159–166

[Kar] Karlsruhe Institute of Technology (KIT), Humanoids and Intelligent Systems Lab - Institute for Anthropomatics: *The Armar Family.*. – http://his.anthropomatik.kit.edu/english/241.php (23.12.2012)

[KAW07] KAWADA Industries, Inc. : *Humanoid Robot HRP-3 Promet MK-II*. (2007). – http://global.kawada.jp/mechatronics/hrp3.html (23.12.2012)

[Kaw11] Kawaguchi, K.: *Jenkins Continuous Integration*. (2011). – http://jenkins-ci.org/ (23.12.2012)

[KB11] Klotzbücher, M. ; Bruyninckx, H.: Hard real-time Control and Coordination of Robot Tasks using Lua. Technical report / Katholieke Universiteit Leuven. 2011. – Forschungsbericht

[KHG+13] Klotzbücher, M. ; Hochgeschwender, N. ; Gherardi, L. ; Bruyninckx, H. ; Kraetzschmar, G. ; Brugali, D. ; Shakhimardanov, A. ;

Paulus, J. ; Reckhaus, M. ; Garcia, H. ; Faconti, D. ; Soetens., P.: The BRICS Component Model: A Model-Based Development Paradigm For Complex Robotics Software Systems. In: *Proceedings of the 28th ACM Symposium on Applied Computing (SAC). Track on Software Architecture: Theory, Technology, and Applications (SA-TTA), Coimbra, Portugal, March 18 - 22, 2013*. New York, NY, USA : ACM, 2013

[KI00] K. Ito, V. B.: *MantisBT*. (2000). – `http://www.mantisbt.org/` (23.12.2012)

[KIK03] Kichkaylo, T. ; Ivan, A. ; Karamcheti, V.: Constrained component deployment in wide-area networks using AI planning techniques. In: *Proceedings of International Parallel and Distributed Processing Symposium, April, 22-26, 2003, Nice, France*. Washington, DC : IEEE Computer Society Press, 2003, S. 10 ff.

[KKK+04] Kaneko, K. ; Kanehiro, F. ; Kajita, S. ; Hirukawa, H. ; Kawasaki, T. ; Hirata, M. ; Akachi, K. ; Isozumi, T.: Humanoid robot HRP-2. In: *Proceedings of the IEEE International Conference on Robotics & Automation (ICRA)*. Piscataway, NJ : IEEE, May 2004, S. 1083–1090

[Klo12] Klotzbücher, M.: *rFSM Coordination Statecharts*. (2012). – `http://people.mech.kuleuven.be/~mklotzbucher/rfsm/README.html` (23.12.2012)

[KMO+05] Kim, Bong K. ; Miyazaki, M. ; Ohba, K. ; Hirai, S. ; Tanie, K.: Web Services Based Robot Control Platform for Ubiquitous Functions. In: *Proceedings of the IEEE International Conference on Robotics & Automation (ICRA), April 18-22, 2005, Barcelona, Spain*. Piscataway, NJ : IEEE, 2005, S. 691 – 696

[Kro05] Kroll, Per: The Eclipse Process Framework project. (2005). – `http://www.ibm.com/developerworks/rational/library/05/1011_kroll/index.html` (23.12.2012)

[Kro07] Kroll, Per: OpenUP in a nutshell. (2007). – `http://www.ibm.com/developerworks/rational/library/sep07/kroll/index.html` (23.12.2012)

[Kru04] Kruchten, P.: *The Rational Unified Process, an introduction*. 3. Aufl. Amsterdam : Addison-Weasly Longman, 2004

[KUK] KUKA Roboter GmbH: *KUKA Remote Control.* – `http://www.kuka-robotics.com/germany/de/products/software/remotecontrol/` (23.12.2012)

[LaV00] LaValle, S. M.: *Motion strategy library.* (2000). – `http://msl.cs.uiuc.edu/msl/` (23.12.2012)

[LDG11] Lortal, G. ; Dhouib, S. ; Gérard, S.: Integrating ontological domain knowledge into a robotic DSL. In: *Proceedings of the 2010 international conference on Models in software engineering, Oct 2-8, 2010, Oslo, Norway.* Berlin u.a. : Springer, 2011, S. 401–414

[Leb] Lebherz, J.: *WinAPT-GET - The apt-get tool for Windows.* – `http://winapt-get.sourceforge.net/` (23.12.2012)

[Lit95] Litke, H.-D.: *Projektmanagement - Methoden, Techniken, Verhaltensweisen.* München : Carl Hanser Verlag, 1995

[LJH06] Li, X. ; Jin, Y. ; Hu, X.: An XML-Driven Component-Based Software Framework for Mobile Robotic Applications. In: *Proceedings of the 2nd IEEE/ASME International Conference on Mechatronic and Embedded Systems and Applications, Peking, Aug. 2006.* Piscataway, NJ : IEEE, 2006, S. 1 –6

[LPP04] Lacour, S. ; Pérez, C. ; Priol, T.: Deploying CORBA Components on a Computational Grid: General Principles and Early Experiments Using the Globus Toolkit. In: Emmerich, W. (Hrsg.) ; Wolf, A. (Hrsg.): *Component Deployment.* Berlin u.a. : Springer, 2004, S. 35–49

[LS06] Liu, Y. D. ; Smith, S. F.: A formal framework for component deployment. In: *Proceedings of the 21st annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications.* New York, NY, USA : ACM, 2006, S. 325–344

[LSS09] Lotz, A. ; Steck, A. ; Schlegel, C.: *SmartSoft.* (2009). – = http://sourceforge.net/projects/smart-robotics/ (23.12.2012)

[M+] Mackall, M. u. a.: *Mercurial.* – `http://mercurial.selenic.com/` (23.12.2012)

[MAJJ08] Mohamed, N. ; Al-Jaroodi, J. ; Jawhar, I.: Middleware for Robotics: A Survey. In: *Proceedings of the IEEE Conference on Robotics, Automation and Mechatronics, Sep 21-24, 2008, Chengdu, China.* Piscataway, NJ : IEEE, 2008, S. 736 –742

[Man10] Mancisidor, Ana L.: Continuous Integration Best Practices with Rational Team Concert / IBM Rational Technical Sales. 2010. – Forschungsbericht

[MAR09] *The UML Profile for MARTE: Modeling and Analysis of Real-Time and Embedded Systems.* (2009). – http://www.omg.org/spec/MARTE/1.0/PDF (23.12.2012)

[Mat] Mathworks, Inc: *Simulink: Simulation und Model-Based Design..* – http://www.mathworks.de/products/simulink/ (23.12.2012)

[Mat12] Matt, C.: Workflow-Management-Systeme. In: *Controlling & Management* 56 (2012), Nr. 1, S. 8–10

[MBK06] Makarenko, A. ; Brooks, A. ; Kaupp., T.: Orca: Components for Robotics. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'06) Workshop on Robotic Standardization, Peking, Oktober 2006.* Piscataway, NJ : IEEE, 2006

[Mica] Microsoft: *Microsoft Command-line Reference.* – http://msdn.microsoft.com/de-de/library/cc754340.aspx (23.12.2012)

[Micb] Microsoft: *Microsoft NetMeeting.* – http://microsoft-netmeeting.softonic.de/ (23.12.2012)

[Micc] Microsoft: *Microsoft System Center..* – http://www.microsoft.com/systemcenter/en/us/default.aspx (23.12.2012)

[Micd] Microsoft: *Microsoft Visual SourceSafe- The Version Control System for Development Teams Using Visual Studio .NET.* – http://msdn.microsoft.com/en-us/vstudio/aa700907.aspx (23.12.2012)

[Mice] Microsoft: *Microsoft Windows Update..* – http://windows.microsoft.com/de-DE/windows/help/windows-update (23.12.2012)

[Micf] Microsoft: *MSRS.* – http://www.microsoft.com/robotics/ (23.12.2012)

[Micg] Microsoft: *Skype.* – http://www.skype.com/ (23.12.2012)

[Mich] Microsoft: *Windows Server 2008: Remote Desktop Services.* – http://www.microsoft.com/en-us/server-cloud/windows-server/remote-desktop-services.aspx (23.12.2012)

[mici] microTOOL GmbH: *objectiF - das UML-Tool für modellgetriebene Entwicklung..* – http://www.microtool.de/objectif/de/index.asp (23.12.2012)

[Min06] Minium, D.: *Team Foundation Server Fundamentals: A Look at the Capabilities and Architecture.* (2006). – http://msdn.microsoft.com/en-us/library/ms364062%28v=vs.80%29.aspx (23.12.2012)

[MLKN09] Madhavan, R. ; Lakaemper, R. ; Kalmar-Nagy, T.: Benchmarking and standardization of intelligent robotic systems. In: *Proceedings of the 14th International Conference on Advanced Robotics (ICAR 2009), June 22-26, 2009, Munich.* Piscataway, NJ : IEEE, 2009, S. 1 –7

[MMR01] Medvidovic, N. ; Mikic-Rakic, M.: Architectural Support for Programming-in-the-Many. / USC-CSE-2001-506, University of Southern California. 2001. – Forschungsbericht

[MOR03] *MORPHA - Kommunikation, Interaktion und Kooperation zwischen Menschen und anthropomorphen Assistenzsystemen.* 1999-2003. – gefördert durch das Bundesministerium für Bildung und Forschung

[Moz] Mozilla Foundation: *Bugzilla.* – http://www.bugzilla.org/ (23.12.2012)

[MR09] Marino, J. ; Rowley, M.: *Understanding SCA (Service Component Architecture).* 1. Aufl. Amsterdam : Addison-Wesley Longman, 2009

[MRM02] Mikic-Rakic, M. ; Medvidovic, N.: Architecture-Level Support for Software Component Deployment in Resource Constrained Environments. In: *Proceedings of the IFIP/ACM Working Conference on Component Deployment, Berlin, Germany, June 20-21, 2002.* London, UK : Springer, 2002, S. 31–50

[MRT+08] Montemerlo, M. ; Roy, N. ; Thrun, S. u. a.: *CARMEN Robot Navigation Toolkit.* (2008). – http://carmen.sourceforge.net/ (23.12.2012)

[MSV+08] Metta, G. ; Sandini, G. ; Vernon, D. ; Natale, L. ; Nori, F.: The iCub humanoid robot: an open platform for research in embodied cognition. In: *Proceedings of the 8th Workshop on Performance Metrics for Intelligent Systems, Washington, D.C., USA, August 19-21, 2008.* New York, NY, USA : ACM, 2008, S. 50–56

[MTW09] Müller, M. ; Thurner, V. ; Wassermann, M.: Koordination von Teamarbeit mit Jazz. In: *Java Magazine* 7.2009 (2009), S. 100–103

[Nes07] Nesnas, I A D.: The CLARAty Project: Coping with Hardware and Software Heterogeneity. In: *Software Engineering for Experimental Robotics* 30 (2007), S. 31–70

[NFL10] Niemüller, T. ; Ferrein, A. ; Lakemeyer, G.: A lua-based behavior engine for controlling the humanoid robot nao. In: Baltes, J. (Hrsg.) ; Lagoudakis, M. G. (Hrsg.) ; Naruse, T. (Hrsg.) u. a.: *RoboCup 2009: Robot Soccer World Cup XIII*. Berlin u.a. : Springer, 2010, S. 240–251

[Nie09] Niemüller, Tim: *Developing A Behavior Engine for the Fawkes Robot-Control Software and its Adaptation to the Humanoid Platform Nao*, RWTH Aachen University, Knowledge-Based Systems Group, Master's Thesis, April 2009

[NMH+10] Notheis, S. ; Milighetti, G. ; Hein, B. ; Wörn, H. ; Beyerer, J.: Skill-based telemanipulation by means of intelligent robots. In: *IEEE/RSJ 2010 International Conference on Robots and Intelligent Systems, Oct. 18-22, 2010 Taipei, Taiwan*. Piscataway, NJ : IEEE, 2010, S. 5258–5263

[NMNF09] Nilsson, A. ; Muradore, R. ; Nilsson, K. ; Fiorini, P.: Ontology for robotics: a roadmap. In: *Proceedings of the 14th International Conference on Advanced Robotics (ICAR 2009), June 22-26, 2009, Munich*. Piscataway, NJ, USA : IEEE, 2009, S. 1–6

[NRP12] Nowak, W. ; Reiser, U. ; Prassler, E.: Robot development process in the DESIRE project. In: Prassler, E. (Hrsg.) ; Bischoff, R. (Hrsg.) ; Burgard, W. (Hrsg.) u. a.: *Towards Service Robots for Everyday Environments: Recent Advances in Designing Service Robots for Complex Tasks in Everyday Environments*. Berlin : Springer, 2012 (Springer Tracts in Advanced Robotics - STAR 76), S. 517–526

[O. 87] O. Laumann and J. Weigert and M. Schröder: *GNU Screen*. (1987). – `http://www.gnu.org/software/screen/` (23.12.2012)

[Obj06a] Object Management Group: CORBA Component Model 4.0 Specification / Object Management Group. Version: April 2006. `http://www.omg.org/docs/formal/06-04-01.pdf`. 2006 (Version 4.0). – Specification

[Obj06b] Object Management Group: *Deployment and configuration of component-based distributed applications specification*. (2006). – `http://www.omg.org/cgi-bin/doc?formal/06-04-02.pdf` (23.12.2012)

[Obj08] Object Management Group: Software & Systems Process Engineering Meta-Model Specification. (2008). – `http://www.omg.org/cgi-bin/doc?formal/2008-04-01` (23.12.2012)

[OJC+11] Osentoski, S. ; Jay, G. ; Crick, C. ; Pitzer, B. ; DuHadway, C. ; Jenkins, O. C.: Robots as web services: Reproducible experimentation and application development using rosjs. In: *Proceedings of the IEEE International Conference on Robotics & Automation (ICRA), Shanghai, China, 9-13 May 2011*. Piscataway, NJ : IEEE, 2011, S. 6078 – 6083

[Ope99] OpenBSD Foundation: *Secure shell (SSH).* (1999). – `http://www.openssh.com/` (23.12.2012)

[Oraa] Oracle: *Hudson Continuous Integration.* – `http://hudson-ci.org/` (23.12.2012)

[Orab] Oracle: *Java message service (JMS)..* – `http://www.oracle.com/technetwork/java/jms/index.html` (23.12.2012)

[Orac] Oracle: *Java web start technology..* – `http://www.oracle.com/technetwork/java/javase/javawebstart/index.html.`

[Orb] OrbiTeam Software GmbH & Co. KG: *BSCW - Basic Support for Cooperative Work.* – `http://www.bscw.de/` (23.12.2012)

[OSE94] *OSEK VDX.* 1994. – `http://www.osek-vdx.org/.` (23.12.2012)

[OW209] OW2 Consortium: *FraSCAti - a component framework providing runtime support for SCA.* (2009). – `http://wiki.ow2.org/frascati` (23.12.2012)

[PB01] Pankoke-Babatz, U.: Kommunikationsorientierte asynchrone Werkzeuge. In: Unland, R. (Hrsg.): *CSCW-Kompendium. Lehr- und Handbuch zum computerunterstützten kooperativen Arbeiten.* Berlin : Springer, 2001, S. 167–173

[PC06] Pontisso, N. ; Chemouil, D.: TOPCASED Combining Formal Methods with Model-Driven Engineering. In: *Proceedings of the 21st IEEE/ACM International Conference on Automated Software Engineering, Sept 18-22, 2006, Tokyo, Japan.* Piscataway, NJ : IEEE, 2006, S. 359 –360

[PCCW95] Paulk, Mark C. ; Curtis, Bill ; Chrissis, Mary B. ; Weber, Charles V.: *The Capability Maturity Model : Guidelines for improving the Software Process.* Boston : Addison-Wesley Professional, 1995

[Pil04] Pilato, Michael: *Version Control With Subversion.* Sebastopol, CA, USA : O'Reilly & Associates, Inc., 2004

[Pra09] Pradeep, V.: *ROS - actionlib.* (2009). – `http://www.ros.org/wiki/actionlib` (23.12.2012)

[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: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2009)

[RCF+09] Reiser, U. ; Connette, C. ; Fischer, J. ; Kubacki, J. ; Bubeck, A. ; Weisshardt, F. ; Jacobs, T. ; Parlitz, C. ; Hägele, M. ; Verl, A.: Care-O-bot 3 - Creating a product vision for service robot applications by integrating design and technology. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems : Oct 11-15, 2009 St. Louis, Missouri, USA*. Piscataway, NJ : IEEE Computer Society Press, 2009, S. 1992–1997

[Rea] RealVNC Limited: *RealVNC.*. – `http://www.realvnc.com/` (23.12.2012)

[Red97] Red Hat: *RPM package manager.* (1997). – `http://www.rpm.org/` (23.12.2012)

[Rei01] Reinhold, M.: Rational Unified Process 2000 versus V-Modell'97: A Comparison of the two most common used Process Models in Germany. In: Kneuper, R. (Hrsg.): *Leichte Vorgehensmodelle - 8. Workshop der Fachgruppe 5.11 der Gesellschaft für Informatik e.V.* Aachen : Verlag Shaker, 2001, S. 111–128

[Rei12] Reiser, U.: DESIRE WEB 2.0 - Integration Management and Distributed Software Development for Complex Service Robots. In: Prassler, E. (Hrsg.) ; Bischoff, R. (Hrsg.) ; Burgard, W. (Hrsg.) u. a.: *Towards Service Robots for Everyday Environments: Recent Advances in Designing Service Robots for Complex Tasks in Everyday Environments* Bd. 76. Berlin : Springer, 2012, S. 517–526

[RGS+08] Riccobene, E. ; Gargantini, A. ; Scandurra, P. u. a.: *ASMETA Toolset.* (2008). – `http://asmeta.sourceforge.net/` (23.12.2012)

[Rie00] Riehle, Dirk: *Framework Design: A Role Modeling Approach.* Zürich, Switzerland, ETH Zürich, Diss., 2000

[Riv] Riverbank Computing Limited: *SIP.* – `http://www.riverbankcomputing.co.uk/software/sip/intro` (23.12.2012)

[RKPV09] Reiser, U. ; Klauser, R. ; Parlitz, C. ; Verl, A.: DESIRE WEB 2.0 - Integration Management and Distributed Software Development for Service Robots. In: *Proceedings of the 14th International Conference on Advanced Robotics (ICAR 2009), June 22-26, 2009, Munich.* Stuttgart : Fraunhofer Verlag, 2009, S. 517–526

[RMB00] Ruh, W. A. ; Maginnis, F. X. ; Brown, W. J.: *Enterprise Application Integration: A Wiley Tech Brief.* 1. Aufl. Newy York, NY : John Wiley & Sons, 2000

[RMP08] Reiser, U. ; Mies, C. ; Plagemann, C.: Verteilte Software-Entwicklung in der Robotik - ein Integrations- und Testframework. In: *Robotik 2008. Leistungsstand, Anwendungen, Visionen, Trends, 11. und 12. Juni 2008, München.* Düsseldorf : VDI-Verlag, 2008, S. 175–178

[Rob11] Robert Bosch LLC: *PR2 Remote Lab.* (2011). – `http://pr2-remotelab.com/doku.php?id=start` (23.12.2012)

[Roy70] Royce, Walker W.: Managing the development of large software systems: concepts and techniques. In: *Proceedings of the IEEE Western Electronic Show and Convention (WESCON),Aug 25-28, 1970, Los Angeles.* Piscataway, NJ : IEEE, 1970, S. 1–9. – Reprinted in *Proceedings* of the Ninth International Conference on Software Engineering, March 1987, S. 328–338

[Roy87] Royce, W. W.: Managing the development of large software systems: concepts and techniques. In: *ICSE '87: Proceedings of the 9th international conference on Software Engineering.* Los Alamitos, CA, USA : IEEE Computer Society Press, 1987, S. 328–338

[Roz] Rozental, G.: *Boost Test Library.* – `http://www.boost.org/doc/libs/1_42_0/libs/test/doc/html/index.html` (23.12.2012)

[RSA11] Riccobene, E. ; Scandurra, P. ; Albani, F.: A Modeling and Executable Language for Designing and Prototyping Service-Oriented Applications. In: *Proceedings of the 2011 37th EUROMICRO Conference on Software Engineering and Advanced Applications.* Washington, DC, USA : IEEE Computer Society, 2011, S. 4–11

[SB01] Schwaber, K. ; Beedle, M.: *Agile Software Development with Scrum.* 1. Aufl. Upper Saddle River, NJ, USA : Prentice Hall PTR, 2001

[Sch06a] Scheutz, M.: ADE - Steps Towards a Distributed Development and Runtime Environment for Complex Robotic Agent Architectures. In: *Applied Artificial Intelligence* 20 (2006), Nr. 2-4, S. 275–304

[Sch06b] Schlegel, Christian: Communication Patterns as Key Towards Component-Based Robotics. In: *International Journal of Advanced Robotic Systems* 3 (2006), März, Nr. 1, S. 49–54

[SGH+97] Simmons, R. ; Goodwin, R. ; Haigh, K. ; Koenig, S. ; O'Sullivan, J.: A Layered Architecture for Office Delivery Robots. In: *Proceedings of the First International Conference on Autonomous Agents, Feb 05-08, 1997, Marina del Rey, CA*. New York, NY : ACM, 1997, S. 245–252

[SHLS09] Schlegel, C. ; Hassler, T. ; Lotz, A. ; Steck, A.: Robotic Software Systems: From Code-Driven to Model-Driven Designs. In: *Proceedings of the 14th International Conference on Advanced Robotics (ICAR 2009), June 22-26, 2009, Munich*. Piscataway, NJ, USA : IEEE, 2009, S. 1–8

[SJJK08] Song, B. ; Jung, S. ; Jang, C. ; Kim, S.: An Introduction to Robot Component Model for OPRoS. In: *International Conference on Simulation, Modeling and Programming for Autonomous Robots (SIMPAR), Workshop on Standards and Common Platform for Robotics, Nov 3, 2008, Venice, Italy*. Berlin : Springer (Lecture Notes in Artificial Intelligence), 2008

[SK08] Siciliano, Bruno (Hrsg.) ; Khatib, Oussama (Hrsg.): *Springer Handbook of Robotics*. New York, Inc. : Springer, 2008

[SKAD01] Scholl, K. U. ; Kepplin, V. ; Albiez, J. ; Dillmann, R.: Developing robot prototypes with an expandable modular controller architecture. In: *Proceedings of the International Conference on Intelligent Autonomous Systems, June 2000, Venedig*. Amsterdam u.a. : IOS Press, 2001, S. 67–74

[Soe02] Soetens, P.: *The OROCOS/RTT Component Model*. (2002). – `http://people.mech.kuleuven.be/~orocos/pub/documentation/rtt/v1.12.x/doc-xml/orocos-components-manual.html` (23.12.2012)

[Som06] Sommerville, I.: *Software Engineering*. 8. Auflage. Harlow, England : Addison-Wesley, 2006

[SRS13] *Multi-Role Shadow Robotic System for Independent Living (SRS)*. 2010-2013. – 7th Framework Programme (FP7), Challenges 7: Independent living, inclusion and Governance

[SS10] Steck, A. ; Schlegel., C.: SmartTCL: An Execution Language for Conditional Reactive Task Execution in a Three Layer Architecture for Service Robots. In: *Workshop on DYnamic languages for RObotic and Sensors systems (DYROS/SIMPAR), Nov. 15-18, 2010, Darmstadt*. Berlin : Springer, 2010, S. 274–277

[SSBK10] Schlegel, C. ; Steck, A. ; Brugali, D. ; Knoll, A.: Design Abstraction and Processes in Robotics: From Code-Driven to Model-Driven Engineering.

In: Ando, N. (Hrsg.) ; Balakirsky, S. (Hrsg.) ; Hemker, T. (Hrsg.) u. a.: *Simulation, Modeling, and Programming for Autonomous Robots, Nov. 15-18, 2010, Darmstadt*. Berlin u.a. : Springer, 2010 (Lecture Notes in Computer Science vol. 6472), S. 324–335

[SSL12] Schlegel, C. ; Steck, A. ; Lotz, A.: Robotic Software Systems: From Code-Driven to Model-Driven Software Development. In: Dutta, A. (Hrsg.): *Robotic Systems - Applications, Control and Programming*. New York, NY : Intech, 2012, S. 473–502

[SSU01] Schwabe, G. ; Streitz, N. ; Unland, R. ; Unland, R. (Hrsg.): *CSCW-Kompendium: Lehr- und Handbuch zum computerunterstützten kooperativen Arbeiten*. Berlin : Springer, 2001

[SVE+07] Stahl, T. (Hrsg.) ; Völter, M. (Hrsg.) ; Efftinge, S. (Hrsg.) u. a.: *Modellgetriebene Softwareentwicklung: Techniken, Engineering, Management*. 2. Aufl. Heidelberg : dpunkt, 2007

[Sym] Symantec: *Altiris deployment solution*. – `http://www.symantec.com/business/deployment-solution` (23.12.2012)

[Szy02] Szyperski, Clemens: *Component Software: Beyond Object-Oriented Programming*. MA : Addison-Wesley, 2002

[TB09] Tenorth, Moritz ; Beetz, Michael: KNOWROB: knowledge processing for autonomous personal robots. In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), October 11-15, 2009, St. Louis, MO, USA*. Piscataway, NJ, USA : IEEE Press, 2009 (IROS'09), 4261–4266

[Tea] TeamViewer GmbH: *TeamViewer - die All-In-One Software für Fernwartung und Online Meeting*. – `http://www.teamviewer.com/de/index.aspx` (23.12.2012)

[T.G07] T.Gross, M. K. ; Herczeg, Michael (Hrsg.): *Computer-Supported Cooperative Work*. Oldenburg Verlag, 2007

[TIB] TIBCO Software Inc: *TIBCO Rendezvous.*. – `http://www.tibco.com/products/automation/messaging/low-latency/rendezvous/default.jsp` (23.12.2012)

[TVRI+12] Tsui, K. M. ; Von Rump, S. ; Ishiguro, H. ; Takayama, L. ; Vicars, P. N.: Robots in the loop: telepresence robots in everyday life. In: *Proceedings*

of the seventh annual ACM/IEEE international conference on Human-Robot Interaction, Boston, Massachusetts, USA. New York, NY, USA : ACM, 2012 (HRI '12), 317–318

[V+] Vidal, S. u. a.: *Yum: Yellow dog updater.*. – `http://yum.baseurl.org/` (23.12.2012)

[Vira] VirtualGL: *TurboVNC.*. – `http://www.virtualgl.org/Downloads/TurboVNC` (23.12.2012)

[Virb] VirtualGL: *User's Guide for VirtualGL 2.2.* – `http://www.virtualgl.org/vgldoc/2_2/` (23.12.2012)

[Virc] VirtualGL: *Using VirtualGL with the VGL Transport.*. – `http://www.virtualgl.org/vgldoc/2_2/#hd007` (23.12.2012)

[Vird] VirtualGL: *VirtualGL - 3D without Boundaries.* – `www.virtualgl.org` (23.12.2012)

[VM12] Vohr, F. ; May, T.: Robot hardware Design in the DESIRE project. In: Prassler, E. (Hrsg.) ; Bischoff, R. (Hrsg.) ; Burgard, W. (Hrsg.) u. a.: *Towards Service Robots for Everyday Environments*. Springer, 2012 (Springer Tracts in Advanced Robotics - STAR 76), S. 11–18

[VN08] Verl, A. ; Naumann, M.: Automatic generation of executable code for a robot cell using UPnP and XIRP. In: *Proceedings of the 5th International Conference on Informatics in Control, Automation and Robotics (ICINCO), 11-15 May, 2008, Funchal, Portugal*. Setubal, Portugal : INSTICC Press, 2008, S. 242–248

[VNE+00] Volpe, R. ; Nesnas, I. ; Estlin, T. ; Mutz, D. ; Petras, R. ; Das, H.: CLARAty: Coupled layer architecture for robotic autonomy / Jet Propulsion Laboratory, California Institute of Technology. 2000. – Forschungsbericht

[Wat09] Watts, K.: *ROS diagnostics*. (2009). – `http://www.ros.org/wiki/diagnostics` (23.12.2012)

[WFBS04] Wrede, S. ; Fritsch, J. ; Bauckhage, C. ; Sagerer, G.: An XML Based Framework for Cognitive Vision Architectures. In: *Proceedings of the 17th International Conference on Pattern Recognition, August 23-26, 2004, Cambridge, UK* Bd. 1. Los Alamitos, CA, USA : IEEE Computer Society, 2004, S. 757–760

[WFT+01] Witten, I. ; Frank, E. ; Trigg, L. ; Hall, M. ; Holmes, G. ; Cunningham., S.: *Weka: Practical machine learning tools and techniques with java implementations*. (2001). – http://www.cs.waikato.ac.nz/ml/weka/ (23.12.2012)

[Wik03] Wikimedia Foundation: *MediaWiki*. (2003). – http://www.mediawiki.org/wiki/MediaWiki (23.12.2012)

[Wil10] Willow Garage: *PR2 Manual*. (2010). – http://pr2support.willowgarage.com/wiki/PR2%20Manual (23.12.2012)

[WLH99] Wörn, H. ; Längle, T. ; Heck, F.: Distributed Diagnosis and Teleservice for Automated Production Cells. In: *Proceedings of the European Control Conference - ECC' 99, Karlsruhe, Germany, August 31 - September 3 1999*. Düsseldorf : VDI/VDE-Gesellschaft Meß- und Automatisierungstechnik -GMA-, 1999, S. 930–936

[Wor11] *World Robotics Report 2011 - Service Robots*. Frankfurt, Germany : IFR Statistical Department, 2011

[WRPV10] Weisshardt, F. ; Reiser, U. ; Parlitz, C. ; Verl, A.: Making High-Tech Service Robot Platforms Available. In: Neumann, Kristin (Hrsg.) u. a.: *International Federation of Robotics u.a.: Joint International Conference of ISR/ROBOTIK2010*. Offenbach u.a. : VDE-Verlag, 2010, S. 1115–1120

[XMP] XMPP Standards Foundation: *XMPP Software Clients*. – http://xmpp.org/xmpp-software/clients/ (23.12.2012)

10 Anhang A: Referenz-Fallstudie DESIRE

Im folgenden Kapitel wird das vom BMBF geförderte Forschungsprojekt „Deutsche Service-Robotik Initiative“, DESIRE (2005-2009) [DES09], der Leitinnovation Service-Robotik beschrieben, das die in Kapitel 1.1 dargelegten Problemstellungen sehr gut verdeutlicht. Weiterhin wurden teilweise Implementierungen des in dieser Arbeit vorgestellten Lösungsansatzes in diesem Projekt eingesetzt, so dass sich Teile des Kapitels Auswertung 7 darauf beziehen.

10.1 Struktur und Chronologie des Projektes

Ziel des Projektes war ein Sprung der Service-Robotik in Richtung Alltagstauglichkeit, einhergehend mit der Konvergenz von Technologien sowie der Schaffung einer Referenzarchitektur. Diese Ziele sollten mit einem Konsortium umgesetzt werden, das aus 14 Industrie- und Forschungs-Partnern bestand. Alle Hardware- und Software-Komponenten der Partner sollten auf einer gemeinsamen Technologieplattform integriert werden, um den gesetzten Zielen gerecht zu werden. Durch diese Randbedingung eignet sich das Projekt sehr gut als Anschauungs- und Evaluationsobjekt für die Problemstellung dieser Arbeit.

Die Komponenten der Partner stammten aus den Themenfeldern:

- Wahrnehmung
- Architektur und Ontologien
- Manipulation
- Mechatronische Schlüsselkomponenten
- Mobilität und Navigation
- Interaktion und Lernen

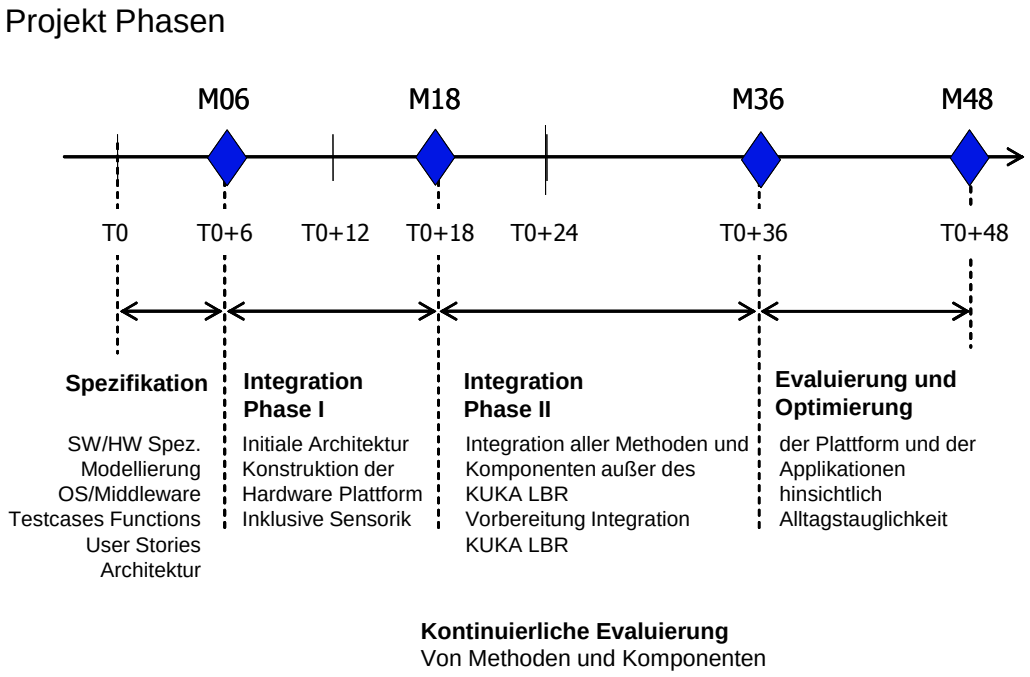


Abbildung 10.1: Geplante Projektphasen des DESIRE Projekts

.

Mit diesen heterogenen Komponenten sollte die Alltagstauglichkeit in Szenarien wie „Tisch abräumen“ oder „Zimmer aufräumen“ demonstriert werden.

Abb. 10.1 zeigt den geplanten Ablauf des Projektes über die Laufzeit von 4 Jahren. Das Projekt wurde in vier Hauptphasen aufgeteilt, die jeweils durch einen Projektmeilenstein - bestehend aus einem Spezifikationsdokument, der Fertigstellung der Hardware bzw. der Demonstration von implementierten Applikationen - abschlossen.

Zu Beginn des Projekts stand die Erstellung eines detaillierten Spezifikationsdokuments, das aus den Zielszenarien abgeleitete Basis-Funktionen des Technologieträgers samt Schnittstellenspezifikationen enthielt. Auf Basis dieses Dokumentes erfolgte die Überprüfung aller Funktionen auf Vollständigkeit und Plausibilität sowie die Aufstellung eines Integrationsplans. Die Vorgehensweise zur Erstellung des Spezifikationsdokuments basierte vor allem auf der Durchführung von Workshops mit den Projektpartnern, in denen zunächst die Zielapplikationen und daraus “generische Funktionen” abgeleitet wurden. Die Schnittstellen dieser generischen Funktionen wurden in der Schnittstellenbeschreibungsssprache IDL spezifiziert, so dass die Spezifikation formal auf Vollständigkeit bzw. Redundanz überprüft werden konnte.

In der ersten Integrationsphase wurde zunächst eine erste Version der Technologieplattform erstellt, die aus einem zweiarmigen Torso mit integrierten Händen bestand, der fix auf ein statisches Tragegestell montiert war. Weiterhin wurde in dieser Phase eine neue Software-Architektur entwickelt, die den Anforderungen (Niveau der Schnittstellen, Taktung der Kommunikation, etc.) aller zu integrierenden Komponenten erfüllen sollte. In der verbleibenden Zeit begann die Integration der einzelnen Komponenten (mit Basisfunktionalitäten) in diese Zielarchitektur und auf der Technologieplattform entsprechend der Schnittstellenspezifikationen. 18 Monate nach Projektstart konnte ein erster integrierter Test zum Meilenstein durchgeführt werden.

Die Vorgehensweise für diese erste Integration bestand zunächst in der Durchführung großer mehrtägiger Integrationstreffen, bei denen jeweils fast alle Projektpartner anwesend sein mussten, um Fortschritte erzielen zu können. Zum Meilensteins dieser ersten Integrationsphase konnte ein einfaches Szenario gezeigt werden, das alle integrierten Komponenten enthielt. Es bestand darin, dass der Roboter auf ein vom Benutzer durch einen Sprachbefehl spezifiziertes Objekt zeigte. Weiterhin war der Roboter in der Lage, Personen in einem gewissen Radius mit dem Sensorkopf zu verfolgen. Abgesehen von der Mobilität konnte zu diesem Zeitpunkt festgestellt werden: Alle Haupt-Komponenten (Manipulation, d.h. Arm- und Greiferansteuerung und Bewegungsplanung, Situationsanalyse, d.h.Personenidentifikation und -verfolgung, Szenenanalyse, d.h. Objekterkennung und -lokalisierung, Nutzerinterface, d.h. Spracherkennung sowie koordinative Komponenten, Planer, Eigenmodell und Ablaufsteuerung, sind in einer Basisversion in das Gesamtsystem integriert. Nach diesem Meilenstein ging die Projektleitung davon aus, dass die Partner sich in der restlichen Projektlaufzeit nun weitgehend unabhängig voneinander voll auf die Weiterentwicklung der einzelnen Komponenten konzentrieren konnten.

Abbildung 10.2 verdeutlicht jedoch, dass dies eine grobe Fehleinschätzung war. Dies ist vor allem auf die mangelhafte Effizienz der verteilten Entwicklung und Zusammenarbeit auf einer gemeinsamen Hardware (siehe Punkt 3 in 1.1) zurückzuführen. Anmerkung: Die dargestellte Integrationsarbeit in Abbildung 10.2 wurde von den gemessenen Aktivitäten im gemeinsamen (Integrations-)Repository abgeleitet (s. Abbildung 7.1).

Der Integrationsfortschritt erfolgte jedoch auch nach dem ersten Meilenstein fast ausschließlich während der Integrationstreffen. Die Arbeiten in der dritten Projektphase konzentrierten sich auf die Integration der Mobilität, sowohl hardwaretechnisch als auch softwaretechnisch. Wieder stellte sich ein Integrationsthema in den Vordergrund, das in der Arbeitsplanung nicht berücksichtigt war und sich auch keiner Komponente direkt zuordnen ließ: die Umrechnung von Objekt- bzw. Personenkoordinaten bezüglich verschiedener Koordinatensysteme (insbesondere von Kamerakoordinaten nach Weltkoordinaten). Weiterhin waren hierbei intensive Arbeitstreffen erforderlich, die die Reisebudgets der Partner frühzeitig

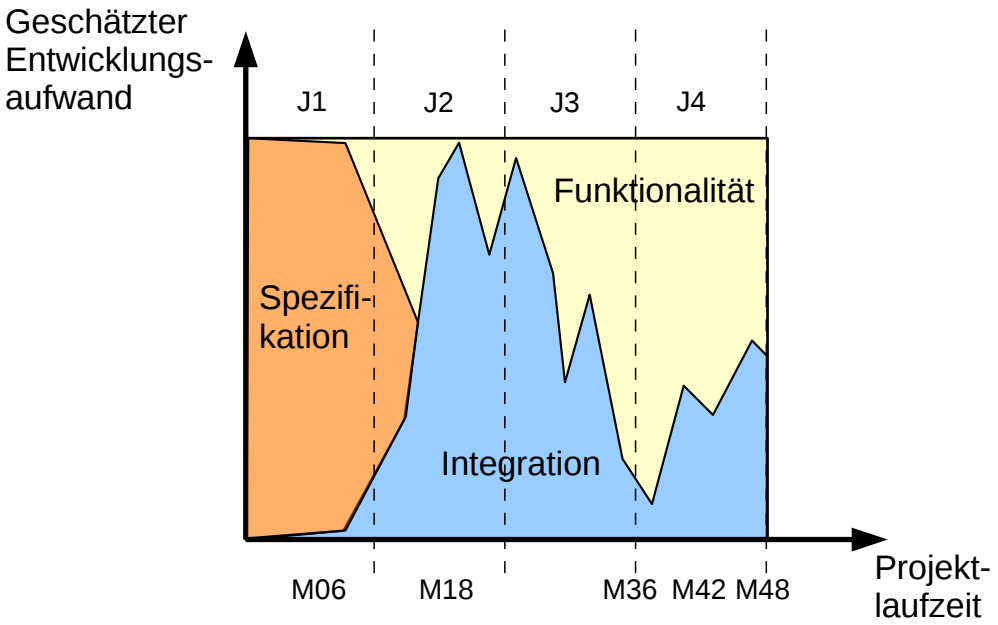


Abbildung 10.2: Anteile von Spezifikation, Integration und Funktionsentwicklung am Gesamtaufwand

leerten. Deshalb wurde ein Virtuelles Netzwerk eingerichtet (VPN), das den Entwicklern zwecks Integrationstests den Fernzugriff auf den Roboter ermöglichen sollte. Allerdings stellte sich schnell heraus, dass sich - insbesondere für integrierte Tests- viele Entwickler vor Ort bereithalten mussten, z.B. um ihre Komponenten jeweils lokal zu starten und zu stoppen. In der dritten Phase wurden weiterhin Benchmarks und Leistungstests entwickelt, um den Fortschritt der Entwicklung überprüfen sowie im weiteren Verlauf des Projekts eine Aussage hinsichtlich der Alltagstauglichkeit des Systems treffen zu können.

Die letzte Phase stand unter den Vorzeichen Optimierung und Evaluierung. Die Optimierung bezog sich dabei sowohl auf die Hardwareplattform, die in der letzten Phase hinsichtlich Größe, Robustheit und einfacher Benutzbarkeit überarbeitet wurde, als auch auf die einzelnen Software- komponenten und das integrierte Gesamtsystem. Diese Optimierung sowie die Entwicklung von Applikationen für einen CeBIT-Auftritt und den Abschlussmeilenstein des Projekts (Abräumen von Haushaltsobjekten auf einem Tisch) erforderte weiterhin kontinuierliche Integrationstests. Diese konnten durch die Einführung der Integrations- und Testplattform erheblich effizienter gestaltet werden, so dass die Anzahl der Integrationstreffen gegen Projektende deutlich reduziert werden konnte.

Bezüglich des verteilten Entwicklungs- und Integrationsprozesses in DESIRE wurden während der Projektlaufzeit folgende Beobachtungen gemacht [RKPV09]:

- Kleine Änderungen an den Schnittstellen während der Integrationstreffen nahmen viel Zeit in Anspruch

- Oft konnte nur ein Entwickler an einer Aufgabe arbeiten, von deren Fertigstellung alle anderen abhängig waren.
- Für das Debuggen einer Komponente waren oft viele Personen erforderlich.
- Viele Komponenten konnten nur durch deren Entwickler betrieben werden, so dass die personellen Abhängigkeiten sehr hoch waren.
- Obwohl ein gemeinsames Software-Repository existierte, war die jeweils aktuellste Version meist nur auf den Rechnern der Entwickler verfügbar. Dies verschärfte die Abhängigkeitssituation weiter.
- Falls der Entwickler einer zentralen Komponente z.B. durch Krankheit kurzfristig ausfiel, war das Ziel eines Integrationstreffen oft nicht mehr erreichbar.
- Selbst wenn alle aktuellen Versionen der Komponenten auf der Plattform installiert waren, konnte das Gesamtsystem sehr bald nur noch bedient werden, wenn alle Komponentenentwickler vor Ort waren, da die Einrichtung, Aktivierung und Konfiguration aller Komponenten unterschiedlich waren.
- Da es keinen gemeinsamen Integrationsprozess gab, hatte fast jede Komponente eine unterschiedliche Implementierung der Anbindung an das Gesamtsystem. Dadurch gab es keine Synergieeffekte bei der Integration für die verschiedenen Partner.
- Zwischen den Integrationstreffen gab es kaum einen Fortschritt im Integtrationsplan: Notwendige Tests (wenn auch nur einfache Schnittstellentests) erforderten meist andere Projektparter, so dass die Weiterentwicklung bis zum nächsten Integrationstreffen verzögert wurde.
- Die geplanten Zeit- und Kostenaufwände der einzelnen Partner für die Integration wurden insgesamt weit überschritten.

Viele dieser Beobachtungen, insbesondere diejenigen zur Teameffizienz, sind nicht neu und treffen in einer ähnlichen Art wohl auf die meisten Entwicklungs-Projekte zu. Ein Besonderheit von DESIRE war jedoch, dass durch die gemeinsame Hardware eine räumlich getrennte Entwicklung bzw. Integration sehr schwierig war. Diese Besonderheit trifft jedoch auf die meisten größeren Service-Robotik Projekte zu. Die hohe Anzahl der Integrationstreffen war somit hauptsächlich den hohen personellen Abhängigkeiten unter den Komponentenentwicklern geschuldet, die auch über die Einführung des Fernzugriffes über VPN nicht beseitigt werden konnte. Erst mit der Einführung des Webportals, das die Steuerung aller integrierten Komponenten und somit eine einfachere Bedienung des komplexen Roboters erlaubte, konnten die personellen Abhängigkeiten maßgeblich reduziert werden.

10.2 Messwerte und Parameter für Aufwandsschätzung für die Integration der DESIRE Komponenten

Das Forschungsprojekt war charakterisiert durch die folgenden Punkte, die sich in den Parameterwerten des Schätzmodells widerspiegeln:

- einen für Forschungsprojekte typischen hohen Innovationsgrad und somit eine relativ niedrige Präzedenz
- eine teils niedrige Teamkohäsion durch hohe Personalfluktuation und eine schwach ausgeprägte gemeinsame Zielvorstellung
- fehlende Prozesse (typisch für ein Forschungsprojekt)
- eine hohe Komplexität der einzelnen Komponenten sowie eine hohe Abhängigkeit zwischen diesen
- die Neuentwicklung von Projektplattform und -architektur und damit einer naturgemäß fehlenden Vorerfahrung der Entwickler mit der Zieldomäne
- häufige Änderungen der Projektplattform (Hardware) und Architektur bzw. Middleware (Software)
- einen inhomogenen und teilweise komplett fehlenden Einsatz von Werkzeugen (insbesondere in den ersten drei Jahren)
- einen singulären Standort der Zielhardware und somit einer hohen zusätzlichen Bürde durch die räumliche Verteilung des Teams

Tabelle 10.1 stellt die COCOMO II Skalierungs- und Kostentreiberfaktoren dar, die aus den oben genannten Punkten mit Hilfe des COCOMO II Modellierungs-Handbuchs [BCC+00] abgeleitet wurden.

Tabellen 10.2 und 10.3 beinhalten die Parameter der Integrationskostenschätzung für die DESIRE-Software-Komponenten Die dazu notwendigen SLOC-Werte f wurden mit Hilfe des Quellcode-Zeilenzählers CLOC [D+] ermittelt.

Tabelle 10.1: Konkrete Werte der COCOMO II Skalierungsfaktoren (links) und Kostentreiberfaktoren (rechts) für DESIRE

Parameter	Wert
PREC	4,96
FLEX	1,01
RESL	2,83
TEAM	4,38
PMAT	6,24

Parameter	Wert
RELY	1
CPLX	1,34
DOCU	0,81
DATA	1
RUSE	1
TIME	1
PVOL	1,15
STOR	1
ACAP	1
PCON	1,12
PCAP	0,88
APEX	1,10
PLEX	1,19
LTEX	1,09
TOOL	1,17
SCED	1
SITE	1,22

Tabelle 10.2: Konkrete Werte für die Parameter der Integrationskostenschätzung aus DESIRE - Wahrnehmungskomponenten

Parameter	SR	GR	ORB	Sit	FR	SA	IO	OOM
BRAK	15	20	20	20	20	20	20	20
CM	20	0	0	1	0	0	0	0
DM	10	0	0	0	0	0	0	0
IM	20	0	0	0	0	0	0	0
KASLOC	0	0	0	110	0	0	0	0
KNSLOC	0,3	1	1	3,5	0,9	0,9	0,7	1
SU	30	0	0	30	0	0	0	0
UNFM	0,2	0,2	0,2	0,2	0,2	0,2	0,2	0,2

SR - Speech Recognition, GR - Gesture Recognition, ORB - Object Recognition Bochum, Sit - Situationsanalyse, FR - Face Recognition, SA - Scene Analysis, IO - IpaObstacle (Hindernismodellierung), OOM - Online Object Modelling

Tabelle 10.3: Konkrete Werte für die Parameter der Integrationskostenschätzung aus DESIRE - Koordination und Aktorikkomponenten

Parameter	HC	M	DB	PL	AS	EM	FW
BRAK	15	20	20	20	40	30	10
CM	20	0	5	20	40	40	0
DM	10	0	1	20	20	30	0
IM	20	0	1	20	20	20	0
KASLOC	0,2	0	32	7,3	11,3	1.5	0
KNSLOC	0,9	1	0,3	1,2	1,1	2,7	1,1
SU	30	0	10	30	30	30	0
UNFM	0,2	0,2	0,2	0,2	0,2	0,2	0,2
A				2,94			
B				0,91			

HC - Head Control, M - Manipulation (Arme, Hände, Bahnplahnung), DB - Objektdatenbank, PL - symbolischer Planer, AS - Ablaufsteuerung, EM - Eigenmodell, FW - Fahrwerk

Charakteristisch für die Entwicklung von Robotersystemen ist ein sehr hoher Integrationsanteil bestehender Technologien. Insbesondere für komplexe Serviceroboter, die eine immer breitere Anwendung außerhalb der Produktion finden, werden Methoden zur Vereinfachung der Systemintegration benötigt. Serviceroboter kommen in der Regel in dynamischen Umgebungen zum Einsatz und sind daher im Vergleich zu Industrierobotern mit höheren Unsicherheiten konfrontiert. Diese informatorischen Defizite müssen mit geeigneten Sensoren und entsprechender Auswertesoftware kompensiert werden. Die steigenden Anforderungen an die Autonomie und Mensch-Roboter-Interaktion dieser Systeme erhöhen die Anzahl der benötigten Softwarekomponenten weiter. Aufgrund der Diversität der erforderlichen Technologien werden Serviceroboter häufig in multi-disziplinären Teams an verteilten Standorten entwickelt. Um eine effiziente Entwicklung und eine Trennung der Entwicklerrollen zu ermöglichen, sind Technologien zur Abstraktion der Systemkomplexität erforderlich. In dieser Arbeit werden Methoden und Werkzeuge zur Unterstützung des verteilten Entwicklungs- und Integrationsprozesses komplexer Serviceroboter behandelt.

ISBN 978-3-8396-0675-9

9 783839 606759

FRAUNHOFER VERLAG