Universität Stuttgart
Fakultät Informatik, Elektrotechnik und Informationstechnik

Studiengang	Softwaretechnik
Prüfer	Prof. Dr. Kurt Rothermel
Betreuer	Dr. Christian Becker
	Dipl.-Inf. Marcus Handte
	Dipl.-Inf. Gregor Schiele
Beginn	1. Oktober 2003
Ende	1. April 2004
CR-Klassifikation	C.2.4, D.2.4, D.2.11, D.3.3

Diplomarbeit Nr. 2142

Entwicklung eines geräteübergreifenden Adaptionsmechanismus für komponentenbasierte Anwendungen in ubiquitären Rechnersystemen

Stephan Urbanski

Institut für Parallele und Verteilte Systeme
Abteilung Verteilte Systeme
Universitätsstr. 38
D-70569 Stuttgart

Kurzfassung

Eine herausragende Eigenschaft ubiquitärer Anwendungen ist deren Anpassungsfähigkeit an eine sich ändernde Umgebung. Die Anpassung von Anwendungen kann durch geeignete ubiquitäre Verteilungsinfrastrukturen automatisch durchgeführt werden. Durch automatische Anwendungskonfiguration entstehen für die Verteilungsinfrastruktur Entscheidungsfreiräume, die es effizient zu nutzen gilt.

Unter dieser Prämisse werden in dieser Arbeit verschiedene Möglichkeiten der Anwendungsadaption sowie diverse komponentenbasierte, ubiquitäre Verteilungsinfrastrukturen untersucht. Im Anschluss wird ein Konzept zur Nutzung der Adaptionsfreiräume in der an der Universität Stuttgart entwickelten komponentenbasierten, ubiquitären Verteilungsinfrastruktur PCom erarbeitet. Das Konzept basiert auf der Bereitstellung einer anwendungsspezifischen Informationsquelle - eine anwendungsumfassende und somit knotenübergreifende Sicht in Form einer Anwendungsbeschreibung durch den Anwendungsprogrammierer. Der Aufbau, der Umfang und die Integration in die bestehende Verteilungsinfrastruktur werden im Folgenden thematisiert.

Verschiedene Selektions- und Adaptionsstrategien, die sich durch Nutzung der neuen Möglichkeiten auszeichnen, werden vorgestellt und an Hand von Anwendungsbeispielen untersucht. Eine Evaluierung der Anwendungsbeschreibung und der dazugehörigen Implementierung erfolgt im Hinblick auf die zuvor aufgestellten Entwurfsziele und ausgesuchten Effizienzgesichtspunkte.

Mögliche Erweiterungen und Ergänzungen zu dem vorgestellten Konzept werden am Ende der Arbeit in Verbindung mit einer Zusammenfassung und einem Fazit diskutiert.

Inhaltsverzeichnis

Abbildungsverzeichnis v

Tabellenverzeichnis vii

1 Einleitung 1

1.1 Überblick 1

1.2 Motivation 1

1.3 Aufgabenstellung 3
1.3.1 Vorgehensweise 3
1.3.2 Abgrenzung von verwandten Gebieten 3
1.3.3 Einordnung in das Projekt 3PC 4

1.4 Struktur dieser Arbeit 5

2 Analyse des Themengebiets 7

2.1 Grundlagen 7
2.1.1 Ubiquitous Computing 7
2.1.2 Dynamik und Adaption in ubiquitären Systemen 8
2.1.3 Adaptionsarten 8
2.1.4 Adaptionsfälle 9

2.2 Komponentenbasierte Anwendungssysteme 9

2.3 Infrastrukturen für ubiquitäre Systeme 11
2.3.1 Aura 12
2.3.2 Gaia 15
2.3.3 One.World 17
2.3.4 PCom 19

2.4 Architekturbasierte Adaption 21
2.4.1 Acme 21

2.5 Adaption in nicht-ubiquitären Systemen 23
2.5.1 Odyssey 23
2.5.2 Grid Computing 24
2.5.3 Autonomic Computing 24
2.5.4 Load Balancing 25
2.5.5 Rainbow 25

2.6 Vergleich von Adaptionsmechanismen 27
2.6.1 Art der Verteilungsinfrastruktur 27

2.6.2 Transparenz . . . 28
2.6.3 Betrachtungsebene . . . 28
2.6.4 Adaptionsstrategien . . . 28
2.6.5 Adaptionsebene . . . 29
2.6.6 Dienste zur Adaption . . . 29
2.6.7 Programmiermodell . . . 29
2.6.8 Form der automatischen Adaption . . . 30

2.7 Freiheitsgrade bei der Adaption . . . 30
2.7.1 Informationsquellen für die Adaptionsentscheidung . . . 31
2.7.2 Zugriff auf Informationsquellen . . . 32
2.7.3 Arten von Freiheitsgraden . . . 32
2.7.4 Auswirkungen für die automatische Adaption . . . 33

3 Anforderungen und Lösungsansatz . . . 35

3.1 Anforderungen an einen Lösungsansatz . . . 35
3.1.1 Funktionale Anforderungen . . . 35
3.1.2 Nicht-funktionale Anforderungen . . . 36

3.2 Anwendungen und Adaption in PCom . . . 37
3.2.1 Komponentenstruktur . . . 37
3.2.2 Verträge . . . 38
3.2.3 Nicht-funktionale Anforderungen . . . 38
3.2.4 Konfiguration . . . 39
3.2.5 Strategien zur Konfiguration . . . 39

3.3 Anwendungsszenarien . . . 42
3.3.1 Präsentationsszenario . . . 42
3.3.2 E-Mail-Szenario . . . 43

3.4 Lösungsansatz . . . 45
3.4.1 Anwendungsbeschreibung . . . 45
3.4.2 Begründung . . . 46

3.5 Einordnung . . . 46

4 Konzeption . . . 49

4.1 Konzepte der Anwendungsbeschreibung . . . 49
4.1.1 Sicht auf die Anwendung . . . 49
4.1.2 Beschreibungssprache . . . 50
4.1.3 Gewichtung . . . 50
4.1.4 Spezifikation der Anwendungsstruktur . . . 50
4.1.5 Identifikation von Komponenten . . . 52
4.1.6 Spezifikation von Aggregationen in der Struktur . . . 52
4.1.7 Spezifikation von funktionalen Eigenschaften . . . 53
4.1.8 Spezifikation von nicht-funktionalen Eigenschaften . . . 53
4.1.9 Spezifikation von Alternativen . . . 54
4.1.10 Definition der Beschreibungseinheiten . . . 55

4.2 Konzepte der Komponentenauswahl . . . 56

4.3 Konzepte der Selektionsstrategien . . . 58
4.3.1 Sofortige Auswahl . . . 58
4.3.2 Verzögerte Auswahl . . . 58

4.4 Konzepte der Adaptionsstrategien . . . 58
4.4.1 Möglichst lokale Adaption . . . 59
4.4.2 Möglichst optimale Adaption . . . 59

4.5 Konzepte der Verhandlungsstrategien . . . 59

5 Implementierung und Integration . . . 61

5.1 Syntax der Beschreibungssprache . . . 61

5.2 Objektmodell der Beschreibungssprache . . . 62
5.2.1 Struktur . . . 63
5.2.2 Implementierung der Beschreibungseinheiten . . . 63
5.2.3 Zugriff und Identifikation . . . 65
5.2.4 Verfolgen von Änderungen . . . 65
5.2.5 Fehlerhafte Anwendungsbeschreibungen . . . 65
5.2.6 Erzeugung der Anwendungsbeschreibung . . . 66
5.2.7 Dynamisches Laden . . . 66

5.3 Integration in PCom . . . 66
5.3.1 Statisches Modell . . . 66
5.3.2 Dynamisches Modell . . . 68

5.4 Selektionsstrategien . . . 69
5.4.1 InstantSelector . . . 69
5.4.2 LazyRemoteContainerSelector . . . 70

5.5 Adaptionsstrategien . . . 71
5.5.1 LocalAdaptor . . . 71
5.5.2 QualityAdaptor . . . 71

5.6 Verhandlungsstrategien . . . 72

6 Evaluation . . . 73

6.1 Metriken . . . 73
6.1.1 Auswertung . . . 74

6.2 Messungen . . . 75
6.2.1 Testumgebung . . . 76
6.2.2 Speicherbedarf . . . 76
6.2.3 Zusätzliche entfernte Aufrufe . . . 76
6.2.4 Messungen anhand einer Beispielanwendung . . . 77

6.3 Evaluation der Anforderungen . . . 82
6.3.1 Funktionale Anforderungen . . . 82
6.3.2 Nicht-funktionale Anforderungen . . . 83

6.4 Evaluierung der Szenarien . . . 86
6.4.1 Präsentationsszenario . . . 86
6.4.2 E-Mail-Szenario . . . 87

7 Zusammenfassung und Ausblick . . . 89

7.1 Zusammenfassung . . . 89

7.2 Ausblick . . . 91

7.3 Fazit . . . 92

Literaturverzeichnis . . . 95

Abbildungsverzeichnis

Abbildung 1.1 Architektur von 3PC 5
Abbildung 2.1 Beschreibung eines Dienstes zur Textbearbeitung 13
Abbildung 2.2 Bestandteile von Aura 14
Abbildung 2.3 Schichtmodell von Aura 14
Abbildung 2.4 Active Space 15
Abbildung 2.5 Modell einer MPCC Anwendung 16
Abbildung 2.6 Architektur von Gaia 17
Abbildung 2.7 Architektur von One.World 19
Abbildung 2.8 Schichtmodell von PCom 20
Abbildung 2.9 Auszug aus einer Acme-Beschreibung 22
Abbildung 2.10 Autonomic Element 24
Abbildung 2.11 Das Rainbow Rahmenwerk 25
Abbildung 2.12 Beispiel in Pseudocode für eine Adaptionsstrategie in Rainbow. 26
Abbildung 2.13 Beispiel für einstufige (Aura) und mehrstufige (PCom) Bindungen 27
Abbildung 2.14 Informationsfluß bei der Adaption 31
Abbildung 3.1 Kontraktstruktur in PCom 38
Abbildung 3.2 Bisheriger Ablauf der Anwendungsinstanziierung in PCom 41
Abbildung 3.3 Ausnahmebehandlung bei entfernten Aufrufen 41
Abbildung 3.4 Beispielkonfigurationen für den Präsentationsmodus 43
Abbildung 3.5 Beispiele für verschiedene Netzwerkalternativen 45
Abbildung 4.1 Semantische Labels 51
Abbildung 4.2 Mehrfache Labels 52
Abbildung 4.3 Aufbau der Anwendungsbeschreibung. 55
Abbildung 4.4 Aufbau der Komponentenbeschreibung 55
Abbildung 4.5 Aufbau der Bindungsbeschreibung 56
Abbildung 4.6 Iterator-Konzept für die Selektion 57
Abbildung 5.1 Auszüge aus dem XML-Schema der Anwendungsbeschreibung. 62
Abbildung 5.2 Vererbungshierarchie der Anwendungsbeschreibung 63
Abbildung 5.3 Objektmodell der Anwendungsbeschreibung. 63
Abbildung 5.4 Referenzen auf das Anwendungsmodell 67
Abbildung 5.5 Laden des Anwendungsmodells 68
Abbildung 5.6 Neuer Ablauf bei der Komponenteninstanziierung in PCom. 69
Abbildung 5.7 Struktur des LazyIterators 70
Abbildung 5.8 Strategy-Pattern für Adaptionsstrategien 71
Abbildung 6.1 Grafische Anwendungsbeschreibung der Beispielanwendung. 77
Abbildung 6.2 Containerkonfiguration. 78
Abbildung 6.3 Grafisches Beispiel einer Anwendungsbeschreibung (Ausschnitt) 86

Tabellenverzeichnis

Tabelle 1 Metriken des Gesamtsystems 74
Tabelle 2 Metriken von Teilbereichen 75
Tabelle 3 Testumgebung. 76
Tabelle 4 Mindestgröße der Beschreibungseinheiten in Bytes. 76
Tabelle 5 Größe der Anwendungsbeschreibung zur Laufzeit in Bytes 78
Tabelle 6 Speicherverbrauch vor und nach der Selektion in Kilobytes 79
Tabelle 7 Zeitmessungen der Selektion in Sekunden (* für 10 Iterationen) 80
Tabelle 8 Zeitmessung der Einzelalgorithmen in Sekunden 80

Kapitel 1

Einleitung

Die Motivation für diese Arbeit wird im folgenden Kapitel dargelegt. Aus einem Überblick über die momentan vorhandenen Möglichkeiten und der Vision von Ubiquitous Computing wird die Notwendigkeit der Fortentwicklung von komponentenbasierten Verteilungsinfrastrukturen und deren Adaptionsmechanismen begründet. Anschließend wird die Struktur der Arbeit dargestellt.

1.1 Überblick

Verteilungsinfrastrukturen für Ubiquitous Computing müssen den speziellen Anforderungen gerecht werden, die durch die hohe Dynamik und Heterogenität der Systemumgebung geschaffen werden. Im Rahmen des 3PC-Projektes[1] der Universität Stuttgart wurde eine Anwendungsarchitektur, PCom[26], entwickelt, die die Erstellung von komponentenbasierten Anwendungen erlaubt. In PCom werden die Komponenten einer Anwendungen bei der Instanziierung der Anwendung erstmals ausgewählt und damit die Konfiguration der Anwendung festgelegt. Darüber hinaus besitzt PCom die Möglichkeit, Anwendungen zur Laufzeit zu rekonfigurieren, d.h. einem geänderten Umfeld anzupassen. In beiden Fällen werden zurzeit allerdings die Entscheidungen nur lokal getroffen und nicht die verteilte Anwendung insgesamt betrachtet. In diese Arbeit wird einen knotenübergreifenden Adaptionsmechanismus vorgestellt, der Entscheidungen zur initialen Konfiguration und Rekonfiguration der Anwendung unter Berücksichtigung des anwendungsspezifischen Gesamtzustands ermöglicht.

1.2 Motivation

Computer durchdringen unseren Alltag in vielen Bereichen. Personalcomputer am Arbeitsplatz und in vielen Haushalten sind nicht mehr wegzudenken. Durch Miniaturisierung ist ein Trend zu kleinen und mobilen Geräten erkennbar. Tragbare Computer, Mobiltelefone mit Organizer-Funktionen und Personal Digital Assistants erfreuen sich hoher Beliebtheit. Der Hardwareentwicklung tragen auch neue Netzwerktechnologien Rechnung, die Kommunikation im mobilen Bereich vereinfachen, wie z.B. Bluetooth und Wireless LAN. In Zukunft werden sich Computer in vielen Gebrauchsgegenständen in unserem Alltag finden - in das Auto haben sie beispielsweise bereits Einzug gehalten. Die Anwendungsszenarien für heutige Computersysteme folgen weitestgehend den Paradigmen, die durch den Personal Computer eingeführt wurden. Die Szenarien beschränken sich auf eine statische Umwelt - Nutzer, Geräte und Ressourcen im Allgemeinen befinden sich an fest definierten Plätzen und Änderungen stellen Ausnahmesituationen dar. Anwendungen sind in der Regel an einen Rechner gebunden und nutzen gegebenenfalls noch bestimmte Server. Dies gilt

auch für die meisten Anwendungen, die auf mobilen Geräten ausgeführt werden. So bleiben viele der Möglichkeiten, die sich durch die neuen Entwicklungen im Bereich der vorher genannten Technologien ergeben, ungenutzt.

Anstelle einer statischen Welt tritt die Vision einer hochdynamischen Welt, in der Nutzer, Geräte, Ressourcen allgemein mobil sind und sich durch eine mit Rechnersystemen gefüllten Welt bewegen. Anstelle klassischer Anwendungen, die an einzelne Rechner gebunden sind, treten Geräteverbunde, die zur Unterstützung des Nutzers bei seinen Aufgaben kooperieren. Diese Vision wird auch Ubiquitous Computing (siehe 2.1.1) genannt.

Eine der großen Herausforderungen von Ubiquitous Computing ist die hohe Dynamik der Umwelt. Die natürliche Eigenschaft von Nutzern und Geräten - ihre Mobilität - wird von Anfang an als zentraler Bestandteil des Paradigmas eingeführt. Mobilität bedeutet, dass sich die Systemumgebung aus Sicht der Anwendung ändert. Dies kann sich typischerweise durch veränderte Verfügbarkeit von Ressourcen äußern. Hervorgerufen werden die Veränderungen zum Beispiel durch Standortwechsel des Nutzers, durch Abschaltung, anderweitiger Nutzung oder auch Standortwechsel von Ressourcen.

Die aus dieser Dynamik resultierende Komplexität darf natürlich nicht zu Lasten des Nutzers umgesetzt werden. So besteht eine weitere wesentliche Forderung des Ubiquitous Computing in der Orientierung an den Bedürfnissen des Nutzers. Wenn die Systeme und Dienste nicht einfach und intuitiv zu nutzen sind, so werden sie bei den Nutzern keine Akzeptanz finden. Der Idealfall eines ubiquitären Systems erfordert keinerlei zusätzliche Eingriffe seitens des Nutzers um seine Aufgabe zu erfüllen, oder anders ausgedrückt, das System arbeitet unsichtbar für den Nutzer.

Die Hilfestellung für den Nutzers ist folglich eine Belastung des Anwendungsprogrammierers. Viele verschiedene Geräte erfordern Anpassungen an viele Plattformen. Kooperation erfordert Kommunikation zwischen den Geräten. Mobilität, d.h. spontane Vernetzung, Flexibilität in Bezug auf Verfügbarkeit und Fehlertoleranz erfordert Adaptionsfähigkeit. Die Entlastung des Nutzers erfordert, dass Adaption so weit wie möglich transparent für den Nutzer gestaltet wird. Genannte Anforderungen führen die Notwendigkeit einer Hilfestellung für den Anwendungsprogrammierers vor Augen. Die aufgestellten zentralen Anforderungen gelten für alle ubiquitären Systeme. So liegt eine Umsetzung dieser Anforderungen bei gleichzeitiger Entlastung des Anwendungsprogrammierers in einer gemeinsamen Softwareinfrastruktur nahe. Eine geeignete Softwareinfrastruktur muss eine einheitliche Kommunikationsplattform und gewisse Basisdienste bereitstellen.

Softwareinfrastrukturen, die dieses leisten, sind zurzeit Forschungsgegenstand an verschiedenen Universitäten weltweit. Eines dieser vielen Projekte, das sich dieses Problems annimmt, ist das 3PC Projekt der Universität Stuttgart. Die von Holger Gubbels im Rahmen einer Diplomarbeit vorgestellte Middleware BASE bildet die Grundlage einer ubiquitären Verteilungsinfrastruktur. Die dienstbasierte Middleware adressiert vorwiegend die Herausforderungen im Bereich der Kommunikation zwischen den Ressourcen. Durch Einführung einer flexiblen modularisierten Kommunikationsarchitektur erlaubt BASE eine Anpassung an die Anforderungen verschiedenster Geräte. Die Abstraktion der Kommunikation in BASE ermöglicht es, auch zur Laufzeit Kommunikationsprotokolle zwischen Ressourcen auszuwählen und zu wechseln. Für Anwendungen, die auf BASE aufbauen, ist die Wahl der Kommunikationsart transparent. Ein einheitliches Programmiermodell für alle auf BASE aufbauenden Anwendungen ermöglicht die Bereitstellung gemeinsam genutzter Basisdienste, wie z.B. Lookup- und Kommunikationsdienste. Diese Dienste sind plattformunabhängig in dem Sinne, dass sie für beliebige BASE-Anwendungen unabhängig von Gerät und Betriebssystem dieselbe Schnittstelle anbieten. Die Adaptionsmöglichkeiten von BASE sind allerdings auf Kommunikation beschränkt - ein Modell zur Adaption auf Dienstebene ist nicht vorgesehen. An dieser Stelle setzt PCom an. PCom ist eine auf BASE basierende komponentenorientierte Verteilungsinfrastruktur. Über die Möglichkeiten von BASE hinaus stellt sie dem Anwendungsprogrammierer ein Architekturmodell zur Verfügung. Anwendungen in PCom bestehen aus Verbunden von Komponenten. Die Bindungen zwischen zwei Komponenten basieren dabei auf einem Vertrag, in dem eine genutzte Komponente ihr Angebot festschreibt. Wird eine Anwendung instanziiert, so wird rekursiv versucht, die durch Verträge festgelegten Anforde-

rungen der beteiligten Komponenten durch Auswahl geeigneter Komponenten zu erfüllen. Die Komponenten der Anwendung werden somit erst zur Instanziierung ausgewählt und zu einer Anwendung zusammengestellt. PCom unterstützt in einer solchen Anwendungsstruktur die Adaption. Kann eine Komponente ihren Vertrag nicht mehr erfüllen, so wird in diesem Fall versucht, eine andere Komponente zu finden, die die Anforderungen erfüllen kann - und dies transparent für den Anwendungsprogrammier und auch für den Anwender. Bei der initialen Zusammenstellung wie auch bei einer Umstrukturierung der Anwendung existieren viele Freiheitsgrade in Bezug auf die Komponentenauswahl, z.B. ob nun versucht wird, ein Vertragsproblem möglichst lokal zu lösen, d.h. anstelle der Problem-Komponente eine Alternative mit den gewünschten Eigenschaften zu finden, oder gleich die gesamte Anwendung umzubauen. In PCom bestimmt momentan der so genannte Selektor, wie die Freiheitsgrade genutzt werden. Auf Basis einer lokalen Selektionsstrategie trifft dieser Algorithmus seine Entscheidungen. Die lokale Selektionsstrategie betrachtet und berücksichtigt dabei nur die direkt beteiligten Komponenten, d.h. die nutzende und genutzte Komponente. Wünschenswert wäre aber eine umfassendere Betrachtung durch Einbeziehung der Anwendung bei der Entscheidungsfindung. Eine knotenübergreifende Betrachtung der gesamten Anwendungen kann andere Adaptionsmöglichkeiten eröffnen, die einer lokalen Strategie verborgen bleiben. Die Verwendung der neuen Möglichkeiten kann gegebenenfalls in besseren Konfigurationen für die Anwendung gegenüber einer rein lokalen Adaption resultieren. Zu einer effizienteren Nutzung der Freiheitsgrade ist eine knotenübergreifende Sicht bei der Entscheidungsfindung unumgänglich.

1.3 Aufgabenstellung

Ziel dieser Arbeit ist die Konzeption, Umsetzung, Integration und Evaluation eines Mechanismus, der dem Anwendungsprogrammierer einerseits die Nutzung der Freiheitsgrade bei Adaption und andererseits eine knotenübergreifende Betrachtung der Anwendung bei der Adaption ermöglicht. Es soll vor allem die hohe Dynamik und die Ressourcenbeschränktheit ubiquitärer Rechnersysteme berücksichtigt werden.

1.3.1 Vorgehensweise

Die vorliegende Arbeit untersucht die Möglichkeiten, die bisher bestehenden Freiheitsgrade bei der Entscheidungsfindung auszufüllen. Dabei soll eine Methode gefunden werden, die es dem Anwendungsprogrammierer ermöglicht, die Freiheitsgrade zu nutzen. Zu diesem Zweck wird eine Möglichkeit der Anwendungsbeschreibung entworfen, die es dem Anwendungsprogrammierer ermöglichen soll, komponentenübergreifend Einfluss auf die Nutzung der Freiheitsgrade bei der Zusammenstellung und Adaption einer Anwendung zu nehmen. Diese Anwendungsbeschreibung erlaubt eine knotenübergreifende Betrachtung der Anwendung bei der Zusammenstellung und Adaption. Eine mögliche Effizienzsteigerung bei der Adaption durch die neuen Beschreibungsmöglichkeiten wird in diesem Zusammenhang erörtert. Die in PCom vorhandenen Adaptionsmechanismen werden im Rahmen der Arbeit erweitert, um Nutzen aus der neu eingeführten Anwendungsbeschreibung zu ziehen. Dabei werden verschiedene Strategien zur Selektion untersucht und verglichen.

1.3.2 Abgrenzung von verwandten Gebieten

Im Rahmen der Untersuchungen werden verschiedenste Forschungsgebiete der Informatik berührt. Der Schwerpunkt liegt natürlich im Bereich des Ubiquitous Computing. Doch sind beispielsweise durch die Anwendungsbeschreibung auch viele softwaretechnische Aspekte betroffen, wie beispielsweise Architektursprachen und CASE-Tools. Im Folgenden werden die Schwerpunkte der Arbeit durch Abgrenzung von verwandten Gebieten herausgestellt.

Diese Arbeit beschränkt sich auf die Betrachtung der Nutzung von Adaptionsfreiräumen durch den Anwendungsprogrammierer durch Bereitstellung anwendungsspezifischer Informationsquel-

len in Form einer Anwendungsbeschreibung. Ebenso ist die Nutzung der Freiräume auf Basis von systemspezifischen Informationsquellen denkbar. Der gewählte Ansatz ist nicht als Wertung der verschiedenen Konzepte zur Nutzung von Freiheitsgraden zu sehen, sondern als Wahl zur Untersuchung, in wie weit das Konzept der Bereitstellung einer Anwendungsbeschreibung für den Anwendungsprogrammierer den Anforderungen gerecht wird. Ein Vergleich verschiedener, grundlegend unterschiedlicher Ansätze zur Bereitstellung anwendungs- und systemspezifischer Informationen ist auf Grund des Prototypenstadiums der ubiquitären Verteilungsinfrastrukturen und dem begrenzten Umfang dieser Arbeit nicht möglich.

Diese Arbeit beschäftigt sich nicht mit der Definition und Untersuchung von Architektursprachen. Die erstellte Anwendungsbeschreibungssprache weist zwar Ähnlichkeiten zu Architektursprachen auf, doch ist sie Mittel zum Zweck, dem Anwendungsprogrammierer die Nutzung der Adaptionsfreiräume in ubiquitären Verteilungsinfrastrukturen zu ermöglichen. Der Schwerpunkt liegt eindeutig im Aufzeigen und Untersuchen der Möglichkeiten einer Anwendungsbeschreibung, die Freiräume bei der Adaption in ubiquitären Verteilungsinfrastrukturen zu nutzen, und nicht in der Erschaffung einer Architektursprache zur Anwendungsbeschreibung.

Die Umsetzung der Anwendungsbeschreibung wird in PCom erfolgen, so dass im Folgenden vor allem die Freiheitsgrade, die in PCom zur Verfügung stehen, betrachtet werden. Soweit wie es möglich ist wird versucht, für PCom gültige Aussagen zu verallgemeinern und unabhängig von der Verteilungsinfrastruktur zu treffen. Doch ein vollkommen generalisierter Ansatz würde sich weder in sinnvoller Form auf Grund der stark unterschiedlichen Anwendungsmodelle in den Verteilungsinfrastrukturen umsetzten noch verifizieren lassen. Die Untersuchung beschränkt sich aus diesem Grund auch auf die Betrachtung von komponentenbasierten, ubiquitären Verteilungsinfrastrukturen. Die Anwendbarkeit von Anwendungsbeschreibungen in herkömmlichen Infrastrukturen wie beispielsweise CORBA Components[3] ist nicht Thema dieser Arbeit, da diese Verteilungsinfrastrukturen nicht über die für Ubiquitous Computing notwendigen Mechanismen verfügen.

Eine automatisierte Ressourcenverwaltung und -optimierung, wie sie beispielsweise in Odyssey[33] implementiert ist, wird hier nicht angestrebt. Zwar bietet die Anwendungsbeschreibung die Möglichkeit, Ressourcennutzung und Quality of Service-Kriterien für den Anwendungsprogrammierer beeinflussbar zu machen, jedoch wird die Möglichkeit nicht weiter vertieft. Die Version von PCom, die für diese Arbeit die Grundlage bildet, verfügt nicht über die notwendigen Bestandteile und Mechanismen für ein solches System. Die Aufgabe besteht explizit in der Schaffung einer Möglichkeit für den Anwendungsprogrammierer, über Bereitstellung von anwendungsspezifischen Informationen Freiräume bei der Adaption zu nutzen. Die im Folgenden entworfene Beschreibungssprache wird erweiterbar gestaltet und schließt eine Aufnahme von Parametern, die einer Ressourcenverwaltung zur Verfügung gestellt werden könnten aber nicht aus.

Ebenso wie eine Ressourcenverwaltung im Speziellen ist die Bereitstellung von nicht-funktionalen Parametern durch Komponenten nicht Untersuchungsgegenstand. Die Arbeit beschränkt sich auf die in PCom dafür vorgesehenen Mechanismen und lässt die Form der Nutzung durch die Komponentenentwickler offen.

1.3.3 Einordnung in das Projekt 3PC

Das Projekt 3PC der Universität Stuttgart hat sich die Entwicklung möglichst minimaler, aber flexibler Unterstützung durch Infrastrukturen für ubiquitäre Anwendungen zum Ziel gemacht. Die Grundlage wurde mit BASE gelegt. Auf dieser dienstbasierten Verteilungsinfrastruktur ist eine inzwischen eigenständige komponentenbasierte Erweiterung entstanden - PCom. Die konzeptuelle Gliederung der Projekte ist wie in folgender Abbildung dargestellt.

Verschiedene Projekte beschäftigen sich mit der Einbeziehung von systemspezifischen Informationen. Die Diplomarbeit von Alexander Rau untersucht die Bereitstellung von Quality of Service-Kriterien auf Plug-in-Ebene in BASE. Andreas Lachenmann untersucht in seiner Arbeit die Möglichkeiten für die Verwaltung von Ressourcen in PCom und deren Einfluss auf die Adaption.

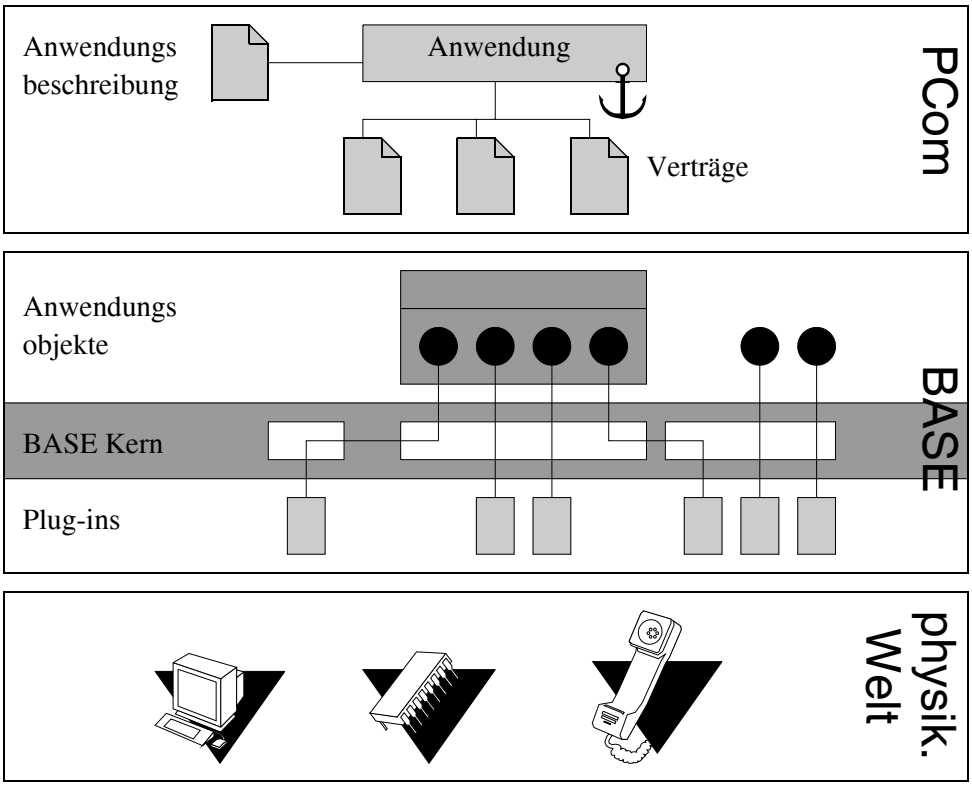


Abbildung 1.1 Architektur von 3PC

Diese Arbeit ergänzt das in PCom vorhandene Vertragskonzept durch die Anwendungsbeschreibung, die dem Anwendungsprogrammierer eine Möglichkeit bietet, die Freiräume bei der Adaption von Anwendungen zu nutzen. Es soll explizit die Sicht des Anwendungsprogrammierers auf die Anwendung, also anwendungsspezifische Informationen, für die Adaption verwendet werden.

1.4 Struktur dieser Arbeit

Die Arbeit ist wie folgt strukturiert. Kapitel 2 enthält eine Einführung in die verschiedenen, mit dieser Arbeit verwandten Themengebiete sowie eine Analyse von bestehenden Forschungsprojekten mit ähnlicher Forschungsrichtung. Basierend auf diesen Resultaten wird in Kapitel 3 ein Lösungsansatz zur Nutzung von Freiheitsgraden bei der Adaption durch den Anwendungsprogrammierer vorgestellt. In Kapitel 4 werden die zentralen Entwurfskonzepte für den Lösungsansatz - die Einführung einer Anwendungsbeschreibung - diskutiert. Die programmiertechnische Umsetzung der Anwendungsbeschreibung und deren Integration in PCom werden in Kapitel 5 thematisiert. Ein Rückblick auf die Entwurfsziele im Vergleich zum Erreichten einschließlich verschiedener Leistungsmessungen erfolgen in der Evaluation in Kapitel 6. Kapitel 7 schließt die Arbeit mit einer Zusammenfassung und damit verbunden einem Ausblick auf mögliche, zukünftige Arbeiten in diesem Themengebiet.

Kapitel 2

Analyse des Themengebiets

Das diese Arbeit umgebende Themengebiet ist Ubiquitous Computing. Die Klärung der grundlegenden Begriffe und Technologien in diesem Bereich soll die Anforderungen und das Umfeld des späteren Lösungsansatzes eingrenzen. Die Analyse der Fähigkeiten bereits bestehender ubiquitärer Verteilungsinfrastrukturen in Bezug auf die Aufgabenstellung ermöglicht einen Blick auf diejenigen Ansätze in den Systemen, die bereits Teile der Problemstellung berühren oder abdecken. Im Anschluss an die Untersuchung verwandter Lösungsansätze wird die Problemstellung konkretisiert und diskutiert. In diesem Rahmen wird eine Klassifizierung der untersuchten Ansätze vorgenommen.

2.1 Grundlagen

Das Themengebiet dieser Arbeit ist Ubiquitous Computing. Für ein umfassendes Verständnis der Zielsetzungen und Probleme, die durch die Vision des Ubiquitous Computing entstehen, werden im Folgenden verschiedene grundlegende, und im Rahmen dieser Arbeit wesentliche Begriffe erläutert.

2.1.1 Ubiquitous Computing

Die Vision von Ubiquitous Computing wurde in der Einleitung bereits angesprochen. Sie geht zurück auf verschiedene Veröffentlichungen[40, 41, 42] von Weiser, in denen er ein Paradigma für die zukünftige Nutzung von Rechnersystemen beschreibt. Dieses Paradigma bezeichnet Weiser als Ubiquitous Computing.

> *"Ubiquitous computing is the method of enhancing computer use by making many computers available throughout the physical environment, but making them effectively invisible to the user."[40]*

Im Gegensatz zu den bisher gängigen Formen von Rechnersystemen, die tief in das alltägliche Leben eingreifen oder sogar versuchen, die Welt im Computer abzubilden (*Virtual Reality*[13]), soll im Ubiquitous Computing die *allgegenwärtige Verfügbarkeit* bei gleichzeitiger *Unsichtbarkeit* der Rechnersysteme realisiert werden. Ein ideales ubiquitäres System ist für den Nutzer nur durch die notwendigsten Ein- und Ausgabegeräte sichtbar, ansonsten arbeitet das System für den Nutzer transparent.

Die verwendeten Geräte in ubiquitären Systemen sind vielfältig und zum großen Teil wirklich unsichtbar für den Nutzer. Als Beispiele sind hier Sensoren und Regelsysteme anzuführen, aber auch

leistungsfähige Rechner, die ressourcenintensive Aufgaben umsetzen können. Ein- und Ausgabegeräte sind idealerweise die einzigen für den Nutzer sichtbaren Geräte.

So vielfältig wie die Geräte können auch die eingesetzten Netzwerktechnologien sein, um Kommunikation in einem ubiquitären System zu ermöglichen. Verkabelung per klassischem Ethernet ist ebenso denkbar wie moderne Funkübertragungstechnologien à la Bluetooth oder WiFi. Festzuhalten bleibt, dass die Bandbreite der verwendeten Geräte innerhalb eines Systems von einem simplen Sensor bis zu einem Hochleistungsrechner reichen kann und das integrierende Softwaresystem dem Rechnung tragen muss.

Eine weitere Eigenschaft von ubiquitären Systemen ist eine hohe Dynamik. Nutzer betreten und verlassen ein ubiquitäres System und bewegen sich darin. Geräte, z.B. solche, die der Nutzer an sich trägt, werden einem System hinzugefügt oder aus ihm herausgenommen. Anwendungen müssen sich dieser sich ständigen Ändernden unterworfenen Umwelt anpassen. Dies impliziert die Anforderung einer hohen Fehlertoleranz gegenüber Ausfällen von Teilsystemen, unabhängig ob nun beispielsweise durch Unterbrechung einer Funknetzwerkverbindung, einem Defekt oder einer simplen Abschaltung eines Geräts verursacht.

Ein Blick in die heutige Hard- und Softwarewelt zeigt, dass der Weg zu der Umsetzung dieser Vision noch sehr weit ist. Angefangen bei Beschränkungen im Bereich der Geräte, z.B. durch nicht ausreichende Miniaturisierung, fehlender Standardisierung für Kommunikation, mangelnde Alltagstauglichkeit und zu hohen Kosten bis hin zu Beschränkungen im Softwarebereich, z.B. fehlende Interoperabilität, fehlende Berücksichtigung von Mobilität und damit fehlender Adaptionsfähigkeit an neue Umgebungen, existieren noch viele ungelöste Probleme bevor die Vision des Ubiquitous Computing Wirklichkeit werden kann.

2.1.2 Dynamik und Adaption in ubiquitären Systemen

Aus Entwicklersicht ist die herausragende Eigenschaft von ubiquitären Systemen die Vielfalt der beteiligten Ressourcen und deren Dynamik. Anwendungen für Ubiquitous Computing müssen dieser Dynamik durch geeignete Mechanismen Rechnung tragen. Anwendungen in ubiquitären Systemen werden oft erst zur Instanziierung aus zum Instanziierungszeitpunkt verfügbaren Ressourcen zusammengestellt und somit dynamisch konfiguriert. Adaption bezeichnet die Methode, durch die Anwendungen rekonfiguriert werden und sich in ubiquitären Systemen an geänderte Umgebungen anpassen. Beide Punkte sind wesentliche Unterscheidungskriterien zu herkömmlichen Anwendungen.

Typische Anwendungen heutzutage sind nicht auf Änderungen in der Umgebung ausgelegt. Im folgenden Szenario wird das Verhalten einer weit verbreiteten Textverarbeitungssoftware auf einem mobilen Gerät betrachtet. Der Nutzer bearbeitet auf seinem Notebook eine Textdatei. Das Notebook ist über eine Funknetzwerkverbindung mit einem Server verbunden, auf dem die Textdatei abgelegt ist. Entfernt sich nun der Nutzer aus dem durch das Funknetzwerk abgedeckten Bereich, so meldet die Textverarbeitung beim Versuch, die Datei auf dem Server zu sichern, einen Fehler, da die Netzwerkverbindung und somit der Dateiserver nicht mehr verfügbar ist.

In einem ubiquitären System sollten Mechanismen zur Verfügung stehen, so dass sich das Problem - der Wegfall der Netzwerkverbindung - automatisch beheben lässt. Eine Möglichkeit der automatischen Problembehebung in diesem Fall wäre, dass an Stelle der Serverversion dann eine lokal gespeicherte Variante verwendet wird, die später bei wiederhergestellter Netzwerkverbindung mit der Serverversion synchronisiert werden kann. Auf jeden Fall sollte der Anwender in der Welt des Ubiquitous Computing nicht mit diesem Problem behelligt werden.

2.1.3 Adaptionsarten

Die Anpassung einer Anwendung durch Adaption kann auf zwei Arten erfolgen. Zum einen kann das Verhalten der Anwendung angepasst werden. Die Anwendung bleibt in ihrer Struktur bestehen und passt ihr Verhalten den geänderten Umständen an. Ein typisches Beispiel wäre hier die

Adaption der Videoqualität bei einer Videoanwendung in Abhängigkeit der zur Verfügung stehenden Netzwerkbandbreite.

Die andere Art der Adaption ist die Rekonfiguration der Anwendung. In diesem Fall wird versucht, die Anwendung dahingehend zu rekonfigurieren, dass das gewünschte Verhalten erhalten bleibt bzw. erzielt wird. Auch hier kann die Netzwerkbandbreite als Beispiel dienen - in diesem Fall würde die Anwendung aber nicht die Videoqualität verändern, sondern andere Netzwerkverbindungen probieren, um die Qualität beizubehalten.

Mischformen aus Verhaltensanpassung in Kombination mit Anwendungsanpassung sind natürlich auch möglich, wenn nicht sogar der Normalfall. In Bezug auf die Videoanwendung könnte dies durch das Einrichten einer Qualitätsuntergrenze (Verhaltensanpassung) geschehen, bei deren Unterschreitung dann die Netzwerkverbindung adaptiert (Rekonfiguration) wird.

2.1.4 Adaptionsfälle

Die Notwendigkeit für Adaption und deren Ausprägung wurde aus den vorangegangenen Ausführungen deutlich. Um die Ursachen für Adaption besser zu verstehen ist eine Klassifizierung von verschiedenen Adaptionsfällen sinnvoll. Die folgende Betrachtung bezieht sich auf ein generisches Komponentensystem (siehe 2.2), d.h. eine Anwendung ist eine Zusammenstellung aus Komponenten, die jeweils einen Teil der Gesamtfunktionalität realisieren.

Adaption auf Grund von anwendungsinternen Ursachen. Die Dynamik in ubiquitären Systemen erstreckt sich auch auf die Anwendungen. Komponenten innerhalb einer Anwendung können ihr Angebot ändern, so dass ggf. zugesicherte Eigenschaften gegenüber nutzenden Komponenten nicht aufrechterhalten werden können. Ebenso können Komponenten ihre Anforderungen anpassen. Beispielsweise könnte eine Komponente ihren Speicherbedarf verdoppeln. Kann die genutzte Speicherverwaltung den Speicher nicht bereitstellen, so muss adaptiert werden. Weitere anwendungsinterne Ursachen bestehen in Fehlern, die in Komponenten auftreten. In diesem Fall kann die Anwendung adaptiert werden, um die fehlerhafte Komponente zu ersetzen.

Adaption auf Grund von externen Ursachen. Externe Ursachen können auch Quellen für Adaption darstellen. So ist die Anwendung immer auf die durch die Umgebung angebotenen Komponenten beschränkt. Ändert sich dieses Angebot, so besteht einerseits die Möglichkeit, dass an der Anwendung beteiligte Komponenten wegfallen - in diesem Fall ist der Adaptionsbedarf offensichtlich - oder andererseits, dass neue Komponenten angeboten werden. Auf Grund der neuen Komponenten könnte eine Anwendung adaptiert werden, um gegebenenfalls eine günstigere Konfiguration zu erzielen. Verwendet eine Anwendung beispielsweise ein Bluetooth-Netzwerk und stellt die Verfügbarkeit eines WLAN-Netzes fest, so kann es durchaus günstig für die Anwendung sein, zu rekonfigurieren um die bessere Netzwerkverbindung zu nutzen.

Fehlverhalten durch externe Ursachen stellen ebenso Adaptionsgründe dar. Tritt beispielsweise eine Netzwerkpartitionierung auf, so kann es zum Ausfall von Komponenten kommen, die in einer Anwendung verwendet werden, und somit Adaption für die betroffene Anwendung notwendig werden.

Externe Ursachen, wie beispielsweise Auslastung bestimmter Ressourcen, können natürlich Gründe für Angebotsänderungen von Komponenten darstellen. Dieser Fall wird aber als anwendungsinterne Ursache gewertet, da es letztendlich von der Komponente abhängt, ob und wenn ja, mit welchem Angebot sie weiterhin ausgeführt wird.

2.2 Komponentenbasierte Anwendungssysteme

In einem ubiquitären System müssen verschiedenste Komponenten miteinander kommunizieren und interagieren um dem Anwender ihre Dienste anbieten zu können. Nicht nur für Adaption sind gemeinsame Mechanismen zur Behandlung von Umgebungs- und Angebotsänderungen vorteil-

haft oder sogar Voraussetzung. Anwendungsprogrammierer stehen somit vor der Aufgabe, sich mit den unterschiedlichsten Kommunikations- und Hardwaretechnologien der verschiedenen Geräte auseinanderzusetzen. Um die Komplexität und die damit verbunden Entwicklungskosten zu senken, sind Erleichterungen für den Anwendungsprogrammierer notwendig. Eine solche Erleichterung stellen Verteilungsinfrastrukturen dar. Die Technologie von Verteilungsinfrastrukturen, auch als Middleware bekannt, existiert wesentlich länger als die der ubiquitären Systeme. Vorhandene Verteilungsinfrastrukturen eignen sich gerade auf Grund der zentralen Eigenschaften ubiquitärer Systeme - hohe Dynamik und möglicher Adaption - in vielen Fällen ohne besondere Anpassungen nicht für diesen Einsatz. Weiterhin sind viele herkömmliche Verteilungsinfrastrukturen durch die in der Verteilungsinfrastruktur implementierte Dienste, die den Anwendungen zur Verfügung stehen, sehr komplex und sind somit nicht auf im Ubiquitous Computing verwendeten mobilen, leistungsschwachen Geräten lauffähig. Doch die grundlegende Idee hinter Verteilungsinfrastrukturen ist auch in speziellen Verteilungsinfrastrukturen für Ubiquitous Computing die Gleiche.

Bernstein beschreibt Middleware als Dienste mit standardisierten Schnittstellen und Protokollen in verteilten Systemen zwischen der Betriebssystemschicht und Anwendungsschicht.

> *"To help solve customers' heterogeneity and distribution problems, and thereby enable the implementation of an information utility, vendors are offering distributed system services that have standard programming interfaces and protocols."*[14]

Ein wesentliches Merkmal von Verteilungsinfrastrukturen ist die Fähigkeit von Programmen, die für diese Struktur geschrieben wurden, problemlos zusammenzuarbeiten. Diese *Interoperabilität* innerhalb einer Struktur ermöglicht es Softwareentwicklern, Programme unabhängig voneinander zu entwickeln, trotzdem aber Programme von Drittherstellern verwenden zu können. Weiterhin ermöglichen Verteilungsinfrastrukturen die Nutzung von Programmen auf allen Plattformen, die von dieser Middleware unterstützt werden. So wird die *Portabilität* der Anwendungen erhöht und dem Anwendungsprogrammierer eine homogene Programmierschnittstelle für die unterstützten Systeme bereitgestellt.

Zusammenfassend stellt eine Verteilungsinfrastruktur für den Anwendungsprogrammierer eine Abstraktion von den verschiedenen Hard- und Softwareinfrastrukturen bereit und bietet weitere Vorteile wie beispielsweise die Wiederverwendung von Anwendungsteilen oder die Bereitstellung von gemeinsam nutzbaren Diensten (Namensdienste, Transaktionsdienste, etc.). Viele der heutigen Geschäftsanwendungen basieren auf einer Middleware. Insbesondere komponentenbasierte Middleware wie CORBA Components[2, 3] oder J2EE[5] erfreuen sich dabei hoher Beliebtheit.

Die kleinste Modellierungseinheit für komponentenbasierte Middleware ist eine Komponente, die Szyperski wie folgt definiert:

> *"A software component is a unit of composition with contractually specified interfaces and explicit dependencies only. A software component can be deployed independently and is subject to composition by third parties."*[37]

Eine Komponente weist nach dieser Definition die im Folgenden genannten Eigenschaften auf. Komponenten bieten ihre Funktionalität nur über spezifizierte Schnittstellen an, d.h. für andere Komponenten ist diese eine Art Black Box. Sie enthält keine impliziten Bindungen zu anderen Komponenten. Aus diesem Grund ist eine Komponente in sich abgeschlossen und kann somit unabhängig ausgeliefert werden. Ein Anwendungssystem entsteht schließlich durch das Zusammenwirken verschiedener Komponenten.

Komponentenarchitekturen stellen eine Weiterentwicklung und Zusammenführung von Konzepten, und damit Eigenschaften und Vorteilen anderer Architekturen dar. Das Konzept der Module wurde in Modula und Ada eingeführt, um definierte Schnittstellen für Funktionalität zu bilden. Allerdings mangelt es dem Konzept an der Fähigkeit zur Versionierung und Vererbung dieser Schnittstellen. Ubiquitäre Systeme sind aber, wie bereits beschrieben, durch hohe Dynamik ge-

prägt und überspannen oftmals mehrere Administrationsdomänen. Somit können keine Annahmen in Bezug auf die in einem System vorhandenen Modulversionen gemacht werden. Zudem ist mit der häufigen Erweiterung von Schnittstellen zu rechnen. Diesen Erfordernissen wird der Modulansatz nicht gerecht bzw. stellt hohe Ansprüche sowohl an die Entwickler wie auch an die Administratoren.

Vererbung und Typisierung von Schnittstellen wird durch die Objektorientierung ermöglicht. In objektorientierten Systemen werden Daten und Funktionalität in Objekten zusammengefasst. Ein Zugriff auf Daten ist nur über wohldefinierte Schnittstellen möglich. Vererbung erlaubt bei Erhaltung der Schnittstelle die Modifikation von vorhandener Funktionalität und erleichtert somit Wartung und Erweiterung von bestehenden Systemen. Objekte stellen allerdings in vielen Fällen die falsche Abstraktionsebene dar, da Objekte im Allgemeinen sehr kleine Einheiten bilden. Ein Objekt alleine hat in aller Regel viele, auch implizite Abhängigkeiten, die zur Lauffähigkeit des Objekts erfüllt sein müssen. Die Wiederverwertung in anderen Systemen ist nur möglich, wenn sich die Abhängigkeiten auch in diesen Systemen auflösen lassen. Komponenten dagegen stellen größere Einheiten dar, die aus vielen Objekten bestehen können. Sie kombinieren die Vorteile der Objektorientierung - die Möglichkeit der Typisierung und Vererbung - ohne dabei Einschränkungen bezüglich der Wiederverwertbarkeit zu verursachen und besitzen per Definition nur explizite Abhängigkeiten.

Komponentenbasierte Verteilungsinfrastrukturen wie CORBA Components und J2EE - im Folgenden auch als “herkömmliche” Verteilungsinfrastrukturen bezeichnet - bieten vor allem Unterstützung für die Abbildung von Geschäftslogik und -prozessen. Mobilität und hohe Dynamik sind entweder nicht vorgesehen oder nicht zentraler Bestandteil des Anwendungsmodells. Die Eignung der meisten dieser Verteilungsinfrastrukturen ist aus diesen Gründen für den Einsatz in ubiquitären Systemen in Frage gestellt. Eine mögliche Verwendung würde zumindest tief greifende Anpassungen an der jeweiligen Verteilungsinfrastruktur erfordern.

2.3 Infrastrukturen für ubiquitäre Systeme

Die Betrachtung herkömmlicher Verteilungsinfrastrukturen hat verdeutlicht, dass in diesen Verteilungsinfrastrukturen wichtige Designkriterien für Ubiquitous Computing nicht aufgegriffen werden und sie daher nicht ohne weiteres in ubiquitären Systemen verwendet werden können. Die daraus resultierende Notwendigkeit von speziell für ubiquitäre Systeme entwickelten Verteilungsinfrastrukturen hat eine Vielzahl von Projekten an Forschungseinrichtungen hervorgerufen. Vier wesentliche Designkriterien ubiquitärer Systeme verdeutlichen die von herkömmlichen Verteilungsinfrastrukturen abweichenden Anforderungen.

Heterogenität. Ubiquitäre Systeme sind inhärent heterogen. Die Vielzahl an verwendeten Geräten und Netzwerkinfrastrukturen bedingt eine extrem heterogene Systemumgebung mit großen Unterschieden in der Leistungsfähigkeit der einzelnen Geräte. Im Gegensatz zu herkömmlichen Verteilungsinfrastrukturen muss die Anpassungsfähigkeit an eine heterogene Umgebung aus unterschiedlichen Geräten und Netzwerktechnologien gewährleistet werden.

Mobilität. In konventionellen Verteilungsinfrastrukturen werden oft Annahmen über die Verfügbarkeit und den Ort bestimmter Systemkomponenten oder -dienste getroffen. Dies ist auch in vielen Anwendungsfällen zulässig und sinnvoll. In ubiquitären Systemen sind jedoch die Nutzer wie auch die Geräte mobil. Anwendungen können sich nicht auf die Verfügbarkeit bestimmter Ressourcen verlassen, weder bei der Instanziierung noch während der Laufzeit. So sollten ubiquitäre Verteilungsinfrastrukturen Mechanismen zur dynamischen Zusammenstellung von Ressourcen zu einer Anwendung verfügen. Auch ist die plötzliche Nichtverfügbarkeit einer Systemkomponente kein Sonderfall und auch kein Fehlerfall, sondern muss im Design bereits als Normalfall berücksichtigt werden. Beispielsweise sollte die Anwendung eines Nutzers sein Herausbewegen aus dem Netzbereich eines Wireless LANs und die damit verbundene Trennung der Netzwerkverbindung nicht als Fehler behandeln sondern dies als reguläre Nutzung ansehen und

umsetzen. Eine Verteilungsinfrastruktur für ubiquitäre Systeme sollte Mechanismen zur Behandlung von Umgebungsänderungen bereitstellen.

Kontext. Eine Anwendung kann unter verschiedenen Umgebungsbedingungen ausgeführt werden. Kontextinformationen können beispielsweise momentane Aktivitäten des Nutzers sein, sein Aufenthaltsort oder auch Tageszeit und Wetter. Die Betrachtung einer Anwendung in ihrem Kontext erlaubt Optimierungen in der jeweiligen Situation und verringert damit die Belastung des Benutzers und setzt so gesehen eine der Zielvorgaben des Ubiquitous Computing um. Da viele Anwendungen von der Bereitstellung von kontextuellen Informationen profitieren können, sollte eine Verteilungsinfrastruktur über Mechanismen zur Bereitstellung von Kontextinformationen verfügen.

Die genannten Anforderungen erfassen die wichtigen Aspekte ubiquitärer Verteilungsinfrastrukturen. Besondere Erwähnung im Rahmen dieser Arbeit verdient sich der Bereich der Adaptionsfähigkeit von Anwendungen in einer komponentenbasierten ubiquitären Verteilungsinfrastruktur. Adaption ist, wie bereits erörtert, eine Strategie, die durch Mobilität entstehenden Anforderungen umzusetzen. Sie stellt somit einen Teilaspekt der Mobilität dar. Adaptionsfähigkeit erlaubt dynamische Zusammenstellung und Anpassung von Anwendungen auch zur Laufzeit und damit eine flexible Reaktion auf Umgebungsänderungen. Die Bereitstellung von Funktionalität im Bereich Adaption auf der Ebene der Verteilungsinfrastruktur erlaubt es Anwendungsprogrammierern, Entwicklungsarbeit einzusparen und auf vorhandene Strukturen zurückzugreifen. Weiterhin wird auf diesem Weg auch eine anwendungsübergreifende Kooperation ermöglicht.

Vier Forschungsprojekte mit teilweise sehr unterschiedlichen Ansätzen werden im Folgenden vorgestellt. Aura und Gaia untersuchen das Gebiet der *Smart Environments*. Ein Smart Environment repräsentiert eine wohldefinierte physikalische Umgebung, in der sich Geräte und Nutzer aufhalten und bewegen können. Ein wesentliches Merkmal dieser beiden Infrastrukturen ist die einstufige Bindungsebene von Komponenten, d.h. eine Anwendung spezifiziert nur ihre direkten Abhängigkeiten in die Umgebung und für anbietende Komponenten sind solche Mechanismen nicht vorgesehen. Der Ansatz von One.World verwendet nicht das Konzept einer Umgebung und erlaubt Realisierungen ohne vordefinierte Umgebungen. Aus diesem Ansatz heraus erlaubt One.World, wie auch die letzte der untersuchten Verteilungsinfrastrukturen, PCom, mehrstufige Bindungen von Komponenten.

Bei der Untersuchung der Systeme wird vor allem der für diese Arbeit relevante Bereich der Adaption herausgearbeitet. In diesem Zusammenhang ist eine Übersicht über die Anwendungsmodellierung für das jeweilige System und die generelle Architektur wesentlich. Auf dieser Basis werden die Adaptionsmöglichkeiten und daraus entstehenden Freiheitsgrade bei der Adaption in den jeweiligen Systemen betrachtet. Besonderes Augenmerk wird zudem auf gegebenenfalls vorhandene Ansätze von Anwendungsbeschreibungen gelegt.

2.3.1 Aura

Aura[21, 22, 39] ist ein Middleware-Projekt der Carnegie Mellon University. Der Wahlspruch von Aura, *"Distraction Free Ubiquitous Computing",* zeigt sehr gut die Zielrichtung dieses Projekts.

In der Welt von Aura bewegen sich Nutzer mit ihren Aufgaben innerhalb von und zwischen *Umgebungen*. Eine Umgebung beschreibt dabei eine Ansammlung von Geräten und Anwendungen, die von einem Nutzer an einem definierten (physikalischen) Ort nutzbar sind.

Aura verfolgt das Konzept des *Task-driven Computing*. In diesem Konzept verwendet der Benutzer Anwendungen als Mittel zum Zweck der Erfüllung einer dahinter stehenden Aufgabe. Der Nutzer wird dabei von einer persönlichen *Aura* umgeben, die mit der Umgebung kommuniziert um für die Erledigung der Aufgaben des Nutzers zu sorgen. Die Aufgaben des Nutzers definieren sich dabei unabhängig von Anwendungen, Diensten oder Zuständen. Aufgaben können sich dabei über den Zeitraum einer konkreten Interaktionsphase hinaus erstrecken. Eine nicht erfüllte Aufgabe, mit der sich der Nutzer aber momentan nicht in Interaktion befindet, wird als passiv bezeich-

net. Ist eine Aufgabe aktiv, so befindet sich der Nutzer in einer *Session* mit der Umgebung. Eine Session beinhaltet eine Konfiguration von Diensten zur Erfüllung der Benutzeraufgaben. Das System ist dann für die automatische Rekonfiguration und Adaption zuständig, um bei Umgebungsänderung weiterhin die Aufgabe erfüllen zu können. Kann eine Aufgabe nicht erfüllt werden, so sollen dem Nutzer proaktiv Hinweise bereitgestellt werden, wie er dennoch seine Aufgabe erfüllen kann. Proaktivität findet sich auch an anderen Stellen im Konzept des Task-driven Computing. Auf Basis der Aufgabe soll es der Verteilungsinfrastruktur ermöglicht werden, vorausschauende Ressourcenplanung zu betreiben. Beispielsweise könnte Aura auf Basis der typischen Laufzeit für eine Aufgabe, der Akkulaufzeit des Notebooks eines Nutzers und dem spezifizierten Energiebedarf verfügbarer Dienste eine optimale Konfiguration zu bestimmen.

Das Paradigma von Ubiquitous Computing verfolgend, besteht die Zielsetzung von Aura, diese Welt für den Nutzer transparent zu gestalten und somit die in vielen Systemen unterbewertete Ressource der Aufmerksamkeit des Nutzers mit einzubeziehen. Die den Nutzer umgebende Aura fungiert für ihn dabei als Proxy, d.h. die Aura sorgt für die Adaption an die verschiedenen Umgebungen. Aus Nutzersicht ist Aura aufgabenorientiert, d.h. Aufgaben eines Nutzers werden als Entität unabhängig einer Umgebung definiert. Unter dem Aspekt der Verteilungsinfrastruktur ist Aura dienstorientiert, d.h. eine Aufgabe wird durch das Zusammenwirken von Diensten bearbeitet.

Anwendungsmodell. Das zentrale Konzept für Anwendungen in Aura sind Aufgaben (*Tasks*). Im Gegensatz zu vielen anderen Verteilungsinfrastrukturen werden in Aura Aufgabenbeschreibungen an Stelle von Schnittstellenbeschreibungen verwendet, um die Anforderungen einer Anwendung zu Charakterisieren. So existiert in Aura auch nicht das Konzept einer zentralen Anwendung, die auf Nutzerseite ausgeführt wird. Architektonisch besteht eine Anwendung in Aura aus ein oder mehreren Suppliern. Semantisch gesehen wird eine Anwendung durch Basisaufgaben (*primitives*) aufgebaut. Sie beschreiben die Funktionalität der Anwendung aus Nutzersicht. In Abbildung 2.1 ist ein Beispiel[22] für eine Aufgabenbeschreibung, die zwischen der Aura des Nutzers und einem Dienst ausgetauscht wird. Zum einen wird die Basisaufgabe näher spezifiziert, mit allgemeinen Eigenschaften wie der voraussichtlichen Laufzeit und speziellen Diensteigenschaften, in diesem Beispiel die Rechtschreibkorrektur. Zum anderen werden getrennt davon die vom Dienst genutzten Daten spezifiziert. Diese Information benötigt Aura, um beim Wechsel der Umgebung die Daten migrieren zu können.

```
<auraTask id="demo">
  <service type="editText">
    <duration unit="minutes" bad="10" good="30"/>
    <settings pane_height="360" pane_width="200">
      <spelling enabled="yes" ignoreAllCaps="yes"/>
      <editing overstrike="no" replaceSelection="yes"/>
    </settings>
  </service>
  <material origin="myTextFile" format="txt">
    <state cursor="104" scroll="28" zoom="100"/>
  </material>
</auraTask>
```

Abbildung 2.1 Beschreibung eines Dienstes zur Textbearbeitung

Architektur. Der Aufbau von Aura reflektiert die oben genannten Sichten. In Abbildung 2.2 und Abbildung 2.3 wird die Architektur von Aura dargestellt.

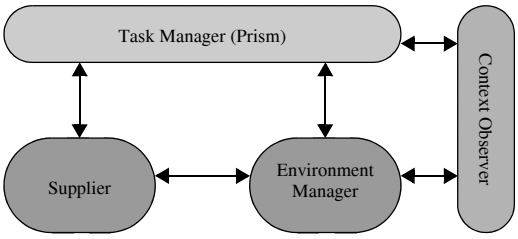


Abbildung 2.2 Bestandteile von Aura

Die persönliche Aura des Nutzers wird durch *Prism* verkörpert. Prism versucht durch Adaption, Umgebungswechsel und Änderungen an der vorhandenen Umgebung vor dem Nutzer zu verbergen. Weiterhin adaptiert Prism die Anwendung, wenn der Nutzer die Aufgabe ändert oder sich der Anwendungskontext ändert.

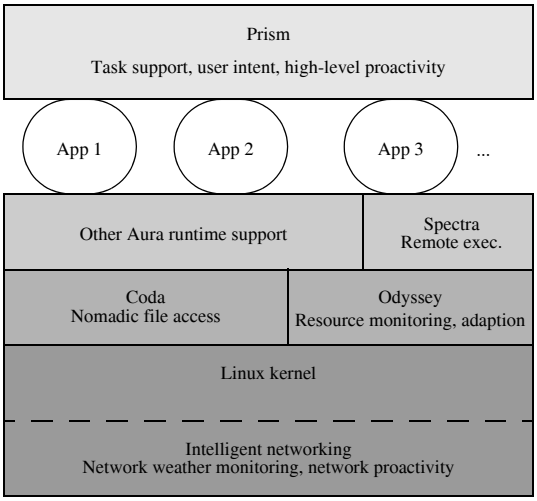


Abbildung 2.3 Schichtmodell von Aura

(Service) *Suppliers* sind Anbieter von Diensten, die eine Anwendung kapseln. In der Regel handelt es sich hierbei um eine Kapselung von vorhandenen, nicht speziell für Aura entwickelten Diensten und Anwendungen, die dadurch ihre Funktionalität für Aura bereitstellen. Die Beschreibung des Dienstangebots wird durch eine XML-Beschreibungsdatei realisiert. Dabei wird ein gemeinsames Vokabular für die Anforderungsbeschreibung vorausgesetzt.

Der *Context Observer* stellt die Verbindung zwischen der Umgebung und Prism her. Einerseits stellt er Prism Informationen über den Zustand und Ereignisse in der Umgebung zur Verfügung. Andererseits bietet er der Umgebung die Möglichkeit, den Zustand und Ereignisse der Aura eines Nutzers in Erfahrung zu bringen.

Der *Environment Manager* verwaltet alle Suppliers, d.h. Dienstanbieter in einer Umgebung. Der Manager ist auch für den verteilten Dateizugriff zuständig. Dienste müssen sich beim Environment Manager der entsprechenden Umgebung registrieren. Bestehen Nutzungsbeschränkungen für Dienste, so wird die Einhaltung dieser Beschränkungen durch den Environment Manager durchgesetzt. Zusätzlich zur Dienstregistrierung kann der Environment Manager auch dahingehend erweitert werden, verschiedene Dienstzusammenstellungen, die dieselbe Aufgabe erfüllen könnten, miteinander zu vergleichen und eine auszuwählen, die “am Besten” zu den Wünschen des Nutzers passt. Die vorhandene Implementierung nutzt diese Möglichkeit nicht und es wird auch keine Definition einer Güte der Komponentenauswahl vorgenommen.

Die Interaktion der einzelnen Bestandteile von Aura wird durch *Connectors* realisiert. Aura kennt vier verschiedene Typen von Verbindungen - zwischen Prism und dem Environment Manager, zwischen Prism und dem Context Manager, zwischen Prism und den Suppliern und zwischen dem Context Manager und dem Environment Manager. Interaktion zwischen verschiedenen Diensten, die in einer Aufgabe zusammenwirken, wird von Aura nicht durch spezielle Infrastrukturmechanismen unterstützt. So bleibt es Aufgabe des Programmierers der Komponente, mögliche Kooperation mit anderen Diensten zu realisieren.

Die Realisierung von Aura basiert auf diversen bereits vorhandenen Technologien, die an der Carnegie Mellon University entwickelt worden sind. Die Verteilung von Daten wird über das verteilte Dateisystem Coda[36] realisiert. Odyssey[33] ist ein Dienstgütemanager, der Dienste den Erfordernissen einer Anwendung anpassen kann.

Adaption. Wesentlich für die Adaption in Aura ist der Kontextbezug. Die Implementierung des Context Observers entscheidet, welche Informationen aus der Umgebung an den Task Manager weitergereicht werden. Diese Informationen, beispielsweise der Aufenthaltsort des Nutzers, können Adaption auslösen.

Die Mobilität des Nutzers zwischen Umgebungen und Adaption innerhalb einer Umgebung werden in Aura durch Konnektoren zwischen Prism und den Service Suppliers unterstützt. Auf Prism-Seite fungieren die Konnektoren als Proxies zu den Service Suppliers. Wird die Verbindung zum Service Supplier unterbrochen oder steht der Dienst nicht mehr zur Verfügung, dann übernimmt der Environment Manager die Aufgabe, einen neuen Dienst zu suchen, der die Aufgabe erfüllen kann. Dieser Prozess ist transparent für Prism. Über die Konnektoren kann auch der Zustand einer Aufgabe eingefroren und wiederhergestellt werden. Dabei ist die Protokollebene fest vorgegeben, wohingegen die Implementierung eines Konnektor-Typs gerätespezifisch variieren kann.

2.3.2 Gaia

Gaia[27, 30, 34, 35] ist aus dem Meta-Betriebssystem-Projekt 2K hervorgegangen. Dabei handelt es sich um eine reflexive Middleware, die auf CORBA[2] basiert, genauer auf Douglas Schmidts TAO[12]. Zusätzlich zu den durch 2K geschaffenen APIs wurde Gaia um diverse notwendige Dienste für ubiquitäre Systeme erweitert.

Gaia ist, wie Aura auch, umgebungsbezogen. *Active Spaces* (siehe Abbildung 2.4) beinhalten Nutzer, Anwendungen, Kontext, Soft- und Hardwareinfrastruktur. Der Nutzer befindet sich in einer oder mehreren Sitzungen (*sessions*) mit Anwendungen in einem Active Space, mit deren Hilfe er seine Aufgaben erledigt. Aus Benutzersicht ist Gaia allerdings nicht aufgabenorientiert. Gaia bietet dafür ein Anwendungsrahmenwerk für den Anwendungsprogrammierer, welches die Entwicklung von ubiquitären Anwendungen vereinfachen soll.

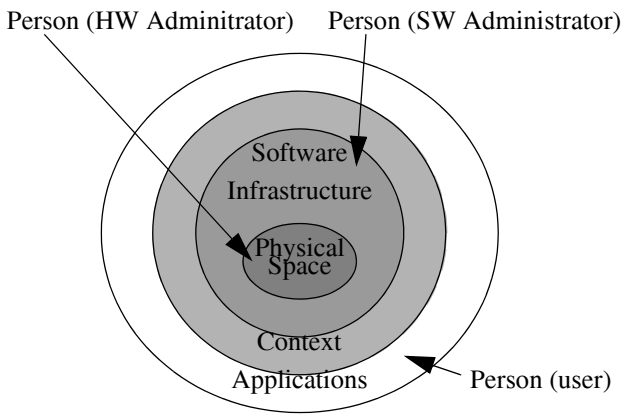


Abbildung 2.4 Active Space

Anwendungsmodell. Gaia bietet nicht nur Dienste zur Unterstützung der Anwendungsentwicklung an, sondern definiert für seine Anwendungen eine eigene Architektur. Die von Gaia unterstützte Anwendungsarchitektur wird als MPCC-Paradigma (*model, presentation, controller and coordinator*) bezeichnet und in Abbildung 2.5 dargestellt. Die Erweiterung des MVC-Paradigmas (*model, view, controller*) wurde zur Anpassung an die Anforderungen eines ubiquitären Systems vorgenommen - die Wahrnehmung der Umwelt (durch Sensoren), die Einbeziehung eines Kontexts, Anwendungsaufteilung, Benutzerfokussierung und Mobilität.

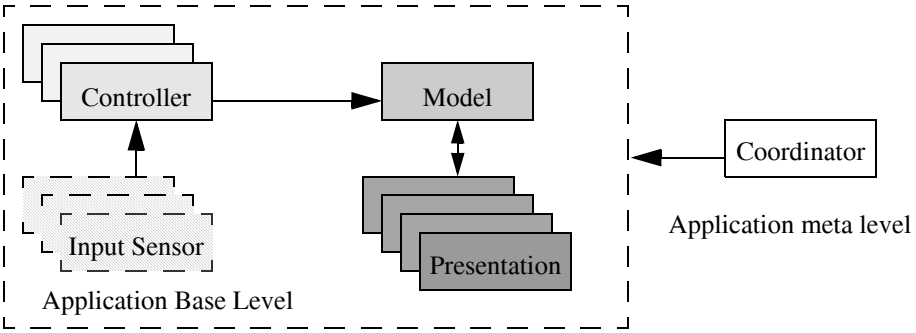


Abbildung 2.5 Modell einer MPCC Anwendung

Die Aufteilung von Anwendungen ermöglicht die Beschreibung einer Anwendung durch die Kombination zweier Anwendungsbeschreibungssprachen, AGD (*Application Generic Description*) und ACD (*Application Customized Description*). In der AGD beschreibt der Entwickler die von einer konkreten Umgebung unabhängigen Parameter einer Anwendung. Dies sind eine Liste der Komponenten der Anwendung, die Anzahl der möglichen Instanzen und die Minimalanforderungen an die Umgebung. In einem Active Space wird dann automatisch auf Basis der vorhandenen Komponenten, die im *Space Repository* (siehe Abbildung 2.6) registriert sind, und den Anforderungen der Anwendung, die in der ADG festgehalten sind, eine Application Customized Description spezifisch für den momentanen Active Space generiert. So ist für jeden Active Space zumindest eine spezielle ACD pro Anwendung erforderlich. Die ACD regelt die Instanziierung und den Zusammenschluss aller an der Anwendung beteiligten Komponenten.

Der *Coordinator* erstellt und verwaltet die Bindungen zwischen den Anwendungskomponenten. Der Anwendungsprogrammierer kann durch Programmieren eigener Regeln (*Policies*) Anwendungsaspekte wie zum Beispiel Reaktionen auf Änderungen im Active Space und Mobilität anpassen. Die Kommunikation zwischen Anwendung und Active Space ist ereignisbasiert.

Architektur. Die Ähnlichkeit zu einem Betriebssystem zeigt sich in der Schichtstruktur von Gaia. Neben den zentralen Verwaltungsfunktionen auf unterster Ebene sind verschiedene weitere Dienste im Gaia Kern angesiedelt. Das Space Repository enthält eine Beschreibung aller im Active Space verfügbaren Dienste und Komponenten. Durch den Event Manager Service wird die Kommunikation zwischen Anwendungen und dem Active Space umgesetzt. Digitale und physische Ressourcen (*Digital and Physical Entities*) werden durch den *Presence Service* verwaltet. Durch Vergabe und Erneuerung von Leases wird die Verfügbarkeit digitaler Ressourcen gesteuert.

Das Vorhandensein physischer Ressourcen wird durch verschiedene Formen von Sensoren überwacht, die an den Presence Service melden. Ein Kontextdienst (*Context Service*) stellt von Sensoren gelieferte Informationen über den Active Space für Anwendungen bereit. Auf den Kontextinformationen aufbauend wird ein Dateisystem für die Anwendungen angeboten. Die Bereitstellung von Speicherplatz, die Organisation der Daten und die Konvertierung in gerätspezifische Repräsentationen sind durch das *Context File System* implementiert. Gaia baut auf einer modifizierten CORBA-Middleware auf und realisiert die angebotenen Dienste des Kerns auch weitgehend durch Verwendung von CORBA-Funktionen.

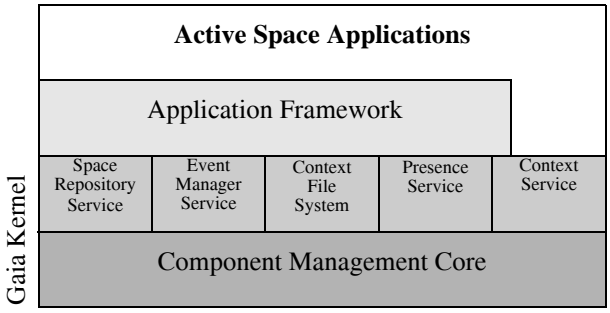


Abbildung 2.6 Architektur von Gaia

Adaption. Automatische Adaption auf Grund von Umgebungsänderungen ist in Gaia nicht vorgesehen. Die Anwendung kann aber über Änderungen in der Umgebung in Kenntnis gesetzt werden. Dies geschieht zum einen durch Events und zum anderen dadurch, dass sie für die genutzten Dienste die Bindungen regelmäßig erneuern muss. Dieser Erneuerungsvorgang kann natürlich auch zum Adaptieren der Anwendung genutzt werden. Im Falle, dass ein Dienst nicht mehr in der Lage ist, eine zugesicherte Leistung zu erbringen, wird durch das Ereignissystem dieser Fehler propagiert. Die Anwendung kann dann durch den Coordinator mittels der Anwendungsbeschreibung die Bindungen neu aufbauen. Änderungen an der Umgebung werden auch über Events bekannt gegeben. Neue Ressourcen werden vom Presence Service gefunden und registriert und ebenso entfernt, wenn sie sich abmelden oder nicht zurückmelden. Den Anwendungen, die solche Events verarbeiten, steht es frei, dementsprechend darauf zu reagieren. Allgemein wird Adaption in Gaia wie auch in Aura eher als der Fall des Umzugs von einer Umgebung in eine andere aufgefasst als die Änderung der momentanen Umgebung.

2.3.3 One.World

Für das Verständnis der Verteilungsinfrastruktur One.World ist eine Betrachtung der ursprünglichen Entwurfsziele notwendig. Die Grundlage von One.World bilden Programmierrichtlinien, die von Grimm in Programming for Pervasive Computing Environments[24] beschrieben werden.

Programmierrichtlinien für Ubiquitous Computing. Ubiquitäre Systeme stellen ein Paradigmenwechsel für die Programmierung von Anwendungen dar. Grimm hat sich in seiner Arbeit dem Einfluss des Paradigmenwechsels auf die Entwicklung von Software gerade in Bezug auf Dynamik und Adaption angenommen, indem er geeignete Programmiermethoden erörtert, mit denen Adaption umgesetzt werden kann. Drei Prinzipien sind nach Grimm zur Wahrung der Adaptionsfähigkeit notwendig.

Sichtbarkeit von Änderungen. Um auf Änderungen reagieren zu können, müssen diese dem Entwickler sichtbar gemacht werden. Eine Abstraktionsschicht, die alle diese Informationen herausfiltert, kann nach Grimm zu einem zu sorglosen Umgang von Seiten des Entwicklers mit der Adaption führen. Weiterhin steht somit dem Entwickler die Möglichkeit offen, eigene Strategien zur Adaption zu implementieren. Trotzdem sollte eine Abstraktionsschicht existieren um unter Umsetzung beider vorangegangenen Forderungen dem Entwickler die Anwendungsentwicklung zu erleichtern.

Dynamische Zusammenstellung. Heutige Anwendungen sind in aller Regel statisch programmiert, d.h. sie haben Abhängigkeiten mit der Umgebung, die beim Starten oder schon zur Übersetzungszeit aufgelöst werden und danach zur korrekten Ausführung der Anwendung immer zur Verfügung stehen müssen. Dieses Modell ist ungeeignet für ubiquitäre Systeme, da sich Rahmenbedingungen für die Anwendung mit hoher Wahrscheinlichkeit, z.B. durch Hinzukommen und Wegfall von Ressourcen ändern werden. An Stelle der statischen Bindungen in die Umgebung sollte eine Anwendung dynamisch auf Basis vorhandener Ressourcen zusammengestellt

werden. Dies kann z.B. durch eine geeignete komponentenbasierte Verteilungsinfrastruktur realisiert werden.

Trennung von Daten und Funktionalität. Im Gegensatz zu nicht verteilten Anwendungen ist die Zusammenfassung von Daten und Funktionalität, z.B. durch Objekte, nicht immer vorteilhaft. Objekte schränken den Zugriff auf die dahinter liegenden Daten ein und müssten so gestaltet werden, dass alle momentanen und zukünftigen Anwendungen die benötigten Manipulationen an den Daten durchführen können. Liegen dagegen die Daten in einem wohldefinierten Format vor, beispielsweise in einer relationalen Datenbank oder sind als XML-Datenstrom verfügbar, so können Daten und Funktionalität unabhängig voneinander weiterentwickelt werden. Festzuhalten bleibt, dass die Zusammenfassung von Daten und Funktionalität in verteilten Systemen erst auf höherer Ebene, z.B. für Anwendungsprogrammierer, die eine Middleware benutzen, sinnvoll ist. Auf niederen Abstraktionsebenen ist dagegen ein direkter Zugriff auf die Daten notwendig.

Die programmiertechnische Umsetzung der Richtlinien kann auf verschiedenen Ebenen vorgenommen werden. Eine Möglichkeit ist, dass jede Anwendung über eigene Adaptionsmechanismen verfügt. Dies erschwert allerdings nicht unerheblich die Umsetzung der oben genannten Programmierprinzipien für verteilte Systeme. Es müsste eine globale Einigkeit über die Form, in der Änderungen sichtbar gemacht werden, über die Schnittstellen, durch die zusammengestellte Systeme kooperieren können und über die Darstellungsform von Daten und Funktionalität existieren. Eine nahe liegende Lösung ist die Zusammenfassung dieser Aspekte in einer gemeinsamen Abstraktionsschicht, auf der Anwendungen für ubiquitäre Systeme nachher aufbauen. Aus diesen Überlegungen heraus ist One.World entstanden.

One.World wurde an der Universität von Washington im Rahmen einer Dissertation von Robert Grimm geschaffen. Die Basis für One.World bilden die von Grimm aufgestellten, eben beschriebenen Programmierprinzipien für Ubiquitous Computing. Ein Datenmodell zur Trennung von Daten und Code, ein Komponentensystem zur ad-hoc Anwendungszusammenstellung und ein eventbasiertes Kommunikationssystem um Änderungen im System zu propagieren sind die Kernbestandteile der Verteilungsinfrastruktur. Im Gegensatz zu den beiden vorangegangenen Verteilungsinfrastrukturen bietet One.World einen umgebungsunabhängigen Ansatz und setzt nicht bestimmte Strukturen wie beispielsweise Smart Environments voraus.

Anwendungsmodell. Ein wesentliches Designelement in One.World ist die Trennung von Daten und Funktionalität. Daten werden in so genannten Datentupeln verwaltet. Kommunikation erfolgt über Ereignisse. Komponenten beinhalten Funktionalität und bieten Behandlungsschnittstellen für asynchrone aus- und eingehende Ereignisse an. Das Eventsystem stellt auch ein Typsystem bereit, das es Komponenten ermöglicht, über angebotene Schnittstellen mit anderen Komponenten zu kommunizieren. Strukturiert werden diese Elemente in so genannten *Environments*. Environments stellen einen Namensraum bereit, in dem sich Datentupel, Komponenten aber auch weitere Environments befinden können. Jede Anwendung befindet sich mindestens in einem Environment, kann aber auch mehrere Umspannen. Der Ressourcenbedarf einer Anwendung wird durch ein Lease-System verwaltet. Ein Lease muss in regelmäßigen Abständen erneuert werden, um die Reservierung der Ressource aufrecht zu erhalten. Die Erneuerung dient somit dazu, Änderungen in der Umgebung sichtbar zu machen.

Architektur. Die Architektur von One.World reflektiert ebenso wie das Anwendungsmodell die von Grimm aufgestellten Programmierprinzipien. Der Kern von One.World bildet eine virtuelle Maschine, die Portabilität ermöglicht. Zur Vereinfachung der Implementierung wird Java als Programmiersprache und die Virtuelle Maschine für Java verwendet. Zu den Foundation Services in One.World zählen zusätzlich zur Virtuellen Maschine die Implementierung der Environments, Tuples und das asynchronen Eventsystem. Weiterhin stellt One.World eine Anzahl von Systemdiensten zur Verfügung, die von Anwendungen genutzt werden können. Checkpointing und Migration stellen Adaptionsdienste für Anwendungen bereit. Discovery und Remote Events ermöglichen dynamische Zusammenstellung von Anwendungen. Gemeinsamer Zugriff auf Daten

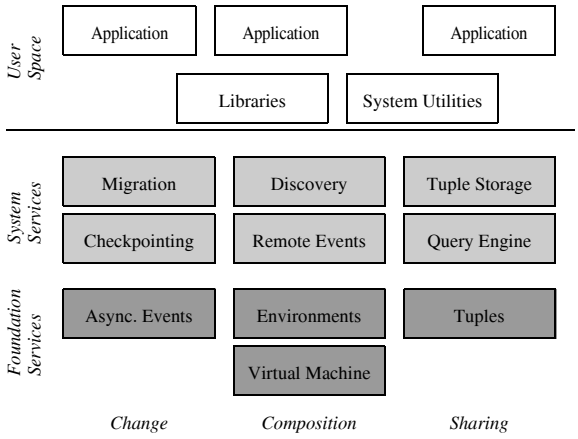


Abbildung 2.7 Architektur von One.World

wird über einen Datencontainer, den Tuple Storage realisiert, der über eine Query Engine abgefragt werden kann.

Adaption. One.World unterstützt die Migration von Anwendungen. Der Checkpointing Service in Verbindung mit dem Migration Service ermöglicht es, den Zustand einer Anwendung, d.h. den Zustand der Environments, das die Anwendung verwendet, zu sichern und wiederherzustellen.

Weitere Adaptionsmöglichkeiten werden durch den Lease-Mechanismus geboten, der Anwendungen zwingt, regelmäßig ihre Bindungen zu überprüfen und zu erneuern. Das dynamische Binden ermöglicht es Anwendungen, zur Laufzeit die Bindungen anzupassen und gegebenenfalls andere Komponenten zu verwenden.

2.3.4 PCom

PCom ist eine komponentenbasierte Verteilungsinfrastruktur für ubiquitäre Systeme. Sie wurde an der Universität Stuttgart als Teil des 3PC-Forschungsprojekts von Handte[26] im Rahmen einer Diplomarbeit vorgestellt. PCom ist ebenso wie One.World nicht auf spezielle Szenarien wie Active Spaces oder Smart Environments beschränkt. Dem Anwendungsprogrammierer wird eine Abstraktion zur Verfügung gestellt, auf deren Grundlage er seine Anwendung aus einzelnen Komponenten und Verträgen als Bindeglied zwischen den Komponenten modelliert. Dabei unterscheiden sich Anwendungen prinzipiell nicht von anderen Komponenten - bis auf die Tatsache, dass Anwendungskomponenten nicht von anderen Komponenten genutzt werden.

Großen Wert wurde bei PCom auf Fehlertoleranz und Portabilität auch auf Kleinstgeräte gelegt. Die Realisierung von Bindungen über Verträge erlaubt verlässliche Aussagen über die Zusammenarbeit von Komponenten und dient gleichzeitig als Entscheidungskriterium um Anwendungen zu konfigurieren. PCom stellt im Vergleich zu Aura und Gaia, die ein Umgebungskonzept besitzen, vergleichsweise wenige Dienste bereit. So verfügt PCom beispielsweise über kein verteiltes Dateisystem. Die Kommunikationsschicht in PCom ist der hohen Portabilität angepasst und basiert auf einer Plug-in-Architektur, die verschiedenste Übertragungsprotokolle auch dynamisch anbieten kann.

Anwendungsmodell. Anwendungen in PCom bestehen aus Komponenten und Verträgen, die die Bindungen zwischen den Komponenten explizit beschreiben. Die Anwendung lässt sich somit als Baum (da pro Komponenteninstanz nur eine eingehende Bindung zulässig ist) mit Knoten als Komponenten und Kanten als Verträge darstellen. In den Verträgen werden die benötigten Schnittstellen und ggf. weitere nicht-funktionale Parameter beschrieben. Auf Grund dieser Beschreibung werden Anwendungen automatisch durch dynamisches Zusammenstellen geeigneter Komponenten konfiguriert. Kommunikation erfolgt von nutzender Komponente in Richtung der

genutzten Komponente durch Verwendung der angebotenen Schnittstellen. Um Fehler oder sonstige Ereignisse, die in einer genutzten Komponente auftreten können, auch entgegen der eigentlichen Kommunikationsrichtung der nutzenden Komponente mitzuteilen, wurde ein Eventsystem implementiert, welches diese entgegengesetzte Kommunikationsrichtung unterstützt.

Beim Instanziieren einer Anwendung wird versucht, rekursiv die Bindungen der Komponenten aufzulösen. Die Auswahl der Komponenten wird dabei durch eine Selektionsstrategie gesteuert, die in ihrer jetzigen Form sowohl lokale als auch Komponenten auf anderen Knoten gleichermaßen in die Auswahl mit einbezieht.

Architektur. PCom basiert auf einer ebenfalls an der Universität Stuttgart entwickelten Middleware, BASE[25]. BASE ist eine dienstbasierte Middleware mit einer hoch modularisierten Kommunikationsarchitektur, die verschiedenste Protokolle und Geräte unterstützt. Auf BASE aufsetzend bietet PCom dem Programmierer ein Komponentenmodell. Analog zu vielen herkömmlichen Verteilungsinfrastrukturen wie J2EE bildet ein *Container* die knotenzentrale bzw. adressraumzentrale Verwaltungsinstanz für Komponenten. Der Container verwaltet beispielsweise den Lebenszyklus der dort vorhandenen Komponenteninstanzen. Auch stellt der Container Dienste für die automatische, dynamische Konfiguration und Adaption von Anwendungen zur Verfügung. Die Auswahlstrategie für die Komponenten einer Abhängigkeit wird containerindividuell durch den *Selector* festgelegt.

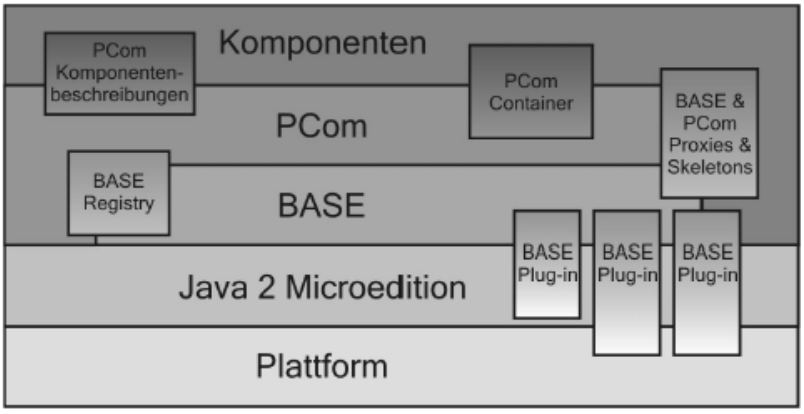


Abbildung 2.8 Schichtmodell von PCom

Die Kommunikation zwischen Komponenten erfolgt, wie bereits beschrieben, in Richtung nutzender zu genutzter Komponente. Dabei werden auf Nutzerseite Stellvertreterobjekte generiert um Verteilungstransparenz herzustellen. Die umgekehrte Kommunikationsrichtung erfolgt über ein Ereignissystem. Dies erlaubt genutzten Komponenten, ihre Nutzer über beliebige, durch Java-Interfaces typisierte Ereignisse in Kenntnis zu setzen. Die von einer Komponente unterstützen Ereignisse, wie auch die implementierten Schnittstellen, werden im Komponentenvertrag festgeschrieben.

PCom und BASE sind vollständig in Java implementiert und auch auf Java-Runtimes für Kleinstgeräte (Java 2 Microedition[7]) lauffähig. Die Virtuelle Maschine schafft dabei, wie auch bei One.World, die Voraussetzung für Portabilität.

Adaption. PCom ist als hochdynamisches System ausgelegt. Die Komponenten, die ein Container zur Verfügung stellt, werden dynamisch geladen, d.h. das Komponentenangebot kann auch zur Laufzeit ergänzt werden. Anwendungen werden dynamisch konfiguriert und die zu einer Abhängigkeit passenden Komponenten automatisch durch einen Selektor ausgewählt. Die vom Selektor verwendete Selektionsstrategie entscheidet im Falle mehrerer verfügbarer Komponenten, welche letztendlich genommen wird. Das gleiche Verfahren findet auch bei der Rekonfiguration bzw. Adaption der Anwendung an geänderte Umgebungsbedingungen Anwendung. Im Falle der Rekonfiguration wird in der Implementierung des bindungsspezifischen *ExceptionHandler* die Reaktion auf einen solchen Vorfall festgelegt.

Komponenten werden durch Verträge mit beschränkter Laufzeit aneinander gebunden, die durch ein Lease-System realisiert wird. Weiterhin ist eine dynamische Anpassung von Angebot und Nachfrage möglich. Falls ein Vertrag nicht mehr erfüllt werden kann, so kann, je nach Selektor, automatisch eine neue Komponente (sofern vorhanden) gefunden und gebunden werden, die die Anforderungen erfüllt. Durch ein Checkpointing-System kann der Zustand einer oder mehrerer Komponenten gesichert und an anderer Stelle wiederhergestellt werden.

Die Möglichkeiten zur automatischen Konfiguration von Anwendungen in PCom sind Bestandteil genauerer Untersuchungen in Kapitel 3.2.

2.4 Architekturbasierte Adaption

Diese Arbeit soll die Nutzung von Adaptionsfreiräumen durch den Anwendungsprogrammierer erforschen. Dazu ist die Darstellung einer komponentenbasierten Anwendung, einschließlich ihrer Architektur wesentlich. Adaption greift tief in die Struktur von Anwendungen ein. Eine formale Beschreibung von Anwendungen in ubiquitären Systemen birgt den Vorteil, die Eingriffe durch Adaption besser ausdrücken zu können. Eine solche Beschreibung kann im Adaptionsfall auch als Hilfe zur Entscheidungsfindung herangezogen werden. Dies erfordert eine Datenrepräsentation der Anwendungsstruktur. Im Rahmen der Softwareentwicklung wurde im Softwareengineering der Begriff des Anwendungsmodells eingeführt. Nach Hesse ist ein Anwendungsmodell wie folgt definiert:

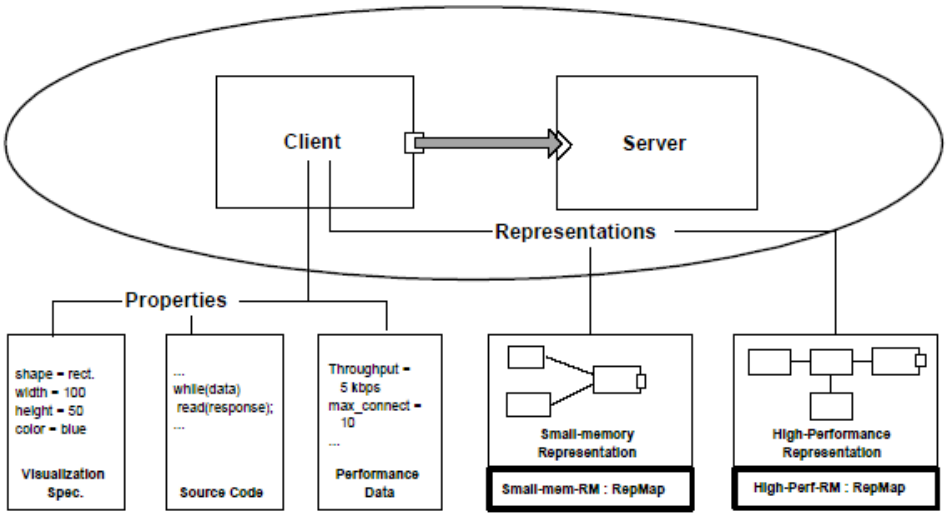
> *Ein Anwendungsmodell ist ein Modell, das sich auf einen Weltausschnitt bezieht, der Gegenstandsbereich einer Software- Anwendung ist. [...] Sie erstrecken sich auf alle Sprachkategorien, d.h. passive (datenorientierte), aktive (funktionsorientierte) und zusammenhangsbildende Elemente. Sie geben Strukturen für die Exemplar-Ebene vor und setzen sich damit aus Elementen der Typ-Ebene zusammen. Sie entstehen in der Analyse- und Definitionsphase, sind aber für den gesamten weiteren Systementwicklungsprozeß maßgeblich.[28]*

Ein Anwendungsmodell beschreibt also die Daten, die Funktionalität und die Struktur einer Anwendung. Allerdings ist das Anwendungsmodell per Definition Teil der Dokumentation eines Systems und daher für Menschen gemacht.

Für die Nutzung durch eine Verteilungsinfrastruktur in ubiquitären Systemen ist eine computerbasierte Beschreibung einer Anwendung erforderlich. Auf dieser Basis wird im Folgenden der Begriff Anwendungsbeschreibung definiert. Eine Anwendung in ubiquitären Systemen wird durch drei Gruppen von Eigenschaften charakterisiert. Dies sind die funktionalen Eigenschaften, d.h. die syntaktische Schnittstelle, die nicht funktionalen Eigenschaften, beispielsweise Energieverbrauch oder Speicherverbrauch und die strukturellen Eigenschaften, d.h. eine Beschreibung der Struktur, in der die Anwendungskomponenten miteinander verbunden sind. Die drei Gruppen von Eigenschaften in einer Datenstruktur zusammengefasst wird in dieser Arbeit als Anwendungsbeschreibung bezeichnet.

2.4.1 Acme

Im Bereich des Softwareengineerings existieren bereits Spezifikationssprachen, mit deren Hilfe sich Strukturen bzw. Anwendungsarchitekturen beschreiben lassen. Dabei handelt es sich um die sogenannten Architecture Description Languages (ADL). Die unter Verwendung einer ADL erstellten Beschreibungen werden dann im Weiteren Softwareentwicklungsprozess als Modell für die beschriebene Anwendung verwendet. Eine der existierenden Sprachen ist Acme[23]. Sie wurde von Garlan et al. speziell für die Beschreibung komponentenbasierter Systeme entworfen. Acme beinhaltet nach Garlan, wie die meisten anderen ADLs auch, einige Standard-Beschreibungsbausteine.




```
System simpleCS = {
 Component client = {
        Port sendRequest;
        Properties { requestRate : float = 17.0; sourceCode : externalFile = "CODE-LIB/client.c" }
 }
 Component server = {
        Port receiveRequest;
        Properties { idempotent : boolean = true; maxConcurrentClients : integer = 1;
               multithreaded : boolean = false; sourceCode : externalFile = "CODE-LIB/server.c" }
        Representation serverDetails = {
         System serverDetailsSys = {
         Component connectionManager = {
          Ports { externalSocket; securityCheckIntf; dbQueryIntf } }
         Component securityManager = {
          Ports { securityAuthorization; credentialQuery; } }
         Component database = {
          Ports { securityManagementIntf; queryIntf; } }
         Connector SQLQuery = { Roles { caller; callee } }
         Connector clearanceRequest = { Roles { requestor; grantor } }
         Connector securityQuery = {
          Roles { securityManager; requestor } }
         Attachments {
          connectionManager.securityCheckIntf to clearanceRequest.requestor;
          securityManager.securityAuthorization to clearanceRequest.grantor;
          connectionManager.dbQueryIntf to SQLQuery.caller;
          database.queryIntf to SQLQuery.callee;
          securityManager.credentialQuery to securityQuery.securityManager;
          database.securityManagementIntf to securityQuery.requestor; }
        }
        Bindings { connectionManager.externalSocket to server.receiveRequest }
 }
}
Connector rpc = { ... }
Attachments { client.send-request to rpc.caller ;
              server.receive-request to rpc.callee }
```

Abbildung 2.9 Auszug aus einer Acme-Beschreibung

- *Komponente*: Komponenten in ADLs werden weitestgehend konform zur Definition von Szyperski verwendet. Sie stellen Funktionalitätseinheiten dar, die durch wohldefinierte Schnittstellen angesprochen werden.
- *Bindung*: Bindungen stellen Interaktionen zwischen Komponenten dar. Sie stehen für die Kommunikation, die zwischen Anwendungskomponenten stattfindet und repräsentiert so die Abhängigkeiten der Komponenten. Je nach Beschreibungsform kann eine Bindung vom Methodenaufruf bis zum kompletten, komplexen Interaktionsprotokoll alles enthalten.
- *Systeme*: Ein System ist eine konkrete Konfiguration bzw. ein Graph, der aus Bindungen und Komponenten besteht. Systeme können auch hierarchisch angeordnet und geschachtelt sein, d.h. ein Gesamtsystem kann sich aus vielen Subsystemen zusammensetzen.

- *Eigenschaften*: Über strukturelle Informationen hinaus erlauben Application Description Languages die Definition semantischer Informationen, die der Beschreibung hinzugefügt werden.
- *Einschränkungen*: Die Spezifikation von Einschränkungen erlaubt die Definition von strukturellen oder semantischen Eigenschaften, die sich im Laufe der Weiterentwicklung der Software nicht ändern sollten.
- *Architektur-Stil*: ADLs beinhalten diverse Grundstile der Softwarearchitektur, bestehend aus Designelementen und Regeln zur Zusammenstellung dieser Elemente. Ein typisches Beispiel für einen Architekturstil ist die Client-Server-Architektur.

Zusätzlich zu den erwähnten Beschreibungseinheiten bietet Acme noch andere Elemente. Repräsentationen stellen konkretere Beschreibungen von Komponenten dar und können als Alternativen für eine Komponente in der Beschreibung verwendet werden. Die Verbindungen zwischen der Repräsentation und der umgebenden Komponente werden in einer speziellen Beschreibungstabelle, der *rep-map*, festgehalten. Acme verfügt über gleichmächtige grafische und textuelle Notationen. In Abbildung 2.9 ist ein Beispiel für eine einfache Client-Server-Anwendung gegeben.

2.5 Adaption in nicht-ubiquitären Systemen

Adaption findet nicht nur in ubiquitären Systemen statt. Es lassen sich dabei zwei Klassen der Adaption unterscheiden. Die erste Klasse ist eine für die Anwendung transparente Form der Adaption. Die Anwendung wird nicht über die Vorgänge der Adaption informiert und hat auch keinen Einfluss darauf. Die Entscheidungen basieren ausschließlich auf internen, im System festgelegten oder berechneten Parametern. Beispielsweise wird im Bereich der Streaming-Technologien (Video-, Audio-) Adaption eingesetzt, um die Datenströme den Netzwerkgegebenheiten anzupassen. Auch werden im Bereich Load Balancing und Grid Computing Anwendungen angepasst, um eine optimale Ressourcennutzung zu erzielen. Ein relativ neues Feld, Self-healing Systems, hat auch eine Verwandtschaft zum Thema Adaption, da hier Anwendungen fehlertolerant entwickelt werden und sich in Fehlersituationen automatisch reparieren.

Die andere Klasse der Adaption findet sich im Bereich des Softwareengineering und greift die Idee der Architekturbeschreibung auf, um mittels expliziter, durch die Anwendung definierter Regeln, Adaption durchzuführen. Als Beispiel wird hier das Rainbow-Framework vorgestellt.

2.5.1 Odyssey

Odyssey wurde im Zusammenhang mit Aura schon kurz angesprochen. Odyssey wurde vor Aura als Ressourcenverwaltung entwickelt. Die Hauptzielsetzung bei der Entwicklung von Odyssey war es, Anwendungen eine Schnittstelle zu Informationen über Ressourcenauslastung zu bieten. Die Anpassung erfolgt durch die Anwendung selbst; Odyssey informiert die Anwendung dabei über relevante Änderungen bei den vorhandenen Ressourcen.

In ihrer Veröffentlichung[33] definieren Noble et al. zwei Größen, um die Qualität der Adaption zu messen. Zum einen definieren sie *Fidelity* als eine Art Qualitätsfaktor aus Anwendungssicht. Die Dimensionen von Fidelity sind dabei variabel. Bei einer Videostreaming-Anwendung würden beispielsweise die Bildwiederholrate und die Auflösung bzw. die verwendete Kompression in die Fidelity mit einfließen. Zum anderen wird *Agility* als Reaktionszeit des Systems auf Änderungen bei den Ressourcen eingeführt. Dabei ist festzustellen, dass es sich hier nicht um wirklich messbare Größen handelt, sondern um eine bewertete, anwendungsspezifische Kosten-Nutzen-Sicht.

Im Rahmen des Projekts wurden verschiedene Adaptionsstrategien ausprobiert. Im Folgenden werden zwei dieser Strategien vorgestellt. Die erste hier vorgestellte Strategie[32] basiert auf einer Historie über den Ressourcenbedarf von Anwendungen. Durch Messen des Ressourcenbedarfs von Anwendungen mit verschiedenen Qualitätsstufen und der Auswertung durch maschinelles Lernen wurde versucht, eine Ressourcenbedarfs-Vorhersage zu entwickeln. Die Strategie hat in den untersuchten Situationen bei der Vorhersage Abweichungen zwischen 15%

und 20% zum realen Ressourcenverbrauch erzeugt. In speziellen Einsatzbereichen ließe sich dieses Vorhersagesystem verfeinern und zur vorausschauenden Ressourcenplanung verwenden.

Die zweite Strategie[19] ist ein spezialisierter Ansatz der ersten. Hier wird nur der Energieaspekt der Anwendung berücksichtigt. Das System basiert auf einer Überwachungskomponente, die die zur Verfügung stehende Energie überwacht. Der zweite Teil ist ein Programm zur Bestimmung des Energieverbrauchs einer Systemkomponente. Aus den gewonnenen Werten kann dann die Anwendung so adaptiert werden, dass die zur Verfügung stehende Energie in Bezug auf anwendungsspezifische Kriterien, beispielsweise die Laufzeit, optimal genutzt wird.

2.5.2 Grid Computing

Anwendungen in Grid Computing-Systemen sind inhärent verteilt. Zur optimalen Nutzung des Grid sind unter anderem eine Ressourcenverbrauchsüberwachung, eine Ressourcenverbrauchsvorhersage, die Adaptionsfähigkeit der Anwendungen sowie Scheduling-Strategien zur Verteilung von Ressourcen notwendig. Die Adaption bei Grid Computing muss vollautomatisch geschehen, da in den meisten Fällen die Anwendungen nicht interaktiv sind. Ein Beispiel für einen Algorithmus, der die Adaption koordiniert, wird in *Coordinating Adaptations in Distributed Systems*[18] von Ensink et al. vorgestellt. Der Anwendungsprogrammierer soll in diesem Ansatz durch eine in einem Framework bereitgestellte automatische Adaptionsstrategie entlastet werden. Eine Anwendung wird in dem Framework durch einen *Static Task Graph (STG)* modelliert, der den Kontrollfluss der Anwendung repräsentiert. Eine Makrosprache - *Program Control Language (PCL)* - erlaubt es, eine durch den STG repräsentierte Anwendung zu adaptieren.

2.5.3 Autonomic Computing

Autonomic Computing ist eine Initiative von IBM, der überhand nehmenden Komplexität im IT-Sektor zu begegnen. Die Vision[29] des Autonomic Computing besteht in Systemen, die sich selber automatisch verwalten. Die Administratoren sollen sich nicht mehr um Details der Benutzung und Wartung von Systemen kümmern müssen und die Anwender sollen in den Genuss zuverlässiger und performanter Systeme kommen. Die im Speziellen genannten Bereiche der Vision sind *Self-Configuration*, *Self-optimization*, *Self-healing* und *Self-protection*.

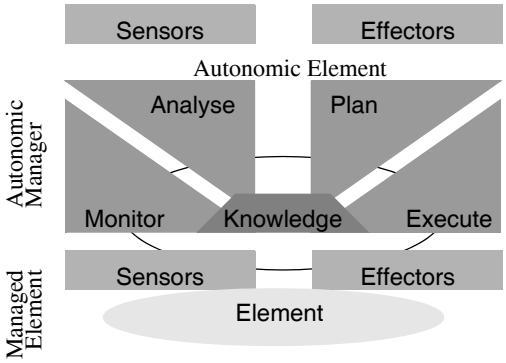


Abbildung 2.10 Autonomic Element

Self-Configuration beinhaltet die automatische Konfiguration von Softwarekomponenten durch ein einfaches Regelwerk und die automatische Anpassung des Gesamtsystems. Self-Optimization soll Mechanismen zur automatischen Steigerung der Geschwindigkeit und Effizienz von Anwendungen enthalten. Automatische Fehlerfindung, -diagnose und -reparatur sind Bestandteil von Self-Healing. Self-Protection beschreibt Einrichtungen zur automatischen, vorausschauenden Abwehr von schädlicher Software (Viren, Würmer, etc.).

Die der Vision zu Grunde liegenden Architektur (siehe Abbildung 2.10[4]) ist eine Erweiterung bisheriger Komponentenarchitekturen. Eine normale Softwarekomponente wird um eine automa-

tische Verwaltungseinrichtung, den *Autonomic Manager* erweitert, der für Überwachung, Analyse, Planung und Ausführung der Komponente zuständig ist. Dabei baut der Autonomic Manager auf einer vorgegebenen und durch die Überwachung und Analyse selbst gewonnenen Wissensbasis auf.

2.5.4 Load Balancing

Load Balancing wird in vielen Bereichen von Client-Server-Architekturen eingesetzt; unter anderem auch in komponenten- oder objektbasierten Verteilungsinfrastrukturen, um die Last zwischen verschiedenen Application Servern aufzuteilen. Load Balancing ist dabei für die Anwendungen transparent - eine Anwendung hat keinen Einfluss auf die Verteilung der genutzten Komponenten und in aller Regel auch keinen Zugriff auf Informationen, auf welchem Application Server eine spezielle Komponente gerade ausgeführt wird.

Die Verteilung der Komponenten kann durch verschiedenste Algorithmen erfolgen. Exemplarisch werden an dieser Stelle drei in einer Veröffentlichung[17] von Lap-Sun Cheung et al. vorgestellten Verfahren kurz im Hinblick auf die Verteilungsstrategie aufgeführt.

Das erste Verfahren verwendet Java Spaces - eine Art gemeinsam genutzter Speicher - als zentralen Taskpool, aus dem sich die Server selbsttätig die zu bearbeitenden Aufgaben holen. Für die Anwendung ist nur der Taskpool sichtbar. Das zweite Verfahren beinhaltet einen Verteilungsalgorithmus, der eingehende Aufgaben auf die Server auf Basis der bei den jeweiligen Servern noch nicht abgearbeiteten Aufgaben verteilt. Ein drittes Verfahren beruht auch auf einem Verteilungsalgorithmus, der versucht, mittels Fuzzy Logic und damit assoziierter Regeln, die Aufgabenlast möglichst gleich zu verteilen.

Alle Verfahren sind aus Anwendungssicht transparent - und somit zumindest nach den Programmierrichtlinien von Grimm für Ubiquitous Computing ungeeignet. Die Verteilungsstrategien könnten jedoch auch in ubiquitären Verteilungsinfrastrukturen Anwendung finden.

2.5.5 Rainbow

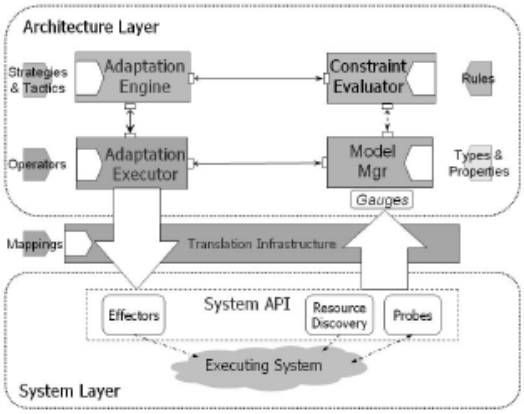
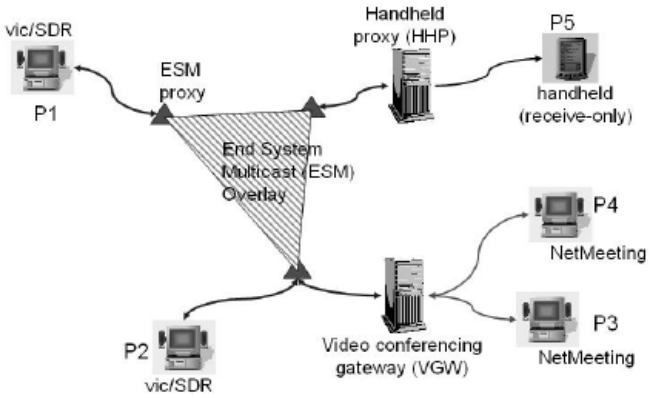


Abbildung 2.11 Das Rainbow Rahmenwerk

Rainbow[16] ist ein an der Carnegie Mellon University entwickeltes Rahmenwerk zur Bereitstellung von Adaptionsmechanismen. Rainbow soll es Anwendungsentwicklern vereinfachen, Adaption in ihre Anwendung zu integrieren und dabei einen möglichst hohen Grad an Wiederverwertung von Adaptionsstrategien ermöglichen. Abbildung 2.11 zeigt den Aufbau des Rahmenwerks. Wiederverwendbarkeit von Adaptionsstrategien wird durch die Trennung von der Adaptionsinfrastruktur und dem systemspezifischen Adaptionswissen erreicht. Die Infrastruktur, die Rainbow auf Systemebene bereitstellt, erlaubt das Überwachen und Protokollieren von Ressourcen. Auf der Architekturebene werden die durch das System bereitgestellten, aggregierten In-

formationen ausgewertet. Die Auswertung der Informationen kann dann Adaption von Anwendungen zur Folge haben. Die Translation Infrastructure ist für die Abbildung der auf der Systemebene erfassten und bereitgestellten Informationen auf eine für die Architekturebene geeignete Repräsentation zuständig. Dieses systemspezifische Wissen bildet die Grundlage sämtlicher Adaptionsentscheidungen.



Types: VicT, NetMeetingT, HandheldT, GatewayT, HandheldProxyT, ESMProxyT, ConnectionT
Properties: GatewayT.cost, GatewayT.load, ConnectionT.bandwidth

Operators:
-HandheldT.move(HandheldProxyT toHHP): switches handheld user to a new handheld proxy.
-NetMeetingT.move(GatewayT toVGW): switches NetMeeting user to a new video gateway.

```
invariant HH : available bandwidth between handheld user HH and its handheld proxy HHP >
minHHBandwidth
!->HHBandwidthStrategy(HH);

invariant VGW : (VGW.cost / number of NM users)
<= maxVGWUnitCost
!->VGWCostStrategy(VGW);

strategy HHBandwidthStrategy (HH : HandheldT) {
 let HHP1 : HandheldProxyT = findBestHHP(HH);
 HH.move(HHP1);
 return true;
}

strategy VGWCostStrategy (VGW : GatewayT) {
 let VGW1 : GatewayT = the gateway with the lowest   cost that can handle the current load;
 if ((query("cost", VGW1) / number of NM users)
     < maxVGWUnitCost) {
  for each NetMeeting user U of VGW {
_  U.move(VGW1);
  }
  return true;
 }
_return false;
}
```

Abbildung 2.12 Beispiel in Pseudocode für eine Adaptionsstrategie in Rainbow.

Für die Adaption eines Softwaresystems in Rainbow wird deren Anwendungsarchitektur verwendet. Diese Architektur muss in einer geeigneten Beschreibungssprache vorliegen, die beispielsweise auf Acme (siehe 2.4.1) basieren kann. Einige zusätzliche Informationen sind jedoch von Nöten, um Adaption auf Basis der Anwendungsarchitektur durchzuführen. Die Erweiterungen gliedern sich in Erweiterungen der Architekturbeschreibung und solche, die Adaptionsmechanismen definieren. Als Erweiterung der Architekturbeschreibung sind erweiterte Beschreibungsmöglichkeiten für die Bindungen zwischen Komponenten nötig, die im Adaptionsfall zur Komponentenauswahl herangezogen werden können. Eine Erweiterung der Eigenschaften-Beschreibung ist die Möglichkeit, Auswertungen über diese Eigenschaften zu definieren. Die nun hinreichenden Informationen über die Anwendungsarchitektur können anschließend durch Be-

schreibung der anwendungsspezifischen Adaptionsmechanismen ergänzt werden. Dazu stellt das Rahmenwerk Adaptionsoperatoren, die für Komponententypen definiert sind und Anfrageoperatoren über Komponenteneigenschaften zur Verfügung. Eine zustandsabhängige Adaptionsstrategie legt das Systemverhalten fest, wenn ein bestimmter Systemzustand erreicht wird. Ein Beispiel für Adaptionsstrategien in einem Video-Konferenz-System findet sich in Abbildung 2.12.

2.6 Vergleich von Adaptionsmechanismen

Die vorangegangenen Ausführungen haben die Entwicklungen im Gebiet des Ubiquitous Computing in mehreren Bereichen gezeigt. Den Anfang für Ubiquitous Computing stellt die Vision von Weiser dar. Die Umsetzung dieser Vision erfordert einen Wechsel in der Denkweise der Programmierer, wie aus Grimms Vorschlägen für Programmierrichtlinien deutlich wird. Allerdings reichen neue Programmierprinzipien nicht aus. Ubiquitous Computing lässt sich nur durch die Bereitstellung geeigneter Verteilungsinfrastrukturen realisieren. Herkömmliche Verteilungsinfrastrukturen können diese Aufgabe nicht übernehmen, da ihre Infrastruktur nicht auf die speziellen Anforderungen ubiquitärer Systeme ausgerichtet ist. Verteilungsinfrastrukturen für ubiquitäre Systeme sind im Moment nur im Forschungsbereich zu finden - marktreife Produkte sind noch nicht in Sicht. So lässt sich auch keine eindeutige Tendenz für bestimmte Formen der Adaption erkennen. Jedes System hat seine eigenen Vorzüge und Einschränkungen. Die aus den untersuchten Systemen gewonnenen Unterscheidungsmerkmale von Adaptionsmechanismen werden im Folgenden beschrieben.

2.6.1 Art der Verteilungsinfrastruktur

Zwischen den in dieser Arbeit vorgestellten Verteilungsinfrastrukturprojekten bestehen erhebliche konzeptionelle Unterschiede und damit auch unterschiedliche Einsatzmöglichkeiten. Am flexibelsten in Bezug auf die Struktur sind One.World und PCom zu verwenden. Sie stellen beide Dienste für die dynamische Bindung von Komponenten zu Anwendungen bereit. Dabei erlauben beide eine beliebige Schachtelung und Bindungstiefe von Komponenten. Die Systeme benötigen einen relativ kleinen Verteilungsinfrastrukturkern und erhalten dadurch die Portabilität auf kleine Systeme. Weiterhin machen sie bezüglich der Umgebung in der ubiquitäre Anwendungen ausgeführt werden keine Vorgaben.

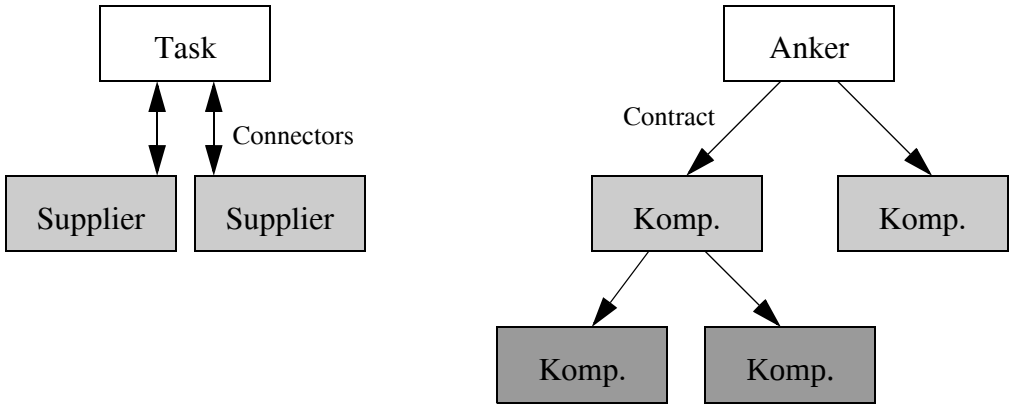


Abbildung 2.13 Beispiel für einstufige (Aura) und mehrstufige (PCom) Bindungen

Die beiden anderen Verteilungsinfrastrukturen, Aura und Gaia definieren Umgebungen (Active Spaces bzw. Smart Environments), in denen sich alle Bestandteile eines ubiquitären Systems befinden. Diese Betrachtungsweise erlaubt gewisse Annahmen in Bezug auf die Umgebung. So ist in einem Active Space oder Smart Environment damit zu rechnen, dass immer ein oder mehrere sehr leistungsfähige Rechner zur Verfügung stehen. Die Annahme spiegelt sich in beiden Architekturen wieder, die über einen erheblich größeren Verteilungsinfrastrukturkern inklusive verteil-

tem Dateisystem im Falle von Aura verfügen. Anwendungen steht hier, verglichen mit den beiden anderen Verteilungsinfrastrukturen, ein wesentlich umfangreicheres Dienstangebot zur Verfügung. Anwendungen für diese Strukturen sind zwar auch komponentenbasiert, doch wird hier nur die Schnittstelle zwischen der Umgebung und der Anwendung betrachtet. In Aura wird sogar von der Anwendung abstrahiert und die Schnittstelle besteht zwischen einer Aufgabe und den Diensten, die diese Aufgabe realisieren. Der Unterschied zwischen den beiden Ansätzen liegt in der Unterscheidung zwischen einstufigen Anwendungen bzw. Aufgaben, die über wohldefinierte Schnittstellen direkt verfügbare Komponenten nutzen und mehrstufigen Systemen, in denen Komponenten andere Komponenten nutzen ohne über deren weiteren Aufbau Annahmen oder Vorgaben machen zu können. Die Unterschiede zwischen Systemen mit einstufigen und mehrstufigen Bindungen wird als Beispiel anhand von Aura und PCom in Abbildung 2.13 gezeigt.

2.6.2 Transparenz

Die Adaptionsstrategien, die bei nicht-ubiquitären und ubiquitären Verteilungsinfrastrukturen zur Anwendung kommen, bewegen sich alle zwischen zwei Extremen in Bezug auf die Transparenz gegenüber der Anwendung. Das eine Extrem besteht in der vollkommenen Anwendungstransparenz. Bei den Adaptionsverfahren im Load Balancing wird beispielsweise die Anwendung adaptiert, ohne dass die Anwendung dies beeinflussen könnte oder überhaupt darüber informiert wird. Das andere Extrem ist die Methode, die in One.World Anwendung findet. Hier werden der Anwendung alle notwendigen Informationen und Mechanismen für die Adaption zur Verfügung gestellt, aber es voll und ganz der Anwendung überlassen, ob und wie adaptiert wird.

2.6.3 Betrachtungsebene

Entscheidend bei einer Adaptionsstrategie ist die Betrachtungsweise, auf deren Basis die Anpassung vorgenommen werden soll. Die beiden extremen Standpunkte sind in diesem Fall eine komponentenlokale Betrachtung, in der nur die Erfüllung von Anforderungen zwischen betroffenen Komponenten im Vordergrund steht und eine globale Betrachtung, die bei Adaption die gesamte Anwendung mit allen ihren Komponenten in die Strategie mit einbezieht. Obwohl eine lokale Betrachtungsweise nur lokale Optima finden kann, liegt der Vorteil in dem geringeren und nur lokalen Informationsbedarf für die Entscheidung. Bei Verteilungsinfrastrukturen mit einstufigen Bindungen relativiert sich das Problem der Betrachtungsebenen, da hier per Definition nur eine Bindungsebene berücksichtigt werden kann.

2.6.4 Adaptionsstrategien

Die Möglichkeiten der Umsetzung von Adaption in Anwendungen sind im Grunde nur durch das vorhandene Anwendungsmodell beschränkt. Die Optionen reichen theoretisch von der kompletten Neukonfiguration einer Anwendung bis hin zu einer Strategie, die versucht, mit möglichst wenigen Änderungen die Anwendung an eine geänderte Umgebung anzupassen. Die Freiräume in der Spanne zwischen diesen beiden Extremsituationen gilt es möglichst sinnvoll zu nutzen. Genutzt werden diese Freiräume durch so genannte Adaptionsstrategien. Eine durch die Strategie getroffene Adaptionsentscheidung sollte immer das Resultat einer Kosten-Nutzen-Berechnung sein. So gut wie alle möglichen Auswahlkriterien lassen sich schlussendlich durch Kosten und Nutzen ausdrücken. Eine große Herausforderung ist es allerdings, diese Kosten und Nutzen objektiv zu bestimmen und in Verhältnis zu setzen. Dabei sollte berücksichtigt werden, dass selbst das Erfassen, Berechnen und Auswerten von Kosten schon einen nicht unwesentlichen Kostenfaktor darstellen kann. Prinzipiell können beliebige Eigenschaften einer Anwendung in die Kostenberechnung mit einbezogen werden. Aus diesem Grund wird das Thema der möglichen verschiedenen Eigenschaften an dieser Stelle nicht vertieft. Manche Systeme verwenden ein vereinfachtes, manuelles Kostensystem, in dem der Programmierer und ggf. der Anwender die Kosten festlegt und den Nutzen selbst bestimmt.

2.6.5 Adaptionsebene

Die unterschiedlichen Ansätze der untersuchten Beispiele in Bezug auf die Umsetzung von Ubiquitous Computing spiegeln sich auch in sehr unterschiedlichen Betrachtungsweisen des Themas Adaption wider. Die Adaption in den umgebungsgebundenen Systemen konzentriert sich zum einen auf die Bereitstellung von Daten durch das verteilte Dateisystem und zum anderen auf die Anpassung der Bindungen einer Anwendung an Dienste in der Umgebung. Diese eine Ebene der Adaption - Anwendung-Komponente-Bindung - erlaubt Anwendungsprogrammierern eine Kontrolle der Adaptionsvorgänge. In den umgebungsunabhängigen Systemen, One.World und PCom, ist dies so einfach nicht möglich. Die Baumstruktur der Anwendungen mit mehrstufigen Bindungen bzw. der Abhängigkeitsgraph der Komponenten oder Environments zeigt, dass Adaption nicht nur an der Schnittstelle Anwendung-Komponente auftreten kann, sondern auch an Komponente-Komponente-Bindungen.

2.6.6 Dienste zur Adaption

Unabhängig von der jeweils verfügbaren Adaptionsebene bieten alle ubiquitären Verteilungsinfrastrukturen verschiedene Dienste an, die den Anwendungsprogrammierer bei der Adaption unterstützen bzw. automatische Adaption durch die Verteilungsinfrastruktur ermöglichen.

- *Daten und Zustand*: Zur Adaption einer Anwendung bzw. vor allem im Spezialfall der Migration einer Anwendung ist es notwendig, den Anwendungszustand zu sichern und wiederherzustellen. One.World und PCom bieten hier einen Checkpointing-Dienst an, der den anwendungsspezifischen Zustand sichern und wiederherstellen kann. Migration von Daten wird in Aura und Gaia durch ein verteiltes Dateisystem realisiert.
- *Dynamik*: Die Verteilungsinfrastrukturen müssen der Dynamik ubiquitärer Systeme gerecht werden - einerseits in Bezug auf die zur Verfügung stehenden Komponenten und andererseits in Bezug auf die Umgebung. Neben dem auch für herkömmliche Verteilungsinfrastrukturen unverzichtbaren Lookup-Dienst bieten alle untersuchten Verteilungsinfrastrukturen Unterstützung für dynamisches Binden und Leasing von Komponenten an. Das dynamische Laden von neuen Komponenten ist auch ein wesentlicher Baustein für die Adaptionsfähigkeit und wird in verschiedenen Formen von allen Verteilungsinfrastrukturen angeboten. Aura und Gaia bieten darüber hinaus noch Dienste an, die von Sensoren erfasste Umgebungsänderungen an die Anwendung weiterleiten.

2.6.7 Programmiermodell

Die bereitgestellten Dienste der Verteilungsinfrastrukturen ermöglichen Adaption von Anwendungen. Allerdings unterscheidet sich die Umsetzung von Adaption im jeweiligen Programmiermodell der Verteilungsinfrastrukturen erheblich.

- *Keine Unterstützung*: Keine Unterstützung bei der Adaption findet sich bei den untersuchten Verteilungsinfrastrukturen lediglich bei herkömmlichen Verteilungsinfrastrukturen wie CORBA Components und J2EE. Diese sind allerdings auch keine auf Ubiquitous Computing ausgerichtete Infrastrukturen.
- *Hervorhebung von Änderungen*: Gemäß den Programmierprinzipien von Grimm et al. ist es wesentlich, dass eine Verteilungsinfrastruktur Änderungen in der Umgebung kommuniziert. One.World, an dessen Entwicklung Grimm ebenfalls beteiligt war, setzt dieses Konzept um. Dieser Ansatz ermöglicht bzw. zwingt den Anwendungsprogrammierer, für jede Anwendung die möglichen Adaptionsereignisse abzufangen und zu behandeln. Die Unterstützung durch die Verteilungsinfrastruktur beschränkt sich hierbei auf die Bereitstellung der systemspezifischen Informationsquellen und erfordert eine anwendungsspezifische Implementierung der Adaptionsstrategien.

- *Automatische Adaption*: Ein anderer Grundsatz bei der Entwicklung von Verteilungsinfrastrukturen besagt, dass dem Komponenten- oder Anwendungsentwickler eine geeignete Abstraktionsebene geboten werden soll, d.h. gewisse Ereignisse sollen für den Programmierer transparent sein und in der Verteilungsinfrastruktur behandelt werden. Diesem Ansatz folgen die drei übrigen Verteilungsinfrastrukturen durch Bereitstellung von Automatismen zur Adaption.

2.6.8 Form der automatischen Adaption

Mechanismen zur automatischen Adaption werden in unterschiedlichen Formen durch die Verteilungsinfrastrukturen bereitgestellt. Bis auf One.World, die über keine automatische Adaption verfügt, bieten die anderen drei untersuchten ubiquitären Verteilungsinfrastrukturen die automatische Wiederherstellung von Bindungen zwischen Komponenten als einen Dienst an. In PCom und Aura wird beim Zusammenbruch einer Bindung zwischen zwei Komponenten bzw. zwischen einer Aufgabe und einem Dienst - sei es dadurch, dass beispielsweise der Vertrag nicht mehr erfüllt werden kann (PCom) oder die Netzwerkverbindung zusammenbricht - automatisch von der Middleware auf Grund der Dienstbeschreibung bzw. des zu Grunde liegenden Vertrags versucht wird, eine andere, passende Komponente zu finden. Im Falle von PCom ist dies eine neue Komponente, in Aura in Zusammenhang mit der Aufgabenabstraktion in der momentanen Implementierung eine neue Anwendung. Gaia benutzt die Anwendungsbeschreibung ACD, um Bindungen wiederherzustellen.

Gaia und Aura stellen durch ihr verteiltes Dateisystem weitere automatische Adaptionsformen zur Verfügung. So werden die Daten einer Anwendung durch in das Dateisystem integrierte Mechanismen transparent für die Anwendung an anderen Orten beziehungsweise auf anderen Rechnern bereitgestellt.

Das Konzept des Task-driven Computing in Aura erlaubt eine Auswahl von Anwendungen auf Basis der von einem Nutzer spezifizierten Anforderungen. Für den Anwendungsprogrammierer ist dieser Mechanismus allerdings keine Unterstützung. Der Nutzer selber spezifiziert die Kriterien, die für seine Aufgabe wichtig sind und das System erstellt auf Grund dieser Beschreibung eine dieser Anforderung entsprechende Konfiguration automatisch. Dies ist in der Form in keiner der anderen untersuchten Infrastrukturen möglich - nicht einmal Aura selber setzt dieses Konzept konsequent um, in dem es das Vorhandensein eines gemeinsamen Vokabulars für Aufgabenbeschreibungen annimmt, ohne dies aber konkret zu definieren. Weiterhin ist auf Grund der aufgabenorientierten Architektur die Kommunikation zwischen Komponenten bzw. Anwendungen innerhalb einer Aufgabe das Problem der Anwendungs- bzw. Komponentenprogrammierers und wird nicht durch die Architektur von Aura darin unterstützt. PCom bietet zwar die Möglichkeit, generische nicht-funktionale Parameter in den Verträgen anzugeben, doch sind zum einen Verträge nur zwischen zwei Komponenten gültig und damit nicht komponentenübergreifend und zum anderen existiert keine semantische Klassifikation dieser Eigenschaften, mit der eine Selektionsstrategie Annahmen in Bezug auf die angegebenen Parameter machen könnte.

2.7 Freiheitsgrade bei der Adaption

Automatische Konfiguration[1] einer Anwendung durch die Verteilungsinfrastruktur wie sie Aura, Gaia und PCom bieten, stellt eine Erleichterung für den Anwendungsprogrammierer dar. Der Anwendungsprogrammierer muss nun nicht mehr für jede Anwendung eigene Adaptionsalgorithmen implementieren. Er kann darauf vertrauen, dass die Verteilungsinfrastruktur diese Aufgabe für ihn übernimmt. Die Erwartung beinhaltet aber auch den Wunsch nach effizienter Adaption, d.h. die Konfiguration sollte ein für die Anwendung möglichst den Anforderungen des Anwenders ent-

1. Im Folgenden wird oftmals von automatischer Konfiguration von Anwendungen in ubiquitären Verteilungsinfrastrukturen gesprochen. Dies beinhaltet sowohl eine mögliche initiale Konfiguration der Anwendung durch die Verteilungsinfrastruktur wie auch eine Rekonfiguration der Anwendung im Adaptionsfall.

sprechendes Resultat erzielen. Bei anwendungsspezifischer Implementierung der Adaption hat der Anwendungsprogrammierer die vollkommene Kontrolle über den Adaptionsprozess. Die Bestimmung und Bewertung der Anforderungen des Anwenders durch die Verteilungsinfrastruktur ist allerdings von sehr vielen verschiedenen und auch gegensätzlichen Faktoren abhängig. Beispielsweise könnte die Anwenderanforderung nach hoher Ausführungsgeschwindigkeit eine Auswahl schneller Komponenten, die aber gleichzeitig andere ausgeführte Anwendungen des Anwenders verlangsamen, zur Folge haben. Eine eindeutige Bestimmung einer "besten" Konfiguration sowie eine absolute Bewertung einer Konfiguration für eine Anwendung sind auf Grund der vielen, teilweise zur Konfigurationszeit nicht bestimmbaren Faktoren nicht realisierbar. Eine Konfiguration einer Anwendung kann nur in Bezug auf vorher definierte Gütekriterien gemessen werden. In One.World programmiert der Entwickler eine speziell auf die Anwendung zugeschnittene Adaptionsstrategie und legt somit die Gütekriterien implizit durch seine Implementierung aus Anwendungssicht fest. Bei Adaption durch die Verteilungsinfrastruktur muss die Infrastruktur die Strategie festlegen und dabei versuchen, verschiedene Anforderungen zu erfüllen. Einerseits sollte eine möglichst "günstige" Konfiguration aus Anwendungssicht gefunden werden, andererseits muss die Konfiguration auch den Interessen des Gesamtsystems bzw. anderer Anwendungen gerecht werden.

2.7.1 Informationsquellen für die Adaptionsentscheidung

Die Verteilungsinfrastruktur benötigt zur Durchführung von "günstiger" Adaption Informationen, um die verschiedenen Interessen bei der Adaptionsentscheidung berücksichtigen zu können. Prinzipiell kann zwischen zwei Arten von Informationsquellen für die Adaptionsentscheidung unterschieden werden (wie in Abbildung 2.14 dargestellt).

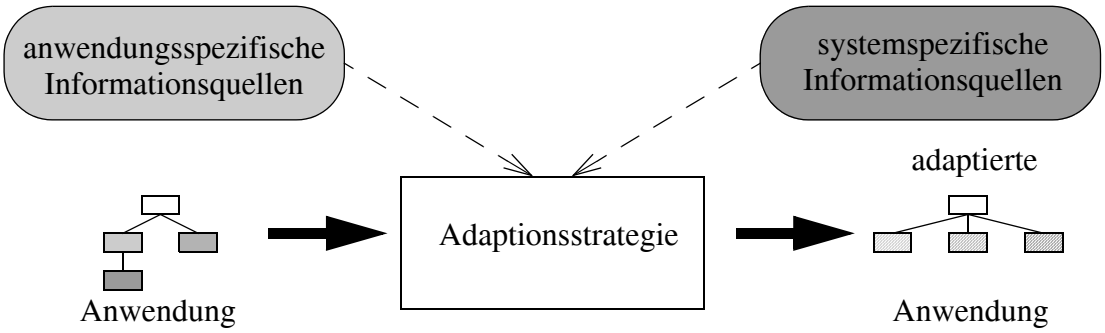


Abbildung 2.14 Informationsfluß bei der Adaption

Anwendungsspezifische Informationsquellen. Anwendungsspezifische Informationsquellen basieren auf Informationen, die anwendungsseitig erhoben werden können. Ein Beispiel für anwendungsspezifische Informationsquellen ist die Aufgabenbeschreibung in Aura. Die vom Anwendungsprogrammierer erstellte Aufgabenbeschreibung steht der Verteilungsinfrastruktur als anwendungsspezifische Informationsquelle zur Verfügung, die je nach Implementierung des Context Observers verwendet wird, um bei der Komponentenauswahl eine für die Anwendung (bzw. in Aura Aufgabe) möglichst günstige Komponente zu finden. Ein Beispiel für eine anwendungsspezifische Information in einer solchen Aufgabenbeschreibung wäre die Anforderung von Rechtschreibhilfe(-komponenten) für eine Textverarbeitungsanwendung (siehe Abbildung 2.1).

Systemspezifische Informationsquellen. Die systemspezifischen Informationsquellen ergeben sich aus allen Informationen, die nicht durch die anwendungsspezifischen Informationsquellen abgedeckt sind. Verschiedene Beispiele für systemspezifische Informationsquellen finden sich im Odyssey-Projekt. Odyssey überwacht durch Sensoren und Protokollierung das umgebende System und stellt die gewonnenen Informationen über eine Schnittstelle der Middleware bzw. den auf Odyssey basierenden Anwendungen als Entscheidungshilfe bei der Adaption bereit.

Ein typisches Beispiel für eine systemspezifische Information ist die Prozessorauslastung eines Systems.

2.7.2 Zugriff auf Informationsquellen

One.World und auch Rainbow ermöglichen dem Anwendungsprogrammierer durch geeignete Schnittstellen Zugriff auf Informationen über das umgebende System. Mit diesen systemspezifischen Informationen und aus den schon vorhandenen anwendungsspezifischen Informationen kann dann in den genannten Verteilungsinfrastrukturen die Adaptionsentscheidung anwendungsseitig getroffen werden. Prinzipiell stehen diese Informationsquellen auch der Verteilungsinfrastruktur selber zur Verfügung. Doch das anwendungsspezifische Wissen muss von der Anwendung bzw. durch den Anwendungsprogrammierer für die Verteilungsinfrastruktur bereitgestellt werden. Dafür ist eine für alle Anwendungen bindende, geeignete Form der Bereitstellung anwendungsspezifischen Wissens notwendig. Die untersuchten Implementierungen von automatischer Adaption bei Bindungen in ubiquitären Verteilungsinfrastrukturen beziehen ihr Anwendungswissen durch Informationen auf Basis formalisierter Dienstbeschreibungen. Sie beinhalten funktionale, nicht-funktionale und strukturelle Anforderungen an die jeweilige Bindung. Die Dienstbeschreibung liegt in PCom in Form von Verträgen, in Gaia in Form von Application Generic Descriptions und in Aura in Form von Aufgabenbeschreibungen vor. In PCom und Gaia werden Komponenten an Hand eines verteilungsinfrastrukturspezifischen Typsystems identifiziert. Aura benutzt die im System vorhandene, vom Anwendungsprogrammierer erstellte Aufgabenbeschreibung zum Auffinden geeigneter Komponenten.

2.7.3 Arten von Freiheitsgraden

Die Freiheitsgrade bei der Adaption können in zwei Bereiche untergliedert werden. Im Folgenden werden unter Bezugnahme auf die vorher vorgestellten Verteilungsinfrastrukturen die möglichen Freiheitsgrade bei der Wahl der Komponenten und bei der Wahl der Adaptionsebene beschrieben.

Freiheit bei der Wahl der Komponenten. Die Suche nach Komponenten, die der Dienstbeschreibung entsprechen, kann in allen untersuchten Systemen mehrere passende Komponenten zu Tage fördern. Die gefundenen Komponenten sollten natürlich alle vorgegebenen Mindestanforderungen erfüllen, können sich aber in anderen, nicht durch in die in der Dienstbeschreibung spezifizierten Eigenschaften, wesentlich unterscheiden. Soll nun die Konfiguration automatisch durch die Verteilungsinfrastruktur vorgenommen werden, so hat diese die Entscheidungsfreiheit bei der Auswahl einer der passenden Komponenten, die bei One.World bzw. bei anwendungsseitiger Adaption dem Anwendungsprogrammierer überlassen wird.

Die Auswahl der Komponenten basiert in allen untersuchten Verteilungsinfrastrukturen im Hinblick auf anwendungsspezifische Informationsquellen entweder ausschließlich auf der Auswertung von funktionalen Eigenschaften, wie beispielsweise der Implementierung bestimmter Interfaces, oder auf einer kombinierten Betrachtung von funktionalen und nicht-funktionalen Eigenschaften von Komponenten.

Freiheit bei der Wahl der Adaptionsebene. Die Freiheit bei der Rekonfiguration einer Anwendung besteht allerdings nicht nur in der eben genannten Auswahl verschiedener, alternativer Komponenten. Im Adaptionsfall in Systemen mit mehrstufigen Bindungen besteht Freiheit bei der Wahl der Bindung bzw. der Bindungen, die adaptiert werden sollen. Auch diese Freiheit muss bei automatischer Konfiguration durch die ubiquitäre Verteilungsinfrastruktur ausgefüllt werden und auch hier ist kein eindeutiges, “bestes” Vorgehen ersichtlich. In mehrstufigen Komponentensystemen kann die Anwendung an anderen Stellen im Anwendungsgraphen als an der Problemstelle im Adaptionsfall rekonfiguriert werden. Verschiedene Strategien könnten beispielsweise möglichst lokale Adaption an der auslösenden Komponente aber auch komplette Rekonfiguration bevorzugen.

Anwendungsspezifische Entscheidungskriterien zur Wahl der Adaptionsebene können ebenfalls auf funktionalen und nicht-funktionalen Eigenschaften von Komponenten basieren. Weiterhin könnten auch Informationen, die die Struktur des Anwendungsgraphen der Anwendung betreffen, zur Entscheidungsfindung verwendet werden.

2.7.4 Auswirkungen für die automatische Adaption

Die Bereitstellung automatischer Konfiguration durch die Verteilungsinfrastruktur und die damit verbundene Nutzung der Freiräume bietet viel Spielraum, eine ubiquitäre Verteilungsinfrastruktur zu erweitern.

Die Verteilungsinfrastruktur verfügt im Gegensatz zu einer einzelnen Anwendung oder Komponente über einen direkten Zugang zu systemspezifischen Informationen. In der Verteilungsinfrastruktur können auch zentral alle Eigenschaften von ausgeführten Komponenten gesammelt werden. Zusätzlich können bei der Adaption auf Verteilungsinfrastrukturebene zur Verfügung stehende, anwendungsspezifische Informationen aller Anwendungen verwendet werden. Die Verteilungsinfrastruktur kann so ihre Entscheidung unter Berücksichtigung dieser vielfältigen Informationen treffen. Der Ansatz, die Adaption auf Verteilungsinfrastrukturebene und nicht auf Anwendungsebene durchzuführen, ermöglicht beispielsweise die Adaption von Anwendungen entgegen deren Präferenzen zur Wahrung von Garantien, die gegenüber anderen Anwendungen abgegeben wurden. Wird dagegen in der Anwendung die Adaptionsstrategie definiert, so sind andere Anwendungen von einer kooperativen Programmierung dieser Anwendung abhängig (vgl. kooperatives vs. preemptives Multitasking).

Für den Anwendungsprogrammierer bietet die Verlagerung der Adaption in die Verteilungsinfrastruktur einen verringerten Implementierungsaufwand. Werden ihm zusätzlich durch geeignete Schnittstellen zur Verteilungsinfrastruktur Möglichkeiten der Einflussnahme auf den Adaptionsvorgang gegeben, so werden hierdurch die Vorteile einer anwendungsindividuellen Adaption (speziell auf die Bedürfnisse dieser Anwendung zugeschnittene Adaption) mit denen einer für die Anwendung transparenten Adaption verbunden.

Kapitel 3

Anforderungen und Lösungsansatz

Zielsetzung dieser Arbeit ist die Bereitstellung einer Möglichkeit zur Nutzung der Adaptionsfreiräume durch den Anwendungsprogrammierer in der ubiquitären Verteilungsinfrastruktur PCom. Die Anforderungen an eine Lösung dieser Aufgabenstellung werden in drei Schritten dargelegt. Im ersten Schritt werden allgemeine Anforderungen an die Lösung aufgeführt und begründet. In einem zweiten Schritt werden die Möglichkeiten erörtert, die die Verteilungsinfrastruktur PCom mit ihrem Anwendungsmodell und ihren Adaptionsmechanismen grundsätzlich bietet. Im dritten Schritt werden zwei Anwendungsszenarien entworfen und analysiert. Die Anwendungsszenarien stellen zu ihrer Umsetzung Anforderungen an die Verteilungsinfrastruktur und somit auch an die zu entwickelnde Lösung. Die aufgestellten Kriterien werden später zur Evaluation des Entwurfs und der Umsetzung herangezogen. Im Anschluss an die Aufstellung der Anforderungen wird der Lösungsansatz - eine Anwendungsbeschreibung durch den Anwendungsprogrammierer - vorgestellt. Die Anwendungsbeschreibung soll dem Anwendungsprogrammierer einerseits eine intuitive und andererseits eine funktional hinreichende Möglichkeit bieten, die Freiheitsgrade bei der Adaption zu füllen. Der Lösungsansatz in Form der Anwendungsbeschreibung wird am Ende dieses Kapitels auf Grundlage der in Kapitel 2 vorgestellten Klassifizierung von ubiquitären Verteilungsinfrastrukturen und deren Adaptionsmechanismen eingeordnet.

3.1 Anforderungen an einen Lösungsansatz

Die Basis für einen Lösungsansatz bilden die aus der Aufgabenstellung herausgearbeiteten Anforderungen. Die im Folgenden aufgeführten allgemeinen Zielsetzungen, die mit dem Lösungsansatz umgesetzt werden sollen, reflektieren die Erwartungen der Zielgruppe, d.h. der Anwendungsprogrammierer sowie die Anforderungen aus Sicht der Systemprogrammierer und die aus den verwendeten Technologien resultierenden Anforderungen. Die Gliederung erfolgt dementsprechend in funktionale Anforderungen und nicht-funktionale Anforderungen, die jeweils nach Priorität absteigend geordnet sind.

3.1.1 Funktionale Anforderungen

Die funktionalen Anforderungen beschreiben den Mehrwert für die Verteilungsinfrastruktur PCom, den diese Erweiterung darstellen soll. In diesem Fall ist der Mehrwert vor allem an den Bedürfnissen des Anwendungsprogrammierers ausgerichtet.

- *Nutzung der Adaptionsfreiräume*: Hauptziel der Arbeit ist die Schaffung eines Mechanismus, der es dem Anwendungsprogrammierer ermöglicht, die in der Verteilungsinfrastruktur vorhandenen Freiräume bei der Adaption nutzen zu können.

- *Knotenübergreifende Adaption*: Die Lösung sollte eine knotenübergreifende Adaption der Anwendung ermöglichen. Bisher werden in PCom nur knotenlokale Kriterien bei der Adaption berücksichtigt.
- *Neue Möglichkeiten der Anwendungsentwicklung*: Die angestrebte Lösung soll es dem Anwendungsprogrammierer erleichtern, Anwendungen für die Verteilungsinfrastruktur PCom zu implementieren. Ihm sollen neue Möglichkeiten zur Einflussnahme auf die Freiheitsgrade bei der Adaption gegeben werden. Auf Basis der Lösung kann der Anwendungsprogrammierer gegebenenfalls Annahmen in Bezug auf die Struktur oder Eigenschaften einzelner verwendeter Komponenten machen, die vorher nicht möglich waren.

3.1.2 Nicht-funktionale Anforderungen

Im Folgenden werden verschiedene, ausgewählte und dementsprechend wesentliche nicht-funktionalen Anforderungen an die Lösung aufgestellt. Viele hier nicht genannte nicht-funktionale Anforderungen werden bereits implizit berücksichtigt bzw. verdienen keiner besonderen Erwähnung.

- *Berücksichtigung von Dynamik*: Dynamik ist eine wesentliche Eigenschaft von ubiquitären Systemen. So muss auch die Lösung dieser Dynamik Rechnung tragen und darf nicht auf einem rein statischen Modell basieren. Die Nutzung der Freiräume muss sich dem dynamischen Verhalten der Verteilungsinfrastruktur und der Umgebung anpassen lassen. Wenn sich zur Laufzeit der Anwendungszweck oder die Umgebung ändert, so kann die Notwendigkeit entstehen, die vorhandenen Freiräume anders als vorher zu nutzen und somit den Einfluss auf die Adaption zu ändern. Hierzu muss der Lösungsansatz dem Anwendungsprogrammierer geeignete Möglichkeiten zur Verfügung stellen.
- *Effizienzgewinn*: Eine Adaptionsstrategie, die die vorhandenen Freiräume mit anwendungsspezifischem Wissen auffüllt, kann gegebenenfalls zur Effizienzsteigerung beitragen. Der Lösungsansatz sollte zum einen die Möglichkeiten der Einflussnahme durch den Anwendungsprogrammierer derart gestalten, dass die durch ihn genutzten Freiheitsgrade zu einer Selektion von Komponenten für die entsprechende Anwendung führen, die besonders geeignet sind. Dies könnte sich als effizienzsteigernd in Bezug auf das Ausführungsverhalten der Anwendung herausstellen. Zum anderen sollte bei der Adaption unter Verwendung der neuen Informationen zur Nutzung der Freiräume versucht werden, schneller eine für die Anwendung geeignete Konfiguration zu finden und auszuwählen. Dies würde die initiale Anwendungskonfiguration sowie die Rekonfiguration der Anwendung im Adaptionsfall beschleunigen.
- *Erweiterbarkeit*: Die in dieser Arbeit vorgestellte Lösung wird für den jetzigen Stand von PCom entworfen. PCom wird jedoch fortlaufend weiterentwickelt. Absehbar ist die Entwicklung diverser neuer Fähigkeiten von PCom, die auch Auswirkungen auf die Adaptionsmöglichkeiten haben werden. Dies sind unter anderem eine Ressourcenverwaltung und die Integration von Quality of Service. Den zukünftigen Ergänzungen muss die Lösung durch einfache Erweiterbarkeit und Anpassbarkeit an Änderungen in PCom Rechnung tragen. Weiterhin muss darauf geachtet werden, dass die Integration in den Adaptionsprozess flexibel gestaltet wird, um weitere Auswahlkriterien und Informationsquellen zur Nutzung der Freiheitsgrade leicht mit einbeziehen zu können.
- *Möglichst geringer Entwicklungsaufwand*: Die Bereitstellung von Möglichkeiten für den Anwendungsprogrammierer, Einfluss auf die Freiheitsgrade der Adaption zu nehmen, sollen ihm die Entwicklung von Anwendungen erleichtern. Darüber hinaus soll durch die Erhaltung der Programmierparadigmen von PCom diese Lösung zu keiner zusätzlichen Belastung bzw. keinem höheren Entwicklungsaufwand sowohl für den Anwendungsprogrammierer wie auch für den Komponentenentwickler führen.
- *Integration in die Verteilungsinfrastruktur*: Die Lösung wird speziell für die Verteilungsinfrastruktur PCom entworfen und wird somit auch zu einem Bestandteil der Verteilungsinfra-

struktur werden. Für den Entwurf und die darauf folgende konkrete Implementierung sollten die gleichen prinzipiellen Regeln und Vorgaben verwendet werden, die auch für die zugrunde liegende Verteilungsinfrastruktur PCom gelten. Im Speziellen sind dies die Erhaltung der Portabilität auf kleinste Geräte und die damit einhergehende erforderliche Ressourcenschonung. Die Vorgaben legen das Vorgehen nach dem Minimalitätsprinzip nahe, in dem versucht wird, eine möglichst kompakte und vom Umfang minimale, d.h. auf den minimal möglichen Funktionsumfang beschränkte Lösung für die Aufgabenstellung zu finden.

- *Wahrung der Kompatibilität*: Kompatibilität ist für zwei Aspekte erforderlich. Die Anforderung, Kompatibilität zu bestehenden PCom-Anwendungen aufrecht zu erhalten, fordert von der Lösung, die Erweiterungen optional zu gestalten und somit weiterhin die bestehenden Anwendungen unverändert verarbeiten zu können. Die Anforderung, die Kompatibilität von PCom gegenüber neuen bzw. gerade in der Entwicklung befindlichen Erweiterungen aufrecht zu erhalten, kann natürlich nicht in vollem Umfang erfüllt werden, doch sollten die Änderungen an der Verteilungsinfrastruktur in einer Form gestaltet werden, die möglichst wenig Auswirkungen auf weitere Entwicklungen im Rahmen von PCom haben.
- *Robustheit*: Die Robustheit einer Anwendung bzw. der Verteilungsinfrastruktur im Allgemeinen darf nicht durch die Nutzung der Freiheitsgrade bei der Adaption durch den Anwendungsprogrammierer gefährdet werden, d.h. die vorhandene Fehlertoleranz muss erhalten bleiben. Speziell die durch diese Lösung erweiterten Mechanismen zur Adaption müssen fehlertolerant implementiert werden.

Die aufgestellten Zielsetzungen lassen sich nicht unabhängig voneinander erfüllen. Beispielsweise wird die dynamische Anpassbarkeit die Komplexität der Implementierung und somit den Ressourcenbedarf erhöhen. Auch steigt der Ressourcenbedarf mit jeder neu geschaffenen Möglichkeit, die dem Anwendungsprogrammierer zur Nutzung der Adaptionsfreiräume zur Verfügung gestellt wird. Die resultierende Lösung wird demnach ein Kompromiss aus den verschiedenen Zielsetzungen sein.

3.2 Anwendungen und Adaption in PCom

Die Schaffung eines Mechanismus zur Nutzung der Freiheitsgrade bei der Adaption in der ubiquitären Verteilungsinfrastruktur PCom erfordert eine Bestandsaufnahme der bereits existierenden Konzepte für Anwendungen in PCom sowie der vorhandenen Konzepte der Adaption, die PCom für Anwendungen vorsieht. Im Folgenden wird dargestellt, wie Anwendungen in PCom strukturiert sind und in welcher Form bisher die Spezifikation von formalen und nicht-formalen Eigenschaften von Komponenten in PCom möglich ist. Die momentan in PCom verwendeten Konzepte und Abläufe für die Selektion von Komponenten und die Adaption von Anwendungen werden im Anschluss vorgestellt.

3.2.1 Komponentenstruktur

PCom ist eine komponentenbasierte Verteilungsinfrastruktur, d.h. eine Anwendung ist ein Verbund von Komponenten. Die Bindungen zwischen den Komponenten sind dabei kontraktbasiert. Für jede Komponente können in einem Vertrag ihr Angebot und die Anforderungen an die Umgebung bzw. die Abhängigkeiten der Komponente spezifiziert werden. Aus diesen Spezifikationen werden dann Bindungen geschaffen, in denen vertraglich die ausgehandelten Eigenschaften festgehalten sind. Die vertragliche Zusicherung impliziert Möglichkeiten, diese Zusicherungen zu überwachen, anzupassen und ggf. bei Nichterfüllbarkeit die Bindung zu lösen. Die Spezifikation der notwendigen Informationen erfolgt durch die Komponentenprogrammierer in verschiedenen Vertragsteilen, die im Folgenden vorgestellt werden.

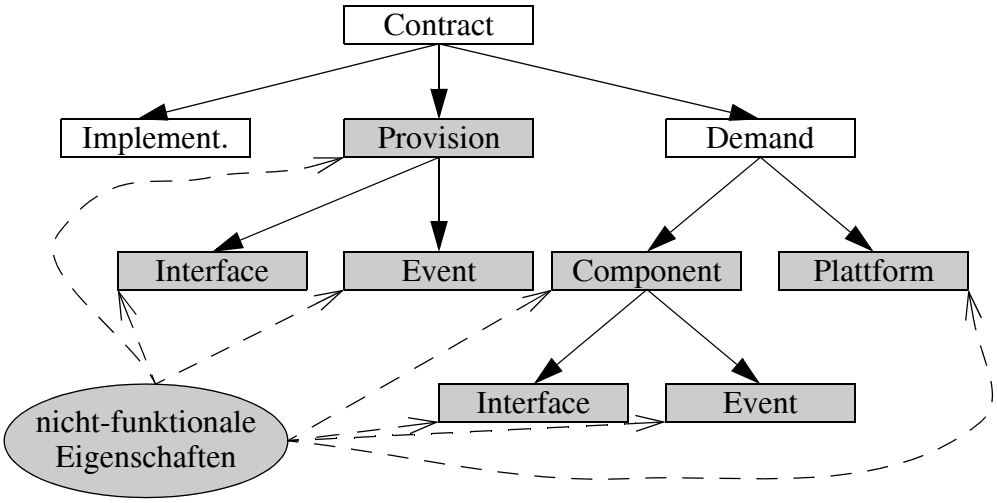


Abbildung 3.1 Kontraktstruktur in PCom

3.2.2 Verträge

In PCom werden alle Komponenten durch Komponentenverträge spezifiziert. In ihnen werden die Angaben zur Implementierung, dem Angebot der Komponente und den Abhängigkeiten bzw. den abgehenden Bindungen der Komponente gemacht. Die Verträge können als XML-Dateien erstellt werden, aus der ein Generator entsprechende Java-Klassen erzeugt. Es steht dem Komponentenentwickler aber frei, den Java-Code für die Verträge manuell zu erzeugen. Die folgende Beschreibung nimmt Bezug auf die XML-Beschreibung, da die Konzepte für die Verträge deutlicher werden und auch nach der Umwandlung in Java-Code erhalten bleiben. Ein Komponentenvertrag enthält drei Sektionen entsprechend der Gruppierung der oben angegebenen Eigenschaften (siehe Abbildung 3.1).

- *Implementation*: Die erste Sektion beschreibt die Implementierungseigenschaften der Komponente. Diese beinhalten unter anderem die Klasse, die die Funktionalität der Komponente implementiert und den Komponentennamen.
- *Provision*: Die zweite Sektion beschreibt das Angebot der Komponente, das durch andere Komponenten genutzt werden kann. Ein Angebot besteht aus den Schnittstellen, die durch nutzende Komponenten aufgerufen werden können und durch Ereignisse, mit denen die Komponente ihre Nutzer benachrichtigen kann. Bei dem Angebot als Ganzes, den Schnittstellen und den Ereignissen lassen sich jeweils auch noch nicht-funktionale Eigenschaften spezifizieren.
- *Demand*: Die dritte Sektion beschreibt die Anforderungen der Komponente an ihre Umgebung. Diese Anforderungen richten sich zum einen an die Systemplattform. Beispielsweise kann hier das zu verwendende Betriebssystem spezifiziert werden. Zum anderen werden in der Component-Sektion die direkten Abhängigkeiten der Komponente spezifiziert. Die Abhängigkeiten beziehen sich auf andere Komponenten und spezifizieren die durch die genutzten Komponenten zu implementierenden Schnittstellen und zu unterstützenden Ereignisse. Auch hier können jeweils bei den Komponenten, bei den Schnittstellen und bei den Ereignissen nicht-funktionale Anforderungen festgeschrieben werden.

3.2.3 Nicht-funktionale Anforderungen

Die aufgeführten Vertragskomponenten lassen erkennen, dass der Schwerpunkt in PCom auf der Spezifikation funktionaler Eigenschaften, d.h. der syntaktischen Schnittstellen von Komponenten, liegt und nicht auf der Beschreibung nicht-funktionaler Eigenschaften. Zwar bietet PCom an verschiedenen Stellen, die in den vorangegangenen Absätzen genannt wurden, die Möglichkeit, nicht-funktionale Eigenschaften und Anforderungen zu spezifizieren, doch existieren weder spe-

zielle Verträge noch eine Ontologie für die Arten von nicht-funktionalen Beschreibungskriterien. Somit bleibt es den Komponentenentwicklern überlassen, welche Parameter sie spezifizieren. Um diese dann nutzen zu können, muss Einigkeit oder zumindest Kenntnis über Bezeichnung, Wertebereich und Bedeutung der nicht-funktionalen Eigenschaften einer Komponente, einer Schnittstelle oder eines Ereignisses vorhanden sein. Des Weiteren beschränkt sich der Spezifikationsumfang der nicht-funktionalen Beschreibungsmöglichkeiten auf die Angabe eines Vergleichsoperators. Darüber hinaus gehende Funktionen, wie beispielsweise die Definition von Intervallen oder Optimierungsrichtungen, sind zum jetzigen Zeitpunkt nicht vorgesehen.

Die in PCom vorhandene Beschreibungsmöglichkeit wird in der Realisierung durch den so genannten *GenericDemandableContract* ermöglicht. Dieser erlaubt die Spezifikation einer nicht-funktionalen Eigenschaft durch Festlegung des Namens, des Vergleichsoperators und des Werts. Die Implementierung ist erweiterbar ausgelegt. Es können neue Arten von nicht-funktionalen Vertragsklassen angelegt werden, die das bisherige Konzept der nicht-funktionalen Anforderungen um zusätzliche Funktionalität erweitern, ohne dabei Anpassungen und Änderung bei der Selektion und Adaption vorzunehmen.

3.2.4 Konfiguration

PCom kann Anwendungen bei der Instanziierung sowie zur Laufzeit durch Änderung der Bindungen rekonfigurieren. Bindungen werden durch Verträge repräsentiert, in denen die Anforderungen zwischen nutzenden und genutzten Komponenten festgeschrieben sind. Es existieren vier Situationen, in denen eine Anpassung dieses Komponentenverbunds notwendig werden kann:

- *Instanziierung*: Bei der Instanziierung einer Anwendung, d.h. der initialen Zusammenstellung des Komponentenverbunds werden zum ersten Mal die Komponenten der Anwendung bestimmt. Die Konfiguration erfolgt automatisch gemäß Abbildung 3.2.
- *Ablauf von Leases*: Die Nutzung von Komponenten ist zeitlich befristet. Wird ein Lease nicht erneuert, so wird je nach Implementierung der Ereignisbehandlung eine Rekonfiguration der Anwendung durchgeführt. Wird der durch die Nichterneuerung des Leases erzeugte Fehler nicht abgefangen, so ist ein Anwendungsfehler die Folge. Eine Standardbehandlung ist in PCom bisher nicht vorgesehen.
- *Wegfall einer (verwendeten) Komponente*: Der Wegfall einer Komponente kann wiederum durch verschiedene Ereignisse ausgelöst werden. Der Absturz einer Komponente, der Absturz des Containers oder auch Netzwerkpartitionierung können dazu führen, dass eine in einer Anwendung verwendete Komponente der Anwendung nicht länger zur Verfügung steht. Auch in diesem Fall wird ein Fehler erzeugt, der von einer an der Anwendung beteiligten Komponente abgefangen werden muss, um einen Anwendungsfehler zu vermeiden.
- *Vertragsbruch*: Vertragsbruch tritt ein, wenn die vertraglich zugesicherten Eigenschaften nicht mehr eingehalten werden können. Dies schließt Änderungen der Anforderung und des Angebots ein, d.h. ändern die jeweiligen Komponenten ihre Angebots- bzw. Anforderungsverträge, so kann dies zum Bruch einer Bindung führen. Je nach verwendeter Implementierung für die Fehlerbehandlung kann eine Rekonfiguration der Anwendung durch Nichteinhaltung der vertraglich zugesicherten Eigenschaften ausgelöst werden. Die Implementierung wird für jede Bindung im entsprechenden Vertrag durch den Komponentenprogrammierer festgelegt.

3.2.5 Strategien zur Konfiguration

In PCom existieren zwei Mechanismen, die das Systemverhalten bei der Konfiguration von Anwendungen beeinflussen. Zum einen wird der *Selektor* bereitgestellt, der für die Auswahl der an einer Anwendung beteiligten Komponenten zuständig ist. Der Selektor wird beispielsweise vom Container dazu benutzt, für eine zu instanziierende Komponente für jede Abhängigkeit die möglichen Komponenten zu bestimmen. Dieser Vorgang wird in Abbildung 3.2 dargestellt. Zum an-

deren entscheidet der *Handler* einer Bindung in den oben genannten Fehlerfällen, ob und an welcher Stelle im Anwendungsbaum die Anwendung rekonfiguriert werden soll. Tritt bei einem entfernten Aufruf ein Fehler auf, so wird dieser im *Proxy* der Komponente entstehende Fehler durch den für diesen Proxy spezifizierten *Handler* verarbeitet. Der Ablauf der Fehlerbehandlung wird in Abbildung 3.3 dargestellt.

Prinzipiell können Komponenten in PCom jederzeit ihre Bindungen lösen bzw. eine Rekonfiguration einleiten. Sie besitzen diese Fähigkeit allerdings nur für ihre direkten Bindungen. In PCom ist bis jetzt keine standardisierte Möglichkeit vorgesehen, dass eine Komponente oder auch der Anwendungsanker als Spezialfall einer Komponente, eine Rekonfiguration nicht an den eigenen Bindungen sondern an anderer Stelle im Anwendungsbaum auslöst. Eine spezielle Handler-Implementierung könnte dies leisten. Die Handler werden aber im Komponentenvertrag spezifiziert und sind somit komponentenindividuell. Der Anwendungsprogrammierer kann sich somit nicht auf spezielle Implementierungen des Handlers verlassen, da die Handler für alle Bindungen, die nicht vom Anwendungsanker abgehen, von Komponentenprogrammierern spezifiziert werden.

Selektionsprozess. Eine Selektion von Komponenten kann im Moment einerseits durch den Versuch, eine Komponente zu instanziieren und andererseits durch den Versuch, eine vorhandene Bindung zu erneuern, ausgelöst werden. Der Verlauf der Selektion bei der Anwendungs- bzw. Komponenteninstanziierung in PCom stellt sich folgendermaßen dar. Bei dem Versuch, eine Komponente zu instanziieren (im Falle einer kompletten Anwendung als erstes der Anwendungsanker, die erste Komponente im Anwendungsbaum), versucht der Container die Abhängigkeiten der Komponente aufzulösen. Für jede in den Anforderungen spezifizierte Bindung startet der Container den Selektor, der aus den verfügbaren Komponenten für die Bindung passende auswählt und in Form einer geordneten Liste dem Container übergibt. Der Container versucht nun wiederum, der Listenreihenfolge entsprechend die Komponenten zu instanziieren, bis zur ersten erfolgreichen Instanziierung. Hierbei werden, wie gerade beschrieben, die Abhängigkeiten der beteiligten Komponenten rekursiv aufgelöst. Die konkrete Instanziierung erfolgt demnach von den Blättern des Anwendungsbaums aufsteigend und findet ihren Abschluss durch die Instanziierung des Anwendungsankers.

Der Nachteil dieses Vorgehens liegt darin, dass die Komponente für eine Abhängigkeit ohne Kenntnis der durch diese implizierte, weitere Anwendungsstruktur ausgewählt wird. Würde man beispielsweise den Energieverbrauch einer Anwendung in PCom optimieren wollen, besteht im Selektionsprozess selber bis jetzt keine Möglichkeit, die Folgekosten der Auswahl einer konkreten Komponente (abgesehen von den direkt durch diese Komponente verursachten Kosten) zu berechnen. Eine solche Funktionalität wird in PCom durch Vertragsverhandlungen angeboten, bei denen alle notwendigen Angebote für eine Anwendung eingeholt werden, die Instanziierung jedoch nicht erfolgt. Das Vorgehen des Selektors ist unter anderem dadurch begründet, dass PCom momentan über kein Reservierungssystem für Ressourcen oder Komponenten verfügt. So könnte das Resultat einer Bestimmung von möglichen Anwendungskonfigurationen zur Auswertungszeit schon wieder veraltet sein. Die einzige Möglichkeit, potentielle Anwendungskonfigurationen während dem Verlauf einer Konfiguration garantieren zu können ist die Instanziierung selbiger. Doch würde ein solches Vorgehen in vielen überflüssigen Komponenteninstanziierungen und deren anschließender Entfernung resultieren, sofern die instanziierte Alternative nicht gewählt wurde. Im Hinblick auf die beschränkten Ressourcen der für PCom vorgesehenen Geräte scheidet ein solches Verfahren aus.

Die Selektion von Komponenten in PCom erfolgt automatisch durch einen am Container angesiedelten Dienst, den bereits genannten Selektor. Verschiedene Selektionsstrategien werden durch ein Strategy-Pattern ermöglicht. Die verwendete Strategie wird in der Containerkonfiguration festgelegt. Wie bereits beschrieben erstellt der Selektor eine geordnete Liste von Komponenten, die die gegebenen Anforderungen erfüllen können. In der Selektorimplementierung, die PCom standardmäßig verwendet, basiert die Auswahl auf Schnittstellen- und Ereigniskompatibilität, d.h. alle Komponenten, die über die geforderten Schnittstellen verfügen und die geforderten Ereignisse senden können, werden in die Ergebnisliste aufgenommen. Dabei berücksichtigt der Selektor

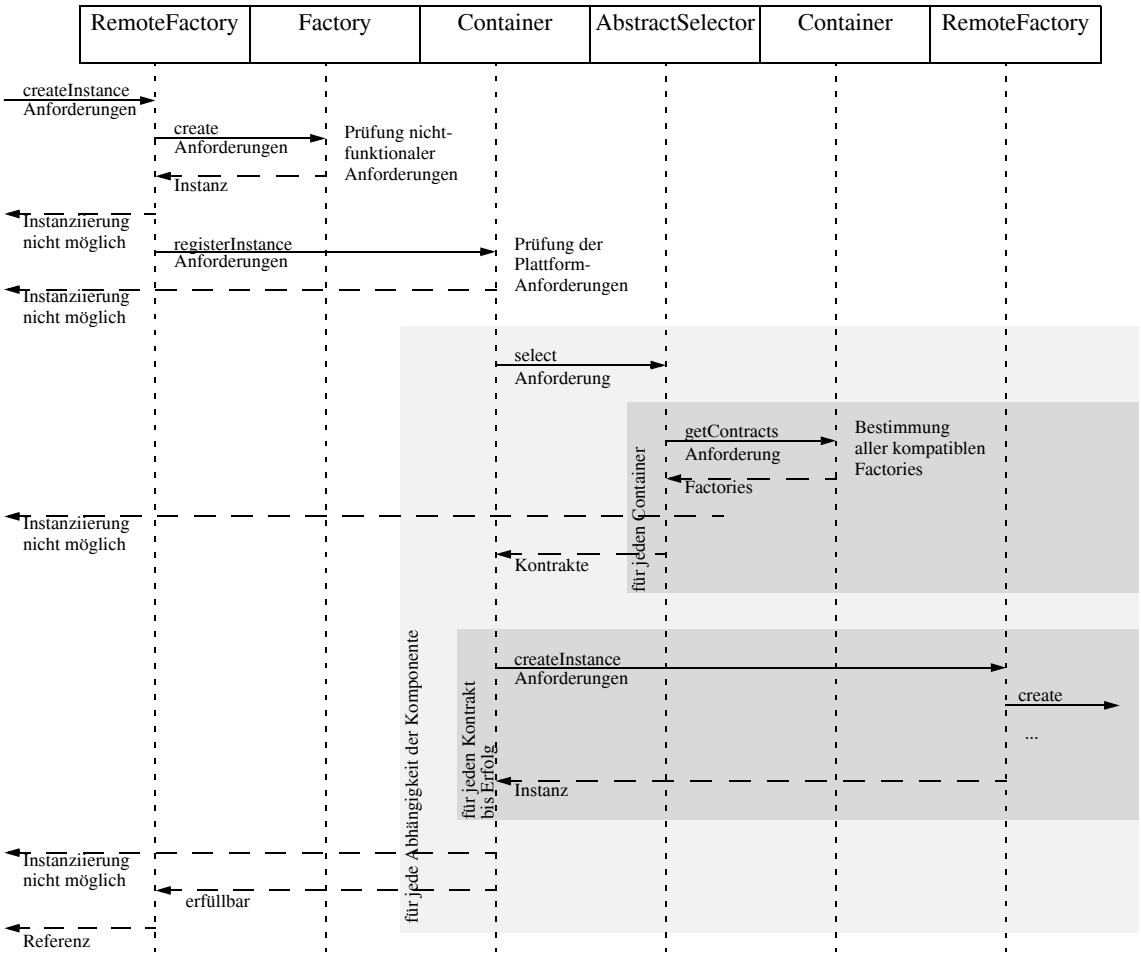


Abbildung 3.2 Bisheriger Ablauf der Anwendungsinstanziierung in PCom

einerseits die lokal verfügbaren Komponenten, andererseits aber auch alle auf entfernten Containern (die mit dem lokalen Container in Verbindung stehen) vorhandenen Komponenten.

Beim Erzeugen der Komponenteninstanz werden dann die nicht-funktionalen Eigenschaften der Komponente bzw. der vorhandenen Schnittstellen und Ereignisse überprüft. Die zweigeteilte Prüfung wurde unter der Annahme implementiert, dass die nicht-funktionalen Anforderungen sich erst bei der instanziierten Komponente überprüfen lassen.

Der Selektionsablauf bei der Erneuerung einer bestehenden Bindung ist analog zum Ablauf bei der Komponenteninstanziierung. Der Startpunkt der Rekonfiguration ist die Bindung, die erneuert werden soll. Für diese Bindung wird der Selektor aufgerufen, um mögliche Komponenten zu bestimmen. Für diese Komponenten erfolgt dann rekursiv der Abstieg, um die jeweiligen Abhängigkeiten aufzulösen und schlussendlich die notwendigen Instanzen zu erzeugen.

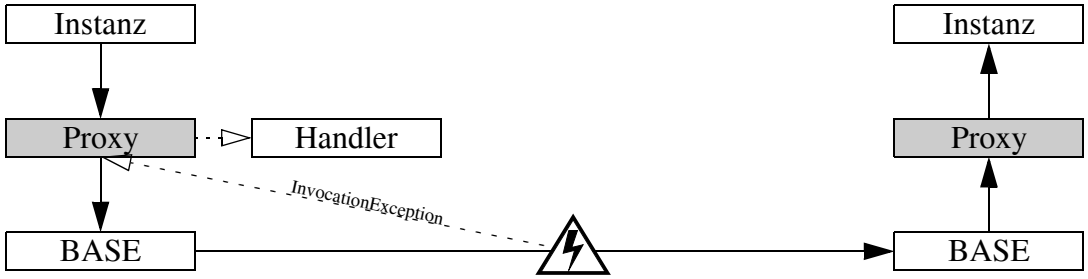


Abbildung 3.3 Ausnahmebehandlung bei entfernten Aufrufen

Ausnahmebehandlung. Ausnahmen bei entfernten Aufrufen in PCom werden durch eine für jede Bindung zu spezifizierende Behandlungsroutine verarbeitet (siehe Abbildung 3.3). Die

Behandlungsroutine wird mit dem aufgetretenen Fehler vom betroffenen Dienstproxy aufgerufen. Nach Behandlung des Fehlers liefert die Routine ein Flag zurück, welches dem aufrufenden Proxy signalisiert, ob der Proxy den Fehler nach oben zur aufrufenden Komponente durchpropagieren oder den fehlgeschlagenen Aufruf wiederholen soll. In PCom existieren keine Standard-Implementierungen für diese Behandlungsroutine, d.h. jede Komponente definiert ihre eigenen Strategien, Fehler zu behandeln. Eine mögliche Strategie, die in der Arbeit von Marcus Handte[26] vorgeschlagen wird, ist beispielsweise das neue Binden - der Proxy bleibt in diesem Fall erhalten und es wird eine neue, genau passende Komponente gesucht.

3.3 Anwendungsszenarien

PCom definiert bezüglich der mit dieser Verteilungsinfrastruktur realisierbaren Anwendungsszenarien keine Einschränkungen. Die folgenden zwei Anwendungsszenarien können somit offensichtlich nicht den gesamten Raum der möglichen Anwendungen abdecken. Allerdings handelt es sich hierbei um in ähnlicher Form oft verwendete und zitierte Beispiele, so dass ihre Relevanz als hoch einzuordnen ist. Eine vollständige Abdeckung ist einerseits aus pragmatischen Gründen nicht erreichbar und andererseits sind viele der Technologien, die in ubiquitären Systemen zum Einsatz kommen sollen (einschließlich der Verteilungsinfrastrukturen) erst in der Entwicklungsphase und somit liegt im Allgemeinen wenig bis keine Anwendungserfahrung mit ubiquitären Anwendungen vor.

Die Anwendungsszenarien werden zunächst auf abstrakter Ebene, ohne direkten Bezug auf die Verteilungsinfrastruktur zu nehmen, vorgestellt. Sie sollen im ersten Schritt Ansätze aufzeigen, bei denen die Freiheitsgrade bei der Adaption der Anwendung sinnvoll durch den Anwendungsprogrammierer genutzt werden könnten. Die in diesem Zusammenhang erfolgende Bezugnahme auf PCom ist stark vereinfachend und klammert viele Umsetzungsprobleme aus, in dem sie sich auf die Konfigurationsproblematik beschränkt. In der Evaluation werden diese Szenarien wieder aufgegriffen, um Festzustellen, in wie weit die neu geschaffenen Möglichkeiten, die sich für die Konfiguration von Anwendungen durch Nutzung der Freiheitsgrade durch den Anwendungsprogrammierer in PCom ergeben, in den hier vorgestellten Szenarien Erleichterung für ihn schaffen.

3.3.1 Präsentationsszenario

Das Szenario beschreibt einen Nutzer, der eine Präsentation zu Hause erstellen und anschließend in einem Besprechungsraum vorführen will. Heute könnte sich der Ablauf für den Nutzer folgendermaßen gestalten. Der Nutzer entwirft die Präsentation an seinem heimischen PC. Nach Erstellung der Präsentation kopiert er diese auf einen USB-Speicherstick. Mit dem Speicherstick und seinem Notebook begibt er sich in den Besprechungsraum. Dieser verfügt über einen vorinstallierten Projektor. Der Nutzer schließt das Notebook an den Projektor an und öffnet die auf dem USB-Speicherstick gespeicherte Präsentation. Der Nutzer konfiguriert das Präsentationsprogramm so, dass seine Notizen auf dem Notebook-Display ausgegeben werden und die Folien über den Projektor an die Leinwand projiziert werden.

Betrachtet man den vorgestellten Arbeitsablauf, der dem Nutzer zugemutet wird, so wird schnell deutlich, dass dieses Szenario viel Optimierungsspielraum bietet. In einer Welt des Ubiquitous Computing könnte das Szenario sich wie folgt abspielen. Auf seinem kleinen PDA wählt der Nutzer die Präsentationsanwendung zum Erstellen einer neuen Präsentation. Da sich der Nutzer zu Hause befindet und somit in Reichweite seines stationären PCs, konfiguriert sich die Anwendung automatisch so, dass der Nutzer am PC arbeiten kann. Nach Abschluss der Bearbeitung setzt sich der Benutzer in sein Auto und fährt in die Firma zur Besprechung. Während der Fahrt will er sich noch einmal die Folien ansehen. Die Anwendung, die immer noch auf seinem PDA ausgeführt wird, erkennt das Vorhandensein des Bordmonitors und das Lautsprechersystem des Autos. Über den Bordmonitor werden die Folien angezeigt und mittels des Lautsprechersystems die Notizen vorgelesen. Im Besprechungsraum angekommen wählt der Nutzer den Vorführmodus in seiner Präsentationsanwendung. Diese wiederum erkennt automatisch die Präsenz des Projektors und

nutzt diesen zweckmäßigerweise zur Ausgabe der Folien. Für die Ausgabe der Notizen wird das ins Rednerpult eingebaute Display von der Anwendung verwendet.

Vergleicht man diese beiden Abläufe, so ist festzustellen, dass im zweiten Ablauf der Nutzer nicht an einer Stelle gezwungen ist, Anwendungen oder Geräte zu konfigurieren, sondern sich voll und ganz auf seine jeweilige Aufgabe konzentrieren kann. Die Konfiguration wird stattdessen von dem ubiquitären System durchgeführt. In diesem Szenario müsste ein Anwendungsprogrammierer eine Menge verschiedener Konfigurationen von Geräten von vornherein berücksichtigen, so dass die Anwendung die gewünschten Geräte verwendet. Soll dies durch die Verteilungsinfrastruktur erreicht werden, so muss der Anwendungsprogrammierer seine Präferenzen der Verteilungsinfrastruktur in geeigneter Form mitteilen können.

Umsetzung in PCom. Eine mögliche Umsetzung des Szenarios in PCom soll im Folgenden aus Sicht des Anwendungsprogrammierers, der die Präsentationsanwendung schreiben muss, im Hinblick auf die Anwendungskonfiguration in den verschiedenen Situationen erfolgen. Es wird davon ausgegangen, dass alle am Szenario beteiligten Geräte die notwendigen Dienste für PCom bereitstellen.

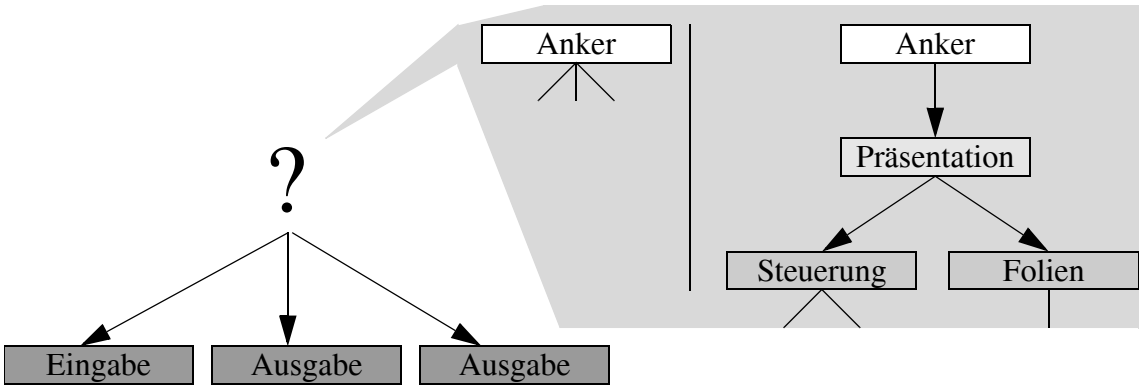


Abbildung 3.4 Beispielkonfigurationen für den Präsentationsmodus

Die verschiedenen zur Verfügung stehenden Ressourcen in dem Szenario sind im Wesentlichen Ein- und Ausgabekomponenten. Sind diese direkt an den Anwendungsanker angebunden, so kann der Anwendungsanker durch Änderung seiner Anforderungen an die gebundenen Komponenten eine Rekonfiguration erzwingen. Doch können keine zusätzlichen Bindungen geschaffen werden, d.h. ohne Neuinstanziierung einer Komponente kann die Anzahl ihrer Bindungen nicht geändert werden. Da eine Neuinstanziierung des Anwendungsankers einem Neustart der Anwendung entspricht, ist dies kein geeigneter Weg die Anwendung zu adaptieren. Verschiebt man diese Bindungen auf tiefere Ebenen oder sind generell die genannten Komponenten nicht direkt mit dem Anwendungsanker verbunden, so stellt sich die Problematik, dass der Anwendungsanker bzw. der Anwendungsprogrammierer keinen Einfluss auf die vertraglichen Anforderungen auf tieferen Ebenen hat (außer gegebenenfalls indirekt durch Auswahl von Komponenten auf höheren Ebenen).

3.3.2 E-Mail-Szenario

Das folgende Szenario beschreibt einen Nutzer, der jederzeit über seine E-Mails verfügen möchte und Zugriff auf sein E-Mail-Postfach benötigt. Die Orte, an denen sich der Beispielnutzer in dem Szenario aufhält sind sein Haus, sein Auto, die Firma, in der er arbeitet und ein Supermarkt. Nach heutigem Stand der Technik ist es bereits möglich, dass dem Nutzer an den genannten Orten seine E-Mails zur Verfügung stehen, wenn auch mit diversen Einschränkungen, die im Folgenden aufgezeigt werden. In seinem Haus wird der Nutzer die E-Mails auf Grund des höheren Bedienkomforts an seinem PC lesen wollen. Die Internetverbindung zu dem angenommenen IMAP-E-Mail-Postfach besteht durch einen DSL-Anschluss. Unterwegs, sei es im Auto oder im Supermarkt, wird die Internetverbindung durch das Handy des Nutzers bereitgestellt. Der im Mobiltelefon in-

tegrierte E-Mail-Client sorgt für die Übertragung der E-Mails vom Server. Am Arbeitsplatz in der Firma schließlich kann der Nutzer durch den Browser seines Arbeitsplatz-PCs auf das Web-Interface seines E-Mail-Providers zugreifen und so Übersicht über seine E-Mails behalten.

Die Einschränkungen dieser Form der Nutzung sind offensichtlich. Drei verschiedene Benutzerschnittstellen - E-Mail-Client am heimischen PC, E-Mail-Client im Handy und Web-Interface - verkomplizieren die Nutzung. Der Nutzer muss, sobald er seinen heimischen PC verlässt, explizit die E-Mail-Übertragung anstoßen. Obwohl unterwegs gegebenenfalls bessere, billigere und schnellere Verbindungen zum Internet möglich wären, ist der Nutzer auf die Verwendung seines Mobiltelefons beschränkt. Insgesamt stellt das Mobiltelefon in diesem Szenario die größte Hürde zu einem angenehmen Arbeiten dar, da es zum Lesen und Bearbeiten von E-Mails auf Grund seiner Größe denkbar ungeeignet ist. Weitere Herausforderungen für den Nutzer bestehen in der Synchronisation der Adressbücher, der gesendeten und der archivierten E-Mails zwischen den verschiedenen Geräten.

Das Szenario aus Sicht von Ubiquitous Computing bietet vielfältige Verbesserungsmöglichkeiten. Analog zu dem Präsentationsszenario könnte hier dem PDA des Nutzers eine zentrale Rolle zukommen. Im Haus wird die auf dem PDA gestartete E-Mail-Anwendung den heimischen PC als Ein- und Ausgabeschnittstelle verwenden, sowie die DSL-Anbindung zur schnellen Datenübertragung nutzen. Entfernt sich der Nutzer von seinem PC, so übernimmt der PDA die kompletten Aufgaben zur E-Mail-Bearbeitung. Für die Internetanbindung wird die bestmögliche Übertragungsweise automatisch bestimmt, sei es nun ein vorhandenes WLAN, Bluetooth oder im Notfall das Handy des Nutzers. Abhängig von dem verwendeten Netzwerk wird von der Anwendung die Übertragung verschlüsselt - gerade im Bereich von öffentlichen Funknetzen ein wesentliches Merkmal. Setzt sich der Nutzer aber beispielsweise in sein Auto, so kann das Audiosystem oder der ggf. vorhandene Bordmonitor zur Ausgabe von E-Mails verwendet werden. Zudem könnte die ggf. durch die Nutzung von Telematik-Diensten bereits bestehende Internetverbindung an Stelle der teuren Verbindung durch das Mobiltelefon mit genutzt werden. Am Arbeitsplatz in der Firma des Nutzers stehen wieder Internetverbindung und eine komfortable Ein- und Ausgabeschnittstelle in Form des Arbeitsplatzrechners bereit. Die Anwendung wird sich dementsprechend anpassen.

Der Vorteil einer ubiquitären Lösung liegt auch hier auf der Hand - der Nutzer wird durch unsichtbare Helfer bei der Erfüllung seiner eigentlichen Aufgaben unterstützt und muss sich nicht um Detailfragen der Konfiguration der einzelnen Geräte kümmern. Diese Entlastung ist die Belastung des Anwendungsentwicklers, der diese Konstellationen in seiner Programmierung berücksichtigen muss. Zusätzlich zu den verschiedenen Konfigurationen ist in diesem Szenario die Angabe von Präferenzen, welche der zur Verfügung stehenden Geräte bzw. Netzwerkverbindungen unter welchen Bedingungen genutzt werden sollen, wesentlich. Ein Mechanismus, der es dem Anwendungsprogrammierer ermöglicht, diese Informationen der Verteilungsinfrastruktur bereitzustellen, könnte auch an dieser Stelle Entlastung für den Anwendungsprogrammierer bringen und automatische Konfiguration durch eine Verteilungsinfrastruktur ermöglichen.

Umsetzung in PCom. Für die Realisierung des E-Mail-Szenarios gelten die gleichen Vorbedingungen, die auch bei dem Präsentationsszenario Anwendung finden. Auch hier stellt sich für den Anwendungsprogrammierer die Problematik der mehrstufigen Bindungen bzw. der fehlenden Möglichkeit, Einfluss auf Bindungen tieferer Ebene nehmen zu können. Als besondere Schwierigkeit in diesem Szenario ist die Behandlung von alternativen Möglichkeiten - im Bereich der Internetanbindungen ist abzusehen, dass häufig mehrere funktional gleiche, aber auf Grund anderer Kriterien nicht gleichwertige Möglichkeiten zur Verfügung stehen. Bestehen aus Sicht der Anwendung Präferenzen, so muss dem Anwendungsprogrammierer eine Möglichkeit gegeben werden, diese der Verteilungsinfrastruktur mitzuteilen, so dass PCom die Komponentenauswahl dementsprechend vornehmen kann.

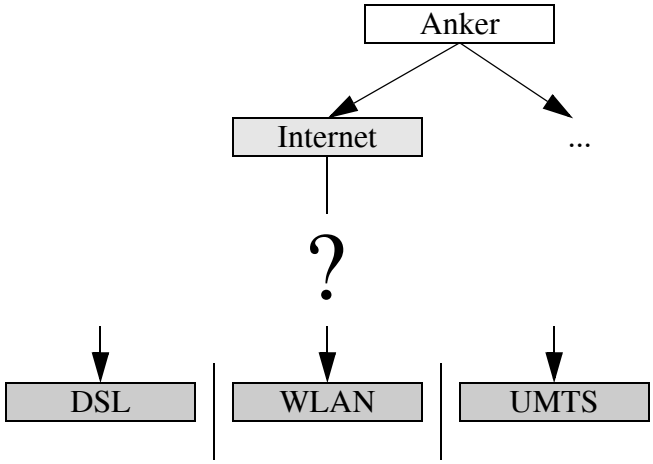


Abbildung 3.5 Beispiele für verschiedene Netzwerkalternativen

3.4 Lösungsansatz

Das Ziel dieser Arbeit besteht darin, eine Lösung zu finden, die dem Anwendungsprogrammierer die Nutzung der Freiräume bei der Adaption in einer Form erlaubt, die den obig aufgestellten Anforderungen gerecht wird.

3.4.1 Anwendungsbeschreibung

Der Ansatz, der in dieser Arbeit gewählt wird, ist der einer *Anwendungsbeschreibung*. Eine Anwendungsbeschreibung enthält funktionale, nicht-funktionale und strukturelle Informationen über die Anwendung und wird durch den Anwendungsprogrammierer erstellt. Der Anwendungsprogrammierer soll mit der Beschreibungssprache die Präferenzen für die Anwendung in Bezug auf die genannten Anforderungen ausdrücken können und so den Konfigurationsmechanismen der Verteilungsinfrastruktur anwendungsspezifische Informationen zur Verfügung stellen. Die Informationen, die durch die Bereitstellung einer Anwendungsbeschreibung dem System zur Verfügung stehen, sollen die Freiheitsgrade bei der Adaption in PCom ausfüllen und so dem Anwendungsprogrammierer neue Möglichkeiten bei der Anwendungsentwicklung bieten. Die bereitgestellten Informationen dienen als Ergänzung zu den bereits vorhandenen Informationen durch die Komponentenverträge.

Das statische Modell der Anwendungsbeschreibung wird auf Basis des in PCom vorhandenen Vertragsmodells entworfen, um keine grundlegend neuen Sprach- und Beschreibungskonzepte für Anwendungs- und Komponentenentwickler einzuführen. Die Anwendungsbeschreibung wird zur Laufzeit dynamisch anpassbar sein, um den Anforderungen der Dynamik in ubiquitären Systemen gerecht zu werden. Weiterhin wird durch das Objektmodell den Anforderungen bezüglich Erweiterbarkeit Rechnung getragen.

Die Anwendungsbeschreibung wird in PCom bei der Konfiguration durch die Verteilungsinfrastruktur als anwendungsspezifische Informationsquelle ausgewertet. Dementsprechend werden die vorhandenen Mechanismen und Algorithmen in PCom angepasst. Das dynamische Modell wird unter Wahrung der Kompatibilität zu bestehenden Anwendungen in den Bereichen der Selektions- und Adaptionsprozesse grundlegend erweitert, um eine effiziente Nutzung der neu geschaffenen Möglichkeiten zu erlauben. Auch hier wird zukünftige Erweiterbarkeit durch Verwendung geeigneter Design Patterns gewährleistet.

3.4.2 Begründung

Im Folgenden wird dargestellt, warum dieser Ansatz gewählt wurde und begründet, wie er die zwei Hauptforderungen der Aufgabenstellung - die Nutzung der Freiheitsgrade durch den Anwendungsprogrammierer und knotenübergreifende Beschreibung- umsetzt.

Bereitstellung von Informationen. Die Beschreibung der Freiräume hat gezeigt, dass einer ubiquitären Verteilungsinfrastruktur zwei Sorten von Informationsquellen bei der Adaption zur Verfügung stehen. Dies sind die systemspezifischen Informationsquellen und die anwendungsspezifischen Informationsquellen. Soll die Konfiguration der Anwendung durch die Verteilungsinfrastruktur und nicht durch die Anwendung selber stattfinden, so müssen die anwendungsspezifischen Informationen der Verteilungsinfrastruktur durch die Anwendung zur Verfügung gestellt werden. Die Bereitstellung der Informationen durch die Anwendung ermöglicht es dem Anwendungsprogrammierer durch Festlegung der bereitgestellten Informationen Einfluss auf die Freiheitsgrade bei der Adaption zu nehmen.

Knotenübergreifende Informationen. In Verteilungsinfrastrukturen mit mehrstufigen Bindungen besteht das bereits beschriebene Problem, dass die Anwendungskomponente - in PCom der Anwendungsanker - ohne weitere Mechanismen nur Einfluss auf die direkt abhängigen Komponenten hat. Die anwendungsspezifischen Informationen sollten aber auch Einfluss auf die gesamte Anwendung haben. Die Anwendungsbeschreibung stellt eine Möglichkeit dar, die gesamte Anwendung zu spezifizieren und damit knotenübergreifend Einfluss auf die Adaption zu nehmen.

Informationsquelle Anwendungsbeschreibung. Im vorherigen Kapitel wurde das Adaptionsrahmenwerk Rainbow vorgestellt. Rainbow erlaubt dem Anwendungsprogrammierer die Spezifikation der Anwendung durch eine Architektursprache. Der Ansatz, eine Anwendungsbeschreibung zur Bereitstellung anwendungsspezifischer Informationen zu nutzen, wird in dieser Arbeit aufgegriffen und für die ubiquitäre Verteilungsinfrastruktur PCom angepasst. Zielsetzung der Anwendungsbeschreibung ist es, dem Anwendungsprogrammierer die Nutzung der bei der Adaption einer Anwendung entstehenden Freiheitsgrade zu ermöglichen. Die Untersuchung der Verteilungsinfrastrukturen im letzten Kapitel hat gezeigt, dass eine solche Beschreibung zwangsläufig stark mit der Verteilungsinfrastruktur verknüpft sein wird, für die sie entwickelt wurde. Eine allgemeine, verteilungsinfrastrukturübergreifende Beschreibung für Anwendungen scheidet auf Grund der sehr unterschiedlichen Anwendungsarchitekturen (Smart Environments, Komponentenverbunde, Active Spaces, ...) aus. Zudem hat die Untersuchung ergeben, dass in den existierenden Infrastrukturen Beschreibungsmöglichkeiten entweder nicht (Gaia, One.World und PCom) oder nur ansatzweise (Anwendungsbeschreibung in Aura) vorgesehen sind. Keine der untersuchten Verteilungsinfrastrukturen verfügt über eine Beschreibungssprache, die es dem Anwendungsprogrammierer erlauben würde, seine Präferenzen in Bezug auf die Struktur und Anforderungen an die einzelnen Komponenten für die gesamte Anwendung zu spezifizieren. Einzig das Rainbow-Rahmenwerk ermöglicht dies, ohne allerdings über die Eigenschaften einer ubiquitären Verteilungsinfrastruktur zu verfügen. In Rainbow wird auf Basis von erweiterten Architekturbeschreibungen Adaption durchgeführt. Die Adaptionsentscheidung bzw. der Adaptionsalgorithmus wird in Rainbow durch die Anwendung implementiert und nicht, wie in PCom, durch die Infrastruktur. Dies ist ein wesentlicher Unterschied zu PCom, in dem die Konfiguration von dem die Komponente umgebenden Container durchgeführt wird.

3.5 Einordnung

Im vorigen Kapitel wurden verschiedene Arten von Adaption diskutiert und klassifiziert. Der hier vorgestellte Lösungsansatz, die Freiräume bei der Adaption durch die Bereitstellung einer Anwendungsbeschreibung durch den Anwendungsprogrammierer nutzbar zu machen, wird im Folgenden an Hand der in Kapitel 2 aufgestellten Kriterien eingeordnet.

- *Art der Verteilungsinfrastruktur*: Die Art der zu Grunde liegenden Verteilungsinfrastruktur hat erheblichen Einfluss auf den Entwurf der Anwendungsbeschreibung. Diese Anwendungsbeschreibung wird für PCom entworfen und implementiert. PCom fällt in die Klasse der ubiquitären Verteilungsinfrastrukturen mit mehrstufigen Bindungen ohne vordefinierte Umgebungen.
- *Transparenz*: Auf Basis der Anwendungsbeschreibung sollen der Adaptionsstrategie durch den Anwendungsprogrammierer mehr Informationen zur Verfügung gestellt werden. Im bisherigen System hat der Anwendungsprogrammierer keinen Einblick in eventuell auftretende Adaptionsvorgänge in der Anwendung sofern sie nicht die direkten Bindungen des Anwendungsankers betreffen. Durch die Einführung der Anwendungsbeschreibung wird eine Möglichkeit, aber keine Notwendigkeit, geschaffen, Einfluss auf den bisher transparenten Vorgang zu nehmen. Die Anwendungsbeschreibung verringert somit die Transparenz des Systems je nach Bedarf des Anwendungsprogrammierers.
- *Betrachtungsebene*: Die Betrachtungsebene unter Verwendung der Anwendungsbeschreibung ist komponentenübergreifend, aber nicht anwendungsübergreifend. Die Anwendungsbeschreibung ermöglicht die Betrachtung der Anwendung als Ganzes bei der Adaption. In PCom ist bisher nur die komponentenlokale Betrachtung der Bindungen bei der Adaption möglich.
- *Adaptionsstrategien*: Die Anwendungsbeschreibung stellt eine anwendungsspezifische Informationsquelle für Adaptionsmechanismen dar. Auf ihrer Basis können komplexere Strategien, die diese neuen Kriterien berücksichtigen, implementiert werden. Die Anwendungsbeschreibung als Informationsquelle stellt keine Anforderungen an und birgt keine Einschränkungen für die Implementierung von Adaptionsstrategien.
- *Adaptionsebene*: Die Anwendungsbeschreibung ändert an der für PCom wesentlichen Adaptionsebene Komponente-Komponente-Bindung nichts. PCom erlaubt mehrstufige Bindungen. Insofern kann aber an verschiedenen Stellen innerhalb des Anwendungsbaums adaptiert werden. Die Wahl der Stelle ist von der verwendeten Adaptionsstrategie abhängig. Sie entscheidet, ob nun nur lokale Komponente-Komponente-Bindungen adaptiert werden oder ob weiter reichende Änderungen am Anwendungsbaum durchgeführt werden. Im Rahmen dieser Arbeit werden zur Untersuchung der Nutzung der Anwendungsbeschreibung im Adaptionsfall zwei Adaptionsstrategien vorgestellt. Eine der Strategien wird versuchen, unter Verwendung der Präferenzen, die in der Anwendungsbeschreibung definiert wurden, immer den höchst präferierten Zustand herzustellen.
- *Dienste zur Adaption*: Das Instrument der Anwendungsbeschreibung benötigt keine zusätzlichen Dienste auf Ebene der Verteilungsinfrastruktur und stellt auch keine neuen Dienste für Anwendungen bereit. Für eine möglichst effiziente Verwendung der Anwendungsbeschreibung durch die Adaptionsstrategie ist es allerdings vorteilhaft, wenn diverse Dienste wie beispielsweise ein Checkpointing-Dienst zur Unterstützung bereit stehen.
- *Programmiermodell*: Die Hauptzielsetzung ist die Nutzung von Adaptionsfreiräumen durch den Anwendungsprogrammierer ohne dass er dabei selbst programmieren muss. Der Anwendungsprogrammierer stellt den Adaptionsmechanismen anwendungsspezifische Informationen bereit. Diese werden dann für die Durchführung automatischer Adaption genutzt. So ist dieser Ansatz eindeutig dem Modell der automatischen Adaption zuzuordnen.
- *Form der automatischen Adaption*: Die Form der automatischen Adaption wird durch die Einführung der Anwendungsbeschreibung in PCom grundlegend geändert. Bisher wurde automatische Adaption in PCom auf Basis der Verträge zwischen jeweils zwei Komponenten durchgeführt. Diese rein lokale Betrachtungsweise wird durch eine knotenübergreifende Anwendungsbeschreibung ergänzt.

Kapitel 4

Konzeption

Die Aufgabenstellung, die durch eine Anwendungsbeschreibung zu lösen ist, wurde im vorherigen Kapitel durch die Aufstellung verschiedener Anforderungen - allgemeine Anforderungen, durch die Verteilungsinfrastruktur bedingte Anforderungen und aus den vorgestellten Anwendungsszenarien resultierende Anforderungen - formuliert. In diesem Kapitel wird ein Konzept für die Anwendungsbeschreibung vorgestellt, welches der Aufgabenstellung Rechnung tragen soll. Das Konzept ist konkret auf die ubiquitäre Verteilungsinfrastruktur PCom ausgelegt. Folglich werden im Zuge der Konzeption auch die Integrationskonzepte für PCom vorgestellt. Des Weiteren werden die sich dadurch ergebenden, neuen Möglichkeiten der Adaption an Hand von je zwei Selektions- und Adaptionsstrategien erläutert.

4.1 Konzepte der Anwendungsbeschreibung

Die Grundlage für die Nutzung der Freiheitsgrade bildet das Konzept der Anwendungsbeschreibung. Dieses Konzept und sein Bezug zu PCom werden im Folgenden erläutert. Wesentliche Elemente bilden die Sicht auf die Anwendung und die Beschreibungsmöglichkeiten, die dem Anwendungsprogrammierer eröffnet werden.

4.1.1 Sicht auf die Anwendung

Die Anwendungsbeschreibung soll für den Anwendungsprogrammierer eine knotenübergreifende Beschreibungsmöglichkeit darstellen und dementsprechend eine Sicht auf die Anwendung als Ganzes ermöglichen. Anwendungen in mehrstufigen Komponentensystemen lassen sich als Graph mit Kanten als Bindungen, in PCom Verträge, und Knoten als Komponenten(-instanzen) beschreiben. Da PCom nur eine eingehende Bindung pro Instanz erlaubt, ist der Anwendungsgraph gleichzeitig auch ein Anwendungsbaum. Die Bindungen zwischen den Komponenten werden in den untersuchten mehrstufigen Systemen One.World und PCom auf Basis von Schnittstellenkompatibilität gesucht. PCom bietet darüber hinaus noch die Möglichkeit, bestimmte, wohlbekannte (nicht-funktionale) Parameter in das Angebot bzw. in die Anforderung mit aufzunehmen. Wohlbekannt definiert sich in diesem Fall dadurch, dass die Parameter sowohl der nutzenden und der genutzten Komponente bekannt sind und eine einheitliche semantische Bedeutung haben. Diese Form der Beschreibung erlaubt aber nur die Spezifikation einer Bindung zwischen einer nutzenden und einer genutzten Komponente. Die abgehenden Bindungen der genutzten Komponente können durch diese Form der Beschreibung nicht beeinflusst werden, sofern nicht die genutzte Komponente die Eigenschaften ihrer Abhängigkeiten für die nutzende Komponente zugänglich macht (beispielsweise in Form eines automatischen Mappings). Die Auf-

gabe der Anwendungsbeschreibung, diese Beschränkung aufzuheben, resultiert daher in der Notwendigkeit, durch die Anwendungsbeschreibung den Anwendungsbaum, und damit die Struktur der Anwendung, spezifizieren zu können.

4.1.2 Beschreibungssprache

Die Beschreibungssprache für die Anwendung sollte zur Entlastung des Anwendungsprogrammierers möglichst einfach gehalten werden. Es bietet sich eine Orientierung an Architekturbeschreibungssprachen an, die in Abschnitt 2.4.1 vorgestellt wurden. Die Bausteine einer Architekturbeschreibung bilden auch die Beschreibungseinheiten in der Anwendungsbeschreibung. Komponenten stellen die zentrale Struktur der Beschreibung dar. Bindungen zwischen Komponenten sind im Falle von PCom Verträge, die die Anwendungsbeschreibung auch in einer Form integrieren muss. Eine Anwendung in PCom setzt sich aus Komponenten und Bindungen zusammen und bildet die Beschreibungseinheit eines Systems. Eigenschaften und Einschränkungen werden zu den jeweiligen Beschreibungseinheiten hinzugefügt um bei der später folgenden Auswertung der Beschreibung berücksichtigt zu werden. Architekturstile sind für die Anwendungsbeschreibung nicht relevant, da PCom durch die Komponentenstruktur bereits Vorgaben in Bezug auf die Darstellung von Anwendungen macht. Ein ergänzendes Beschreibungselement wird durch Alternativen eingeführt. Sie ermöglichen die Beschreibung verschiedener, gewichteter Alternativen der Anwendungsstruktur, um die Spezifikation von je einer kompletten Anwendungsbeschreibung pro Alternative zu vermeiden.

Über die Orientierung an Architekturbeschreibungssprachen hinaus bietet sich eine Orientierung an im System bereits vorhandenen Beschreibungssprachen an. In PCom sind dies die Komponentenverträge, in denen unter anderem die Angebote und Anforderungen der Komponenten einschließlich Eigenschaften und Einschränkungen spezifiziert werden können.

Im Folgenden wird zuerst der Inhalt der Beschreibungssprache - die zu beschreibenden Eigenschaften - untersucht, bevor anschließend die eben aufgeführten Beschreibungseinheiten definiert werden.

4.1.3 Gewichtung

Alle der im Folgenden präsentierten Möglichkeiten zur Spezifikation funktionaler, nicht-funktionaler und struktureller Eigenschaften stellen zusätzliche Einschränkungen für die Adaptionsalgorithmen in PCom dar. Eine durch den Anwendungsprogrammierer erstellte Beschreibung kann die vorhandenen Informationen nicht vollständig ersetzen, da die in den Komponentenverträgen spezifizierten Anforderungen die Mindestanforderungen darstellen, auf denen die Implementierung der nutzenden Komponente beruht. Die Auswertung der Informationsquellen einschließlich der Anwendungsbeschreibung kann daher nur eine speziellere Auswahl an Komponenten zur Folge haben, niemals eine allgemeinere. Die Anwendungsbeschreibung ersetzt nicht die vorhandenen Informationen, beispielsweise die genannte Anforderungssektion der Komponentenverträge, sondern dient als ergänzendes Spezifikationsmerkmal, um speziellere Komponenten auszuwählen.

4.1.4 Spezifikation der Anwendungsstruktur

Die Anwendungsstruktur in PCom besteht aus Komponenten und Bindungen, die zusammen einen Baum pro Anwendung bilden. Für die Umsetzung einer Anwendungsbeschreibung, die eine solche Struktur beschreiben können soll, müssen jede der einzelnen Bindungen und Komponenten in geeigneter Form identifizierbar sein. Bei einstufigen Bindungsbeschreibungen ist dies nicht notwendig, da die nutzende Komponente ihre Struktur und damit alle Bindungen der Anwendung kennt. Die vorhandenen Identifikationsmöglichkeiten in PCom - Identifikation an Hand von Schnittstellen- und Ereigniskompatibilität - ist für eine Gesamtsicht der Anwendung nicht hinreichend. Die Interfaces beschreiben den Typ bzw. die Schnittstellen der Komponente, allerdings nicht die semantische Bedeutung der Komponente. Verdeutlicht wird diese Problematik im Prä-

sentationsszenario aus Kapitel 3. Als Beispiel soll hier eine Komponente dienen, die das Interface Ausgabe implementiert (siehe hierzu auch Abbildung 3.4). Das semantische Wissen über die genutzte Komponente ist implizit in der nutzenden Komponente vorhanden. In diesem Beispiel soll die Ausgabekomponente zur Ausgabe der Präsentationsfolien verwendet werden.

Wenn ein Anwendungsprogrammierer die Anwendung beschreiben soll, so wird für ihn nicht das benutzte Interface Ausgabe relevant sein, sondern vor allem die semantische Bedeutung dieser Komponente - die Ausgabe der Präsentationsfolien. Dazu ist es aber erforderlich dieses bisher nur implizit vorhandene Wissen in explizites Wissen umzuwandeln. Die Umsetzung kann auf verschiedene Arten erfolgen. Eine Möglichkeit ist die Erstellung einer Ontologie auf Basis der verwendeten Interfaces. Dies würde bedeuten, dass jedes Interface gleichzeitig die Semantik der Nutzung beschreibt. Der Vorteil dieser Lösung liegt in der Nutzung vorhandener Strukturen, der Nachteil in einer Interface-Explosion, da für jede Bedeutungsmöglichkeit eines Interfaces ein weiteres erzeugt werden muss. Der genannte Nachteil hat schwerwiegende Auswirkungen auf die Wartbarkeit - jede Änderung an der Ontologie wie auch jede Änderung an den eigentlichen Interfaces erfordert ggf. die Umstellung vieler abhängiger Interfaces. Komponenten in PCom können mehrere Interfaces implementieren. Die Semantik ist aber auf die Komponente-Komponente-Bindung bezogen. So ist eine Trennung der Schnittstellenbeschreibung von der Semantik zu bevorzugen.

Auch für die getrennte Repräsentation der semantischen Bindungsinformationen gibt es verschiedene Optionen. Es kann, wie auch bei den Schnittstellen, ein Interface verwendet werden, in diesem Fall in Form eines Marker-Interfaces wie *Serializable*, d.h. ein leeres Interface, das die Semantik der Bindung beschreibt. Eine andere Beschreibungsform kann beispielsweise eine einfache Zeichenkette sein, die die Semantik beschreibt. Allen Lösungen gemeinsam ist die Notwendigkeit der Schaffung einer Ontologie, die die semantischen Bedeutungen von Komponenten erfasst. Diese Ontologie muss für alle Komponenten- und Anwendungsentwickler bindend sein. Die Aufstellung ist allerdings nicht Teil dieser Arbeit - es wird im Folgenden von der Existenz einer solchen ausgegangen.

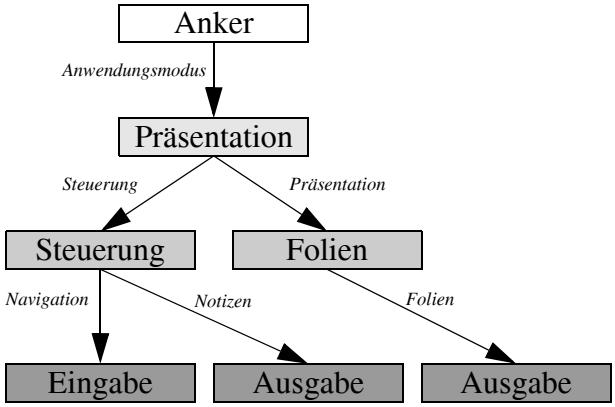


Abbildung 4.1 Semantische Labels

Die für die Anwendungsbeschreibung gewählte Repräsentation ist auf Grund der Einfachheit die eines so genannten *semantisches Labels*. Dieses semantische Label basiert auf besagter Begriffsontologie, mit der sich die semantische Bedeutung einer Komponente im Hinblick auf die Anwendung ausdrücken lässt. Pro Bindung definiert die nutzende Komponente ein semantisches Label für die entsprechende Bindung, unabhängig von den geforderten Interfaces. Mit Bezug auf das vorher genannte Beispiel aus dem Präsentationsszenario würde die Ausgabekomponente in den Anforderungen der nutzenden Komponente mit dem semantischen Label “Folien” versehen werden - natürlich unter der Annahme, dass “Folien” der passende Begriff aus der Begriffsontologie ist. Eine mögliche Konfiguration der Anwendung unter Verwendung semantischer Labels könnte demnach wie in Abbildung 4.1 aussehen.

4.1.5 Identifikation von Komponenten

Eine Identifikation von Komponenten auf alleiniger Basis ihres semantischen Labels oder ihres Interfaces ist weiterhin nicht hinreichend. Bevor der Anwendungsprogrammierer in der Lage ist, mit dem Labelsystem Komponenten im Anwendungsbaum ohne Kenntnis ihrer Schnittstellen zu spezifizieren, muss noch das Problem des Mehrfachvorhandenseins von semantischen Labels gelöst werden, das in gleicher Form für eine reine Schnittstellenidentifikation existiert. Wird die Anwendung aus dem Präsentationsszenario mit Komponenten zur Fehlerbehandlung versehen, so könnte es sein, dass mehrere Komponenten über "Fehlerbehandlung"-Bindungen verfügen, wie in Abbildung 4.2 gezeigt wird. Offensichtlich ist in dem Beispiel auch das Vorhandensein zweier Komponenten, die das "Ausgabe"-Interface implementieren. Für den Anwendungsprogrammierer verursacht dies Probleme sobald er Einschränkungen für die genannten Komponenten spezifizieren möchte. Mehrdeutigkeit in Bezug auf die betroffenen Komponenten entsteht, wenn es dem Anwendungsprogrammierer ermöglicht wird, einzelne Komponenten aus dem Anwendungsbaum ohne die übergeordnete Struktur zu spezifizieren.

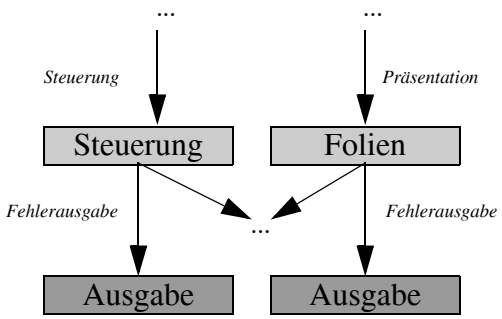


Abbildung 4.2 Mehrfache Labels

In dem in dieser Arbeit verfolgten Ansatz wird diese Möglichkeit nicht existieren. Auf den ersten Blick scheint es sinnvoll, Komponenten ohne Kenntnis der Anwendungsstruktur spezifizieren zu können. Doch führt man sich das dynamische, mehrstufige Bindungssystem von PCom vor Augen, so wird schnell deutlich, dass die Bedeutung dieser Spezifikation aus Sicht des Anwendungsprogrammierers unklar ist. Ohne Angabe der Struktur weiß der Anwendungsprogrammierer weder an welcher Stelle im Baum sie sich befindet noch ob nicht mehrere Komponenten dieser Spezifikation entsprechen. Im Beispiel für das Präsentationsszenario in Abbildung 4.2 wirkt sich dieser Umstand folgendermaßen aus. Die Präsentationsanwendung besitzt in ihrem Baum zwei Bindungen, die mit dem semantischen Label "Fehlerausgabe" versehen sind, sowie zwei Komponenten, die das Interface "Ausgabe" implementieren. Über die eine der beiden Komponenten werden Fehler der Folienausgabe und über die andere die Fehler der Steuerungskomponente ausgegeben. Spezifiziert nun der Anwendungsprogrammierer Einschränkungen für die Komponente "Ausgabe" oder für das Label "Fehlerausgabe" ohne Angabe der Struktur, ist nicht ersichtlich, ob sich die Einschränkungen auf eine oder auf beide Ausgabekomponenten beziehen soll.

4.1.6 Spezifikation von Aggregationen in der Struktur

Die Forderung an den Anwendungsprogrammierer, die Struktur einer Anwendung zu spezifizieren, um Einschränkungen für Komponenten spezifizieren zu können, bringt auch eine Reihe von Problemen mit sich. Das größte Problem ist die Voraussetzung, dass der Anwendungsprogrammierer die gesamte Struktur der Anwendung kennen muss, um sie spezifizieren zu können. Dies kann zum einen auf Grund der dynamischen, mehrstufigen Bindungen in PCom nicht verlangt werden und zum anderen widerspräche es auch dem Konzept der an dieser Stelle gewollten Transparenz der Anwendungsstruktur.

Die Position der Komponenten in der Anwendungsstruktur, für die der Anwendungsprogrammierer Einschränkungen spezifizieren will, sollte ihm aber bekannt sein - zumal er sich auf das Vor-

handensein dieser Komponenten verlässt. Andere Teile der Anwendungsstruktur dagegen können aus Sicht des Anwendungsprogrammierers irrelevant sein. Es muss also eine Möglichkeit für den Anwendungsprogrammierer geschaffen werden, Aggregationen in der Struktur ausdrücken zu können. Das Konzept zur Lösung dieser Problematik basiert auf einem Kompromiss zwischen Flexibilität und Eindeutigkeit. Der Anwendungsprogrammierer muss, wenn er eine Komponente spezifiziert, die Struktur bis zu dieser Komponente vom Anwendungsanker aus gesehen spezifizieren. Für Bereiche des Anwendungsbaums, in denen der Anwendungsprogrammierer keine Komponente spezifizieren möchte, wird die so genannte Stern-Komponente eingeführt. Eine Stern-Komponente ist ein Platzhalter für eine oder mehrere beliebige Komponenten mit beliebigen Folgekomponenten und beliebigem semantischen Label. Auf eine Stern-Komponente kann keine weitere Komponentenspezifikation folgen.

4.1.7 Spezifikation von funktionalen Eigenschaften

Der bisherige Entwurf der Anwendungsbeschreibung ermöglicht nur eine Festlegung der Struktur der Anwendung, aber keine Beeinflussung der Auswahl von Komponenten auf Basis ihrer funktionalen Eigenschaften. Die funktionalen Eigenschaften einer Komponente werden momentan in PCom durch die implementierten Schnittstellen und die unterstützten Ereignisse festgeschrieben. Beide funktionalen Eigenschaften werden auf Seite der nutzenden Komponente im *ComponentDemandContract*, sprich in der Abhängigkeitensektion des Komponentenvertrags spezifiziert. Auf Anbieterseite, d.h. auf Seite der genutzten Komponente, werden analog die angebotenen Schnittstellen und unterstützten Ereignisse in der Angebotssektion des Komponentenvertrags festgelegt. Diese bereits vorhandene Beschreibungsmöglichkeit kann zweckmäßigerweise für die Anwendungsbeschreibung aufgegriffen werden. Für jede der in der Anwendungsbeschreibung spezifizierten Komponenten können zusätzliche, durch die Komponente zu implementierende Schnittstellen und durch die Komponente zu unterstützende Ereignisse definiert werden. Dadurch wird die Spezifikation von zusätzlichen funktionalen Eigenschaften einer Komponente in der Anwendungsbeschreibung ermöglicht.

4.1.8 Spezifikation von nicht-funktionalen Eigenschaften

Nicht-funktionale Eigenschaften stellen wesentliche Kriterien für die Auswahl von Komponenten in ubiquitären Systemen dar. Seien es Dienstgütemerkmale wie beispielsweise die Leistungsfähigkeit eines Rechners für aufwendige Berechnungen, die Netzwerkbandbreite einer Internetverbindung oder einfach Eigenschaften einer Komponenten wie beispielsweise die Größe eines Monitors - Anwendungen benötigen in vielen Fällen nicht nur eine bestimmte syntaktische Schnittstelle, sondern eben bestimmte dieser Eigenschaften darüber hinaus. In Anwendungen für Aura werden beispielsweise diese nicht-funktionalen Eigenschaften in der Aufgabenbeschreibung zusammengefasst. In PCom werden nicht-funktionale Eigenschaften als Vertragszusatz definiert. Diese Zusätze können auf Komponenten-, Schnittstellen- und Ereignisebene definiert werden. Dabei lassen sich aber in den jeweiligen Komponentenverträgen nur immer die Eigenschaften zwischen nutzender und genutzter Komponente festlegen. Durch die mehrstufigen Bindungen ist es somit dem Anwendungs- bzw. dem Komponentenprogrammierer nicht möglich, auf tieferen Ebenen nicht-funktionale Eigenschaften von Komponenten zu spezifizieren.

Die Anwendungsbeschreibung soll diese Einschränkung ausräumen. Analog zu dem Komponentenvertrag können zusätzlich zu denen im Komponentenvertrag definierten, nicht-funktionalen Eigenschaften für jede Komponente, deren Schnittstellen und deren unterstützte Ereignisse weitere nicht-funktionale Eigenschaften spezifiziert werden. Das grundsätzliche Konzept der nicht-funktionalen Eigenschaften in PCom mit seinen Vor- und Nachteilen wird durch die Einführung der Anwendungsbeschreibung nicht berührt. Eine Prüfung auf Widerspruchsfreiheit zwischen den in der Anwendungsbeschreibung und den in den Komponentenverträgen spezifizierten nicht-funktionalen Anforderungen ist vorab auf Grund des dynamischen, mehrstufigen Bindungssystems in PCom nicht möglich. Deshalb erfordert die Spezifikation nicht-funktionaler Anforderungen an Komponenten bzw. an deren Schnittstellen und Ereignisse die Sorgfalt des

Anwendungsprogrammierers. Durch Spezifikation widersprüchlicher (zu denen im Komponentenvertrag festgelegten) Anforderungen kann es zu Situationen kommen, in denen eine Instanziierung der Anwendung auf Grund dieser gegensätzlichen Anforderungen unmöglich wird.

4.1.9 Spezifikation von Alternativen

Mit den oben genannten Konzepten kann der Anwendungsprogrammierer nun eine vollständige Anwendung in PCom spezifizieren. In einer hochdynamischen Umgebung, wie sie für ubiquitäre Systeme typisch ist, muss allerdings auch damit gerechnet werden, dass eine spezifizierte Idealkonfiguration nicht erfüllbar ist oder sich die Verfügbarkeit bzw. die Eigenschaften von Komponenten zur Laufzeit ändern und somit für die Anwendung günstigere oder ungünstigere Bedingungen eintreten. Dem Anwendungsprogrammierer sollte aus diesen Gründen eine Möglichkeit geboten werden, alternative Anwendungsbeschreibungen spezifizieren zu können. Da weiterhin die Verteilungsinfrastruktur die Anwendung automatisch konfigurieren soll, muss der Anwendungsprogrammierer im Falle von alternativen Beschreibungen auch die Möglichkeit haben, seine Präferenzen bezüglich der Alternativen auszudrücken.

Alternativen können in Bezug auf die Struktur, die funktionalen und die nicht-funktionalen Anforderungen existieren. Die einfachste Möglichkeit, alternative Beschreibungen auszudrücken, wäre für jede Alternative eine komplett eigene Beschreibung zu erstellen. Diese Möglichkeit ist allerdings schlicht unpraktikabel, da alle kombinatorischen Möglichkeiten der Alternativen als jeweils eigene Anwendungsbeschreibungen spezifiziert werden müssten. Es muss also eine Beschreibungsform gefunden werden, die Alternativen innerhalb des spezifizierten (Teil-)Baums zulässt. Verschiedene Lösungen sind denkbar. Zum einen könnten die Alternativen auf Ebene der nicht-funktionalen Anforderungen bei den Schnittstellen, Ereignissen und Komponenten spezifiziert werden. Dadurch hätte der Anwendungsprogrammierer aber keine Möglichkeit, Alternativen in Bezug auf die Struktur und funktionalen Anforderungen auszudrücken. Verlegt man die Alternativen auf die funktionale Ebene, so kann der Anwendungsprogrammierer Alternativen sowohl bezüglich der zu implementierenden Schnittstellen und zu unterstützenden Ereignisse wie auch zu deren nicht-funktionalen Eigenschaften spezifizieren. Alternative Strukturen sind jedoch immer noch nicht möglich. So muss zum Spezifizieren alternativer Strukturen die Definition von Alternativen auf Ebene der Abhängigkeiten von Komponenten ermöglicht werden.

Der für PCom verwendete Lösungsansatz bietet ausschließlich die Möglichkeit, alternative Strukturen zu spezifizieren. Sicherlich scheint für den Anwendungsprogrammierer eine Spezifikation von Alternativen auf allen genannten Anforderungsebenen auf den ersten Blick eine günstigere Lösung. Es sprechen aber verschiedene Gründe dagegen. Die Unterstützung für nicht-funktionale Eigenschaften in PCom ist wie in Absatz 3.2.3 beschrieben implementiert und Qualitätsstufen sind in diesem Konzept nicht vorgesehen. Die Spezifikation verschiedener Qualitätsstufen auf funktionaler Ebene wäre prinzipiell möglich. Doch zum einen würde dies die Komplexität des Algorithmus zur Auswertung der Qualitätsstufen auf Grund der zwei Ebenen stark erhöhen und zum anderen lassen sich diese Qualitätsstufen auch über zusätzliche Alternativen auf Bindungsebene ausdrücken. Unter der Annahme, dass in Bezug auf die funktionalen Eigenschaften einer Komponente verschiedene Qualitätsstufen eher unwahrscheinlich erscheinen, ist eine Beschränkung auf alternative Strukturen vertretbar.

Die Qualitätsstufen können, wie in Abbildung 4.3 zu sehen ist, auf beliebig vielen verschiedenen Ebenen in der Anwendungsbeschreibung definiert werden. Die (Gesamt-)Qualitätsstufe einer Anwendung definiert sich dabei immer vom Anwendungsanker aus gesehen, d.h. es wird versucht, auf möglichst hoher Ebene eine hohe Qualität zu erreichen. Die Gründe hierfür liegen einerseits in der Form, in der Anwendungen für PCom instanziiert werden. Dieser Algorithmus arbeitet top-down, d.h. es werden vom Anwendungsanker ausgehend rekursiv die Abhängigkeiten aufgelöst und die entsprechenden Komponenten instanziiert. Würde man nun die Qualität über die gesamte Anwendung gleichwertig bestimmen wollen, müssten alle möglichen Kombinationen von Komponenten bestimmt, instanziiert, und dann bis auf jene mit der höchsten Qualitätsstufe wieder verworfen werden. Der vorgestellte Ansatz dagegen erlaubt sofortige Entscheidungsfindung bei dem

top-down-Algorithmus. Andererseits begründet sich die Ordnung der Qualitätsstufen durch das Verhältnis nutzender zu genutzter Komponente. Die Annahme besteht darin, dass eine qualitativ niederwertige, nutzende Komponente weniger Vorteil aus einer höherwertigen, genutzten Komponente ziehen kann als umgekehrt. Verdeutlicht man sich dies am Anwendungsbaum, so könnte die frühzeitige Auswahl einer hohen Qualitätsstufe auf tiefer Baumebene zur Folge haben, dass die Auswahl der darüber liegenden Komponenten stark eingeschränkt wird und so zu einem insgesamt qualitativ schlechteren Anwendungsverhalten führt.

4.1.10 Definition der Beschreibungseinheiten

Die Mächtigkeit der Anwendungsbeschreibung ist durch die vorangegangenen Abschnitte festgelegt worden. Die Anwendungsbeschreibung besteht aus verschiedenen Entitäten, die die Struktur der Anwendung und die funktionalen und nicht-funktionalen Eigenschaften der verschiedenen Komponenten beschreibt. Im Folgenden werden diese einzelnen Beschreibungsbausteine, die in Abbildung 4.3 abgebildet sind und zusammengenommen die Anwendungsbeschreibung bilden, genauer beschrieben.

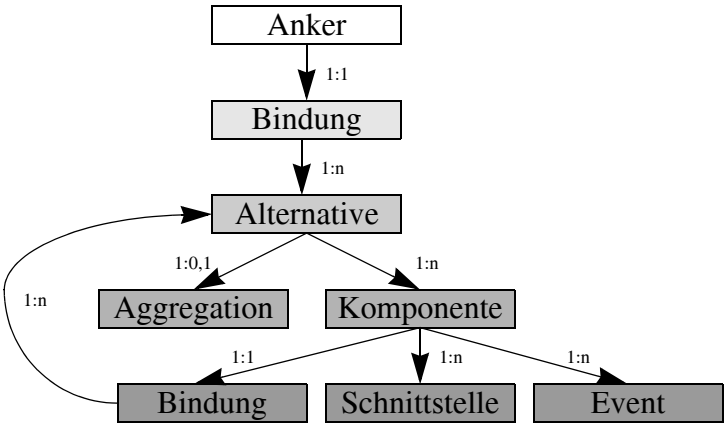


Abbildung 4.3 Aufbau der Anwendungsbeschreibung

- *Komponente*: Eine Komponente in der Anwendungsbeschreibung ist das Gegenstück zu der Komponentendefinition in PCom. Analog zu Absatz 3.2.2 können hier Gruppen von Eigenschaften spezifiziert werden. Das Angebot einer Komponente kann nicht beschrieben werden. Dieses ist und bleibt einzig von der anbietenden Komponente abhängig. Die Definition weiterer zu implementierender Schnittstellen und zu unterstützender Ereignisse stellen aber eine sinnvolle Beschreibungsmöglichkeit dar. Auch Ergänzungen bezüglich der Anforderungen an die Abhängigkeiten einer Komponente können in der Komponentendefinition spezifiziert werden. Auf Komponentenebene können durch die Anwendungsbeschreibung weitere nicht-funktionale Eigenschaften festgelegt werden.

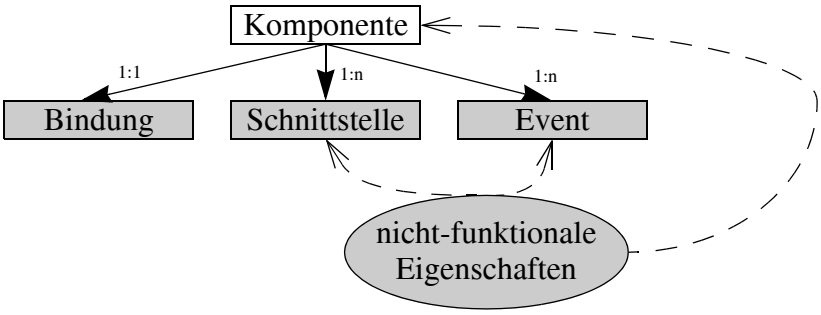


Abbildung 4.4 Aufbau der Komponentenbeschreibung

- *Spezialfall Aggregation*: Aggregationen in der Anwendungsbeschreibung werden in Form einer speziellen Komponente, der Stern-Komponente, realisiert. Sie dient als Platzhalter für

beliebige Komponenten mit beliebigen funktionalen, nicht-funktionalen und strukturellen Eigenschaften und beliebigen semantischen Labels. Aus diesem Grund können an der Stern-Komponente keine Angebote, Abhängigkeiten, Schnittstellen, Ereignisse oder nicht-funktionale Eigenschaften als weitere Einschränkung spezifiziert werden.

- *Spezialfall Anwendungsanker*: Ein weiterer Spezialfall einer Komponente ist der Anwendungsanker. Er ist zwar aus PCom-Sicht auch eine Komponente, doch bildet er den Ausgangspunkt für die Anwendungsbeschreibung und wird im Gegensatz zu allen anderen Komponenten nicht durch den Selektor ausgewählt, sondern ist durch die Anwendung fest vorgegeben bzw. repräsentiert die Anwendung. Insofern ist es nicht sinnvoll, ein Angebot, Schnittstellen oder nicht-funktionale Eigenschaften für den Anwendungsanker spezifizieren zu können.
- *Bindung*: Eine Bindung zwischen zwei Komponenten wird durch Aufführung der genutzten Komponente in der Beschreibung der Abhängigkeiten der nutzenden Komponente spezifiziert. In PCom können zurzeit keine weiteren Eigenschaften von Bindungen spezifiziert werden. Identifiziert wird eine Bindung an Hand ihres semantischen Labels.

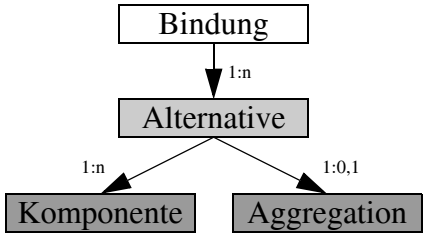


Abbildung 4.5 Aufbau der Bindungsbeschreibung

- *Anwendungsbeschreibung*: Die Anwendungsbeschreibung ist in diesem Fall das beschriebene System, d.h. es wird der Anwendungsbaum vom Anwendungsanker abgehend bis hin zu den Blättern des Baums bzw. zu den Stern-Komponenten spezifiziert. Eine Schachtelung von oder Referenzen zwischen verschiedenen Anwendungsbeschreibungen ist nicht möglich, da dies der Definition einer Anwendung widersprechen würde.
- *Funktionale Eigenschaft*: Die funktionalen Anforderungen an eine Komponente werden in der Anwendungsbeschreibung durch Aufführung der zu implementierenden Schnittstellen und zu unterstützenden Ereignisse bei der entsprechenden Komponente definiert.
- *Nicht-funktionale Eigenschaft*: Nicht-funktionale Anforderungen können bei den Schnittstellen-Definitionen, bei den Ereignis-Definitionen und bei Komponentenbeschreibungen aufgeführt werden. Die Beschreibungsmöglichkeiten beschränken sich auf die generischen, bisher in PCom vorhandenen Möglichkeiten. Die definierten Einschränkungen werden gegen das entsprechende Schnittstellen-, Ereignis- oder Komponentenangebot geprüft.
- *Alternative*: Die Spezifikation von Alternativen wird in der Anwendungsbeschreibung auf Ebene der Bindungen ermöglicht (siehe Diskussion in Absatz 4.1.9). Für jede Komponente lassen sich bewertete Alternativmöglichkeiten für ihre Abhängigkeiten spezifizieren.

4.2 Konzepte der Komponentenauswahl

In der automatischen Komponentenauswahl durch die Verteilungsinfrastruktur liegt einer der Freiräume, die durch die Anwendungsbeschreibung gefüllt werden soll. Die bisher in PCom existierende Lösung (genauer beschrieben in Absatz 3.2.5) berücksichtigt ausschließlich diejenigen Komponenten in lokalen und entfernten Containern, die den Anforderungen entsprechen. Die Auswahl erfolgt bisher durch den Selektor, der für jede Abhängigkeit einer Komponente aufgerufen wird und passende Komponenten aus dem Angebot filtert. Dem Selektor stehen jetzt aber durch die Anwendungsbeschreibung zusätzliche Informationen zur Entscheidungsfindung zur

Verfügung. Die Informationen aus der Anwendungsbeschreibung stellen zusätzliche Anforderungen an die auszuwählenden Komponenten und somit eine Einschränkung bei der Auswahl von Komponenten dar. Eine Möglichkeit, die in PCom vorhandenen Strukturen zu nutzen ist dem Selektor die Anwendungsbeschreibung bei der Auswahl der Komponenten zur Verfügung zu stellen. Die neu eingeführten Qualitätsstufen umfassen alle Bindungen einer Komponente pro Qualitätsstufe. Daraus wird ersichtlich, dass eine Selektion pro Abhängigkeit einer Komponente ungeeignet ist. Unter Berücksichtigung der Anwendungsbeschreibungsstruktur ist eine vollständige Selektion über alle Abhängigkeiten einer Komponente zu bevorzugen. Das Resultat dieses Selektionsvorgangs sind dann eine oder mehrere Qualitätsstufen, die für jede Abhängigkeit der Komponente eine oder mehrere mögliche Komponenten enthalten. Wird der Selektionsprozess angepasst, so kann auch die Instanziierung der abhängigen Komponenten nicht wie bisher pro Abhängigkeit direkt auf die Selektion erfolgen.

Das Resultat des Selektionsprozesses wird - wie oben festgestellt - eine dreidimensionale Struktur mit den Dimensionen Qualität, Abhängigkeit und mögliche Komponenten sein. Bisher werden alle möglichen Komponenten als Resultat zurückgegeben. Der Nachteil dieser Lösung ist gegebenenfalls unnötiger Aufwand, wenn sich die erste gefundene, passende Komponente bereits instanziieren lässt. Die anderen gefundenen Komponenten werden ungeprüft verworfen.

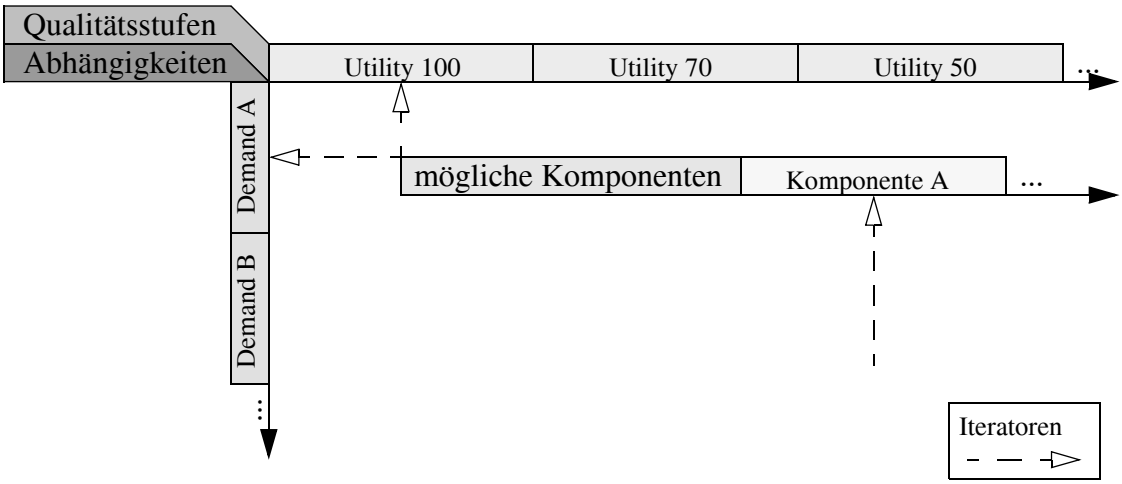


Abbildung 4.6 Iterator-Konzept für die Selektion

Die genannten notwendigen Änderungen und Defizite am bisherigen Selektionsprozess legen ein komplettes Neudesign nahe. Die größte Designänderung stellt die Änderung der Aufrufstruktur dar. Der Selektor soll nicht mehr pro Bindung aufgerufen und dann alle verfügbaren Komponenten auf einmal zurückliefern. Eine Möglichkeit, dies zu umgehen ist ein iterator-basierter Ansatz, um dem Container die Ergebnisse der Selektion zur Verfügung zu stellen. Der Selektor repräsentiert eine Datenstruktur für die drei Ergebnisdimensionen. Der Selektor erlaubt das Iterieren über die zwei Dimensionen, Qualitätsstufen und Abhängigkeiten. Der Zugriff liefert einen weiteren Iterator für die dritte Ergebnisdimension. Diese dritte Dimension besteht aus den Komponenten, die für die momentane Qualitätsstufe die ausgewählte Abhängigkeit erfüllen.

Der vorher funktional orientierte Selektor wird mit diesem Konzept in eine Datenstruktur umgewandelt, die entsprechend eine verzögerte Auswertung ermöglicht. In der Umsetzung werden die neuen Möglichkeiten durch zwei Selektorimplementierungen, dem *InstantSelector* und dem *LazyRemoteContainerSelector* verdeutlicht.

Die Erweiterung des Selektors impliziert auch eine Anpassung aller den Selektor nutzenden Programmteile. Die den Selektor nutzenden Teile der Verteilungsinfrastruktur - bisher die Containerimplementierung bei Komponentenregistrierung und Neubindung - müssen jetzt für jeden Selektionsvorgang einen Selektor bzw. eine Ergebnisdatenstruktur instanziieren, mittels der Iteratoren über den Ergebnisraum iterieren und die Ergebnisse der Selektion verwerten. Aus softwaretechnischer Sicht birgt diese Lösung den Vorteil, dass die komplette Auswahllogik jetzt im Selektor konzentriert ist und die Aufrufer die Ergebnisse auswerten.

4.3 Konzepte der Selektionsstrategien

Die zusätzlich zu verarbeitenden Informationen und die Erweiterung des Selektors bedingen eine Neukonzeption der bisher in PCom verwendeten Selektionsstrategie. Zwei unterschiedliche Selektionsstrategien werden im Folgenden vorgestellt. Die eine orientiert sich an der bisher vorhandenen Selektionsstrategie und die andere zeigt die Möglichkeiten der neuen Architektur auf.

4.3.1 Sofortige Auswahl

Die Selektionsstrategie *sofortige Auswahl* führt die Selektion in einem Durchgang durch. Dabei werden alle verfügbaren lokalen und entfernten Container auf zu den gegebenen Anforderungen passende Verträge angefragt. Diese werden im Selektionsalgorithmus unter Verwendung der Anwendungsbeschreibung den darin spezifizierten verschiedenen Qualitätsstufen zugeordnet. Nach Prüfung aller Angebote werden die unvollständigen Qualitätsstufen - jene, die für mindestens eine Abhängigkeit kein Angebot aufweisen können - aus dem Endergebnis entfernt. Das in der Art auf funktionale und strukturelle Eigenschaften geprüfte und kompaktifizierte Angebot kann anschließend über das Iteratorkonzept ausgewertet und verwendet werden.

4.3.2 Verzögerte Auswahl

Die Selektionsstrategie *verzögerte Auswahl* greift die Eigenschaften der neuen Selektorarchitektur bei der Selektion auf. An Stelle für jede Abhängigkeit alle entfernten Container sofort nach Angeboten anzufragen, wird hier nur im Bedarfsfall die Liste der entfernten Container abgearbeitet. Wird der Iterator auf das Angebot zu einer bestimmten Abhängigkeit-Qualitätsstufe-Kombination weitergesetzt, so wird versucht, zuerst lokal weitere Angebote zu finden und danach so lange entfernte Container angefragt, bis zumindest ein Angebot zurückgeliefert werden kann oder festgestellt wird, dass keine weiteren Angebote existieren.

Aus funktionaler Sicht ist das Verhalten der beiden Selektionsstrategien für die nutzenden Programmteile gleichwertig. Das Laufzeitverhalten der beiden Selektionsstrategien unterscheidet sich jedoch erheblich. Die sofortige Auswahl hat einen einmaligen Aufwand für die Selektion mit fest bestimmbarer (in Abhängigkeit der verfügbaren entfernten Container) Menge an entfernten Aufrufen. Die Auswertung des Selektionsergebnisses beschränkt sich auf das Navigieren über ein bereits vorhandenes Angebot. Die verzögerte Auswahl dagegen fragt erst bei Bedarf entfernte Container an. So können beim Durchlaufen des Ergebnisraums noch unvorhersehbare Mengen (bis maximal zur Anzahl der verfügbaren entfernten Container pro Abhängigkeit) an entfernten Aufrufen anfallen. Andererseits kann die Strategie im Idealfall (alle Anforderungen lassen sich lokal auflösen) ohne entfernte Aufrufe auskommen. Beiden Strategien gemein ist die Miteinbeziehung der Anwendungsbeschreibung zur Bestimmung von Angeboten für die dort definierten Qualitätsstufen.

4.4 Konzepte der Adaptionsstrategien

Der zweite Bereich von Freiheitsgraden bei der Adaption ist die Entscheidung, an welcher Stelle bzw. auf welcher Bindungsebene eine Anwendung adaptiert werden soll. Momentan tritt dieser Fall nur ein, wenn ein entfernter Aufruf fehlschlägt (siehe dazu Abschnitt 3.2.5). Die Implementierung der im Folgenden vorgestellten Konzepte bezieht sich auf die Möglichkeiten, die ein speziell implementierter Handler für Bindungen bieten kann. Doch können die vorgestellten Adaptionsstrategien auch in anderen Fällen Anwendung finden. Mögliche Fälle wären beispielsweise, dass eine Komponente ihre Bindungen erneuern will oder eine neue Komponente dem System hinzugefügt wird und sich dadurch ggf. eine bessere Qualitätsstufe in der Anwendungsbeschreibung erreichen ließe. In PCom sind dafür aber keine standardisierten Mechanismen vorgesehen. Ein Ansatz könnte unter Ausnutzung des eventbasierten Kommunikationssystems versuchen, die Rekonfiguration der Anwendung auf eine geeignete Stufe zu heben. Im

Rahmen dieser Arbeit werden Konzepte für solche Mechanismen aber nicht untersucht und vertieft.

Die Einführung der Anwendungsbeschreibung eröffnet auch für die Adaption neue Möglichkeiten. Zwei davon werden im Folgenden vorgestellt. Während sich die Strategie der möglichst lokale Adaption darauf beschränkt, die Anwendungsbeschreibung als zusätzliches Auswahlkriterium bei der neu zu bindenden Komponente zu verwenden, nutzt die zweite vorgestellte Strategie die Anwendungsbeschreibung, um die Adaptionsebene zu wählen. Dabei stellt dies nur eine der Möglichkeiten dar, die neue Informationsquelle innerhalb der Adaptionsstrategie auszuwerten.

4.4.1 Möglichst lokale Adaption

Die Strategie der möglichst lokalen Adaption versucht zuerst, die Anwendung an der Problemstelle[1] zu rekonfigurieren. Der Algorithmus basiert auf der Annahme, dass je weniger Komponenten neu gebunden werden müssen, desto höher der Nutzen für die Anwendung sein wird. An einer tieferen Stufe im Anwendungsbaum als an der Problemstelle kann nicht adaptiert werden, da eine weiter Annahme besagt, dass die Problemstelle auf jeden Fall neu konfiguriert werden muss (und damit der gesamte Anwendungsbaum ab der Problemstelle obsolet wird). Ansonsten müsste die Adaption auf tieferer Ebene angestoßen, und damit die Problemstelle verlagert werden.

Kann auf Grund mangelnder Verfügbarkeit nicht auf Ebene der Problemstelle rekonfiguriert werden, so wird der Fehler zuerst an die aufrufende Komponente und dadurch auf die nächst höhere Ebene im Anwendungsbaum propagiert. Der Algorithmus versucht nun, an dieser Stelle die Anwendung zu rekonfigurieren. Dies wird fortgesetzt, bis entweder die Rekonfiguration geglückt ist oder schließlich auf Ebene des Anwendungsankers festgestellt wird, dass eine Rekonfiguration unmöglich ist.

Der vorgestellte Algorithmus entspricht dem in der Arbeit von Marcus Handte[26] vorgestellten Algorithmus. Als Erweiterung wird hier beim Versuch der Rekonfiguration die Anwendungsbeschreibung mit berücksichtigt. Ansonsten werden keine Informationen aus der Anwendungsbeschreibung genutzt.

4.4.2 Möglichst optimale Adaption

Die Strategie der möglichst optimalen Adaption bezieht die Anwendungsbeschreibung stärker in den Adaptionsprozess mit ein. Die Zielsetzung dieser Strategie ist es, die Anwendung so zu konfigurieren, dass sie sich immer in dem aus Sicht der Anwendungsbeschreibung höchst präferierten Zustand befindet. Wird an einer Problemstelle die Adaption angestoßen, so wird an Hand der Anwendungsbeschreibung diejenige Komponente zwischen Anwendungsanker und Problemstelle bestimmt, die an höchster Ebene nicht der höchstmöglichen Präferenz in der Anwendungsbeschreibung entspricht. Die Rekonfiguration wird durch den Anwendungsbaum bis zu dieser Stelle propagiert und dort entsprechend durchgeführt. Die zu Grunde liegende Annahme bei dieser Strategie ist, dass der Nutzen einer größer angelegten Rekonfiguration die Kosten übersteigt, in dem mit höherer Wahrscheinlichkeit eine günstigere Konfiguration aus Anwendungssicht erreicht wird.

4.5 Konzepte der Verhandlungsstrategien

Die Selektion in PCom ist für die konkrete Komponentenauswahl verantwortlich, die im Anschluss gleich instanziiert werden. Wünschenswert wäre aber auch eine Art Prüfalgorithmus, der

1. Eine Problemstelle steht hier für diejenige Stelle im Anwendungsbaum, an der der Adaptionsalgorithmus angestoßen wurde. Dies muss nicht unbedingt auf Grund eines Problems sein, wie im vorherigen Absatz dargestellt wurde, sondern kann auch anderweitige Gründe wie beispielsweise geänderte Verfügbarkeit von Komponenten haben.

zwar einen oder mehrere komplette, aber noch nicht instanziierte Anwendungsbäume berechnet. So würde es der Anwendung ermöglicht, aus verschiedenen Konfigurationen auszuwählen. Voraussetzung dafür ist natürlich, dass das Ergebnis von einer Verhandlung wirklich instanziierbar ist.

Der bisherige Selektor in PCom bietet eine Schnittstelle zur Implementierung einer solchen Funktionalität auf Basis einzelner Anforderungen an. Die Prüfung auf Instanziierbarkeit und das Zurückliefern der verschiedenen, passenden Angebote ist implementiert. Allerdings können keine Garantien bezüglich der Instanziierungsfähigkeit der Konfiguration gemacht werden, da in PCom keine Reservierungen von Ressourcen für die Zeit zwischen Verhandlung und Instanziierung vorgenommen werden können. Diese Funktionalität wird auch im Rahmen dieser Arbeit nicht nachgerüstet.

Das bisherige Verhandlungskonzept wird jedoch, ebenso wie der Selektor, an die Gegebenheiten der Anwendungsbeschreibung angepasst. Das Konzept der Datenstruktur für den Selektor zwingt zu einer Trennung von Verhandlung und Selektion - die beiden Konzepte lassen sich nicht wie bisher in einer Einheit zusammenfassen. Die Trennung eröffnet die Möglichkeit, die Verhandlung analog zur Selektion in eine Datenstruktur, mit allen bereits genannten Vorteilen, umzuwandeln. Das Resultat ist eine Datenstruktur, wie in Abbildung 4.6 gezeigt. Im Unterschied zum dort abgebildeten Modell werden hier vorhandene Angebote als Ergebnis geboten.

Die fehlenden Rahmenbedingungen für verlässliche Vertragsverhandlungen in PCom bedingen die Beschränkung auf die Konzeption nur einer einzigen Verhandlungsstrategie im Rahmen dieser Arbeit. Ihr liegt die bisher vorhandene Implementierung und somit die Konzeption, die auch für die sofortige Komponentenauswahl verwendet wurde, zu Grunde.

Kapitel 5

Implementierung und Integration

Die wesentlichen Elemente der Implementierung der Beschreibungssprache und deren Integration in die ubiquitäre Verteilungsinfrastruktur PCom werden in diesem Kapitel vorgestellt. Die für den Anwendungsprogrammierer relevante XML-Syntax der Beschreibungssprache wird als erstes vorgestellt. Diese Beschreibungssprache muss für die Verteilungsinfrastruktur PCom modelliert werden. Das entsprechende Objektmodell der Beschreibungssprache in Java und dessen Erzeugung wird im Folgenden vorgestellt. Im dritten Teil dieses Kapitels werden die notwendigen statischen und dynamischen Erweiterungen in PCom beschrieben. Der letzte Abschnitt widmet sich der Implementierung der im vorangegangenen Kapitel vorgestellten Konzepte für Selektions- und Adaptionsstrategien.

5.1 Syntax der Beschreibungssprache

Die Beschreibungssprache für die Anwendungsbeschreibung soll für den Anwendungsprogrammierer möglichst einfach zu verwenden sein und gleichzeitig alle im vorangegangenen Kapitel spezifizierten Eigenschaften aufweisen. Die bereits in PCom vorhandenen Komponentenspezifikationen können entweder manuell in entsprechende Java-Klassen codiert werden, oder aber als XML-Datei spezifiziert und von einem Generator anschließend in Java-Code umgewandelt werden. Letztere Möglichkeit ist für den Anwendungsprogrammierer die komfortablere. Der XML-Code ist für den Menschen leichter lesbar, ohne dabei aber an Maschinenlesbarkeit einzubüßen. Zudem kann der Generator, der die XML-Datei in Java-Code umwandelt, viel notwendigen, aber nicht anwendungsbezogenen Code erzeugen, der ansonsten vom Anwendungsprogrammierer geschrieben werden müsste.

Eine Zielsetzung für die Anwendungsbeschreibung besteht in der Erleichterung der Anwendungsentwicklung. Unter dieser Zielsetzung in Verbindung mit dem Ziel einer möglichst hohen Affinität zu existierenden Strukturen scheint eine Repräsentation der Anwendungsbeschreibung als XML-Struktur sinnvoll. Ein weiterer Vorteil einer XML-Struktur ist die Möglichkeit, die spezifizierte Anwendungsbeschreibung syntaktisch durch Bereitstellung einer Schema-Definition zu prüfen.

Das XML-Schema enthält die im vorigen Kapitel aufgeführten Beschreibungseinheiten und die jeweils für eine Beschreibungseinheit notwendigen Attribute. In Abbildung 5.1 werden Auszüge der Schema-Definition von Anwendungsbeschreibungen für PCom dargestellt.

```
<!DOCTYPE APPDESCRIPTION [

<!ELEMENT APPLICATION_MODEL (DEPENDENCIES)>
<!ELEMENT DEPENDENCIES (SLEVEL+)>
<!ELEMENT SLEVEL (COMPONENT+)>
<!ELEMENT COMPONENT ((INTERFACE|EVENT|ANY)*,DEPENDENCIES?)>
<!ELEMENT INTERFACE (ANY*)>
<!ELEMENT EVENT (ANY*)>

<!ATTLIST APPLICATION_MODEL CLASS #REQUIRED>
<!ATTLIST APPLICATION_MODEL MODEL #REQUIRED>
<!ATTLIST APPLICATION_MODEL CLIENT #REQUIRED>
<!ATTLIST APPLICATION_MODEL NAME #REQUIRED>

<!ATTLIST DEPENDENCIES CLASS #REQUIRED>

<!ATTLIST SLEVEL CLASS #REQUIRED>
<!ATTLIST SLEVEL UTILITY #REQUIRED>

<!ATTLIST COMPONENT CLASS #REQUIRED>
<!ATTLIST COMPONENT LABEL #REQUIRED>

<!ATTLIST INTERFACE CLASS #REQUIRED>
<!ATTLIST INTERFACE TYPE #REQUIRED>

]>
```

Abbildung 5.1 Auszüge aus dem XML-Schema der Anwendungsbeschreibung

5.2 Objektmodell der Beschreibungssprache

Im vorangegangenen Kapitel wurde die Anwendungsbeschreibung für PCom entworfen. Die Anwendungsbeschreibung muss zur Nutzung bei der Adaption programmiertechnisch modelliert werden. Die Modellierung ist so umzusetzen, dass es dem Anwendungsprogrammierer einfach möglich ist, eine Anwendungsbeschreibung zu erstellen, so dass diese dann von PCom genutzt werden kann. Der Anwendungsprogrammierer erstellt die Anwendungsbeschreibung als eine XML-Datei, die den in obigem Absatz aufgestellten Vorgaben entspricht. Die für den Anwendungsprogrammierer zweckmäßige Darstellungsform als XML-Datei ist aus Verteilungsinfrastruktursicht nicht zweckmäßig.

Der Selektor und andere Teile der Verteilungsinfrastruktur müssen auf Informationen aus dieser Beschreibung zugreifen. Ein jeweiliges Analysieren beim Zugriff scheidet aus Effizienzgründen aus. Weiterhin muss die Beschreibung zwischen Containern ausgetauscht werden. In PCom existieren bereits Mechanismen, um entfernte Aufrufe zu realisieren und somit auch Daten über Adressraumgrenzen hinweg verfügbar zu machen. Dazu müssen die Daten allerdings in Objekten gekapselt sein. Es bietet sich also eine Abbildung der in XML vorliegenden Anwendungsbeschreibung auf ein entsprechendes Objektmodell an. Im Folgenden wird dieses Modell entworfen und die Zuordnung zwischen Beschreibungseinheiten und Objektklassen dargestellt.

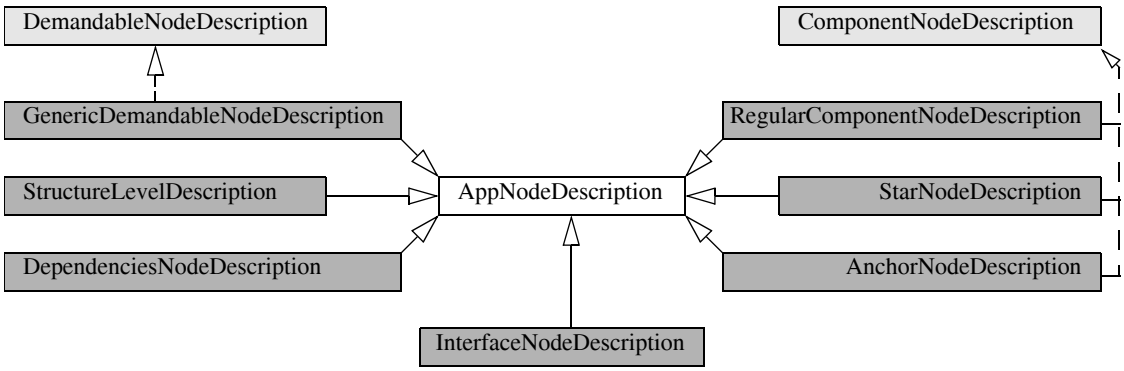


Abbildung 5.2 Vererbungshierarchie der Anwendungsbeschreibung

5.2.1 Struktur

Vor der Definition der einzelnen Beschreibungseinheiten ist die Betrachtung der Gesamtstruktur der Beschreibung hilfreich. Die Anwendungsbeschreibung repräsentiert Teile oder den gesamten Anwendungsbaum mit alternativen Teilbäumen. So lässt sich die Anwendungsbeschreibung wiederum in eine Baumdarstellung überführen. Die verschiedenen Beschreibungseinheiten sind per Definition in 4.1.4 hierarchisch vom Anwendungsanker ausgehend miteinander verkettet.

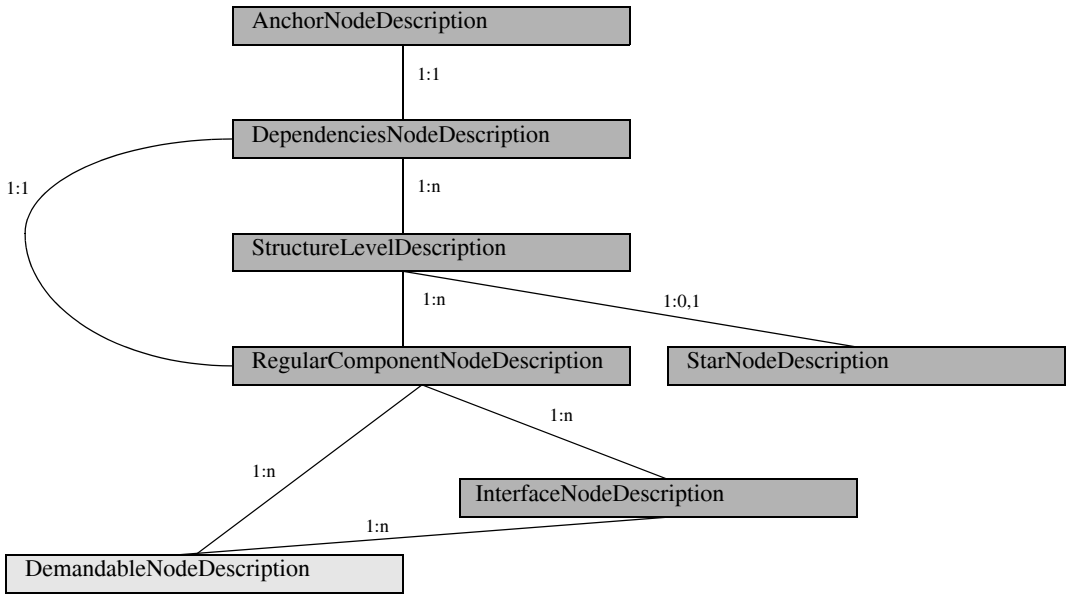


Abbildung 5.3 Objektmodell der Anwendungsbeschreibung

5.2.2 Implementierung der Beschreibungseinheiten

Für die im vorherigen Kapitel eingeführten Beschreibungseinheiten der Anwendungsbeschreibung werden Entsprechungen im Objektmodell geschaffen. Die Grundlage für alle folgenden Klassen bildet die abstrakte Klasse *AppNodeDescription*. Die Klasse implementiert die allen in der Anwendungsbeschreibung vorkommenden Beschreibungseinheiten gemeinsamen Methoden. Sie stellt unter anderem Funktionalität zur Identifikation des Knotens und zur Serialisierung bereit.

- *Komponente*: Alle Komponentenbeschreibungen in PCom implementieren das Interface *ComponentNodeDescription*. Dieses Interface enthält jene Methodenschnittstellen, die allen

Komponenten-Knoten, sei es der Anwendungsanker, eine Sternkomponente oder eine normale Komponente, gemein sind. Beispielsweise wird die Methode zur Prüfung auf Kompatibilität eines Angebots in der *ComponentNodeDescription* definiert. Dies ermöglicht eine einfachere Logik bei der Komponentenauswahl, da für den Auswahlalgorithmus die Schnittstelle für alle diese Knoten gleich ist. Eine normale Komponente wird auf die Klasse *RegularComponentNodeDescription* abgebildet. Die Klasse implementiert Funktionalität, die Kompatibilität zwischen einem Angebot und der spezifizierten Komponente zu prüfen sowie auch die nicht-funktionalen Eigenschaften der Komponente gegen eine Anforderung zu testen. Die Prüfung erfolgt auf Basis der Typkompatibilität zwischen den implementierten und geforderten Interfaces und Ereignissen.

- *Spezialfall Aggregation*: Einen Spezialfall einer Komponente stellt die *StarNodeDescription* dar. Sie repräsentiert eine Aggregation im Anwendungsmodell, wie in Absatz 4.1.6 beschrieben. Bei dieser Klasse sind die Prüfungen auf Kompatibilität und nicht-funktionale Eigenschaften immer erfolgreich, da sie explizit als Platzhalter für beliebige Komponenten definiert wurde.
- *Spezialfall Anwendungsanker*: Dem Anwendungsanker fällt auch im Objektmodell eine besondere Rolle zu. Im Gegensatz zu regulären Komponenten können keine funktionalen und nicht-funktionalen Eigenschaften spezifiziert werden. An der *AnchorNodeDescription* wird zudem der systemweit eindeutige Identifikator der gesamten Anwendungsbeschreibung abgelegt. Diese Eigenschaft wird später im Kapitel ausführlich diskutiert.
- *Bindung*: Bindungen stellen Abhängigkeiten der einzelnen Komponenten dar. Diese werden für jede in der Anwendungsbeschreibung spezifizierte Komponente in der *DependenciesNodeDescription* zusammengefasst. Die *DependenciesNodeDescription* dient als Container für die nach Alternativen gruppierten Bindungen einer Komponente. Sie bietet Funktionalität, um Komponenten anhand eines Angebots und eines semantischen Labels in der Alternative mit der höchsten Präferenz zu finden.
- *Anwendungsbeschreibung*: Die Anwendungsbeschreibung als Ganzes wird an Hand der am Anwendungsanker gespeicherten *ObjectID* identifiziert. So stellt auch das Anwendungsankerobjekt *AnchorNodeDescription* den Einstiegspunkt für den Zugriff auf die Anwendungsbeschreibung bereit. Eine weitere Kapselklasse ist nicht vorgesehen.
- *Funktionale Eigenschaft*: Die funktionalen Eigenschaften, die eine Komponente aufweisen soll, werden auf Komponentenebene durch die *InterfaceNodeDescription* beschrieben. Durch sie kann der Anwendungsprogrammierer sowohl Vorgaben bezüglich der zu implementierenden Interfaces wie auch zu den zu unterstützenden Ereignissen machen.
- *Nicht-Funktionale Eigenschaft*: Nicht-funktionale Eigenschaften können durch Klassen spezifiziert werden, die das Interface *DemandableNodeDescription* implementieren. An dieser Stelle wird im Hinblick auf einfache Erweiterbarkeit der bisher in PCom vorhandenen Unterstützung nicht-funktionaler Parameter das Interface verwendet. Eine konkrete Implementierung der *DemandableNodeDescription* findet sich in der Klasse *GenericDemandableNodeDescription*. Sie unterstützt alle bisher in PCom existierenden Möglichkeiten zur Spezifikation nicht-funktionaler Parameter und kann auf Interface-, Ereignis- und Komponentenebene eingefügt werden.
- *Alternative*: Alternativen können in der Anwendungsbeschreibung auf Bindungsebene spezifiziert werden. Die abgehenden Bindungen einer Komponente können in Qualitätsstufen gegliedert und diese mit einem Nutzwert (*Utility*) geordnet werden. Im Objektmodell werden alle Bindungen einer Qualitätsstufe in einer *StructureLevelNodeDescription* zusammengefasst. Die Instanzen dieser Klasse enthalten auch Zugriffsmethoden auf den jeweils spezifizierten Nutzwert.

5.2.3 Zugriff und Identifikation

Jeder der im vorangegangenen Kapitel entworfenen Beschreibungseinheiten wurde eine entsprechende Klasse im Objektmodell zugewiesen. Eine Anwendungsbeschreibung kann inhaltlich nun auf dieses Objektmodell abgebildet werden. Über die inhaltliche Abbildung hinaus werden im Folgenden die Mechanismen für den Zugriff auf die Anwendungsbeschreibung und deren Beschreibungseinheiten und damit verbunden die Identifikation der Anwendungsbeschreibung und ihrer Bestandteile vorgestellt.

- *Identifikation der Anwendungsbeschreibung*: Eine Anwendungsbeschreibung kann auf verschiedenen Geräten zum Einsatz kommen, abhängig von der Verteilung der in der Anwendung verbundenen Komponenten. Um eine eindeutige Identifikation der Anwendungsbeschreibung über Adressraumgrenzen hinweg zu gewährleisten, muss ein systemweit eindeutiger Identifikator festgelegt werden. Dieser Identifikator wird durch eine *ObjectID* realisiert, die besagte Anforderungen erfüllt.
- *Identifikation von Beschreibungseinheiten*: Innerhalb einer Anwendungsbeschreibung ist es notwendig, einzelne Beschreibungseinheiten identifizieren zu können. Beispielsweise muss eine Komponente ihre Position in der Anwendungsbeschreibung kennen um für ihre Bindungen die in der Anwendungsbeschreibung spezifizierten Anforderungen prüfen zu können. Innerhalb einer Anwendungsbeschreibung werden aus diesem Grund alle Beschreibungseinheiten durchnummeriert. Dabei wird ein einfacher *Integer* verwendet, da nur anwendungsbeschreibungsinterne Eindeutigkeit bestehen muss und somit der Wertebereich ausreicht.
- *ModelID*: Eine Beschreibungseinheit in einer Anwendungsbeschreibung lässt sich global eindeutig durch die *ObjectID* der Anwendungsbeschreibung und den *Integer* der Beschreibungseinheit identifizieren. Auf Grund der Häufigkeit, mit der diese beiden Informationen verknüpft werden, wird die *ModelID* bereitgestellt, die diese beiden Informationen in einer eigenen Klasse kapselt.
- *Zugriff auf Beschreibungseinheiten*: Der Zugriff auf die Beschreibungseinheiten, beispielsweise zur Auswertung von spezifizierten Eigenschaften, kann auf zwei Arten erfolgen. Zum einen kann der Zugriff natürlich über die Nummer der Beschreibungseinheit erfolgen. Zum anderen besteht eine wichtige Zugriffsmethode in der Bestimmung einer Komponentenbeschreibung, die bestimmte Anforderungen erfüllt und ein spezielles semantisches Label aufweist. Dieser Zugriff kann bei der Anwendungskonfiguration zur Bestimmung von kompatiblen Komponentenbeschreibungen verwendet werden. Dabei wird im Falle mehrerer möglicher Resultate automatisch diejenige Komponentenbeschreibung zurückgeliefert, die in der Qualitätsstufe mit dem höchsten Nutzen spezifiziert wurde. Der Zugriff durch semantische Labels kann auch als Resultat eine Stern-Komponente haben, da sie in die Anwendungsbeschreibung als Aggregation für beliebige Komponenten eingefügt werden kann.

5.2.4 Verfolgen von Änderungen

Eine grundlegende Anforderung an ubiquitäre Systeme ist Dynamik. So muss sich auch die Anwendungsbeschreibung zur Laufzeit ändern lassen. Die Änderungen können Auswirkungen auf die momentane Konfiguration der Anwendung haben. Um es Anwendungs- und Komponentenprogrammierern zu ermöglichen, über Änderungen an der Anwendungsbeschreibung benachrichtigt zu werden, wurde der *ApplicationModelListener* geschaffen, der entsprechend dem Observer-Pattern[20] arbeitet.

5.2.5 Fehlerhafte Anwendungsbeschreibungen

Treten bei der Instanziierung einer Anwendungsbeschreibung, beispielsweise bei dem Versuch einer Beschreibungseinheit eine unzulässige weitere Beschreibungseinheit anzufügen, Fehler auf, so wird eine *IllegalModelException* erzeugt, die eine entsprechende Fehlerbeschreibung enthält. Wird die Anwendungsbeschreibung als XML-Datei erzeugt und durch den Generator umgewan-

delt, so werden diese Fehler bereits bei der Umwandlung dem Anwendungsprogrammierer mitgeteilt.

5.2.6 Erzeugung der Anwendungsbeschreibung

Die Spezifikation der Anwendungsbeschreibung wird durch den Anwendungsprogrammierer in Form einer XML-Datei vorgenommen. Diese muss nun in das eben vorgestellte Objektmodell überführt werden. Die Umsetzung der Überführung kann analog zu der bei den Komponentenverträgen bereits vorhandenen Methode erfolgen. Der vorhandene Generator für Komponentenverträge wurde zur Konvertierung von Anwendungsbeschreibungen in eine Java-Struktur erweitert und erzeugt für jede Anwendungsbeschreibung eine Java-Quellcodedatei. Während des Konvertierungsprozesses wird eine Prüfung auf syntaktische Korrektheit der Anwendungsbeschreibung durchgeführt.

5.2.7 Dynamisches Laden

Komponenten und dementsprechend auch Anwendungen werden in PCom dynamisch geladen. Folglich ist es zwingend erforderlich, dass die Klassen der Anwendungsbeschreibung auch dynamisch geladen werden können. Für Komponentenverträge existiert bereits ein Mechanismus zum dynamischen Laden der Vertragsdaten. Auf Basis dieser Implementierung wurde das Laden der Anwendungsbeschreibung realisiert.

Die Anwendungsbeschreibung wird entweder manuell oder durch den Generator auf eine Java-Klasse abgebildet. Die Klasse stellt noch nicht die Repräsentation durch das Objektmodell dar, sondern ermöglicht das dynamische Laden der Anwendungsbeschreibung. Die automatisch oder manuell erzeugte Klasse ist von *LoadableModel* abgeleitet. Nach deren Instanziierung wird durch Aufruf einer Methode eine Instanz der Anwendungsbeschreibung gemäß dem vorher spezifizierten Objektmodell erzeugt.

5.3 Integration in PCom

Die Anwendungsbeschreibung wird an verschiedenen Stellen in PCom referenziert und verwendet. Dies erfordert Strukturen in PCom, in denen Anwendungsmodelle abgelegt, verteilt und ausgewertet werden. Im folgenden Abschnitt wird zunächst die statische Struktur der Integration der Anwendungsbeschreibung in PCom beschrieben. Sie beinhaltet eine Übersicht über die die Anwendungsbeschreibung referenzierenden Programmteile und das containerbasierte Caching von Anwendungsbeschreibungen. Anschließend erfolgt eine Beschreibung der Erweiterungen und Auswirkungen auf das dynamische Modell von PCom. Diese sind notwendig zur Bereitstellung und zum adressraumübergreifenden Transfer von Anwendungsbeschreibungen und schließlich der Verwendung von Anwendungsbeschreibungen in PCom.

5.3.1 Statisches Modell

Das Objektmodell der Anwendungsbeschreibung wurde im vorangegangenen Abschnitt definiert. Die Anwendungsbeschreibung wird anhand ihrer systemweit eindeutigen *ObjectID* identifiziert und der Zugriff auf Beschreibungseinheiten erfolgt über die Repräsentation des Anwendungsankers. Die Gesamtstruktur wird von der Verteilungsinfrastruktur an verschiedenen Stellen verwendet. Die Referenzen, der Zugriff und das Caching von Anwendungsbeschreibungen werden im Folgenden beschrieben.

Verbindung von Anwendung und Beschreibung. Die Anwendungsbeschreibung wird pro Anwendung spezifiziert. Die Verbindung zwischen der Beschreibung und dem Anwendungsanker erfolgt durch eine zusätzliche Angabe im Komponentenvertrag des Anwendungsankers. Der Anwendungsprogrammierer fügt in der *Implementation*-Sektion des Vertrags ein

Attribut mit einem Verweis auf die ladbare Klasse der Anwendungsbeschreibung ein. Falls der Anwendungsprogrammierer keine Anwendungsbeschreibung spezifiziert, so ist die Anwendung trotzdem vollkommen funktionsfähig. Es können lediglich die neuen Möglichkeiten der Anwendungsbeschreibung bei der Adaption nicht genutzt werden.

Das Modelrepository. Jede Anwendung in PCom kann über eine Anwendungsbeschreibung verfügen. Die Anwendungsbeschreibung muss aus verschiedenen Gründen, beispielsweise für das neue Binden von Abhängigkeiten einer Komponente, allen Komponenten der Anwendung in geeigneter Form zur Verfügung stehen. Die Bereitstellung kann auf mehrere Arten erfolgen. Eine Möglichkeit besteht darin, die Anwendungsbeschreibung bei den entsprechenden Methodenaufrufen als Parameter mit zu übergeben. Zwei Nachteile werden bei dieser Methode deutlich. Zum einen werden je nach Umfang der Anwendungsbeschreibung große Datenmengen als Parameter übergeben - dies fällt vor allem bei entfernten Aufrufen ins Gewicht. Zum anderen muss, wenn nicht alle Aufrufe vom Anwendungsanker, der die initiale Referenz auf die Anwendungsbeschreibung hält, die Anwendungsbeschreibung trotzdem zwischengespeichert werden. Eine Methode zur Behebung dieses Problems wäre die Speicherung der Anwendungsbeschreibung in jeder an der Anwendung beteiligten Komponente. Allerdings entstehen durch die genannte Lösung zwei neue Probleme. Besitzt eine Anwendung n Komponenten, so wird die Anwendungsbeschreibung n-Mal gespeichert. Diese Methode ist sehr speicherplatzintensiv, vor allem, da anzunehmen ist, dass verschiedene Komponenten der Anwendung im gleichen Container ausgeführt werden. Weiterhin müssen auch bei jeder Änderung der Anwendungsbeschreibung n Aktualisierungen vorgenommen werden. Die Vorteile lokaler Verfügbarkeit der Anwendungsbeschreibung und minimaler Redundanz und damit geringerem Speicherbedarf und geringeren Aktualisierungskosten vereint ein vom Container verwendeter Cache für Anwendungsbeschreibungen.

Das *ModelRepository* beinhaltet alle der von den Komponenten eines Containers benötigten bzw. referenzierten Anwendungsbeschreibungen. Die Cache-Verwaltung basiert auf einem einfachen Reference-Counting. Jede Komponente, die eine Beschreibung verwendet bzw. nicht mehr benötigt wird, registriert. Periodisch wird der Reference-Counter überprüft. Wenn auf eine Anwendungsbeschreibung keine Referenzen mehr existieren, so wird ein Flag gesetzt, um die Anwendungsbeschreibung im nächsten Prüfzyklus zu löschen. Dieser zusätzliche Zyklus stellt sicher, dass eine während eines Adaptions- oder Selektionsvorgangs geladene Anwendungsbeschreibung nicht sofort wieder entfernt wird.

Referenzen auf Anwendungsbeschreibungen. Eine Anwendungsbeschreibung für eine Anwendung wird erstmalig im Komponentenvertrag des entsprechenden Anwendungsankers spezifiziert. Wird der Komponentenvertrag vom Container geladen und eine entsprechende *Factory*-Instanz erzeugt, so wird im zugehörigen *FactoryTranslator* eine *ObjectID* als Referenz auf die Anwendungsbeschreibungsinstanz gesetzt.

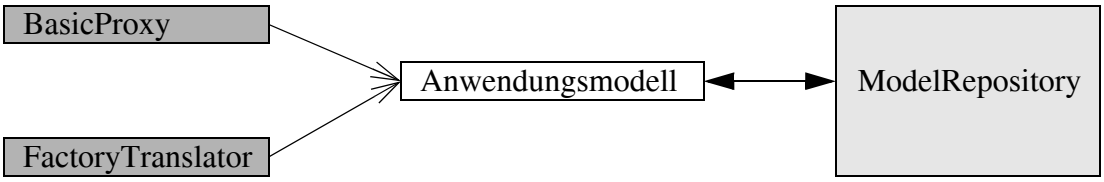


Abbildung 5.4 Referenzen auf das Anwendungsmodell

Jede in einer Anwendung verwendete Komponente kennt den Vertrag, durch den sie an die nutzende Komponente gebunden ist. Die nutzende Komponente greift durch einen Proxy, der von *BasicProxy* abgeleitet ist, auf die genutzte Komponente zu. Muss die Bindung wiederhergestellt werden, so muss der Proxy Kenntnis über die Vertragsanforderungen sowie über die Position der Komponente in der Anwendungsbeschreibung besitzen. Das Caching von Anwendungsbeschreibungen ermöglicht es, dass in *BasicProxy* nur eine *ModelID* auf die Anwendungsbeschreibung und die Position der Komponente innerhalb der Beschreibung als Referenz gespeichert werden muss.

5.3.2 Dynamisches Modell

Das dynamische Modell von PCom muss für die Anwendungsbeschreibung an verschiedenen Stellen erweitert werden. Im Folgenden werden die Erweiterungen in einer an dem Lebenszyklus von Komponenten orientierten Reihenfolge vorgestellt.

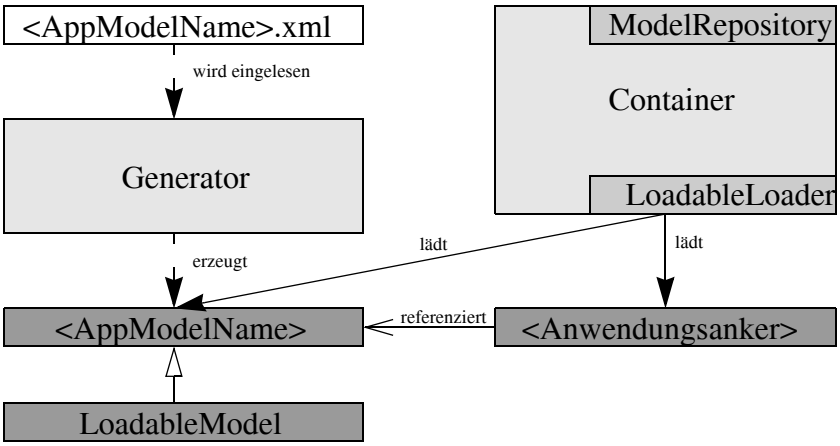


Abbildung 5.5 Laden des Anwendungsmodells

Dynamisches Laden von Anwendungsbeschreibungen. Die Anwendungsbeschreibungen liegen in einer durch PCom dynamisch nachladbaren Form vor. Die vom Generator erzeugten oder manuell erstellten und von *LoadableModel* abgeleiteten Klassen werden vom Container beim Laden der zur Verfügung stehenden Komponenten (*-Factories*) automatisch mitgeladen, sofern sie in Komponentenverträgen referenziert werden. Die geladene Anwendungsbeschreibung wird im *ModelRepository* des Containers abgelegt. Eine Referenz auf die Anwendungsbeschreibung wird bei der entsprechenden *Factory* gesetzt.

Änderungen an Anwendungsbeschreibungen. In Abschnitt 5.2.4 wurde das Observer-Pattern beschrieben, mit dem es Komponenten ermöglicht wird, Änderungen an Anwendungsbeschreibungen nachzuverfolgen. Standardmäßig werden bei Änderungen an einer Anwendungsbeschreibung alle Container, die die Anwendungsbeschreibung in ihrem *ModelRepository* gecacht haben, benachrichtigt. Dazu wird, sobald ein Container eine Anwendungsbeschreibung von einem entfernten Container lädt, ein Listener-Objekt im entfernten Container gesetzt, der das Interface *ApplicationModelListener* implementiert. Im Fall der Änderung der beobachteten Anwendungsbeschreibung sendet dann der entfernte Container, von dem die Anwendungsbeschreibung ursprünglich geladen wurde, die aktualisierte Anwendungsbeschreibung. Die Komponenten, die an der zu der Anwendungsbeschreibung zugehörigen Anwendung beteiligt sind, werden standardmäßig nicht benachrichtigt.

Komponentenauswahl. Die Komponentenauswahl wird durch den *Selector* in PCom bei der initialen Anwendungskonfiguration und der Rekonfiguration einer Anwendung durchgeführt. Das neue Konzept für den Selektor, dass in Abschnitt 4.2 vorgestellt wurde, erfordert eine komplette Neuimplementierung des bisherigen Selektors. An Stelle der abstrakten Basisklasse *AbstractSelector* tritt nun das Interface *Selector*, welches den Zugriff auf die Ergebnisiteratoren ermöglicht. Der Selektor ist ein zweidimensionaler Iterator, der durch die Methoden *highestQualityLevel, nextQualityLevel, firstDependency* und *nextDependeny* entlang den beiden Ergebnisdimensionen Qualität und Abhängigkeit frei bewegt werden kann. Der Rückgabewert eines solchen Methodenaufrufs ist ein *Iterator* über die Menge an für diese Qualität und Abhängigkeit verfügbaren Komponenten. Dieser Iterator kann gemäß dem Java-Iteratorinterface verwendet werden. Elemente dieser Menge sind wie bisher Objekte der Klasse *ComponentContract*. Über den Selektor kann aber noch zusätzlich die Position der selektierten Komponente bzw. des *ComponentContracts* im Anwendungsbaum ermittelt werden. Weiterhin stehen Methoden zum Abrufen der momentanen Qualitätsstufe wie auch des für die momentane Abhängigkeit maßge-

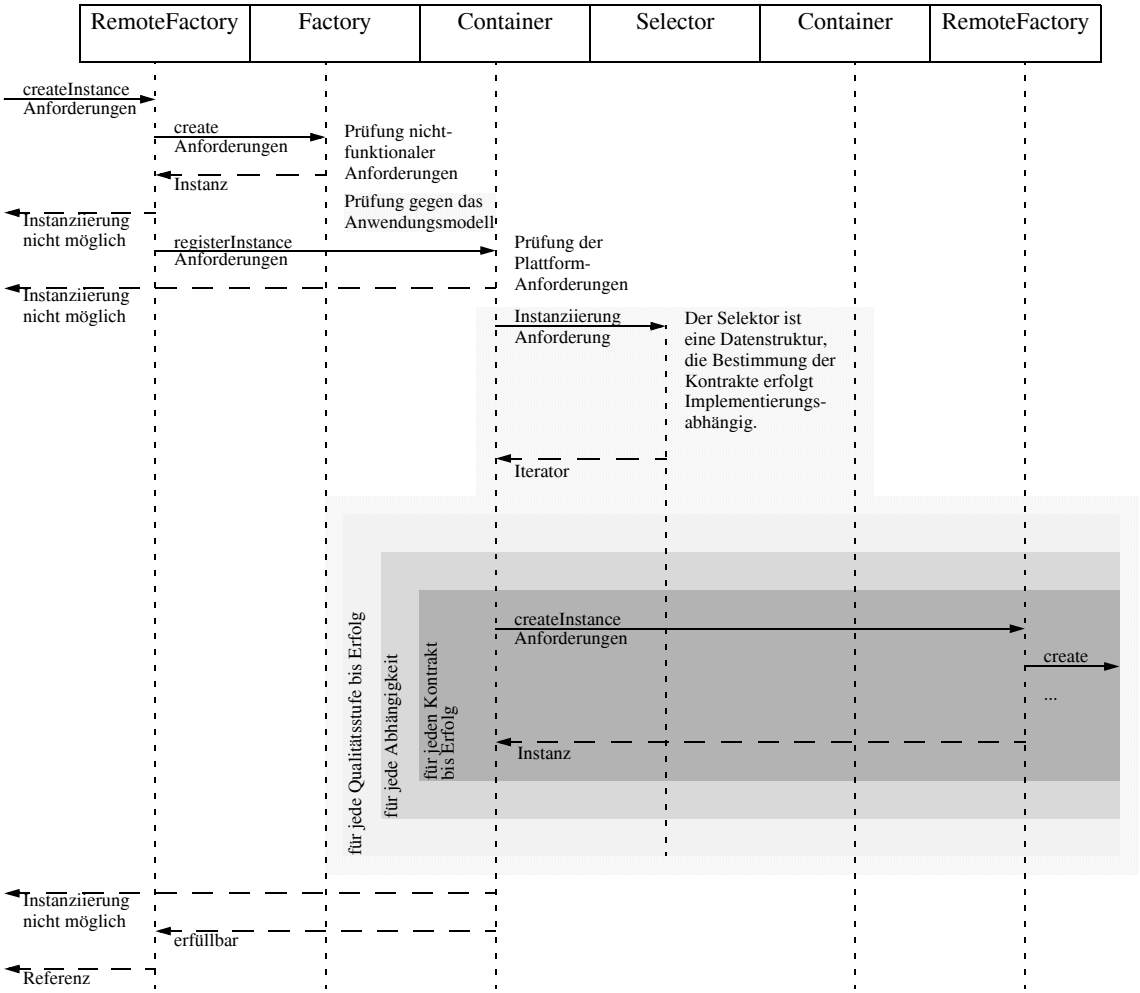


Abbildung 5.6 Neuer Ablauf bei der Komponenteninstanziierung in PCom

benden *ComponentDemandContract* bereit. Der neue Ablauf für die Selektion am Beispiel der Komponenteninstanziierung wird in Abbildung 5.6 dargestellt.

5.4 Selektionsstrategien

Auf Basis der neuen Selektorimplementierung wurden zwei verschiedene Selektionsstrategien implementiert, die jeweils verschiedene, in Abschnitt 4.3 beschriebene Konzepte verfolgen.

5.4.1 InstantSelector

Die Umsetzung der ersten Selektionsstrategie (sofortige Auswahl, 4.3.1) erfolgt analog zur bisher in PCom implementierten Selektionsstrategie. Die Klasse *InstantSelector* implementiert das *Selector*-Interface und bietet zwei Konstruktoren. Der eine Konstruktor nimmt neben dem aufrufenden Container eine Liste von Anforderungen in Form von *ComponentDemandContracts* und eine *ModelID* mit der Position derjenigen Komponente im Anwendungsbaum, deren Bindungen selektiert werden sollen, entgegen. Der andere Konstruktor erhält nur einen *ComponentDemandContract* und nicht die *ModelID* der nutzenden sondern die *ModelID* der genutzten Komponente. Dieser Konstruktor ist für die Erneuerung einer bestehenden Bindung gedacht.

Nach der Initialisierung rufen beide Konstruktoren die interne Methode *select* auf, die die gesamte Selektion in einem Schritt durchführt. Die Bestimmung passender Komponenten erfolgt auf Basis von Schnittstellenkompatibilität unter Berücksichtigung der Anforderungen und der Anwendungsbeschreibung. Die bestimmten Komponenten werden in einer Ergebnismatrix abgelegt. Die Ergebnisse der Selektion stehen also direkt nach Instanziierung des *InstantSelectors* fest. Die Realisierung der Navigationsmethoden beschränkt sich auf das entsprechende Zurückliefern der bereits im Konstruktor bestimmten Ergebnismengen.

5.4.2 LazyRemoteContainerSelector

Die Strategie der verzögerten Auswertung (Abschnitt 4.3.2) nutzt die Möglichkeit des neuen Selektorkonzepts und führt die Selektion erst bei Bedarf durch. Analog zum *InstantSelector* bietet der *LazyRemoteContainerSelector* die *Selector*-Schnittstelle und zwei Konstruktoren für die oben beschriebenen Fälle an. Im Unterschied zur vorangegangenen Implementierung werden allerdings im Konstruktor nur lokal verfügbare Komponenten ausgewertet und nach Herkunftscontainer, Qualitätsstufen und Abhängigkeiten geordnet abgelegt. Zudem wird noch eine Liste aller verfügbaren Container als Instanzvariable angelegt. Die Implementierung der Navigationsmethoden ist ansonsten analog zu der des *InstantSelector* gestaltet. Da die gesamte Ergebnismenge zur Zugriffszeit noch nicht feststeht, liefern sie an Stelle eines Iterators auf die Ergebnismenge einen spezialisierten Iterator - den *LazyIterator* - zurück, der eine verzögerte Auswertung erlaubt.

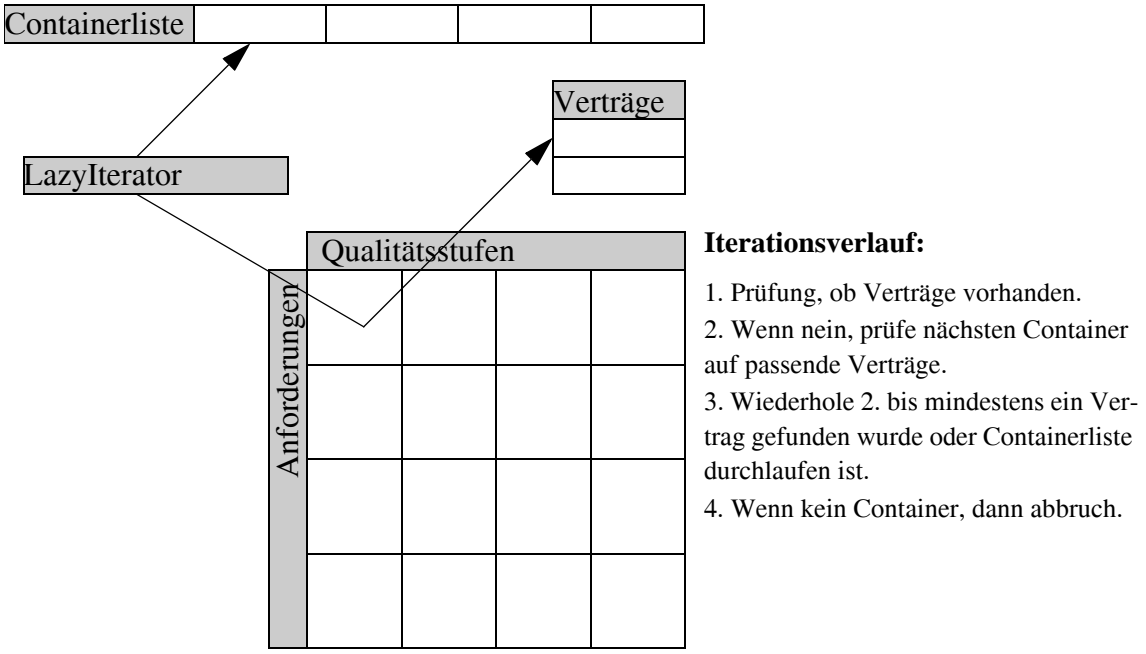


Abbildung 5.7 Struktur des LazyIterators

Die Klasse *LazyIterator* implementiert das *Iterator*-Interface. Wird nun das nächste Element der Ergebnismenge (auch hier ein *ComponentContract*) abgerufen oder auch nur ein *hasNext* aufgerufen, so wird geprüft, ob sich in der vorhandenen Ergebnismenge noch weitere Elemente befinden. Ist dies der Fall, so wird das entsprechende Element zurückgeliefert bzw. *true* zurückgeliefert. Im Falle, dass keine Elemente mehr in der Ergebnismenge vorhanden sind, wird versucht, weitere Elemente zu bestimmen. Dazu wird für die momentane Abhängigkeit kontrolliert, welche entfernten Container schon nach passenden Komponenten gefragt wurden. Die Liste der noch nicht überprüften Container wird so lange durchlaufen, bis entweder keine weiteren Container mehr verfügbar sind, oder mindestens eine Komponente existiert, die den Anforderungen entspricht. Werden auf diesem Weg eine oder mehrere passende Komponenten gefunden, so werden sie entsprechend in die Ergebnismatrix aufgenommen. Ist dies nicht der Fall, so wird für *hasNext false* zurückgeliefert. Anzumerken ist hier, dass beim Durchlaufen der entfernten Contai-

ner jeweils die Ergebnisse für alle Qualitätsstufen einer Abhängigkeit berechnet werden, da das containerseitige Interface zur Abfrage von syntaktisch kompatiblen Komponenten zu einer Anforderung keine Spezifikation bzw. Beschränkung auf eine Qualitätsstufe zulässt.

5.5 Adaptionsstrategien

Die Umsetzung der Adaptionsstrategien ist, wie bereits in Abschnitt 4.4 diskutiert, in dieser Arbeit auf die Implementierung spezieller Handler beschränkt. Um in Zukunft die Entwicklung und die Integration individueller Adaptionsstrategien bei Handlern zu erleichtern, wurde zu diesem Zweck ein Strategy-Pattern eingeführt. Das in Abbildung 5.8 abgebildete Pattern zeigt die weiterhin abstrakte Standard-Implementierung des Interfaces *ExceptionHandler*. Von dieser Implementierung können bindungsspezifische Implementierungen des Verhaltens im Fehlerfall (bzw. nicht Adaptionsfall) abgeleitet werden. Die Klasse *AbstractExceptionHandler* realisiert, dass bei bestimmten Fehlern an Stelle des Standardfehlerverhaltens eine Rekonfiguration der Anwendung ausgelöst wird. Die Adaptionsstrategie wird dabei auf Containerebene festgelegt und durch Weiterleitung des Rekonfigurationsaufrufs durch den Proxy aktiviert. Es wurde als Beispiel eine neue Fehlerklasse eingeführt, die bei Auftreten im *AbstractExceptionHandler* eine Rekonfiguration auslöst. Dabei handelt es sich um die von *InvocationException* abgeleitete Klasse der *ComponentNotAvailableException*.

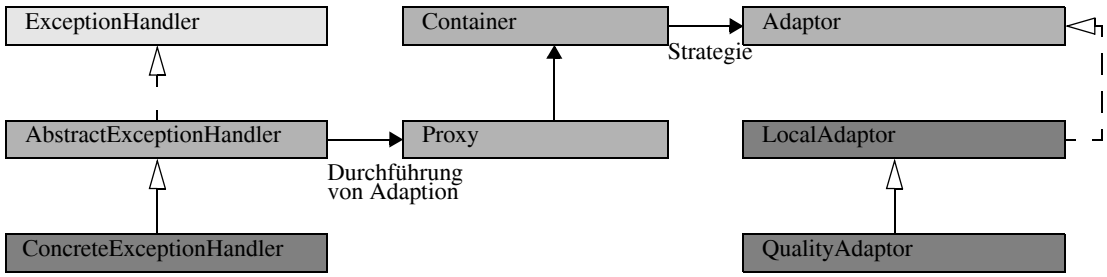


Abbildung 5.8 Strategy-Pattern für Adaptionsstrategien

Das Modell ermöglicht weiterhin eine bindungsspezifische Fehlerbehandlung durch Ableitung und Erweiterung der Klasse *AbstractExceptionHandler*. Die im Folgenden vorgestellten Adaptionsstrategien basieren auf dieser neu geschaffenen Struktur.

5.5.1 LocalAdaptor

Die Strategie der möglichst lokalen Adaption (siehe Abschnitt 4.4.1) wird durch die Klasse *LocalAdaptor* umgesetzt. Die Implementierung versucht in jedem Fall, die Problemstelle neu zu binden. Dies geschieht durch Aufruf der *rebind*-Methode des Proxies und - im Falle des Misserfolgs - wird der Fehler auf höhere Ebenen im Anwendungsbaum propagiert. Die Umsetzung basiert auf einer Beispielimplementierung eines Handlers, die im Rahmen der Arbeit von Marcus Handte[26] erstellt wurde.

5.5.2 QualityAdaptor

Der *QualityAdaptor* setzt die Konzepte der möglichst optimalen Adaption um, die in Absatz 4.4.2 vorgestellt wurden. Der *QualityAdaptor* nutzt die Anwendungsbeschreibung, um zu bestimmen, ob sich die Problemstelle in einem Ast des Anwendungsbaums mit höchstmöglicher Qualitätsstufe befindet. Der Anwendungsanker stellt hierzu die Methode *isOptimalComponent* bereit. Wird festgestellt, dass die Problemstelle keine solche Komponente ist, so wird der Fehler an die aufrufende Komponente und damit im Anwendungsbaum nach oben propagiert. Andernfalls wird analog zur möglichst lokalen Adaption verfahren und versucht, mittels Neubinden der Problemstelle die Bindung wiederherzustellen.

5.6 Verhandlungsstrategien

Für das Konzept der Vertragsverhandlungen werden im Rahmen dieser Arbeit nur grundlegende Mechanismen implementiert, um einen späteren Ausbau zu ermöglichen, ohne die Verteilungsinfrastruktur anpassen zu müssen.

Das Interface *Negotiator* bietet analog zum *Selector* Methoden, über die Ergebnisse der Verhandlungen zu iterieren. Der *Negotiator* erlaubt die Navigation entlang der zwei Ergebnisdimensionen Qualität und Anforderung und liefert einen Iterator auf die jeweils zur entsprechenden Qualitätsstufe und Anforderung passenden Angebotsverträge zurück.

Eine konkrete Realisierung der Datenstruktur stellt der *InstantNegotiator* dar. Bei Instanziierung prüft er die vorhandenen Angebote auf dem lokalen und allen entfernt verfügbaren Containern. Die verfügbaren Angebote werden anschließend (die Verfügbarkeit einer Anwendungsbeschreibung vorausgesetzt) entsprechend den Qualitätsstufen zugeordnet.

Für die gefundenen Ergebnisse kann allerdings keine Garantie auf Instanziierbarkeit gegeben werden, da zwischen dem Zeitpunkt der Prüfung und der Ergebnisauswertung keine Ressourcenreservierung vorgenommen werden kann.

Kapitel 6

Evaluation

Die Umsetzung der Anwendungsbeschreibung in PCom eröffnet viele neue Möglichkeiten für den Anwendungsprogrammierer. Die zusätzliche Komplexität der Verteilungsinfrastruktur wird durch die im ersten Abschnitt des Kapitels erhobenen Metriken versucht zu erfassen. Der Preis der Erweiterung im Hinblick auf das Laufzeitverhalten in PCom wird im zweiten Abschnitt der Evaluation anhand von Speicherverbrauchs- und Geschwindigkeitsmessungen untersucht. Ein Rückblick auf die eigentlichen Entwurfskriterien und in wie weit die konkrete Umsetzung diese erfüllt folgt im dritten Abschnitt. Im vierten Abschnitt wird schließlich untersucht, ob die Anwendungsbeschreibung ein geeignetes Mittel ist, den Anwendungsszenarien, die in Kapitel 3 vorgestellt wurden, Rechnung zu tragen und deren Implementierung zu vereinfachen.

6.1 Metriken

Die im Folgenden erhobenen Kennzahlen sollen einen Überblick über die der Verteilungsinfrastruktur PCom hinzugefügte Komplexität geben. Es wird zwischen dem System neu hinzugefügten Programmteilen und geänderten Programmteilen differenziert, da ein großer Teil der Implementierung in der Anpassung bestehender Strukturen bestand. Als Vergleichsimplementierung dient PCom im Zustand vor der Implementierung der Anwendungsbeschreibung. Zuerst werden die Metriken bezüglich der gesamten Anwendung erhoben und damit der Umfang der Erweiterungen verdeutlicht. Anschließend werden die Kennzahlen ausgewählter Teile der Infrastruktur zum Vergleich zwischen der bisherigen und der neuen Implementierung untersucht. Die einzelnen untersuchten Programmteile setzen sich dabei wie folgt zusammen:

- *PCom&BASE*: PCom&BASE umfasst das Komplettsystem, auf dem die Änderungen für die Anwendungsbeschreibung implementiert wurden.
- *neue/entfernte/modifizierte Klassen/Interfaces*: Dieser Block stellt das Delta zwischen PCom mit der Erweiterung durch die Anwendungsbeschreibung und dem System vor der Erweiterung dar.
- *Anwendungsmodell:* Die Gruppe Anwendungsmodell umfasst alle Klassen und Interfaces, die sich in *pcom.model* befinden. Sie stellen das Objektmodell der Anwendungsbeschreibung einschließlich verschiedener Hilfsklassen dar.
- *Verträge:* Verträge umfassen alle in *pcom.contract* abgelegten Klassen und Interfaces. Sie stellen die Implementierung der verschiedenen Vertragsformen für Komponenten dar und sind im Zuge der Erweiterung nur minimal zum ursprünglichen System angepasst worden.

- *Selektor (neu/alt)*: Die Gruppen des alten und neuen Selektors enthalten jeweils alle Klassen und Interfaces aus *pcom.container.selector* und *pcom.container.negotiator*. Diese Codeblöcke umfassen die vollständige Implementierung des für die Komponentenauswahl zuständigen Selektors. Die betrachtete neue Implementierung schließt dabei nur den *InstantSelector* ein. Die zweite Strategie des *LazyRemoteContainerSelectors* wurde für diese Aufstellung nicht berücksichtigt um die Vergleichbarkeit zur bestehenden Implementierung zu gewährleisten.
- *RegisterInstance (neu/alt)*: Die Methode *registerInstance* ist eine auf Containerebene implementierte Methode, die für die Auflösung der Abhängigkeiten einer Komponente zuständig ist. Zur Auswahl der Komponenten verwendet sie den Selektor.

Die Kennzahlen bestehen aus einer Gruppe statischer Attribute: Zahl der Codezeilen, Zahl der Klassen und Zahl der Interfaces. Weiterhin wurden zur Vergleichbarkeit - wie auch in der Arbeit von Marcus Handte[26] - Messungen bezüglich der zyklomatischen Komplexität nach McCabe[31] und der Verschachtelungstiefe erhoben. Je nach Programmteil wurden alle oder Teile dieser Kennzahlen ermittelt.

Zur Erhebung der Messungen wurde, wie auch in den vorangegangenen Arbeiten, das Eclipse Metrics Plug-In[11], hier in der Version 1.3.4, verwendet. Die Ergebnisse der Messungen werden in Tabelle 1 und Tabelle 2 dargestellt.

6.1.1 Auswertung

Die zusätzlichen Codeteile der Implementierung der Anwendungsbeschreibung stellen vom Umfang her etwa 25% des bisher vorhandenen Systems dar. Für eine so wesentliche Erweiterung der Verteilungsinfrastruktur ist dies eine angemessene Größe. Die Zielvorgabe der Minimierung, damit die Funktionsfähigkeit auf ressourcenschwachen Geräten erhalten bleibt, wird alleine durch die Codegröße nicht gefährdet. Bemerkenswert ist das Verhältnis zwischen der Prozentzahl des neu hinzugefügtem Code und der Zahl der modifizierten Schnittstellen. Beide betragen 25%. Diese Zahl wird vorwiegend durch die Bereitstellung anwendungsspezifischer Informationsquellen bedingt. Die verschiedenen Schnittstellen mussten um entsprechende Methoden erweitert werden, die den Zugriff und Austausch der Anwendungsbeschreibung erlauben. Die Modifikation an vorhandenen Implementierungen, also den Klassen der Verteilungsinfrastruktur, ist dagegen auf relativ wenige Programmteile beschränkt. Die Modifikationen in diesen wenigen Klassen sind jedoch teilweise erheblich, wie beispielsweise die Methode *registerInstance* auf Containerebene zeigt. Auf Grund der Anpassungen an diesen Klassen ändern sich auch vier aus diesen automatisch generierten Klassen (Skeletons und Proxies). Sie werden in der Tabelle geklammert ausgewiesen.

Programmteil	# LOC	# Klassen	# Interfaces	Ø zyklomatische Komplexität	Ø Verschachtelungstiefe
PCom&BASE (alt)	3764	89	32	-	-
neue Klassen/Interfaces	889	22	6	-	-
entf. Klassen/Interfaces	60	2	0	-	-
mod. Klassen/Interfaces	-	7 (+4)	8	-	-

Tabelle 1 Metriken des Gesamtsystems

Die Gegenüberstellung der Programmteile Anwendungsmodell und Verträge zeigt, dass sich die Ähnlichkeit im Beschreibungsumfang der Verträge und der Anwendungsbeschreibung auch in einem ähnlichen Code im Hinblick auf Größe und Komplexität niederschlägt. Es wurden bei der Implementierung auch diverse Algorithmen aus den Vertragsklassen angepasst und in die

Anwendungsbeschreibung übernommen. An dieser Stelle könnte eine Überarbeitung des Designs zur Identifizierung und Separierung von gemeinsam genutzten Codeteilen Vorteile bei der Wartbarkeit und Erweiterbarkeit erzielen.

Programmteil	# LOC	# Klassen	# Interfaces	Ø zyklomatische Komplexität	Ø Verschachtelungstiefe
Anwendungsmodell	415	10	4	1,99	1,495
Verträge	463	12	4	1,982	1,518
Selektor (neu)	172	3	2	2,7	1,34
Selektor (alt)	60	2	0	2,75	1,5
registerInstance (neu)	112	-	-	19	8
registerInstance (alt)	95	-	-	12	5

Tabelle 2 Metriken von Teilbereichen

Der Selektor in der neuen Implementierung weist im Vergleich zur bisherigen Implementierung in PCom die dreifache Anzahl an Codezeilen auf. Dies zeigt deutlich den erhöhten Codeumfang, der durch die Auswertung der Anwendungsbeschreibung und der damit eingeführten verschiedenen Qualitätsstufen entsteht. Zudem erzeugt die höhere Abstraktion mit dem Selektor als Datenstruktur einen Mehraufwand an Code. Die komplexen Teile der Selektorimplementierung, die Algorithmen zur Auswahl der den Anforderungen entsprechenden Komponenten, sind in den beiden verglichenen Implementierungen ähnlich, da der *InstantSelector* eine Nachbildung des bisherigen *BasicSelector* darstellt. Die Komplexität dieser beiden Codeblöcke unterscheidet sich aus diesem Grund nur marginal.

Ein interessantes Ergebnis zeigt sich bei der Betrachtung der Methode *registerInstance* in der Containerimplementierung. In ihr werden die Abhängigkeiten einer Komponente durch Auswahl und anschließende Instanziierung geeigneter Komponenten aufgelöst. Die Methode verwendet den Selektor, um zu den Anforderungen eine Menge passender Komponenten zu bestimmen. Die neue Implementierung ist um ca. 20% umfangreicher als die vorherige und wesentlich komplexer - trotz oder gerade wegen der Umgestaltung des Selektors in eine Datenstruktur. Die Gründe für den Zuwachs an Komplexität liegen auf der Hand. An Stelle bindungsweise die Komponenten zu selektieren und anschließend gleich zu instanziieren wird jetzt über sämtliche Qualitätsstufen und Bindungen iteriert. Die daraus entstandene höhere Komplexität geht allerdings mit einer (subjektiv) logischeren und übersichtlicheren Codestruktur einher.

6.2 Messungen

Die folgenden Messungen sollen verschiedene Laufzeiteigenschaften der Implementierung der Anwendungsbeschreibung untersuchen. Die Implementierung wird in drei Gebieten untersucht. Das erste Gebiet ist der statisch ermittelbare Speicherbedarf von Anwendungsbeschreibungen. Hier wird die Größe der einzelnen Beschreibungseinheiten bestimmt. Das zweite Gebiet stellt eine Abschätzung der Anzahl von potentiell kostenintensiven entfernten Aufrufen auf, die durch die Einführung der Anwendungsbeschreibung zusätzlich notwendig werden. Das letzte Gebiet ist die Messung verschiedener Kriterien anhand einer Beispielanwendung. Es werden Messungen zum Speicherbedarf verschiedener Systemteile gemacht und die Laufzeit verschiedener Algorithmen ermittelt.

6.2.1 Testumgebung

Es werden zwei Systeme zum Testen verwendet. Diese sind in Tabelle 3 beschrieben. Sofern nicht anders spezifiziert, werden die Untersuchungen auf System S1 vorgenommen. Geschwindigkeitsmessungen wurden auf Basis der in Java integrierten Zeit-Funktionen durchgeführt. Die Speichergröße der Instanzen wurde an Hand eines von Martin Roth vorgeschlagenen Verfahrens[8] ermittelt.

Eigenschaft	Wert
Prozessor S1	Pentium 4, 3.00GHz, Hyperthreading
Speicher S1	1024 MByte
Prozessor S2	Pentium 3, 667MHz
Speicher S2	512 MByte
OS S1&S2	Microsoft Windows XP SP1
Java VM	Sun Java Hotspot Client VM, 1.4.2-b28
Java Bibliotheken	Java 2 Runtime Environment, Standard Edition, 1.4.2-b28

Tabelle 3 Testumgebung

6.2.2 Speicherbedarf

Der Speicherbedarf von Anwendungsbeschreibungen hängt natürlich wesentlich von der Beschreibungsumfang, d.h. der Anzahl der spezifizierten Komponenten und der Größe der angegebenen Parameter ab. Die Bestimmung der Gesamtgröße würde als Untersuchungsbeispiel eine typische Anwendungsbeschreibung erfordern, die mangels Anwendungserfahrung nicht vorliegt. Aus diesem Grund werden im Folgenden die Mindestgrößen der einzelnen Beschreibungseinheiten bestimmt. Im weiteren Verlauf der Evaluation wird die Größe einer vollständigen Anwendungsbeschreibung im Rahmen eines Beispiels untersucht.

Instanz von Klasse	Größe in Bytes
AnchorNodeDescription	320
DependenciesNodeDescription	296
GenericDemandableNodeDescription	274
InterfaceNodeDescription	296
RegularComponentNodeDescription	276
StarNodeDescription	296
StructureLevelDescription	296
ModelID	16

Tabelle 4 Mindestgröße der Beschreibungseinheiten in Bytes

6.2.3 Zusätzliche entfernte Aufrufe

Die kostenintensivsten Funktionsaufrufe in einem verteilten System sind jene über Adressraum- und Rechnergrenzen hinweg. Im Folgenden werden die durch die Bereitstellung der Anwendungsbeschreibung zusätzlich verursachten entfernten Aufrufe aufgeführt.

- *Selektion*: Für die Selektion von Komponenten muss die Anwendungsbeschreibung verteilt werden. Zu diesem Zweck lädt jeder Container, der an einem Selektionsprozess für eine Anwendung mit Anwendungsbeschreibung beteiligt ist, die Anwendungsbeschreibung genau ein mal, sofern die Anwendungsbeschreibung nicht bereits im Cache vorhanden ist. Danach ist sie im Cache des Containers abgelegt und wird hieraus wieder entfernt, wenn keine lokalen Referenzen auf die Anwendungsbeschreibung mehr existieren. Der Selektionsprozess kann bei der initialen Anwendungskonfiguration oder auch bei der Adaption angestoßen werden.
- *Updates*: Wird eine Anwendungsbeschreibung durch die Anwendung modifiziert, so wird ein Update ausgelöst. Im Rahmen dieser Aktualisierung werden alle Container, die die Anwendungsbeschreibung in ihrem Cache führen, mit einer aktuellen Version der Anwendungsbeschreibung ausgestattet. Eine Einschränkung der bestehenden Implementierung ist, dass auch Container benachrichtigt werden, die die Anwendungsbeschreibung bereits wieder aus ihrem Cache entfernt haben.

Die oben genannten Stellen sind die einzigen, bei denen definitiv zusätzliche entfernte Aufrufe entstehen. Die weiteren entfernten Aufrufe sind abhängig von der jeweiligen Implementierung des *Adaptor*, *Negotiator* und *Selector*. Bei der Selektion erzeugt der *InstantSelector* beispielsweise genauso viele entfernte Aufrufe wie der *BasicSelector* in der ursprünglichen Implementierung. Die entfernten Aufrufe, die daraufhin bei der Instanziierung anfallen, lassen sich nur sehr schwer vergleichen, da hier die Verfügbarkeit der Komponenten, die Reihenfolge, in der sich die Container gefunden haben, und noch andere Parameter eine Rolle spielen. Eine Messung, in wie weit durch das neue Selektorkonzept entfernte Aufrufe vermieden werden oder auch zusätzlich entstehen, wäre nur durch Betrachtung eines komplexen Anwendungsszenarios sinnvoll. Hier müsste man dann Mittel- und Extremwerte ermitteln, um fundierte Aussagen zur Effizienz treffen zu können. Allerdings ist es auch in dieser Situation so, dass verschiedene Selektionsstrategien für sehr unterschiedliche Szenarien optimiert werden können. An dieser Stelle wird auf Grund fehlender Anwendungserfahrung auf solche Messungen verzichtet und stattdessen in geringerem Umfang Messungen an einer kleineren Beispielanwendung vorgenommen.

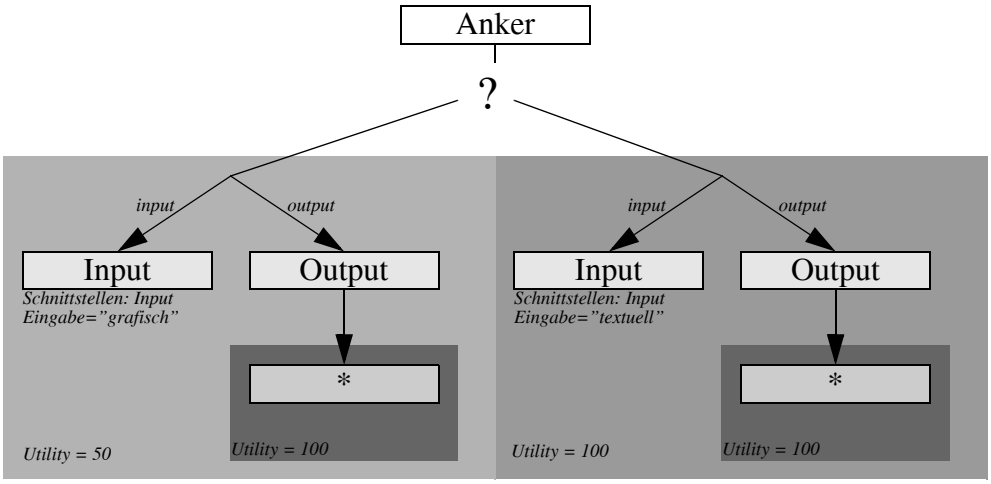


Abbildung 6.1 Grafische Anwendungsbeschreibung der Beispielanwendung

6.2.4 Messungen anhand einer Beispielanwendung

Die tatsächliche Größe einer Anwendungsbeschreibung und der Zeitaufwand zur Selektion lassen sich nur anhand einer Anwendung bestimmen. Zu diesem Zweck wurde eine kleine Beispielanwendung entwickelt. Die Anwendung ist eine verteilte Implementierung einer simplen "Echo"-Anwendung, d.h. eine getätigte Eingabe wird wieder ausgegeben. Ein- und Ausgabe werden dabei von unterschiedlichen Komponenten übernommen. Die Anwendungsbeschreibung, wie sie in Abbildung 6.1 grafisch dargestellt wird, gibt der Verteilungsinfrastruktur Hinweise auf die präferierte Zusammensetzung der Anwendung. Der Anwendungsanker muss nach der

Anwendungsbeschreibung zwei abgehende Bindungen aufweisen, und zwar eine mit “input” und eine mit “output” gelabelte Komponente. Die “Input”-Komponente muss darüber hinaus das Interface *Input* implementieren und darf keine weiteren Abhängigkeiten aufweisen. Die “Output”-Komponente darf beliebige Abhängigkeiten, entsprechend der spezifizierten Stern-Komponente, haben. Die spezifizierten Qualitätsstufen unterscheiden sich durch ein nicht-funktionales Attribut, das bei der “Input”-Schnittstelle spezifiziert wurde. Die Anwendung bevorzugt eine Komponente, die die Schnittstelle *Input* mit der nicht-funktionalen Eigenschaft “Eingabe” und deren Wert “textuell” aufweist.

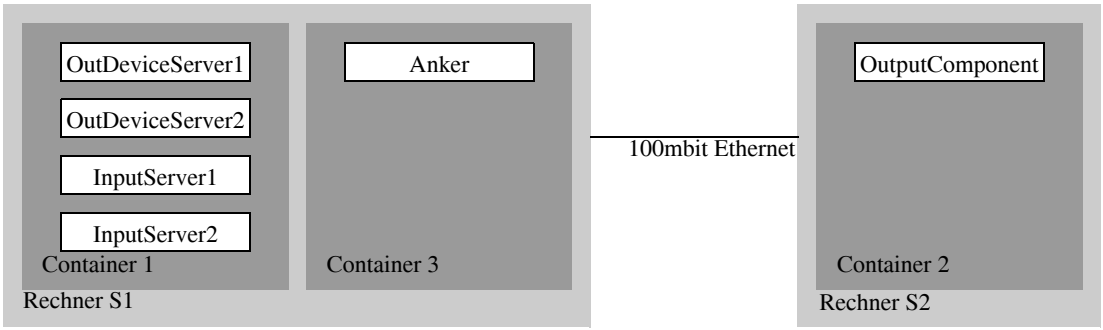


Abbildung 6.2 Containerkonfiguration

Im Folgenden wird die Anwendung je nach Test mit Anwendungsbeschreibung und ohne Anwendungsbeschreibung mit dem bisherigen PCom und dem erweiterten PCom getestet. Die Anwendung ist, sofern nicht anders angegeben, wie in Abbildung 6.2 dargestellt für die Messungen auf verschiedene Container und Rechner verteilt.

Speicherbedarf für das Beispiel. Auf Basis des Beispiels wurde der Speicherbedarf für verschiedene Komponenten ermittelt. Zum einen wurde der Speicherbedarf für die Anwendungsbeschreibung bestimmt. Sie besteht aus einem Anker, vier Abhängigkeitseinheiten, vier Qualitätsstufen, zwei nicht-funktionalen Beschreibungseinheiten, vier normalen Komponentenbeschreibungen und zwei Sternkomponenten. Die Größe sollte nach Tabelle 4 mindestens 1*320B + 4*296B +2*274B + 2*296B + 4*276B + 2*296B + 4*296B = 5524 Bytes betragen. Die gemessene tatsächliche Größe beträgt 6873 Bytes. Serialisiert (d.h. die Datenmenge, die ohne Protokolloverhead über das Netzwerk übertragen werden muss) benötigt die Beschreibung schließlich noch 4190 Bytes.

Messung	Größe in Bytes
Instanziierte Anwendungsbeschreibung	5524
Serialisierte Anwendungsbeschreibung	4190

Tabelle 5 Größe der Anwendungsbeschreibung zur Laufzeit in Bytes

Das nächste untersuchte Gebiet ist der Speicherbedarf während der Selektion. Die Messung des Speicherbedarfs bei der Selektion kann nicht statisch wie beispielsweise die Größe der Anwendungsbeschreibung bestimmt werden. Aus diesem Grund wird der Speicherbedarf des Gesamtsystems zu einem bestimmten Zeitpunkt bei der Selektion ausgegeben. Die Messungen sind dementsprechend mit Vorsicht auszuwerten - zwar wurden mehrere Messungen durchgeführt, doch können die Resultate durch den Systemzustand beeinflusst worden sein. Die gemessenen Werte (Neu mit und ohne Modell) werden mit Messungen im bisherigen PCom-System (Alt ohne Modell) verglichen. Die Speichermessung erfolgt zum Zeitpunkt der Auflösung der Abhängigkeiten des Anwendungsankers. Vor der Speichermessung wurde jeweils eine Garbage Collection durchgeführt.

Die gemessenen Werte in Tabelle 6 verdeutlichen zweierlei. Einerseits spiegelt sich im Speicherverbrauch der neuen Selektorimplementierung (*InstantSelector*) die Tatsache wieder, dass die komplette Selektion über alle Qualitätsstufen und alle Abhängigkeiten auf einmal durchgeführt und das Ergebnis zwischengespeichert wird. Die alte Implementierung selektiert nur über jeweils eine Abhängigkeit und kennt keine Qualitätsstufen. Weiterhin zeigt sich, dass die Anwendungsbeschreibung trotz der dort spezifizierten zwei Qualitätsstufen keinen Mehrverbrauch an Speicher bei der Selektion im Vergleich zum Test ohne Anwendungsbeschreibung zur Folge hat. Der Anstieg im absoluten Speicherverbrauch zwischen der neuen Implementierung mit und ohne Modell entspricht in etwa der Größe der Anwendungsbeschreibung und ist somit erklärt. Der Unterschied zwischen der alten Implementierung und der neuen ohne Modell wird durch die Systemerweiterungen in Zusammenhang mit der Anwendungsbeschreibung wie beispielsweise das *ModelRepository* bedingt. Die Implementierung der Anwendungsbeschreibung erfüllt, zumindest für dieses Beispiel, klar die Anforderungen der Ressourcenschonung in dem sie gegenüber der vorhandenen Implementierung kaum zusätzliche Speicherressourcen benötigt.

Konfiguration	vor Selektion	nach Selektion	Delta
Neu mit Modell	147	154	7
Neu ohne Modell	138	145	7
Alt ohne Modell	131	135	4

Tabelle 6 Speicherverbrauch vor und nach der Selektion in Kilobytes

Zeitmessung für die Selektion. Eine Aussage über die Effizienz der Selektion insgesamt ist schwer zu treffen, da hier der Erfolg bei der Auswahl günstiger und ungünstigerer Konfigurationen mit einbezogen werden müsste. Doch lassen sich verschiedene Teilbereiche und verwendete Algorithmen im Rahmen der Anwendungsbeschreibung und des Selektionsprozesses aus Geschwindigkeitsgesichtspunkten betrachten. Dazu wurden die im Folgenden aufgeführten Teilbereiche untersucht:

- *Selektion*: Die Zeitmessung der Selektion misst im Falle der neuen Implementierung die Zeit, die zur Instanziierung des *InstantSelector* benötigt wird. Dies schließt auf Grund der Implementierung dieser Selektionsstrategie den kompletten Selektionsvorgang mit ein. Im Falle der alten Implementierung wird für jede Abhängigkeit die Zeit für den Methodenaufruf *select()* gemessen und am Ende aufaddiert. So wird in der alten Implementierung auch der komplette Selektionsvorgang erfasst.
- *Instanziierung*: Die Kosten für die Instanziierung werden durch Messung des Zeitraumes der Selektion und der darauf folgenden Instanziierung der Komponenten bestimmt. Von diesen Kosten werden die im vorigen Punkt bestimmten Selektionskosten wieder herausgerechnet.

Die Messungen wurden mit verschiedenen, im Folgenden aufgelisteten Konfigurationen durchgeführt:

- *Neu mit Modell (1)*: Die Konfiguration entspricht der in Abbildung 6.1 abgebildeten Beschreibung. Die Verteilung der Komponenten ist entsprechend Abbildung 6.2.
- *Neu mit Modell (2)*: Als Alternative wurde hier der Nutzen (Utility) der Qualitätsstufen getauscht.
- *Neu ohne Modell*: Die Konfiguration ist ohne Anwendungsbeschreibung, entspricht aber ansonsten der ersten.
- *Alt ohne Modell*: Die vorherige Implementierung wird hier, verteilt gemäß Abbildung 6.2, untersucht.
- *Neu mit Modell (L)*: Diese Konfiguration entspricht der ersten, nur wird hier alles lokal in einem Container ausgeführt. Auf Grund der hohen lokalen Ausführungsgeschwindigkeit

wurde eine zehnfache Iteration des kompletten Selektions- und Instanziierungsvorgangs durchgeführt.

- *Alt ohne Modell (L)*: Bei dieser Messung wird die Laufzeit der vorherigen Implementierung in einem lokalen Szenario bestimmt. Auf Grund der hohen lokalen Ausführungsgeschwindigkeit wurde eine zehnfache Iteration des kompletten Selektions- und Instanziierungsvorgangs durchgeführt.

Die Messungen wurden in den entsprechenden Konfigurationen je 20 Mal durchgeführt. Die Werte in Tabelle 7 stellen die Durchschnittswerte über diese Messungen dar.

Art der Messung	Neu mit Modell (1)	Neu mit Modell (2)	Neu ohne Modell	Alt ohne Modell	Neu mit Modell (L)	Alt ohne Modell (L)
Selektion	1,250	1,230	1,232	1,163	0,060(*)	0,035(*)
Instanziierung	2,744	3,101	1,945	1,912	0,195(*)	0,179(*)

Tabelle 7 Zeitmessungen der Selektion in Sekunden (* für 10 Iterationen)

Zeitmessung für Einzelalgorithmen. Im Folgenden werden Zeitmessungen für einzelne im Bereich der Anwendungsbeschreibung verwendeten Algorithmen durchgeführt. Die gemessenen Algorithmen betreffen dabei Kosten, die zusätzlich durch die Einführung der Anwendungsbeschreibung im Vergleich zur bisherigen Implementierung entstehen. Die zusätzlichen Kosten entstehen beim Laden von Komponentenverträgen mit Referenz auf eine Anwendungsbeschreibung durch das Laden der Anwendungsbeschreibung, dem Zugriff auf Anwendungsbeschreibungen im lokalen Cache des Containers, der Suche nach Knoten in der Anwendungsbeschreibung für bestimmte Anforderungen, einer weiteren Kompatibilitätsprüfung für bestimmte Anforderungen und schließlich dem Transfer einer Anwendungsbeschreibung über Container- bzw. Adressraumgrenzen hinweg.

- *Laden der Anwendungsbeschreibung*: Die initiale Ladezeit für die Anwendungsbeschreibung vom *LoadableLoader* wird mit dieser Messung bestimmt. Der Vorgang wurde über 10000 Iterationen gemessen.
- *Lookup der Anwendungsbeschreibung*: Durch diese Messung wird die Zeit zum Auffinden einer Anwendungsbeschreibung im *ModelRepository* (dem lokalen Containercache) gemessen. Die Messung erfolgt über 10000 Zugriffe.
- *Suche in der Anwendungsbeschreibung*: Die Suche in der Anwendungsbeschreibung ist die Zeit, die benötigt wird, einen bestimmten Knoten in der Anwendungsbeschreibung an Hand einer Anforderung zu bestimmen. Die entsprechende Methode *getComponentNode()* ist auf Ebene der *StructureLevelDescription* implementiert und wird für die Messung 10000 Mal aufgerufen.
- *Kompatibilität für eine Qualitätsstufe bestimmen*: Ein Angebot wird während dieser Messung auf die Kompatibilität (Entsprechung der funktionalen Eigenschaften) zu dem entsprechenden Knoten im Anwendungsbaum geprüft. Die Messung beinhaltet auch 10000 Vergleiche.
- *Transfer der Anwendungsbeschreibung*: Der Zeitverbrauch für den Transfer der Anwendungsbeschreibung zwischen zwei Containern wurde hier bestimmt.

Die Messungen wurden, unabhängig von der Anzahl der gemessenen Durchläufe, wie auch schon bei den vorangegangenen Messungen je 20 Mal durchgeführt. Die Ergebnisse werden in Tabelle 8 präsentiert.

Art der Messung	Ergebnis
Laden der Anwendungsbeschreibung (10000 Iterationen)	3,601

Tabelle 8 Zeitmessung der Einzelalgorithmen in Sekunden

Art der Messung	Ergebnis
Lookup der Anwendungsbeschreibung (10000 Iterationen)	0,037
Suche in der Anwendungsbeschreibung (10000 Iterationen)	0,120
Kompatibilität für eine Qualitätsstufe bestimmen (10000 Iterationen)	0,374
Transfer der Anwendungsbeschreibung (1 Iteration)	0,452

Tabelle 8 Zeitmessung der Einzelalgorithmen in Sekunden

Auswertung der Zeitmessungen. Aus den Messungen ergibt sich, dass (in der verteilten Konfiguration) die neue Implementierung der Selektion mit und ohne Verwendung einer Anwendungsbeschreibung gegenüber der alten Vorgehensweise keine nennenswerten Geschwindigkeitsunterschiede aufweist. Der zusätzliche Aufwand, der durch die Prüfung der Anwendungsbeschreibung entsteht, ist, wie aus den Werten in Tabelle 8 deutlich wird, sehr gering und erst über eine sehr große Anzahl von Zugriffen im messbaren Bereich.

Die Instanziierung in der verteilten Konfiguration entspricht vom zeitlichen Aufwand auch dem Aufwand der bisherigen Implementierung. Die Konfigurationen ohne Anwendungsbeschreibung zeigen jedoch einen erheblichen Zeitvorteil zu den Konfigurationen mit Anwendungsbeschreibung. Der zusätzliche Zeitverbrauch zwischen “Neu mit Modell (1)” und “Neu ohne Modell” erklärt sich durch den notwendigen Transfer der Anwendungsbeschreibung zwischen den Containern. Addiert man den Wert aus Tabelle 8 für den Transfer der Anwendungsbeschreibung auf den Wert von “Neu ohne Modell”, so gleichen sich die Werte an. Die Unterschiede zwischen den beiden Konfigurationen mit Modell werden durch die Instanziierungsreihenfolge der Komponenten verursacht. Da in PCom nicht-funktionale Eigenschaften momentan erst bei der Instanziierung geprüft werden, wird in der zweiten Konfiguration zuerst eine “falsche” Eingabekomponente instanziiert und bei der Prüfung auf Konformität zu den nicht-funktionalen Anforderungen der Qualitätsstufe wieder verworfen.

Die Messungen in den lokalen Konfigurationen sind im Hinblick auf einen Vergleich wenig aufschlussreich, da sich der Zeitaufwand für einen Einzeldurchlauf der Selektion am Rande der Messgenauigkeit befindet. Die Unterschiede zwischen der bisherigen und der neuen Implementierung sind auf dem Testsystem nicht nennenswert. Die Beobachtung an dieser Stelle ist, dass unabhängig der PCom-Version erst verteilte Konfigurationen, und somit entfernte Aufrufe, die Selektion und Instanziierung erheblich verteuern.

Weitere Messungen im Rahmen der Beispielanwendung zeigen, dass bis auf den Transfer der Anwendungsbeschreibung alle weiteren die Anwendungsbeschreibung betreffenden Algorithmen einen sehr geringen Zeitaufwand zur Folge haben. Das Laden der Anwendungsbeschreibung ist von den vier genannten Algorithmen noch der aufwändigste, der allerdings nur einmalig pro Anwendungsbeschreibung systemweit auftritt. Der Transfer von Anwendungsbeschreibungen durch entfernte Aufrufe dagegen verursacht vergleichsweise erhebliche Kosten, die sich auch in den Ergebnissen in Tabelle 7 widerspiegeln. Die Kosten an dieser Stelle entstehen durch den Aufwand des entfernten Aufrufs, denn die serialisierte Größe einer Anwendungsbeschreibung, wie in Tabelle 5 gemessen, ist für das im Testsystem verwendete 100mBit-Netzwerk keine Belastung. Die Kosten fallen bei jedem Transfer der Anwendungsbeschreibung zwischen Containern an, d.h. für jeden Container, der an dem Selektionsprozess für eine Anwendung mit Anwendungsbeschreibung beteiligt ist. Anschließend befindet sich die Anwendungsbeschreibung im lokalen Cache des Containers, dem *ModelRepository,* und es fallen keine weiteren Transferkosten an. Aktualisierungen der Anwendungsbeschreibung verursachen dementsprechend Kosten in Höhe der Anzahl der Container, die die Anwendungsbeschreibung in ihrem Cache abgelegt haben multipliziert mit dem Aufwand für den Anwendungsbeschreibungstransfer.

6.3 Evaluation der Anforderungen

Im Rahmen der Vorstellung des Lösungsansatzes in Kapitel 3 wurden allgemeine Zielsetzungen definiert. Nach dem Entwurf einer Lösung und deren Umsetzung werden im Folgenden die zu Grunde liegenden Kriterien erneut reflektiert und die Realisierung mit den ursprünglichen Anforderungen verglichen. Die Gliederung nach funktionalen und nicht-funktionalen Anforderungen, die bei deren Aufstellung eingeführt wurde, findet auch hier Anwendung.

6.3.1 Funktionale Anforderungen

Im Folgenden wird untersucht, ob die geschaffene Lösung mit ihren Konzepten und dem damit umgesetzten Funktionsumfang den ursprünglich aufgestellten funktionalen Anforderungen genügt.

Nutzung der Adaptionsfreiräume. Die Hauptzielsetzung dieser Arbeit besteht darin, die in der ubiquitären Verteilungsinfrastruktur PCom vorhandenen Freiräume bei der Adaption durch den Anwendungsprogrammierer nutzbar zu machen. Hierbei wurden zwei Dimensionen von Freiräumen ausgemacht - Freiräume bei der Wahl von zu bestimmten Anforderungen passenden Komponenten und Freiräume bei der Wahl der Adaptionsebene.

Die Nutzung der zuerst genannten Freiräume fällt auf Grund der Struktur von PCom leichter, da die vorhandenen Strukturen in PCom bereits eine konfigurierbare Selektion vorsehen. Die Arbeit hat durch das Konzept der Anwendungsbeschreibung eine Bereitstellung von anwendungsspezifischen Informationen bei der Selektion ermöglicht. Diese Anwendungsbeschreibung wird durch den Anwendungsprogrammierer erstellt und erlaubt ihm, für die Anwendung Einschränkungen in Bezug auf die Anwendungsstruktur und in Bezug auf funktionale sowie nicht-funktionale Eigenschaften von den an der Anwendung beteiligten Komponenten zu spezifizieren. Dabei ermöglicht die Realisierung der Anwendungsbeschreibung die Nutzung aller bisher vorhandenen Spezifikationsmöglichkeiten für Komponenten in PCom.

Zu einer aus Sicht des Anwendungsprogrammierers zweckmäßigeren und vor allem eindeutigen Identifikation von Komponenten in einer Anwendung wurde ein neues Konzept - das der semantischen Labels - eingeführt. Das Konzept ermöglicht die eindeutige Identifikation von Komponenten anhand ihrer Verwendungsart, und somit auf Grundlage der Bedeutung, die die Komponente für die Anwendung aus Sicht des Anwendungsprogrammierers aufweist.

Um funktionale und nicht-funktionale Anforderungen an eine Komponente spezifizieren zu können, muss jedoch die Position der Komponente im Anwendungsbaum dem Anwendungsprogrammierer bekannt sein. Die Frage, ob eine freie Spezifikation von Einschränkungen für alle Komponenten, die ein bestimmtes Interface implementieren oder an einer Bindung mit einem bestimmten semantischen Label beteiligt sind, sinnvoll ist, wird in dieser Arbeit verneint, da die Bedeutung einer solchen Einschränkung momentan auf Grund von möglichen Mehrdeutigkeiten nicht geklärt werden kann. Dies könnte sich allerdings durch eine geeignete Interface- oder Labelontologie ändern.

Dem Anwendungsprogrammierer wird auch die Spezifikation von Alternativen bezüglich der Anwendungsbeschreibung ermöglicht. Die Alternativen sind auf die strukturelle Ebene beschränkt, doch lassen sich durch dieses Konzept für alle drei Spezifikationskategorien Alternativenformen definieren. Der dadurch entstehende Mehraufwand bei der Spezifikation durch den Anwendungsprogrammierer wird zu Gunsten einer einfacheren Auswertung in Kauf genommen.

Knotenübergreifende Adaption. Die Adaption in PCom beschränkte sich bisher auf eine knotenlokale Betrachtung, es wurden Anforderungen und Angebote jeweils auf der Komponente-Komponente-Bindungsebene verglichen. Die Anwendungsbeschreibung bietet eine knotenübergreifende Sicht auf die Anwendung, da in ihr zumindest ein Teil des Anwendungsbaums spezifiziert ist. Eine knotenübergreifende Sicht ermöglicht neue Wege bei der Adaption. An Stelle der bisher vorhandenen Strategie, Anwendungen möglichst lokal zu adaptieren, können jetzt auch

neue Strategien, wie beispielsweise durch den *QualityAdaptor* umgesetzt werden, der versucht, eine aus Sicht der gesamten Anwendung möglichst günstige Konfiguration zu erzielen.

Die knotenübergreifende Sicht steht auch allen an der Anwendung beteiligten Komponenten und Mechanismen zur Konfiguration zur Verfügung. Dementsprechend bietet sich die neue anwendungsspezifische Informationsquelle auch für die Nutzung in anderen Algorithmen an. Beispielsweise kann eine knotenübergreifende Betrachtung bei den Vertragsverhandlungen im *Negotiator* auch von Nutzen sein.

Neue Möglichkeiten der Anwendungsentwicklung. Durch die Anwendungsbeschreibung wird dem Anwendungsentwickler eine Möglichkeit geboten, mehr Einfluss auf die Auswahl der Komponenten, die sich zu einer Anwendung zusammenschließen, zu nehmen. Dies ermöglicht komplexere Anwendungsstrukturen, in denen der Anwendungsprogrammierer trotzdem hinreichend Annahmen bezüglich der verwendeten Komponenten treffen kann. Mittels Einflussnahme durch die Anwendungsbeschreibung könnte ein Anwendungsentwickler eine Präsentationsanwendung (siehe Beispiele zum Präsentationsszenario) mit mehreren Bindungsstufen erstellen und dabei sicherstellen, dass - obwohl keine direkte Bindung zum Anwendungsanker vorhanden ist - ein für ihn günstiges Ausgabegerät, beispielsweise ein Projektor, auf tieferer Ebene gewählt wird.

6.3.2 Nicht-funktionale Anforderungen

Die wesentlichen nicht-funktionalen Anforderungen, die in Kapitel 3 aufgestellt wurden, werden an dieser Stelle mit den Eigenschaften der Konzepte und deren Umsetzung in PCom verglichen und bewertet.

Berücksichtigung von Dynamik. Die Anwendungsbeschreibung beeinflusst die Mechanismen für Dynamik in PCom. Die Implementierung der Anwendungsbeschreibung muss aber selber auch den Anforderungen der Dynamik gerecht werden. Die automatische Verteilung der Anwendungsbeschreibung zwischen den an einer Anwendung beteiligten Containern ist gewährleistet. Im Objektmodell der Anwendungsbeschreibung wird dem Anwendungsentwickler eine Schnittstelle geboten, die eine Anpassung der Anwendungsbeschreibung zur Laufzeit zulässt. In PCom wurde ein Update-Mechanismus integriert, der alle gecachten Anwendungsbeschreibungen automatisch im Falle von Änderungen aktualisiert. Weiterhin ist es Komponenten möglich, einen *ApplicationModelListener* auf die Anwendungsbeschreibung zu setzten und so von Änderungen in Kenntnis gesetzt zu werden. Nicht implementiert wurde ein System zur automatischen Anpassung der Bindungen, falls diese nach Änderung der Anwendungsbeschreibung die Anforderungen nicht mehr erfüllen.

Die Fehler, die im Rahmen der entfernten Aufrufe zur Übertragung und Aktualisierung von Anwendungsbeschreibungen auftreten können werden durch bereits in PCom vorhandene Mechanismen behandelt.

Effizienzgewinn. Die Effizienz der Komponentenauswahl lässt sich in zwei Dimensionen steigern. Die erste Dimension - Effizienzsteigerung durch Auswahl besonders geeigneter Komponenten für eine Anwendung - kann durch das Instrument der Anwendungsbeschreibung realisiert werden. Dem Anwendungsprogrammierer ist es nun möglich, Einfluss auf die Komponentenauswahl auf tieferen Bindungsebenen zu nehmen und auch verschiedene Qualitätsstufen für die Anwendung zu definieren. Eine Einschränkung bleibt allerdings bestehen. Ohne eine Ontologie für semantische Labels und ein durchgehendes Konzept für nicht-funktionale Anforderungen wird das Konzept der Anwendungsbeschreibung zu keinen Erfolgen führen können. Ohne eine Festlegung von Standards ist zu viel implizites Wissen zwischen Anwendungs- und Komponentenprogrammierern notwendig, um eine wirkliche Effizienzsteigerung zu erzielen.

Die zweite Dimension - Effizienzsteigerung durch Beschleunigung des Selektions- und Instanziierungsvorgangs - kann schwer beurteilt werden. Die in Abschnitt 6.2 durchgeführten Messungen legen nahe, dass sich die Performance in diesen Bereichen durch Hinzunahme der Anwendungsbeschreibung nicht verschlechtert hat. Allerdings wurde nur der Zeitaufwand der Algorithmen un-

ter gleichen Bedingungen geprüft und die neuen Optimierungsmöglichkeiten nicht berücksichtigt. Eine Untersuchung mit beispielsweise einer optimierten Version des *LazyRemoteContainerSelectors* und einer entsprechenden Beispielanwendungen könnte eine solche Effizienzsteigerung belegen oder auch verneinen, würde jedoch den Umfang dieser Arbeit überschreiten, da die Gewichtigkeit der Aussagen über die Messung von der Realitätsnähe der Beispielanwendung abhängt.

Erweiterbarkeit. Die Erstellung des Objektmodells sowie dessen Integration in PCom wurden im Hinblick auf mögliche Erweiterungen im System vorgenommen. Ein Bereich für Erweiterungen ist das Konzept der nicht-funktionalen Anforderungen. Die Umsetzung der Beschreibungseinheiten für nicht-funktionale Eigenschaften als Interface-Definition erlaubt verschiedene Implementierungen, die ohne größeren Aufwand Erweiterungen am Konzept der nicht-funktionalen Eigenschaften zulassen. Als eine Einschränkung der Erweiterbarkeit könnte sich herausstellen, dass die Komponenten und die Anwendungsbeschreibung einen ähnlichen, aber nicht den gleichen Code benutzen und somit Änderungen sowohl an den Komponentenverträgen wie auch an den entsprechenden Klassen in der Anwendungsbeschreibung vorgenommen werden müssen.

Eine Verbesserung der Erweiterbarkeit ist durch die Einführung des neuen Selektors als Datenstruktur erreicht worden. Das Konzept ermöglicht eine Vielzahl von neuen Strategien, den Selektionsprozess zu optimieren. Gleiches gilt für den Ablauf der Adaption im Fehlerfall. Das hier implementierte Strategy-Pattern bietet eine bessere Abstraktionsstufe und somit Möglichkeiten, die Verteilungsinfrastruktur zu erweitern. Der Verhandlungsprozess in PCom wurde in seiner neuen Form vom Selektor getrennt. Die Trennung erhöht zum einen die Flexibilität, verschiedene Selektor- und Verhandlungsstrategiekombinationen einzusetzen und zum anderen auch die Erweiterbarkeit, da beide Strategien unabhängig voneinander implementiert werden können.

Die Berücksichtigung weiterer Informationsquellen bei der Selektion, wie beispielsweise systemspezifische Informationen, die von einer Ressourcenverwaltung bereitgestellt werden, lassen sich durch eine spezielle Implementierung der *Selector*-Schnittstelle realisieren. Weitere Änderungen in durch die Anwendungsbeschreibung berührten Bereichen der Verteilungsinfrastruktur sind nicht von Nöten.

Möglichst geringer Entwicklungsaufwand. Die Einführung der Anwendungsbeschreibung und der entsprechenden Auswertungsmechanismen in die Verteilungsinfrastruktur PCom sollte den Anforderungen nach keinen Mehraufwand für Komponenten- und Anwendungsentwickler bedeuten. Der einzig zwingend notwendige Mehraufwand entsteht durch die Spezifikation der semantischen Labels für Bindungen, ansonsten ist auf Grund der existierenden Abwärtskompatibilität kein zusätzlicher Aufwand erforderlich.

Die neuen Möglichkeiten der Anwendungsbeschreibung zielen im Gegenteil auf eine Erleichterung bei der Anwendungsentwicklung. Der Anwendungsprogrammierer kann auf Basis der Beschreibungsmöglichkeit Annahmen in Bezug auf die Struktur und die Eigenschaften von Komponenten machen, die vorher in der Form nicht möglich waren. Eine Evaluation dieses Kriteriums würde allerdings den Rahmen dieser Arbeit überschreiten.

Die bestehenden Entwicklungsparadigmen von PCom wurden um das Konzept der Anwendungsbeschreibung erweitert. Die Erweiterung ist ein optionales Konzept, das dem Anwendungsprogrammierer bei Bedarf die Entwicklung von Anwendungen vereinfachen kann. Wird sie nicht benötigt, so entsteht für den Anwendungsprogrammierer im Vergleich zum bisherigen Entwicklungsaufwand kein Mehraufwand. Gleiches gilt für den Komponentenentwickler.

Integration in die Verteilungsinfrastruktur. Die Integration der Anwendungsbeschreibung in die Verteilungsinfrastruktur ist nahtlos erfolgt. Die verwendeten Mechanismen sind - bis auf die Einführung semantischer Labels - vollkommen transparent für die Komponenten- und Anwendungsentwickler. Die Implementierung der Anwendungsbeschreibung orientiert sich in vielen Bereichen an bereits vorhandenen Strukturen in PCom, so dass auch keine grundlegend neuen Programmstrukturen in der Verteilungsinfrastruktur geschaffen wurden.

Verschiedene, wesentliche Designkriterien von PCom wurden für den Entwurf und für die Implementierung der Anwendungsbeschreibung auch mit einbezogen. Ubiquitäre Systeme fordern die Unterstützung leistungsschwächerer Geräte. Folglich sollte Software für solche Geräte nur die notwendigen Funktionen bereitstellen und diese möglichst ressourcenschonend umsetzen. Die Anwendungsbeschreibung setzt dieses Designkriterium an verschiedenen Stellen um. Die Orientierung an bereits vorhandener Funktionalität bei gleichzeitiger Beschränkung auf ein sinnvolles Mindestmaß an Funktionalität hat zu dem Umfangszuwachs von 25% mehr Code geführt. Der Codeumfang dient natürlich nur als Anhaltspunkt für den wirklichen Ressourcenbedarf zur Laufzeit. Die Messungen im vorangegangenen Abschnitt zeigen jedoch, dass die Resultate nicht gegen die Anforderung sprechen. Der Speicherbedarf steigt in Maßen auf Grund der Anwendungsbeschreibung um ca. 12% an. Der zusätzliche Ressourcenbedarf an Rechenleistung und Netzwerkbandbreite ist auf den untersuchten Plattformen nicht im Bereich des Messbaren. Die für diese konkrete Implementierung verwendeten Konzepte bzw. Designpattern sind im Bereich der Selektion und Adaption so flexibel ausgelegt, dass sich optimierte Implementierungen - gerade bezüglich des Ressourcenverbrauchs - umsetzen lassen. Als Beispiel ist der in Bezug auf entfernte Zugriffe und Speicherverbrauch optimierte *LazyRemoteContainerSelector* anzuführen.

Die Portabilität der Implementierung der Anwendungsbeschreibung wird durch Java gewährleistet und berücksichtigt, wie auch die vorherige Implementierung, den CDLC-Standard[6]. Dies stellt die Lauffähigkeit auf Kleinstgeräten, die nur diesen Standard unterstützen, sicher.

Wahrung der Kompatibilität. Die Einführung der Anwendungsbeschreibung sollte unter Erhaltung der Kompatibilität zu bisherigen Anwendungen erfolgen. Dieses Ziel ist mit einer Einschränkung erreicht worden. Die einzige Änderung, denen sich sowohl Komponenten- wie auch Anwendungsentwickler unterwerfen müssen ist die, dass sie für jede Abhängigkeit einer Komponente ein semantisches Label vergeben müssen. Der Generator, der XML-basierte Komponentenbeschreibungen in entsprechende Verträge umwandelt, wurde an diese Neuerung angepasst, so dass nur ein zusätzliches Attribut pro Bindung spezifiziert werden muss.

Im Rahmen des Entwurfs wurde versucht, die Auswirkungen auf andere Erweiterungen von PCom so gering wie möglich zu halten. Dieses Ziel konnte nur teilweise erreicht werden - eine Anpassung von 25% aller im System vorhandenen Schnittstellen stellt einen wesentlichen Eingriff dar. Dagegen sind die Änderungen an der bestehenden Implementierung auf sieben Klassen beschränkt, und dort auch nur in die Adaption betreffenden Bereichen. Diese Änderungen sollten für andere Erweiterungen keine Integrationshürde darstellen. Die prinzipiellen Abläufe in PCom wurden bis auf den Selektions- und Adaptionsvorgang nicht geändert, d.h. grundlegende Mechanismen wie beispielsweise der Lebenszyklus von Komponenten wurden nicht angepasst.

Robustheit. Die Robustheit bestehender Anwendungen in PCom wird durch die Anwendungsbeschreibung nicht gefährdet. Anwendungen können sowohl mit wie auch ohne Anwendungsbeschreibung ausgeführt werden und Anwendungen ohne Anwendungsbeschreibung werden von der neuen Implementierung nicht berührt.

Ist für eine Anwendung eine Anwendungsbeschreibung definiert, so wird diese in den beteiligten Containern gespeichert und bleibt während der Laufzeit der Anwendung immer verfügbar. Wird eine Anwendungsbeschreibung durch den Anwendungsanker angepasst, kann der Fall eintreten, dass bis zur Aktualisierung aller beteiligten Container die alte Anwendungsbeschreibung beispielsweise zur Komponentenselektion verwendet wird. In diesem Fall wird gegebenenfalls für eine kurze Zeit eine für die Anwendung ungünstigere Komponente ausgewählt, doch da weiterhin auch die Komponentenverträge erfüllt werden, ist die Selektion einer inkompatiblen Komponente nicht möglich.

Zusätzliche Fehler können nur bei durch die Anwendungsbeschreibung verursachten entfernten Aufrufen auftreten. Die Implementierung der Anwendungsbeschreibung verwendet die standardmäßig in PCom vorhandenen Möglichkeiten zur Fehlerbehandlung im Falle von Übertragungsfehlern bei dem Transfer von Anwendungsbeschreibungen.

Eine Einschränkung in Bezug auf die Robustheit ist die fehlende Konsistenzprüfung zwischen der Anwendungsbeschreibung und verfügbaren Komponentenverträgen. Durch Spezifikation einer Anwendungsbeschreibung kann der Anwendungsprogrammierer mit ungeschickt oder widersprüchlich gewählten Eigenschaften eine Anwendungsinstanziierung verhindern.

6.4 Evaluierung der Szenarien

Im Entwurf der Anwendungsbeschreibung sollten die Anforderungen, die sich aus den Anwendungsszenarien ableiten lassen, reflektiert werden. In wie weit die neu geschaffenen Mechanismen dem Anwendungsprogrammierer Vorteile in Bezug auf die Entwicklung der Beispielanwendungen ermöglichen können, wird in den folgenden Abschnitten untersucht.

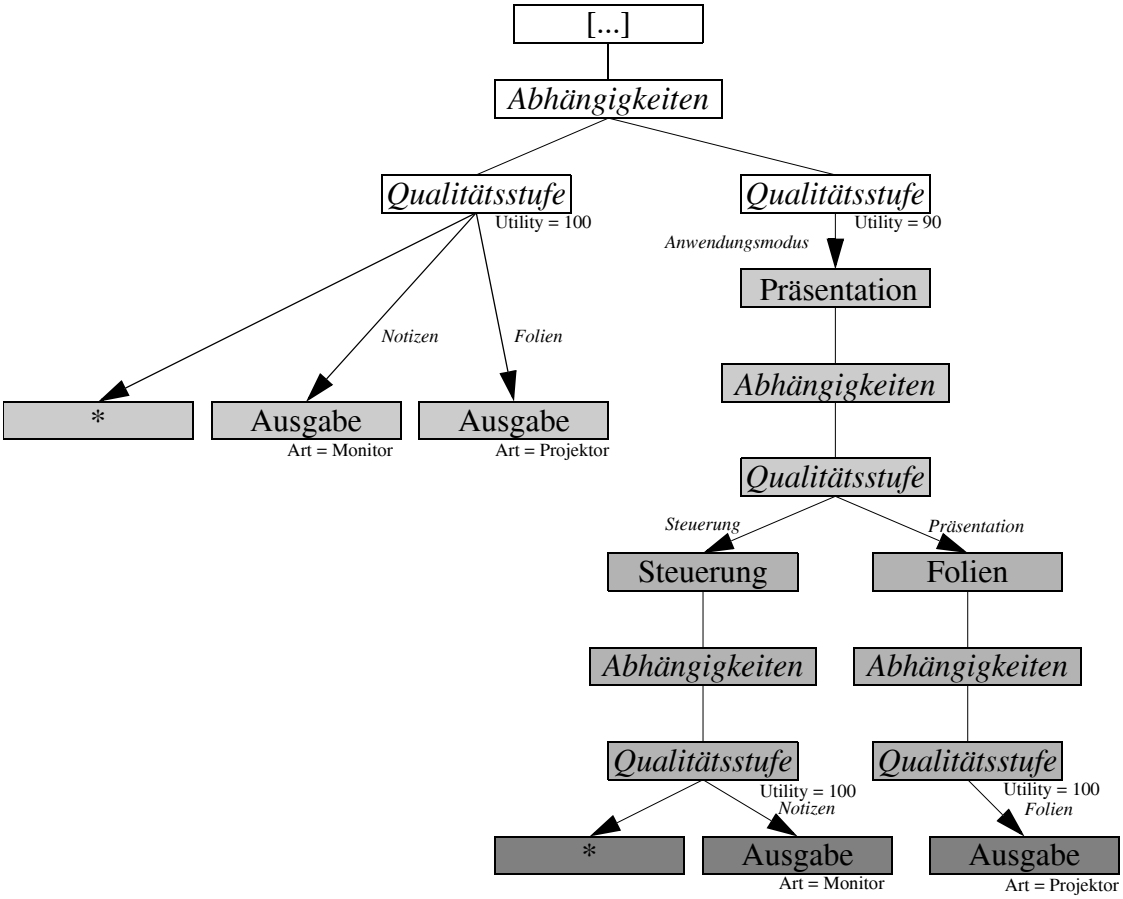


Abbildung 6.3 Grafisches Beispiel einer Anwendungsbeschreibung (Ausschnitt)

6.4.1 Präsentationsszenario

Im Präsentationsszenario wurde die Problematik aufgeworfen, dass eine Anwendung verschiedene Ausgabegeräte mit gleicher Schnittstelle aber unterschiedlicher Bedeutung für die Anwendung benutzt. Die in Absatz 3.3.1 genannten Möglichkeiten, dieses Problem zu lösen, hätten wesentliche Nachteile für den Anwendungsprogrammierer impliziert. In wie weit das Konzept der Anwendungsbeschreibung für die genannte Problemstellung eine hinreichende Lösung darstellen kann wird im Folgenden diskutiert.

Die Ursache für die genannten Umsetzungsprobleme in diesem Szenario ist die nicht vorhandene Möglichkeit der Einflussnahme des Anwendungsprogrammierers auf tiefere Bindungsebenen im Anwendungsbaum. Durch die Anwendungsbeschreibung wird eine solche Möglichkeit geschaf-

fen - mit der Einschränkung, dass dem Anwendungsprogrammierer die Struktur bis zur spezifizierten Bindung oder Komponente bekannt sein muss. Sind beispielsweise die genannten Möglichkeiten aus Abbildung 3.4 Strukturen, die die Anwendung annehmen kann, so kann der Anwendungsprogrammierer durch eine Anwendungsbeschreibung geeignete Einschränkung für eine Auswahl von günstigen Komponenten definieren. Abbildung 6.3 zeigt eine beispielhafte grafische Repräsentation einer Anwendungsbeschreibung, die im Präsentationsszenario Anwendung finden könnte.

Der Anwendungsprogrammierer kann an den Ausgabekomponenten nun die nicht-funktionalen Eigenschaften, die für die Auswahl der Komponenten relevant sind, definieren. Das Beispiel in Abbildung 6.3 zeigt den spezifizierten Anwendungsbaum, in dem der Anwendungsprogrammierer die Struktur der Anwendung bis auf Ebene der Ausgabekomponenten beschreibt. Die in dieser Struktur beschriebenen Komponenten können, müssen aber keine einschränkenden Eigenschaften aufweisen. Es ist nur notwendig, ein semantisches Label pro Bindung zu spezifizieren. Die Stern-Komponente setzt der Anwendungsprogrammierer dabei ein, um auszudrücken, dass er die weitere Struktur an dieser Stelle nicht einschränken möchte. An den Ausgabekomponenten beschreibt er schließlich die von ihm gewünschte Einschränkung für die Geräteauswahl - unter Annahme der Existenz einer gemeinsamen Ontologie für nicht-funktionale Anforderungen und semantische Labels. In dem gegebenen Szenario wäre die Spezifikation von weiteren Alternativen denkbar, beispielsweise für den Fall, dass kein Projektor zur Verfügung steht.

Eine für dieses Szenario sinnvoll scheinende Möglichkeit, die Definition von “wenn-dann”-Bedingungen (Beispiel: Wenn Typ=”Projektor”, dann Bindung an “Folien”-Ausgabe), ist im Rahmen der Anwendungsbeschreibung nicht vorgesehen, kann jedoch durch die Spezifikation von Alternativen nachgebildet werden.

Der Ablauf des Beispielszenarios sieht verschiedene Orte und Anwendungsmodi (Entwurf, Präsentation, etc.) vor. Die Anpassbarkeit der Anwendungsbeschreibung zur Laufzeit ermöglicht es, die daraus entstehenden Anforderungen umzusetzen. Wechselt der Anwender den Anwendungsmodus, so kann die Anwendung die Anwendungsbeschreibung dementsprechend anpassen bzw. gegen eine andere, vorher vom Anwendungsprogrammierer erstellte Beschreibung, austauschen. Dieser Mechanismus gewährleistet die erforderliche Flexibilität in Bezug auf die sich ändernden Anforderungen der Anwendung. Die Spezifikation struktureller Alternativen sorgt für weitere Flexibilität, in diesem Fall bezüglich der verfügbaren Geräte.

Im Szenario wird die automatische Verwendung der im Auto des Nutzers vorhandenen Hardware beschrieben. Dies ist nur unter Einschränkung in PCom realisierbar. Es ist momentan kein Mechanismus zur Rekonfiguration einer Anwendung auf Grund “positiver” Änderung der Umgebung, d.h. wenn beispielsweise neue Komponenten zur Verfügung stehen, implementiert.

6.4.2 E-Mail-Szenario

Das E-Mail-Szenario unterscheidet sich vom Präsentationsszenario durch eine gleich bleibende Anwendungsfunktion bei einer sich stärker ändernden Umgebung. Die Umsetzung dieses Szenarios in der ursprünglichen Implementierung von PCom hätte wieder die Problematik der Einflussnahme auf tiefere Bindungsebenen hervorgebracht. Diese Fähigkeit ist durch die Anwendungsbeschreibung in speziell für diese Anwendung hinreichendem Umfang gegeben. Der Anwendungsprogrammierer kann unter Kenntnis der Anwendungsstruktur bis zu der für die Kommunikation verantwortlichen Komponente diese spezifizieren und dann durch priorisierte Alternativen mit unterschiedlichen nicht-funktionalen Eigenschaften eine Wahl von günstigen Netzwerkverbindungen erreichen. Die Nutzung verschiedener Ein- und Ausgabegeräte kann auf die gleiche Art und Weise durch Spezifikation nicht-funktionaler Eigenschaften an den entsprechenden Komponenten und Beschreibung struktureller Alternativen bewerkstelligt werden.

Die Einschränkungen für dieses Szenario entsprechen auch den im ersten genannten. Einerseits besitzt PCom keinen standardisierten Mechanismus, Anwendungen über neu verfügbare Komponenten in Kenntnis zu setzen und andererseits existieren keine Möglichkeiten zur Spezifikation

von “wenn-dann”-Eigenschaften für Komponenten. Diese können allerdings (mit geringfügig höherem Aufwand) durch strukturelle Alternativen ersetzt werden.

Kapitel 7

Zusammenfassung und Ausblick

Dieses Kapitel wird in kompakter Weise noch einmal die anfangs genannten Ziele und die zu deren Erreichung verwendeten Lösungsansätze gegenüberstellen. Möglichkeiten der Weiterführung und Ergänzungen der in dieser Arbeit vorgestellten Konzepte werden im Ausblick vorgestellt. Den Abschluss der Arbeit bildet eine Zusammenfassung der durch die Entwicklung der Anwendungsbeschreibung gewonnenen Schlussfolgerungen.

7.1 Zusammenfassung

Die Realisierung von ubiquitären Systemen erfordert Verteilungsinfrastrukturen, die die Entwicklung von Anwendungen für solche Systeme vereinfachen bzw. erst ermöglichen. Verschiedene Arten von Verteilungsinfrastrukturen wurden im Rahmen dieser Arbeit im Hinblick auf die Anwendungsarchitektur und die Form der Unterstützung von Anwendungsentwicklern untersucht. Allen gemein ist immer noch eine hohe Komplexität bei der Anwendungsentwicklung auf Grund der ubiquitären Prinzipien wie beispielsweise Mobilität und Dynamik. In komponentenbasierten Verteilungsinfrastrukturen besteht eine Herausforderung in der für eine Anwendung vorteilhaften Zusammenstellung ihrer benötigten Komponenten und in deren Adaption in Problemsituationen. Die Unterstützung für die automatische Konfiguration von Anwendungen durch die Verteilungsinfrastruktur ist nicht in allen untersuchten Systemen vorhanden. One.World basiert auf dem Prinzip, dem Anwendungsprogrammierer lediglich die Informationen über Änderungen im System bereitzustellen und eine anwendungsspezifische Implementierung für die Konfiguration vorzusehen. Die anderen Verteilungsinfrastrukturen bieten den Anwendungs- und Komponentenprogrammierern Unterstützung in Form von Mechanismen zur automatischen Anwendungskonfiguration. In der Verteilungsinfrastruktur zentralisierte Mechanismen zur Konfiguration bzw. Adaption von Anwendungen sollten für die Anwendung möglichst günstige Entscheidungen treffen. Die hierbei vorhandenen Freiheitsgrade möglichst sinnvoll zu füllen stellt die Verteilungsinfrastruktur vor eine große Herausforderung.

Um eine möglichst vorteilhafte Nutzung der Freiheitsgrade zu erreichen, stehen der Verteilungsinfrastruktur prinzipiell zwei Arten von Informationsquellen zur Verfügung. Dies sind systemspezifische Informationsquellen und anwendungsspezifische Informationenquellen. Die Untersuchung der verschiedenen Verteilungsinfrastrukturen hat gezeigt, dass keines der Systeme über Mechanismen verfügt, mit denen der Anwendungsprogrammierer der Verteilungsinfrastruktur Informationen zur Selektion von Komponenten für die gesamte Anwendung bereitstellen kann, die über rein syntaktische Informationen hinausgehen und mehr als eine Bindungsebene betreffen.

Zielsetzung dieser Arbeit war die Schaffung eines Mechanismus, der es dem Anwendungsprogrammierer ermöglicht, die vorhandenen Freiheitsgrade zu nutzen. Die Bereitstellung einer Anwendungsbeschreibung erschien nach Analyse der Anforderungen dabei als ein geeignetes Mittel, um die vielfältigen Anforderungen zu erfüllen. Durch die Anwendungsbeschreibung wird es dem Anwendungsprogrammierer ermöglicht, seine Präferenzen in Bezug auf die Struktur und die funktionalen und nicht-funktionalen Eigenschaften einer Anwendung auszudrücken und diese so der Verteilungsinfrastruktur als anwendungsspezifische Informationsquelle bei der Konfiguration zur Verfügung zu stellen.

Der Umfang der Beschreibungssprache hat sich zwangsläufig als stark von der verwendeten Verteilungsinfrastruktur abhängig herausgestellt. Ohne Erweiterungen an der Verteilungsinfrastruktur vorzunehmen ist der Beschreibungsumfang auf die vorhandenen Möglichkeiten der Verteilungsinfrastruktur beschränkt. Als strukturelles Beschreibungsmittel werden in PCom in den Komponentenverträgen die Abhängigkeiten der Komponenten spezifiziert. Für die Beschreibung der funktionalen Eigenschaften werden in PCom die Ereignisse, die eine Komponente unterstützen soll sowie die Schnittstellen, die sie implementieren soll spezifiziert. Die nicht-funktionalen Eigenschaften können in PCom auf Komponentenebene wie auch auf Ereignis- und Schnittstellenebene spezifiziert werden. Der hier gebotene Sprachumfang wurde in die Anwendungsbeschreibung übernommen.

Für die Spezifikation der strukturellen Eigenschaften einer Anwendung wurde das bisherige Bindungskonzept durch die Einführung eines semantischen Labels zur Beschreibung der Bedeutung einer abgehenden Bindung eingeführt. Dies ermöglicht dem Anwendungsprogrammierer, eine Komponente im Anwendungsbaum anhand des semantischen Labels an Stelle des implementierten Interfaces zu identifizieren.

Die weitere Spezifikation der strukturellen Eigenschaften ist durch die Notwendigkeit der eindeutigen Identifizierbarkeit von Komponenten im Anwendungsbaum gekennzeichnet. Der Anwendungsprogrammierer muss vom Anwendungsanker aus alle Komponenten spezifizieren, bis zu den Blättern des Anwendungsbaums oder bis zu Aggregationsbeschreibungseinheiten, mit denen der Anwendungsprogrammierer ausdrückt, dass ab hier beliebige Komponenten und Strukturen folgen dürfen. Nur auf diese Weise kann die eindeutige Zuordnung der spezifizierten Eigenschaften zu den entsprechenden Komponenten gewährleistet werden.

Der Dynamik von ubiquitären Systemen wird in zweierlei Hinsicht Rechnung getragen. Einerseits wird dem Anwendungsprogrammierer die Spezifikation von strukturellen Alternativen ermöglicht. Diese Beschreibungsform erlaubt die Spezifikation von Anwendungsbeschreibungen für verschiedene Umgebungen mit der Angabe von Präferenzen, so dass eine aus Anwendungssicht möglichst gute Konfiguration gefunden werden kann. Andererseits erlauben es Mechanismen, die Anwendungsbeschreibung zur Laufzeit anzupassen, und so Umgebungsänderungen oder Bedarfsänderungen der Anwendung zu reflektieren.

Ein Hauptentwurfsziel, die Erweiterbarkeit, wurde sowohl für die verwendeten Konzepte als auch für die konkrete Implementierung umgesetzt. Die Anwendungsbeschreibung ist durch die Verwendung von XML offen für neue Beschreibungskonstrukte. Der sich an der Vertragsstruktur von PCom orientierende Aufbau ermöglicht eine analoge Anpassung und Erweiterung beider Verteilungsinfrastrukturbestandteile. Die im Bereich der Adaption, Selektion und Vertragsverhandlungen verwendeten Strategy-Patterns erlauben auch ohne Modifizierung der Verteilungsinfrastruktur die Implementierung neuer Strategien.

Die Auswertung der neuen Informationsquelle Anwendungsbeschreibung hat ein neues Design der Adaptions-, Selektions- und Vertragsverhandlungsprozesse in PCom bedingt. Die neu geschaffenen Strukturen wurden durch Verwendung des Strategy-Pattern erweiterbar gestaltet und gewährleisten zudem die Kompatibilität zu bestehenden PCom-Anwendungen. Die Umwandlung des Selektors in eine Datenstruktur hat darüber hinaus neue Optimierungsmöglichkeiten geschaffen, die ansatzweise in der Implementierung des *LazyRemoteContainer*-Selektors vorgestellt werden.

Die Verwertung der Informationsquelle Anwendungsbeschreibung zur Nutzung der vorhandenen Freiheitsgrade wurde anhand je zweier neuer Strategien für die Selektion von Komponenten und Adaption von Anwendungen demonstriert. Dabei hat sich herausgestellt, dass durch die Anwendungsbeschreibung eine knotenübergreifende Sicht auf die Anwendung ermöglicht wird, wie beispielsweise die Adaptionsstrategie des *QualityAdaptors* zeigt.

7.2 Ausblick

Die Bereitstellung der Anwendungsbeschreibung in PCom legt verschiedene Erweiterungen nahe, die den Nutzwert der Anwendungsbeschreibung und somit auch den der Verteilungsinfrastruktur steigern könnten.

Ein wesentlicher Punkt für zukünftige Erweiterungen ist die Schaffung einer Ontologie für semantische Labels und nicht-funktionale Eigenschaften. Nur ein gemeinsam von Anwendungsprogrammierern und Komponentenprogrammierern genutztes Vokabular kann die Nutzung dieses Spezifikationsinstruments ermöglichen. In diesem Zusammenhang bieten sich auch eine Untersuchung der nicht-funktionalen Eigenschaften und gegebenenfalls deren Erweiterung um weitere Spezifikationsmöglichkeiten wie Intervalle und Güterichtungen innerhalb der Intervalle an.

Ein zweiter wesentlicher Komplex ist der Entwurf, die Implementierung und anschließend die vergleichende Untersuchung verschiedener Adaptions-, Selektions- und Verhandlungsstrategien. Die Strategien können Vorteile aus den zusätzlichen knotenübergreifenden, anwendungsspezifischen Informationen ziehen, die die Anwendungsbeschreibung bereitstellt. Zudem können durch die neue Struktur des Selektors im Hinblick auf Speicherverbrauch und Zeitaufwand Optimierungen vorgenommen werden, wie dies bereits ansatzweise im *LazyRemoteContainerSelector* gezeigt wurde.

Eine Erleichterung für den Anwendungsprogrammierer und gleichzeitig eine Steigerung der Robustheit könnte durch eine Form der semantischen Prüfung von Anwendungsbeschreibungen erreicht werden, die Widersprüche zu vorhandenen Komponentenverträgen oder nicht instanziierbare Konstellationen aufdeckt. Bisher wird die Anwendungsbeschreibung nur syntaktisch auf Erfüllung des XML-Schemas (siehe Abbildung 5.1) geprüft. Vollständige Prüfungen werden sich jedoch wahrscheinlich nicht realisieren lassen, da die Informationen über die im System vorhandenen Komponenten erst zur Vertragsverhandlungs- bzw. Instanziierungszeit verfügbar sind.

Sprachliche Erweiterungen der Anwendungsbeschreibung stellen auch ein möglicherweise lohnendes Projekt dar. Bis jetzt sind die spezifizierten Anforderungen in einer Anwendungsbeschreibung durch feste Attribute realisiert. Im Verlauf der Untersuchung der Anwendungsszenarien wurde die Möglichkeit von “wenn-dann”-Beziehungen genannt. Im Rahmen der Anwendungsbeschreibung könnten solche Konstrukte in einer ersten Ausbaustufe zu einer Aggregation von verschiedenen Alternativen genutzt werden, die bis jetzt noch durch getrennte Strukturalternativen spezifiziert werden müssten. Eine zweite Ausbaustufe wäre eine Untersuchung, in wie weit Querverweise zwischen Knoten der Anwendungsbeschreibung möglich sind. Ein Beispiel hierfür wäre “Wenn die Netzwerkkomponente eine DSL-Leitung ist, dann verwende die Hochqualitätsvideoausgabekomponente.”. Der Vorteil dieser Beziehungen besteht in weiteren Einschränkungen für die Selektion von Komponenten. Durch die zusätzlichen Bedingungen könnten Komponenten im Selektionsvorgang gezielter ausgewählt werden.

Die “wenn-dann”-Bedingungen könnten auch durch den Ausbau eines weiteren, in PCom vorhandenen Mechanismus realisiert werden. Der Mechanismus der Vertragsverhandlungen wird bisher nicht genutzt - auch auf Grund der fehlenden Möglichkeit, Ressourcen zu reservieren. Während den Vertragsverhandlungen wird eine Anwendungskonfiguration gesucht, die eine gültige, instanziierbare Konfiguration darstellt. Damit das Ergebnis der Verhandlung auch über den Moment der konkreten Prüfung hinaus Bestand hat, muss ein Reservierungssystem für Komponenten und Ressourcen in PCom integriert werden. Durch die Bereitstellung eines Reservierungsmechanismus

könnten im Rahmen der Konfiguration einer Anwendung zuerst verschiedene Angebote eingeholt und anschließend verglichen werden. Die Anwendungsbeschreibung ermöglicht hier auch Möglichkeiten, im Vorfeld der Instanziierung eine möglichst günstige Konfiguration für die Anwendung zu bestimmen. Die Instanziierung einer Anwendung, für die eine Konfiguration schon über vorangegangene Vertragsverhandlung ausgewählt wurde, würde keine Selektion von Komponenten mehr benötigen, sondern sich durch direkte Instanziierung der ausgehandelten Struktur auszeichnen.

Die Anwendungsbeschreibung stellt in Zusammenhang mit der Komponentenselektion eine anwendungsspezifische Informationsquelle, also die Sicht des Anwendungsprogrammierers dar. Denkbar wäre eine Ergänzung dieser Sichtweise durch eine containerspezifische Sicht. Der Ansatz von Aura und Gaia besteht in relativ statischen Umgebungen, in denen sich Nutzer bewegen. Umgebungen mit verhältnismäßig wenigen Änderungen werden auch in PCom-basierten Systemen existieren. Ein Container in einer solchen Umgebung könnte Informationen über die Umgebung sammeln, und für bestimmte Anwendungen günstige Konfigurationen ermitteln und ablegen. Auf diese Weise könnte mit der gewonnenen Erfahrung eine Datenbank für Anwendungskonfigurationen aufgebaut werden. Wird nun eine Anwendung in einer solchen Umgebung instanziiert, könnten die im Container vorhandenen Informationen mit denen der Anwendungsbeschreibung abgeglichen werden und so die Konfiguration beschleunigt werden bzw. eine bereits in der Datenbank vorhandene Konfigurationsvorlage verwendet werden. Im günstigsten Fall bräuchte nur, wie bereits bei den Vertragsverhandlungen erwähnt, eine solche Vorlage instanziiert werden. Ein weiterer Untersuchungsgegenstand wären die sich daraus ergebenden Vorteile oder Nachteile auf die konkreten Konfigurationen im Hinblick auf die von der Anwendung aufgestellten Präferenzen.

Generell kann die Bereitstellung systemspezifischer Informationsquellen bei der Selektion eine sinnvolle Ergänzung zu den anwendungsspezifischen Informationen durch die Anwendungsbeschreibung darstellen. Konzepte, wie sie beispielsweise in Odyssey zur Überwachung, Aufzeichnung und schließlich Bereitstellung systemspezifischer Informationen existieren würden die Möglichkeiten zur Nutzung der Freiheitsgrade in PCom weiter erhöhen. Eine tragende Rolle könnte hier einem Kontext-Dienst zukommen. Die Erhebung von Kontextinformationen und deren Verwertung im Rahmen der Adaption von Anwendungen könnte eine für den Nutzer günstigere Konfiguration seiner Anwendungen zur Folge haben bzw. die Interaktion mit dem Nutzer auf das für die Bedienung der Anwendung notwendige Maß zu reduzieren.

7.3 Fazit

Ubiquitäre Systeme stellen hohe Anforderungen an Anwendungsprogrammierer. Verteilungsinfrastrukturen für solche Systeme müssen Mechanismen bieten, die den Anwendungsprogrammierer in geeigneter Form entlasten. Eine Form der Entlastung ist die Bereitstellung von Mechanismen zur automatischen Adaption und Konfiguration von Anwendungen.

Die Freiheitsgrade bei der Entscheidung, wie eine Anwendung adaptiert wird, können auf verschiedene Arten genutzt werden. Eine Möglichkeit der Nutzung dieser Freiheitsgrade ist die einer Anwendungsbeschreibung. Sie ist ein Mittel, das es dem Anwendungsprogrammierer ermöglicht, Einfluss auf die Adaptionsentscheidung zu nehmen.

Die anwendungsspezifische Informationsquelle Anwendungsbeschreibung ermöglicht eine knotenübergreifende Betrachtung der Anwendung im Adaptionsfall und schafft dadurch für PCom neue Möglichkeiten und Optimierungspotential für die Adaption.

Die Anwendungsbeschreibung hat sich als geeignetes Mittel zur knotenübergreifenden Spezifikation von Präferenzen in Bezug auf funktionale, nicht-funktionale und strukturelle Anforderungen aus Sicht des Anwendungsprogrammierers gezeigt. Die konkrete Umsetzung fügt sich in die bestehende Verteilungsinfrastruktur nahtlos ein und wird dabei dem ubiquitären Umfeld gerecht in dem sie für die betrachteten Anwendungen den Ressourcenbedarf nur unerheblich steigert.

Das gesamte Potential der Anwendungsbeschreibung lässt sich erst durch deren Verwendung in realen Anwendungsszenarien - und der damit einher gehenden Entwicklung von optimierten Adaptions-, Selektions- und Verhandlungsstrategien - genau bestimmen. Die notwendigen Strukturen wurden in der Verteilungsinfrastruktur PCom geschaffen und es ist nun die Aufgabe weiterer Projekte, Nutzen aus den geschaffenen Möglichkeiten zu ziehen.

Literaturverzeichnis

[1] **3PC**, *http://www.3pc.info*, Universität Stuttgart

[2] **CORBA**, *http://www.corba.org*, Open Management Group

[3] **CORBA Components (Version 3.0)**, *http://www.omg.org/docs/formal/02-06-65.pdf*, Object Management Group

[4] **IBM and autonomic Computing: An architectural blueprint for autonomic Computing**, April 2003

[5] **J2EE**, *http://java.sun.com/j2ee/*, Sun Microsystems

[6] **Java 2 Plattform Micro Edition Connected Limited Device Configuration (CDLC)**, *http://java.sun.com/products/cldc/index.jsp*, Sun Microsystems

[7] **Java 2 Platform Micro Edition Overview**, *http://java.sun.com/j2me/j2me-ds.pdf*, Sun Microsystems

[8] **Java SizeOf()**, *http://martin.nobilitas.com/java/sizeof.html*, Martin Roth

[9] **Jini**, *http://wwws.sun.com/software/jini/*, Sun Microsystems

[10] **Jini Networking Technology,** *http://wwws.sun.com/software/jini/whitepapers/jini-execoverview.pdf*, Sun Microsystems

[11] **Metrics plugin for Eclipse**, *http://metrics.sourceforge.net/*, Frank Sauer

[12] **TAO**, *http://www.cs.wustl.edu/~schmidt/TAO.html*, Douglas Schmidt

[13] S. AUKSTAKALNIS, D. BLATNER, **Silicon Mirage: The Art and Science of Virtual Reality**, *Peachpit Press*, Berkeley, CA, 1992.

[14] PHILIP A. BERNSTEIN, **Middleware: An Architecture for Distributed System Services**, *Digital Equipment Corporation Technical Report CRL 93/6*, Cambridge Research Lab, 1993

[15] GRADY BOOCH, **Objektorientierte Analyse und Design**, *Addison-Wesley*, 1994

[16] SHANG-WEN CHENG, AN-CHENG HUANG, DAVID GARLAN, BRADLEY SCHMERL, PETER STEENKISTE, **Rainbow: Architecture-based Self-Adaption with Reusable Infrastructure**, *Submitted for publication*, CMU, 2004

[17] LAP-SUN CHEUNG, YU-KWONG KWOK, **A Quantitative Comparison of Load Balancing Approaches in Distributed Object Computing Systems**, *IEEE COMPSAC 2001, 257-262*, Chicago, USA, 2001

[18] BRIAN ENSINK, VIRKAM ADVE, **Coordinating Adaptations in Distributed Systems**, UIUC

[19] JASON FLINN, M. SATYANARAYANAN, **Energy-aware adaption for mobile applications**, *Proc. of the 17th Symposium on Operating Systems Principles (SOSP)*, Kiawah Island, 1999

[20] ERICH GAMMA, RICHARD HELM, RALPH JOHNSON, JOHN VLISSIDES, **Design Patterns: Elements of Reusable Object-Oriented Software**, *Addison-Wesley*, 1995

[21] DAVID GARLAN, DANIEL P. SIEWIOREK, ASIM SMAILAGIC, PETER STEENKISTE, **Project Aura: Towards Distraction-Free Pervasive Computing**, *IEEE Pervasive Computing, 1(2), 22-31*, April-Juni 2002

[22] DAVID GARLAN, JOÃO PEDRO SOUSA, **Aura: An Architectural Framework for User Mobility in Ubiquitous Computing Environments**, *Proceedings of the 3rd Working IEEE/ IFIP Conference on Software Architecture*, August 2002

[23] DAVID GARLAN, ROBERT T. MONROE, DAVID WILE, **Acme: Architectural Description of Component-Based Systems**, *Foundations of Component-Based Systems, pp. 47-68, Cambridge University Press*, 2000

[24] ROBERT GRIMM, JANET DAVIS, ERIC LEMAR, ADAM MACBETH, STEVEN SWANSON, STEVEN GRIBBLE, TOM ANDERSON, BRIAN BERSHAD, GAETANO BORRIELLO, DAVID WETHERALL, **Programming for Pervasive Computing Environments**, *Technical Report UW-CSE-01-06-01,* University of Washington, Juni 2001

[25] HOLGER GUBBELS, **Basisdienste für Anwendungen in ubiquitären Rechnersystemen**, Universität Stuttgart, Dezember 2002

[26] MARCUS HANDTE, **Entwicklung eines Komponentensystems für Anwendungen in ubiquitären Rechnersystemen**, Universität Stuttgart, 2003

[27] CHRISTOPHER K. HESS, FRANCISCO BALLESTEROS, M.DENNIS MICKUNAS, ROY H. CAMPBELL, **An Adaptive Data Object Service for Pervasive Computing Environments**, UIUC

[28] W. HESSE, **Objekt-orientierte Anwendungsmodellierung - ein Weg zu einem durchgängigen Software-Entwicklungsprozeß** in: G. Kugel (Hrsg.), Praxiserprobte Software-Entwicklungswerkzeuge im Überblick, Reihe Kontakt und Studium, *Expert-Verlag*, 1992

[29] JEFFREY O. KEPHART, DAVID M.CHESS, **The Vision of Autonomic Computing**, *IEEE Computer, 36(1), 41-50*, January 2003

[30] FABIO KON, ROY H. CAMPBELL, **Supporting Automatic Configuration of Component-Based Distributed Systems**, UIUC

[31] T. MCCABE, **A Complexity Measure**, *IEEE Transactions of Software Engineering, Volume SE-2, No. 4, Seite 308-320*, 1976

[32] DUSHYANTH NARAYANAN, JASON FLINN, M. SATYANARAYANAN, **Using History to Improve Mobile Application Adaptation**, *Proceedings of the 3rd IEEE Workshop on Mobile Computing Systems and Applications*, Monterey, CA, December 2000

[33] BRIAN D. NOBLE, M. SATYANARAYANAN, DUSHYANTH NARAYANAN, JAMES ERIC TILTON, JASON FLINN, KEVIN R. WALKER, **Agile Application-Aware Adaption for Mobility**, *ACM SIGOPS Operating Systems Review 31, 5 (Dec. 1997), 276--287*, Saint-Malo, France, 1997

[34] MANUEL ROMÁN, CHRISTOPHER HESS, RENATO CERQUEIRA, ROY H. CAMPBELL, KLARA NAHRSTEDT, **Gaia: A Middleware Infrastructure to Enable Active Spaces**, UIUC, Juni, 2002

[35] MANUEL ROMÁN, CHRISTOPHER K. HESS, ROY H. CAMPBELL, **Gaia: An OO Middleware Infrastructure for Ubiquitous Computing Environments**, *ECOOP-OOOSWS*, Magala, Spanien, 2002

[36] M. SATYANARAYANAN, **Coda: A Highly Available File System for a Distributed Workstation Environment**, *Proceedings of the Second IEEE Workshop on Workstation Operating Systems*, Pacific Grove, CA, 1989

[37] CLEMENS SZYPERSKI, **Component Software 2nd Edition**, *Addison-Wesley*, 2002

[38] JIM WALDO, KEN ARNOLD, **The Jini(TM) Specifications 2nd Edition**, *Addison-Wesley*, 2000

[39] ZHENYU WANG, DAVID GARLAN, **Task-Driven Computing**, *Technical Report, CMU-CS-00-154*, CMU, Mai 2000

[40] MARK WEISER, **Some Computer Science Issues in Ubiquitous Computing**, *Communications of the ACM, 36(7), 74-84*, July 1993

[41] MARK WEISER, **The Computer for the 21st Century**, *Scientific American, 265(3), 94-104,* September 1991

[42] MARK WEISER, **Ubiquitous Computing**, *IEEE Computer Hot Topics, 71-72,* Oktober 1993

Erklärung

Ich versichere, dass ich diese Arbeit selbständig verfasst und nur die angegebenen Hilfsmittel verwendet habe.

_____

(Stephan Urbanski)