Erweiterung einer Komponentenplattform zur Unterstützung multimodaler Anwendungen mit föderierten Endgeräten

Dissertation

zur Erlangung des akademischen Grades Doktoringenieur (Dr.-Ing.)

vorgelegt an der
Technischen Universität Dresden
Fakultät Informatik

eingereicht von

Dipl.-Inf. Kay Kadner
geboren am 1. Oktober 1979 in Plauen

Betreuender Hochschullehrer:
Prof. Dr. rer. nat. habil. Dr. h. c. Alexander Schill

Gutachter:

Prof. Dr. rer. nat. habil. Dr. h. c. Alexander Schill	TU Dresden
Prof. Dr. rer. nat. habil. Gerhard Weber	TU Dresden
Prof. Dr. rer. nat. Max Mühlhäuser	TU Darmstadt

Tag der Abgabe: 29. November 2007
Tag der Verteidigung: 05. Mai 2008

Dresden im Mai 2008

Danksagung

Die Idee für diese Arbeit entstand während meiner Tätigkeit bei SAP Research CEC Dresden und am Institut für Systemarchitektur der TU Dresden. Ich danke besonders Herrn Prof. Dr. rer. nat. habil. Dr. h. c. Alexander Schill für die Begleitung dieser Arbeit und insbesondere für die eingeräumten Freiheiten, regelmäßigen Motivationsschübe und konstruktiven Diskussionen. Mein Dank gilt weiterhin Herrn Prof. Dr. Gerhard Weber und Herrn Prof. Dr. Max Mühlhäuser für ihre hilfreichen Hinweise sowie die Bereitschaft, das Koreferat zu übernehmen.

Weiterhin möchte ich allen Kollegen vom SAP Research CEC Dresden und der TU Dresden danken, die während der Bearbeitung ein stets kollegiales und freundliches Arbeitsklima ermöglichten. Dabei gilt mein Dank besonders Dr. Christoph Pohl, der mich bei der Themenfindung unterstützt hat und Dr. Thomas Springer, der mir besonders in der Endphase beratend zur Seite stand. Außerdem bedanke ich mich bei Dipl.-Inform. Falk Hartmann, Dr. Steffen Göbel, Dipl.-Inf. Gerald Hübsch für die interessanten Diskussionen und Anregungen.

Darüber hinaus danke ich Dipl.-Inf. Anja Klein, Dipl.-Wirt.-Inf. Jürgen Anke, Dipl.-Ing. Martin Knechtel und Dr. Thomas Springer für die hilfreichen Anmerkungen zu Struktur, Form und Orthografie sowie Grammatik der Arbeit. Mit ihrer Hilfe ist es gelungen, Fehler zu beseitigen und die Verständlichkeit zu verbessern.

Zuletzt gilt mein Dank meiner Familie und meinen Freunden, die mich während der Bearbeitung stets unterstützt und motiviert haben. Insbesondere bedanke ich mich bei Doreen, die durch viel Rücksicht, kontinuierlichen Ansporn und nicht zuletzt durch das Korrekturlesen zum Erfolg der Arbeit beigetragen hat.

Dresden, im November 2007

Kay Kadner

Inhaltsverzeichnis

Abkürzungsverzeichnis

ACL ..... Agent Communication Language

AJAX ..... Asynchronous JavaScript and XML

AMM ..... Aircraft Maintenance Manual

API ..... Application Programmer Interface

ATA ..... Air Transport Association

AWT ..... Abstract Windows Toolkit

BDI ..... Belief-Desire-Intention

BL ..... Business Logic

BPEL ..... Business Process Execution Language

BPEL4WS ... BPEL for Web Services

BSC ..... Base Station Controller

BTS ..... Base Transceiver Station

CCM ..... CORBA Component Model

CCXML ..... Call Control XML

CDC ..... Connected Device Configuration

CLDC ..... Connected Limited Device Configuration

COM+ ..... Weiterentwicklung des Component Object Model

CPU ..... Central Processing Unit

CSCW ..... Computer Supported Cooperative Work, auch Computer Supported Collaborative Work

D3ML ..... Device-independent MultiModal Mark-up Language

DVZ ..... Dienstverzeichnis

DOM ..... Document Object Model

KVZ ..... Komponentenverzeichnis

EJB ..... Enterprise Java Beans

EMMA ..... Extensible Multimodal Annotation Markup Language

FIPA ..... Foundation of Intelligent Physical Agents

GSM ..... Global System for Mobile Communications

HCI ..... Human-Computer-Interaction

HTML ..... HyperText Markup Language

HTTP ..... HyperText Transfer Protocol

HTTPS ..... HyperText Transfer Protocol Secure

IETF ..... Internet Engineering Task Force

InkML ..... Ink Markup Language

IP ..... Internet Protocol

JADE ..... Java AGent Development Framework

JAR ..... Java ARchive

JME ..... Java Micro Edition

JPEG ..... Joint Photographic Experts Group

JSE ..... Java Standard Edition

JSGF ..... Java Speech Grammar Specification

JVM ..... Java Virtual Machine

KQML ..... Knowledge Query and Manipulation Language

LAN ..... Local Area Network

MMI-F ..... MultiModal Interaction Framework

MVC ..... Model-View-Controller

NLP ..... Natural Language Processing

NoE ..... Network of Excellence

NTP ..... Network Time Protocol

OSGi ..... Open Services Gateway Initiative

PDA ..... Personal Digital Assistant

PNG ..... Portable Network Graphics

QoS ..... Quality-of-Service

RAM ..... Random Access Memory

RFC ..... Request for Comments

RFID ..... Radio-Frequency Identification

RIA ..... Rich Internet Applications

RMI ..... Remote Method Invocation

SCXML ..... State Chart XML

SMIL ..... Synchronized Multimedia Integration Language

SNOW ..... Services for NOmadic Workers

SNTP ..... Simple NTP

SOAP ..... Simple Object Access Protocol

SSML ..... Speech Synthesis Markup Language

SVG ..... Scalable Vector Graphics

UI ..... User Interface

UIML ..... User Interface Markup Language

UMTS ..... Universal Mobile Telecommunications System

UPnP ..... Universal Plug and Play

URI ..... Uniform Resource Identifier

URL ..... Uniform Resource Locator

W3C ..... World Wide Web Consortium

WiDex ..... Widget Description Exchange Service

WLAN ..... Wireless LAN

WML ..... Wireless Markup Language

WS-BPEL .... Web Services BPEL

WS-CDL ..... Web Service Choreography Description Language

WSDL ..... Web Services Description Language

XHTML ..... Extensible HyperText Markup Language

XML ..... eXtensible Markup Language

Abbildungsverzeichnis

Tabellenverzeichnis

Es ist wichtiger Fragen stellen zu können, als auf alles eine Antwort zu haben.

James Thurber

1 Einleitung

Die Entwicklung unserer Gesellschaft ist geprägt von technologischen Innovationen im Bereich Informationsverarbeitung und Kommunikation. In den letzten Jahren haben sich z. B. Mobiltelefone so stark verbreitet, dass es in Deutschland schon mehr Mobiltelefonanschlüsse als Einwohner gibt[1]. Europaweit (EU27) besaßen 2005 immerhin 96% der Einwohner einen Mobilfunkvertrag[2]. Dieser starke Anstieg der Verbreitung ist nicht zuletzt auf die immer umfangreicher werdenden Funktionen der Mobiltelefone zurückzuführen, welche vor allem im Bereich Multimedia (Kamera, Musik, Video, etc.) angesiedelt sind. Projekte wie das BIB3R-Projekt (Berlin's Beyond-3G Testbed and Serviceware Framework for Advanced Mobile Solutions, [Thea]) streben die verstärkte Integration verfügbarer Kommunikationstechnologien an. Das ermöglicht die Entwicklung neuartiger Dienste beispielsweise im Bereich der *location-based services* als Teilmenge kontextsensitiver Anwendungen [SKSY06]. Von zunehmendem Interesse sind auch PDA (Personal Digital Assistant) und Smartphone, die in gewissem Maß die Funktionalität eines Laptops mit der eines Mobiltelefons vereinen, da man außer den normalen Telefonfunktionen u. a. auch Internet und E-Mail über WLAN (Wireless LAN) sowie Rich-Client-Anwendungen nutzen kann.

Neben dem Trend im Mobilfunkmarkt lässt sich diese Entwicklung auch durch die sinkenden Preise und damit steigende Verbreitung von Laptops belegen, welche nicht zuletzt auf die zunehmende Verfügbarkeit von öffentlichen WLAN-

[1] Pressemitteilung des Bundesverband Informationswirtschaft, Telekommunikation und neue Medien e.V. vom 14. August 2006, `http://www.bitkom.org/de/presse/8477_40990.aspx`

[2] Pressemitteilung der eurostat vom 27. November 2007, `http://epp.eurostat.ec.europa.eu/pls/portal/docs/PAGE/PGP_PRD_CAT_PREREL/PGE_CAT_PREREL_YEAR_2007/PGE_CAT_PREREL_YEAR_2007_MONTH_11/3-27112007-DE-AP.PDF`

Zugriffspunkten zurückzuführen ist. Bis vor wenigen Jahren waren tragbare PCs noch relativ groß und teuer, sodass sich nicht jeder diesen Luxus erlauben konnte. Mittlerweile haben jedoch Laptops ihren Weg in viele Haushalte gefunden und sind keine Besonderheit mehr. Dazu kommt, dass immer mehr Geräte aus dem Alltag auf verschiedene Arten mit Computern ausgerüstet sind bzw. mit computergenerierten Signalen umgehen können, wie z. B. Wandprojektoren oder Digitalfernsehgeräte.
Insgesamt ist es zwar so, dass sich immer mehr Computer in der einen oder anderen Weise in unserem Alltag befinden. Jedoch kann der Nutzer nur selten die vielfältigen Möglichkeiten effektiv und effizient nutzen, weil die meisten Geräte für ihren eigenen, isolierten Verwendungszweck entwickelt wurden und keine Schnittstellen zu anderen Geräten anbieten bzw. die angebotenen Schnittstellen nicht genutzt werden. Darüber hinaus interagiert der Mensch mit seiner Umwelt auf natürliche Weise mit Hilfe von gleichzeitig verwendeten Informationskanälen [DFAB04, MMF00, Bun98], sodass eine technische Unterstützung dieser nur konsequent ist. Statt von Informationskanälen kann man auch von Modalitäten sprechen. Eine Modalität wird als Art und Weise des Informationsaustauschs zwischen Kommunikationspartnern verstanden. Die ausführliche Definition befindet sich in Abschnitt 3.1.

1.1. Motivation

Bei der Interaktion mit Computern verfolgt jeder Nutzer ein bestimmtes Ziel, das in der Regel zum Erledigen einer Aufgabe notwendig ist. Auf dem Weg zur Erreichung des Ziels möchte sich der Nutzer voll und ganz auf die Aufgabe konzentrieren und nicht durch Eigenschaften, Fähigkeiten oder Probleme der dafür verwendeten Endgeräte abgelenkt werden. Dies ist insbesondere dann der Fall, wenn ein Endgerät eine vom Nutzer gewünschte Modalität nicht oder nur unzureichend zur Verfügung stellt. Dies begründet sich in der Tatsache, dass kein Endgerät existiert, welches alle Nutzeranforderungen hinsichtlich funktionaler (z. B. verfügbare Modalitäten) und nichtfunktionaler (z. B. Größe, Gewicht, Modalitätsqualität) Eigenschaften der Interaktion erfüllt. Zwar gibt es oft Spezial-Hardware, die zusätzliche Möglichkeiten der Interaktion anbietet, jedoch empfiehlt sich diese auf Grund des Anschaffungspreises nicht für jedermann. Und selbst in hohen Preisklassen ist beispielsweise ein Endgerät mit hosentaschentauglichem Gewicht und Größe sowie einem 19"Monitor, der bei Bedarf zugeschaltet werden kann, nicht verfügbar.
Dazu kommt, wie einleitend schon erwähnt, die Tatsache, dass sich immer mehr Endgeräte in unserem Alltag etablieren, die ihre Dienste mittels verschiedener Modalitäten anbieten. Jedoch sind viele dieser Geräte entworfen und hergestellt worden, ohne dass besonderer Wert auf Interoperabilität zwischen Geräten bzw. Möglichkeiten zur Föderation von Geräten gelegt wurde. Eine Geräteföderation

ist eine Menge von Geräten, die gemeinsam zur Erreichung eines Ziels verwendet werden. So ist es beispielsweise nicht ohne großen Aufwand möglich, die Stifteingabe eines PDAs für Anwendungseingaben zu nutzen, wobei die Anwendung (z. B. eine Zeichenanwendung) auf einem Laptop läuft.
Es gibt demnach kein „ultimatives“ Endgerät, welches dem Nutzer alle erdenklichen Modalitäten zur Verfügung stellt. Der Nutzer muss sich vor dem Beginn seiner Arbeit für ein Endgerät entscheiden (siehe Abbildung 1.1a), ein späteres Ändern oder gemeinsames Benutzen verschiedener Geräte ist bisher nicht möglich. Durch Kombinieren von Geräten könnten verschiedene Ein- und Ausgaben in verschiedenen Modalitäten realisiert werden. Bisher ist der Nutzer stets auf die Modalitäten angewiesen, die das gerade ausgewählte Endgerät anbietet, ohne dass er die Vorteile bei der Interaktion mit weiteren, unabhängigen Endgeräten nutzen kann.

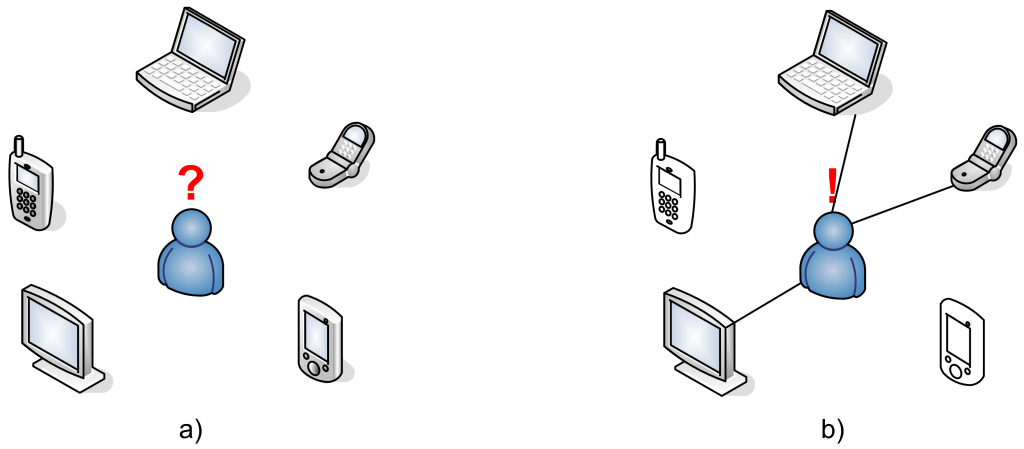

Abbildung 1.1.: Motivation der Arbeit: a) momentane Situation, der Nutzer muss sich für ein Endgerät entscheiden, b) zukünftige Situation, der Nutzer wählt die gewünschten Geräte aus einer Menge aus

1.2. Zielstellung

Aus den vorgenannten Gründen ergibt sich die Forderung nach Unterstützung der gemeinsamen Verwendung verschiedener Geräte für die Interaktion mit einer Anwendung (siehe Abbildung 1.1b). Dafür scheint eine Integrationsschicht geeignet, die sich über verschiedene Geräte erstreckt und dem Nutzer die gewünschte Freiheit bei der Wahl der Endgeräte und somit der Modalitäten ermöglicht. Da es sich bei den Endgeräten meist um mobile Endgeräte handelt, muss die Integrationsschicht die sich dadurch ergebenden Anforderungen und Besonderheiten berücksichtigen. Um dem Nutzer entsprechende Flexibilität in der Wahl seiner

Endgeräte zu ermöglichen ohne dass er den Überblick verliert, muss die Integrationsschicht dem Nutzer entsprechende Mechanismen zur Verwaltung der Geräte zur Verfügung stellen.
Ein weiteres Entwurfsziel der Integrationsschicht ist die Robustheit gegenüber Ausfällen einzelner Geräte bzw. Modalitäten, da es für den Nutzer nicht akzeptabel ist, wenn Teile der verteilten Nutzerschnittstelle gelegentlich nicht mehr verfügbar sind oder die Interaktion auf andere Weise unterbrochen wird. Wie bereits erwähnt, will sich der Nutzer auf seine Aufgabe konzentrieren und widmet nur einen geringen Teil seiner Aufmerksamkeit der Konnektivität mit drahtlosen Netzwerken. Das schließt eine Adaption der Integrationsschicht an geänderte Systembedingungen mit ein, sodass auch bei Abkopplung von jeglicher Infrastruktur die vom Nutzer gewählten Interaktionsmöglichkeiten verfügbar sind.

1.3. Kernfragen und Abgrenzung

Da sich die Integrationsschicht über verschiedene Geräte erstreckt, jedes Gerät unterschiedliche Funktionen entsprechend seiner verfügbaren Modalitäten übernimmt und die verschiedenen Funktionen dynamisch zur Laufzeit bestimmt und geändert werden können, bietet sich eine komponentenbasierte Realisierung an. Demnach besteht die Integrationsschicht aus einer Menge von verteilten Komponentenplattformen, auf denen Komponenten die Interaktion des Nutzers mit der Anwendung ermöglichen. Daraus ergibt sich die Kernfrage 1 nach Konzepten auf Protokollebene, mit denen eine gleichzeitige Verwendung der Endgeräte zum Anwendungszugriff möglich ist.
Wesentlicher Bestandteil der Integrationsschicht ist die Komponentenplattform, sodass sich Kernfrage 2 ergibt, welche Konzepte diese unterstützen muss, wenn eine aus Komponenten bestehende Anwendung über verschiedene Komponentenplattformen verteilt ausgeführt werden soll.
Damit die Integrationsschicht für möglichst viele Anwendung benutzbar ist, soll durch Beantwortung von Kernfrage 3 herausgefunden werden, welche Anforderungen an eine Anwendung gestellt werden, damit sie für multimodalen Zugriff durch verteilte Endgeräte geeignet ist.
Zusammenfassend sollen mit der vorliegenden Arbeit die folgenden Kernfragen beantwortet werden:

Kernfrage 1: Welche Konzepte sind auf Protokollebene notwendig, damit dem Nutzer eine gleichzeitige Verwendung unterschiedlicher, mobiler und stationärer Endgeräte zum Zugriff auf dieselbe Anwendung auf eine nutzerfreundliche Art und Weise ermöglicht wird?

Kernfrage 2: Welche Konzepte muss eine Komponentenplattform unterstützen, wenn die Funktionalität der darauf aufsetzenden Anwendung zur Interaktion in einzelne Komponenten unterteilt wird? Insbesondere ist dabei die

Bereitstellung von multimodalen Nutzerschnittstellen durch einzelne, z. T. mobile Endgeräte zu berücksichtigen.

Kernfrage 3: Welche Anforderungen muss eine Anwendung erfüllen, damit die Benutzung föderierter Endgeräte nicht explizit unterstützt werden muss? Wie muss die Integrationsschicht bzw. die Anwendung beschaffen sein, damit für die Anwendung die tatsächliche Art der Nutzerschnittstelle transparent ist, sodass föderierte Endgeräte ohne Einfluss auf die Anwendung oder deren Entwicklungsprozess unterstützt werden können?

Die in der Arbeit angesprochenen Bereiche der Mensch-Maschine-Interaktion, Ubiquitous Computing und Komponententechnologien sind sehr vielfältig, daher ist eine Abgrenzung der Arbeit von nicht betrachteten Aspekten notwendig. Da der Fokus auf der technischen Realisierung einer solchen Integrationsschicht liegt, werden Aspekte der Mensch-Maschine-Kommunikation nur am Rande betrachtet. Es soll durch die Arbeit nicht ermittelt werden, welche besonderen Anforderungen an die kognitive Verarbeitung (z. B. Wechsel der Modalitäten) durch den Nutzer solcher Nutzerschnittstellen gestellt werden. Konzepte zur Integration bzw. Fusion verschiedener multimodaler Eingaben zu zusammengesetzten Kommandos werden in dieser Arbeit nicht betrachtet sondern als vorhanden angenommen. Erste Arbeiten dazu entstanden bereits 1980, als Bolt [Bol80] seinen „Put that there“-Prototyp vorstellte, mit dem man auf Gegenstände zeigen und ein Kommando sagen konnte, das vom System integriert und als zusammengesetztes Kommando ausgeführt wurde. Darüber hinaus wird in der Arbeit keine Beschreibungssprache oder -methodik für geräteunabhängige, multimodale Nutzerschnittstellen konzipiert, da bereits eine Vielzahl an Beschreibungssprachen existieren [Theb, OAS, GHKP06]. Der Fokus der Arbeit liegt auf den technischen Konzepten der Integrationsschicht, die dem Nutzer die gewünschte bzw. notwendige Flexibilität, Nutzerfreundlichkeit und Robustheit bei der Interaktion ermöglicht.

1.4. Gliederung der Arbeit

Die weitere Arbeit gliedert sich folgendermaßen: in Kapitel 2 werden auf Basis zweier Anwendungsfälle die genauen Anforderungen an die Integrationsschicht erarbeitet. In Kapitel 3 werden verwandte Arbeiten untersucht und in Bezug auf die vorher ermittelten Anforderungen verglichen. In Kapitel 4 werden die Integrationsschicht und deren Bestandteile konzipiert. Kapitel 5 beschäftigt sich anhand einer prototypischen Implementierung mit der Validierung der Integrationsschicht. Abschließend wird in Kapitel 6 die Arbeit zusammengefasst und ein Ausblick auf weiterführende Forschungsfragen gegeben.

Den Fortschritt verdanken die Menschen den Unzufriedenen.

Aldous Huxley

2 Anforderungsanalyse

In diesem Kapitel werden die Anforderungen an die zu entwickelnde Integrationsschicht ermittelt, welche bereits teilweise in [Kad06] erarbeitet wurden. Dazu dienen zwei in Abschnitt 2.1 beschriebene Anwendungsfälle, welche so gewählt wurden, dass möglichst allgemein gültige Anforderungen daraus abgeleitet werden können. In Abschnitt 2.2 werden Annahmen beschrieben, die zur Eingrenzung der Arbeit notwendig waren und Auswirkung auf die Anforderungen haben. Die Anforderungen werden schließlich in Abschnitt 2.3 (Interaktionsanwendung), Abschnitt 2.4 (Komponentenplattform) und Abschnitt 2.5 (Zielanwendung) erarbeitet. Den Abschluss des Kapitels bildet eine Gegenüberstellung der Kernfragen der Arbeit mit den aufgestellten Anforderungen (Abschnitt 2.6).

2.1. Anwendungsfälle

Im Folgenden werden zwei Anwendungsfälle beschrieben, welche die Funktionalität der geplanten Integrationsschicht bildhaft darstellen. Dabei wurden die Anwendungsfälle aus unterschiedlichen Anwendungsdomänen entnommen um ein großes Spektrum an möglichen Anforderungen abzudecken. Das erste Szenario beschreibt die Tätigkeit eines Wartungsarbeiters im industriellen Umfeld (Unterabschnitt 2.1.1). Das zweite Szenario stammt aus dem Geschäfts- bzw. Forschungsumfeld (Unterabschnitt 2.1.2), kann aber prinzipiell überall dort eingesetzt werden, wo physisch verteilte, virtuelle Teams zum Einsatz kommen. Die Anwendungsszenarien stellen keine abgeschlossenen Handlungen der Nutzer dar, was z. B. ein Kriterium beim aufgaben-orienterten Entwurf von Nutzerschnittstellen in der HCI (Human-Computer-Interaction) ist [LR93]. Sie illustrieren nur

mögliche Einsatzgebiete der Integrationsschicht und erheben keinen Anspruch auf Vollständigkeit der Interaktionsmöglichkeiten mit den Anwendungen.

2.1.1. Wartungsarbeiter

Ein Wartungsarbeiter, beispielsweise aus dem Luftfahrtbereich, hat unter anderem die Aufgabe, genau spezifizierte Wartungsarbeiten an Flugzeugen durchzuführen. Dazu gibt es Wartungsdokumente, die jeden Schritt der durchzuführenden Aufgabe vorgeben. Derzeit existiert diese Wartungsdokumentation meist auf Papier, da die Umstellung auf elektronische Dokumente für bereits vorhandene Dokumentationen sehr aufwändig ist und Fehler dabei nicht ausgeschlossen werden können. Beispielsweise umfasst die Wartungsdokumentation für einen Airbus A320 mit verschiedenen Konfigurationen 8432 Prozeduren im Sinne von AMM-Aufgaben (Aircraft Maintenance Manual) wie Tests, Prüfen, Entfernen, etc., die in 3218 ATA-Unterkapitel (Air Transport Association) eingeteilt werden. Für die Zukunft ist zu erwarten, dass die Dokumentationen immer häufiger elektronisch erstellt und verwendet werden [The05b, BPBH06, KR07].
Um die elektronischen Dokumentationen zu verwenden, benötigt der Wartungsarbeiter ein Gerät zur Anzeige bzw. zum Zugriff auf die Informationen. Dazu kann je nach Anforderungen an Beweglichkeit und Platz am Arbeitsort ein PDA, Smartphone, TabletPC, Laptop oder ähnliches zum Einsatz kommen.
In Abbildung 2.1 ist dargestellt, welche Endgeräte der Wartungsarbeiter in diesem Anwendungsfall verwendet. Der Wartungsarbeiter hat sich für den Einsatz eines Laptops entschieden (Schritt 1). Der Laptop bietet Interaktionsmöglichkeiten wie Display-Ausgabe, Tastatur- und Mauseingabe und Sprachinteraktion (Ein- und Ausgabe). Der Wartungsarbeiter verwendet die Eingabemöglichkeiten beliebig, um in der Anwendung entsprechend seiner Aufgaben zu navigieren und die benötigten Informationen zu erhalten. An einer Stelle der aktuellen Wartungsprozedur wird vom Arbeiter verlangt, dass er für die Ausführung der nächsten Schritte beide Hände verwendet. Der Arbeiter kennt sich mit diesem Teil der Wartungsprozedur nicht aus, weshalb er in jedem Schritt auf die Dokumentation angewiesen ist. Er stellt den Laptop in der Nähe ab, sodass er noch die Schrift auf dem Display lesen kann, welches jeweils Schritt für Schritt die Anweisungen anzeigt. Eingaben per Tastatur oder maus kann er so jedoch nicht machen. Daher ist er auf die Interaktion per Sprache angewiesen. Um die Sprachfunktion des Laptops mit zufriedenstellender Zuverlässigkeit nutzen zu können, ist der Einsatz eines Headsets zwingend notwendig, da durch die im industriellen Umfeld vorhandenen Störgeräusche bereits eine verhältnismäßig kurze Entfernung von ein bis zwei Metern zwischen Kopf und Laptop die Verwendung von Sprachsteuerung verhindert. Nachdem der Arbeiter festgestellt hat, dass seine Kommandos schlecht erkannt werden und er auch die akustische Rückmeldung kaum versteht, entscheidet er sich daher für den Einsatz eines Headsets. Der Laptop hat jedoch keinen entsprechenden Anschluss, sodass der Arbeiter auf seinen PDA zurück-

greift (Schritt 2). Er trifft die nötigen Einstellungen auf dem PDA, um mit diesem auf die selbe Anwendung zuzugreifen, die auf dem Laptop gerade ausgeführt wird. Den PDA verstaut er in seiner Kleidung und setzt das Headset auf. Nun ist er in der Lage, über den PDA Kommandos einzugeben, um z. B. auf die Anweisungen des nächsten Schritts zuzugreifen. Dieser wird dann sowohl auf dem Display des Laptops angezeigt als auch über das Headset des PDAs akustisch ausgegeben. Nachdem der Arbeiter zwei Wartungsschritte durchgeführt hat, stellt er fest, dass der Akku des Laptops so gut wie leer ist. Er schaltet den Laptop aus und hat nun ausschließlich den PDA zum Zugriff auf die Dokumentation zur Verfügung.

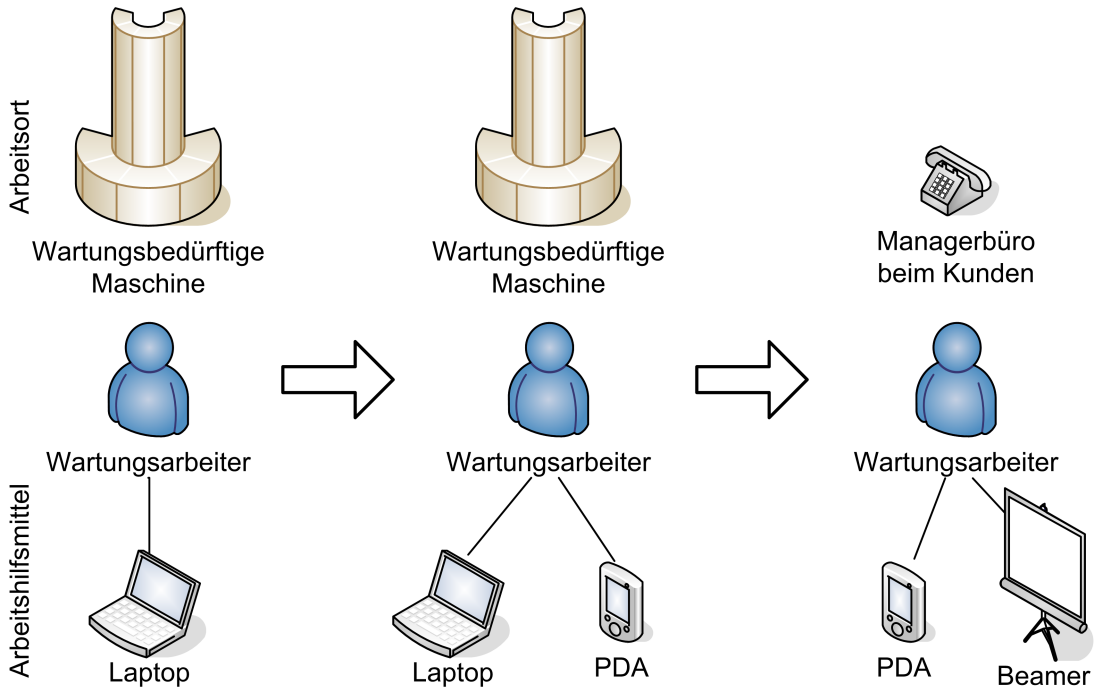

Abbildung 2.1.: Beispielszenario: Mobiler Wartungsarbeiter

Nach Abschluss der Wartungsarbeiten verlässt der Arbeiter die Arbeitshalle und begibt sich zum Manager der Abteilung, um mit ihm die Ergebnisse der Arbeit zu besprechen. Im Büro des Managers befindet sich ein Beamer, welchen der Arbeiter zur Präsentation benutzen möchte (Schritt 3). Mit Hilfe des PDAs wählt er den Beamer zur Föderation aus, woraufhin die Anwendung ihre Ausgabe auf dem PDA und dem Beamer präsentiert. Da der Beamer nicht zur Eingabe geeignet ist, bleibt dem Nutzer hierfür nur der PDA. Nach Abschluss der Präsentation gibt der Arbeiter das Display wieder frei und schaltet seinen PDA aus, womit die Interaktion mit der Anwendung beendet ist.

2.1.2. Videokonferenz

In Zeiten steigender Transferkapazitäten von Netzwerken und zunehmender Internationalisierung von Projektteams werden immer häufiger Videokonferenzsysteme zur effektiveren Kommunikation eingesetzt. Das äußert sich in zweistelligen Wachstumsraten sowohl beim Umsatz von Unternehmen wie Polycom als auch durch Anzahl der Systeminstallationen [Wai07]. Videokonferenzsysteme haben den Vorteil, dass man sein Gegenüber nicht nur hört, sondern auch sieht, wodurch der non-verbale Informationsaustausch ermöglicht wird. Aus [Sch03b] geht hervor, dass die verfügbare Bandbreite des Auges um den Faktor 100 mal größer ist als die der Ohren, sodass der Vorteil eines Videokonferenzsystems nicht von der Hand zu weisen ist.

Der Projektleiter eines Entwicklungsprojektes, dessen Mitarbeiter in weltweit verteilten Standorten arbeiten, reist häufig zu den verschiedenen Standorten und hat dort oft keinen Arbeitsplatz, an dem er seine gewohnte technische Ausrüstung zur Verfügung hat. Das bedeutet in den meisten Fällen, dass er nur seinen Laptop und ggf. einen externen Monitor bzw. Maus und Tastatur vorfindet. Daher kann er bei einer Konferenzschaltung nur per Daten- bzw. Sprachkonferenz teilnehmen und sich die Kamerabilder der anderen Teilnehmer ansehen. Er selbst ist für die anderen unsichtbar. Um diesen Mangel zu umgehen, verwendet der Projektleiter seinen mit einer Kamera ausgestatteten PDA, indem er ihn so konfiguriert, dass das von der Kamera aufgenommene Bild in die Konferenzschaltung einfließt. Da der PDA für die Videoaufnahme genutzt wird und sich die Kamera auf der Rückseite des PDAs befindet, ist es zwecklos, das PDA-Display zum gleichzeitigen Anzeigen der Daten der anderen Teilnehmer zu verwenden. Diese Funktionalität wird weiterhin vom Laptop bereitgestellt.

Auch wenn die eben geschilderte Anwendung ein gutes Beispiel für die flexible Föderation von Endgeräten ist, so handelt es sich jedoch zunächst nur um eine multimediale und nicht um eine multimodale Anwendung. Man spricht erst von einer multimodalen Anwendung, wenn die Eingabekomponenten für Video bzw. Audio eine Verarbeitung dahingehend durchführen, dass sie versuchen bestimmte Gesten in den Bildern bzw. Wörter im Audiostrom zu erkennen, welche dann auf Kommandos für die Anwendung abgebildet werden können (die Definition für Multimodalität befindet sich in Abschnitt 3.1). Typische Anwendungskommandos in diesem Beispiel sind die Stummschaltung von Audio/Video für sowohl Ein- als auch Ausgabe, Beenden der Anwendung etc.

2.2. Annahmen und Voraussetzungen

Um den Rahmen der Arbeit sinnvoll einzuschränken, wurden verschiedene Annahmen getroffen, welche im Folgenden näher erläutert werden.

Das gesamte System besteht aus heterogenen Geräten. Das trifft sowohl auf Desktop-PCs und Server zu, gilt aber auch insbesondere für die mobilen Endgeräte, die ein Nutzer bei sich trägt. Dabei gibt es Geräte, die eine Nutzerschnittstelle anbieten (z. B. Desktop-PCs, PDAs) und solche, die lediglich Verarbeitungen durchführen, ohne für den Nutzer zur direkten Interaktion bereitzustehen (z. B. Server). Zum eindeutigen Verständnis soll hervorgehoben werden, dass der Begriff „Geräte“ alle möglichen Geräte umfasst, während sich „Endgeräte“ nur auf die Geräte mit direkter Interaktionsmöglichkeit durch den Nutzer beschränkt. Somit sind die Endgeräte eine Teilmenge aller Geräte.

Es wird angenommen, dass *ein* Nutzer mit *einer* Anwendung interagiert, wofür er auf eine *Vielzahl* von Modalitäten zurückgreifen kann, die ihm durch verschiedene Endgeräte zur Verfügung gestellt werden. Im Grenzfall kann der Nutzer auch nur die Modalitäten eines einzelnen Endgerätes benutzen, was im Prinzip dem heutigen Stand der Technik beim Interagieren mit Anwendungen entspricht. Es besteht die in Abbildung 2.2 dargestellte Beziehung zwischen Nutzer, Endgerät und Anwendung, wobei das Verhältnis n : e : a mit $n = a = 1$ und e≥1 ist. Es ist zu beachten, dass jedes Endgerät eine oder mehrere Modalitäten anbieten kann. Der Nutzer verwendet beliebig viele aber mindestens eine Modalität pro Endgerät, wobei in der weiteren Arbeit nicht auf die genaue Verteilung von Modalitäten auf Endgeräte eingegangen wird, da die zu Integrationsschicht von konkreten Modalitätsverteilungen unabhängig entworfen werden soll.

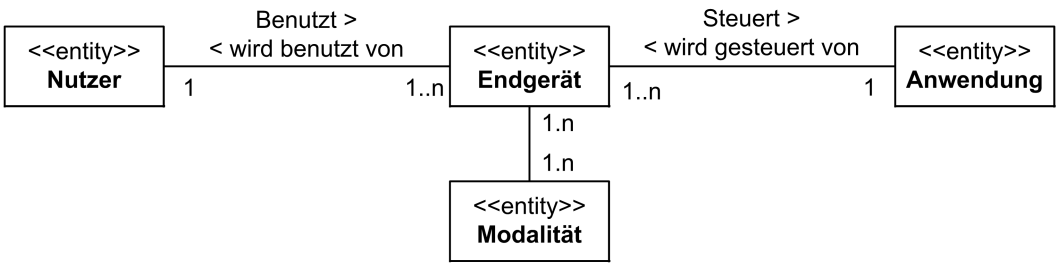

Abbildung 2.2.: Zusammenhang zwischen Nutzer, Endgerät und Anwendung

Durch diese Voraussetzung ist es zunächst nicht möglich, dass ein Nutzer mehrere unabhängige Föderationen erstellt, mit denen er auf mehrere Anwendungen durch noch mehr Endgeräte zugreift, wie z. B. Videokonferenz mit Laptop und PDA während einer Präsentation mit einem anderen PDA und Wand-Display (n : g : a mit $n = 1$ und $g > a$). Dadurch würde das System nur komplexer werden, ohne dass weitere Anforderungen oder zusätzliche Konzepte bezüglich des Grundproblems der Integrationsschicht benötigt werden. Ebenfalls nicht im Fokus der Arbeit befinden sich kollaborative Szenarien, bei denen mehrere Nutzer auf eine Anwendungsinstanz mittels verschiedener Endgeräte zugreifen (n : g : a mit $a = 1$, $n > 1$ und $g > n$). Hingegen ist es durchaus möglich, dass mehrere

Nutzer das System zur Interaktion mit verschiedenen Anwendungen verwenden, wobei die Nutzer und Anwendungen aber voneinander unabhängig sind.
Ein Endgerät kann Interaktionen durch mindestens eine, möglicherweise aber auch durch mehrere Modalitäten anbieten. Ein typischer Anwendungsfall der Integrationsschicht ist außerdem die Nutzung der gleichen Modalität durch mehrere Endgeräte, die unterschiedliche (z. B. Bildschirmgrößen für visuelle Ausgabe von Laptop und PDA), aber auch gleiche Eigenschaften haben dürfen (z. B. zwei Laptops mit gleicher Bildschirmgröße).
Eine Föderation besteht nur aus Endgeräten. Alle anderen Geräte wie Server oder Infrastrukturrechner nehmen nicht an einer Föderation teil. Damit ein Endgerät an einer Föderation teilnehmen kann, muss es die Installation einer minimalen Laufzeitumgebung ermöglichen. In besonderen Fällen ist es möglich, dass auf dem Endgerät selbst keine Laufzeitumgebung vorhanden ist, da dieses beispielsweise bei Verwendung eines Mobiltelefons nur zur Ein- und Ausgabe von Sprache verwendet wird. Hierbei wird dann die zugehörige Laufzeitumgebung von einer Serveranwendung realisiert, welche dem Nutzer die Sprachschnittstelle zur Verfügung stellt. Trotz dieser möglichen Indirektion sind die Endgeräte gleichberechtigte Teilnehmer der Föderation. Die Eigenschaften der Laufzeitumgebung werden in Abschnitt 4.3 spezifiziert.
Ein Endgerät wird in die Föderation aufgenommen, indem es zur Interaktion ausgewählt wird und somit die Nutzerschnittstelle auch durch dieses Endgerät dargestellt wird. Ebenso wird ein Endgerät aus der Föderation entlassen, wenn die Darstellung einer Nutzerschnittstelle nicht mehr gewünscht ist. Demnach besteht eine Föderation nur aus *aktiven* Endgeräten, es ist nicht vorgesehen, ein Endgerät *passiv* an einer Föderation teilhaben zu lassen.

2.3. Anforderungen an die Interaktionsanwendung

Die Interaktionsanwendung (IA) ermöglicht dem Nutzer die Interaktion mit der vom Nutzer eigentlich gewünschten Anwendung, der Zielanwendung (ZA). Damit ist die Interaktionsanwendung vergleichbar mit einem Web-Browser, der die Interaktion zwischen Nutzer und einer Web-Anwendung realisiert. Dieser Zusammenhang ist in Abbildung 2.3 dargestellt.

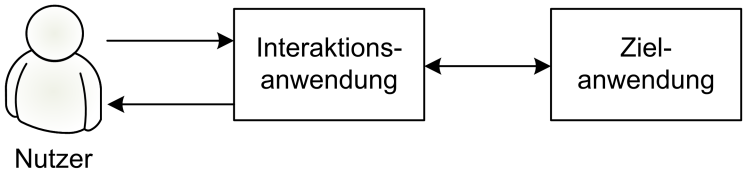

Abbildung 2.3.: Abstrakter Zusammenhang zwischen Nutzer, Interaktionsanwendung und Zielanwendung

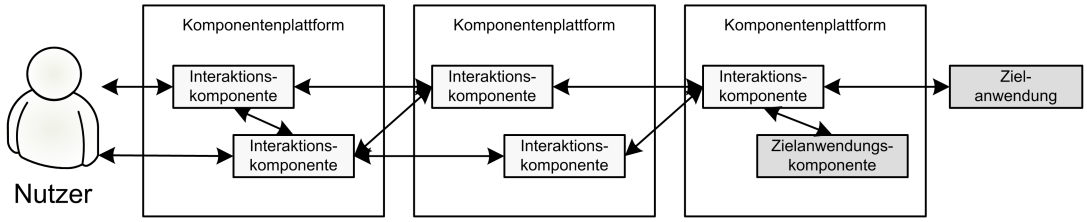

Abbildung 2.4.: Detaillierter Zusammenhang zwischen Nutzer, Interaktionsanwendung und Zielanwendung

Beim Einsatz föderierter Endgeräte erstreckt sich die Interaktionsanwendung über verschiedene Endgeräte und Server, die sowohl drahtgebunden als auch drahtlos miteinander kommunizieren können. Für eine solche verteilte Anwendung ist ein modularer oder komponentenbasierter Aufbau erforderlich, wobei die einzelnen Komponenten auf unterschiedlichen Komponentenplattforminstanzen zur Ausführung kommen und über ein Netzwerk miteinander kommunizieren können. Dabei müssen sich die Komponenten nicht notwendigerweise auf Endgeräten der Föderation befinden, da diese Endgeräte zunächst nur die Nutzerschnittstelle präsentieren. Beispielsweise zur Koordinierung der verteilten Nutzerschnittstellen oder wegen ressourcenintensiver Aufgaben können Komponenten auch auf Server-basierten Komponentenplattform ausgeführt werden. Zusammen mit der Komponentenplattform bildet die Interaktionsanwendung die Integrationsschicht. In Abbildung 2.4 ist beispielhaft dargestellt, wie sich die Interaktionsanwendung in Komponenten unterteilt und auf Komponentenplattformen verteilt ausgeführt wird. Die vom Nutzer zur Interaktion verwendete Komponente (z. B. ein Browser) muss von einer auf einem Endgerät vorhandenen Komponentenplattform dargestellt werden, wobei die anderen Komponenten (z. B. Integration) beispielsweise auf Servern zur Ausführung kommen können. Zielanwendungen in Form von Web-Anwendungen existieren ohnehin meist auf entfernten Servern.

IA-Anforderung 1 *Die Interaktionsanwendung muss durch* ***Modularität*** *gekennzeichnet sein, d. h. sie muss aus Komponenten bestehen, die u. U. auf verschiedene Komponentenplattformen verteilt zur Ausführung kommen können.*

Damit der Nutzer die größtmögliche Kontrolle über die Geräte in seiner Föderation hat, muss die Integrationsschicht eine Verwaltung der Föderation ermöglichen, sodass der Nutzer nach Bedarf Endgeräte aufnehmen oder entfernen kann. Dazu gehört, dass der Nutzer definieren kann, welche Endgeräte mit welcher Modalität benutzt werden sollen, sofern ein Endgerät unterschiedliche Modalitäten zur Verfügung stellt. Mit der Verwaltung müssen auch insbesondere Endgeräte angesprochen werden können, die über keine Eingabemöglichkeiten verfügen und demnach

„entfernt“ ausgewählt werden müssen. In [BCL07] wurde die Notwendigkeit einer manuellen bzw. expliziten Kopplung von Interaktionsressourcen beschrieben.

IA-Anforderung 2 *Die Integrationsschicht muss eine (manuelle)* ***Föderationsverwaltung*** *ermöglichen.*

Die Interaktionsanwendung erfordert eine hohe Robustheit gegenüber Ausfällen und Veränderungen, z. B. Nichtverfügbarkeit einzelner Komponentenplattformen und damit der darauf befindlichen Komponenten. Aus diesem Grund stellt eine zentrale Koordination der einzelnen Komponenten einen Single-Point-of-Failure dar, der beispielsweise durch einen dezentralen Ansatz zur Koordination umgangen werden kann. Durch die hohe Robustheit soll auch erreicht werden, dass das Entfernen von Endgeräten aus der Föderation die Föderation selbst nicht beeinflusst. Dieses Problem kann aber vorkommen, wenn ein zu entfernendes Endgerät Komponenten enthält, die für die übrige Föderation notwendig sind. Die Forderung nach hoher Robustheit ist außerdem darin begründet, dass die Interaktionsanwendung die Interaktion mit der Zielanwendung realisiert und somit vergleichbar mit Web-Browsern auf der Ausgabeseite, aber auch Tastatur oder Maus auf der Eingabeseite ist. Dementsprechend muss auch die Robustheit mit diesen Interaktionselementen vergleichbar sein, d. h. die Interaktionsanwendung muss verfügbarer als die Zielanwendung selbst sein, da sonst eine nutzerfreundliche Bedienung nicht gewährleistet ist.

IA-Anforderung 3 *Die* ***Robustheit*** *der Interaktionsanwendung gegenüber Ausfällen von Komponenten, z. B. durch Veränderungen in der Plattformverfügbarkeit, muss gewährleistet sein.*

Durch die Interaktionsanwendung werden Informationen vom Nutzer zur Zielanwendung transportiert. Daher muss dieser Weg gegen Sicherheitsrisiken gesichert werden, indem z. B. die Kommunikation verschlüsselt wird. Die Sicherheit des Informationsaustauschs innerhalb der Interaktionsanwendung muss mindestens so sicher sein, wie der Informationsaustausch zwischen Interaktionsanwendung und Zielanwendung, beispielsweise zwischen Browser und Web-Anwendung.

IA-Anforderung 4 *Die Kommunikation innerhalb der Komponenten der Interaktionsanwendung muss die* ***Sicherheit*** *gegenüber Angriffen wie Abhören und Modifikation des Inhalts gewährleisten.*

Jegliche Nutzerinteraktion mit dem System durch eine Teilnutzerschnittstelle muss in den anderen Teilnutzerschnittstellen, die meist auf anderen Geräten existieren, reproduziert werden. Dazu gehören sowohl Eingaben, die den Dialogzustand beeinflussen (z. B. Navigation zwischen einzelnen Seiten), als auch solche, die lediglich zum Füllen von Eingabefeldern verwendet werden. Grobgranularere Synchronisierungen führen zu inkonsistenten Sichten auf die Anwendung durch

verschiedene Teilnutzerschnittstellen, daher muss die Synchronisierung von Eingaben auf *event*-Ebene erfolgen. Die Anforderung bezieht sich auf Ereignisse, die eine Auswirkung auf den Inhalt des Datenmodells des aktuellen Dialogs und somit auf andere Teilnutzerschnittstellen haben, wie z. B. Tastatureingaben in ein repliziertes Eingabefeld in zwei Browsern, wobei zusätzlich auch unsichtbare Ereignisse wie das Drücken der Umschalttaste eingeschlossen sind. Ist keine Auswirkung auf die anderen Teilnutzerschnittstellen vorhanden (wie z. B. beim Scrollen durch Tasten- oder Mauseingabe), muss auch kein Abgleich durchgeführt werden. Diese Auffassung von Synchronisierung auf *event*-Ebene entspricht in etwa der Definition aus [Wor03], welche sich an DOM (Document Object Model) Events orientert. Man kann auch von einer Synchronisierung auf Basis des Datenmodells sprechen, da alle zu synchronisierenden Änderungen sich auf Änderungen am Datenmodell der jeweiligen Anwendung beziehen.

IA-Anforderung 5 *Die Wahrung der* ***Konsistenz*** *von Nutzereingaben in verschiedenen Teilen der verteilten Nutzerschnittstelle muss auf* event*-Ebene erfolgen.*

Da der Nutzer eine möglichst zusammenhängende Darstellung der Anwendung haben möchte, muss eine Synchronisierung der Ausgabe erfolgen, sodass die Aktualisierung von Nutzerschnittstellen als Reaktion auf Änderungen am Dialogzustand möglichst gleichzeitig erfolgt. Damit soll die unterschiedliche Leistungsfähigkeit heterogener Endgeräte und Netzwerktechnologien ausgeglichen werden. Unterschiedliche Zeitpunkte der Aktualisierung einzelner Nutzerschnittstellenteile vermitteln dem Nutzer nicht das Bild von einer einheitlichen, wenn auch aufgeteilten Nutzerschnittstelle. Insbesondere bei aus verschiedenen Informationsströmen bestehenden Präsentationen ist die zeitliche Ausrichtung der einzelnen Ströme gegenseitig von großer Bedeutung für das Gesamtergebnis (z. B. Untertitel zu Video), was in [PFL+02] herausgestellt wurde.

IA-Anforderung 6 *Die Integrationsschicht muss einen Mechanismus zur* ***Synchronisierung*** *der Aktualisierung der verteilten Nutzerschnittstellen unterstützen.*

Um eine von der Zielanwendung unabhängige Interaktionsanwendung zu ermöglichen, ist es wichtig, dass die beiden Anwendungen nicht nur im Sinne der Modularität getrennt sind, sondern auch sonst keine Abhängigkeiten aufweisen. Für die Zielanwendung muss transparent bleiben, welche Art von Nutzerschnittstelle dem Nutzer tatsächlich zur Verfügung steht. Die Interaktionsanwendung darf also keine Voraussetzungen bezüglich der Interna der Zielanwendung haben.

IA-Anforderung 7 *Die Interaktionsanwendung muss* ***transparent*** *mit der Zielanwendung zusammenarbeiten, ohne dass dafür eine explizite Unterstützung durch die Zielanwendung vorausgesetzt wird.*

Die Flexibilität des modularen Ansatzes ermöglicht es, dass der Nutzer beliebige Modalitäten zur Interaktion benutzen kann. Dabei wird der Nutzer nur durch das eventuell beschränkte Vorhandensein von Hardware und seine persönlichen Aufnahmefähigkeiten begrenzt. Um die Flexiblität besonders auszunutzen, muss das System dynamische Änderungen der vom Nutzer zur Interaktion ausgewählten Modalitäten unterstützen, d. h. der Nutzer kann zur Laufzeit Modalitäten hinzufügen oder entfernen.

IA-Anforderung 8 *Die Interaktionsanwendung bzw. die Integrationsschicht muss* ***dynamische*** *Änderungen zur Laufzeit der vom Nutzer zur Interaktion verwendeten Modalitäten ermöglichen.*

Mit den vorgestellten Anforderungen IA-Anforderung 1 bis 8 wird ein detaillierter Rahmen vorgegeben, der für die sich daraus abzuleitenden Anforderungen an die Komponentenplattform und die Konzeption der Interaktionsanwendung verwendet wird.

2.4. Anforderung an eine Komponentenplattform

Aus den Anforderungen an die Interaktionsanwendung lassen sich Anforderungen an die Komponentenplattform ableiten, die die Komponenten ausführt, aus denen die Interaktionsanwendung besteht.
Aus IA-Anforderung 1 ergibt sich die Notwendigkeit, dass Komponenten dynamisch zur Laufzeit auf die entsprechenden Plattforminstanzen verteilt werden. Die Verteilung kann unterschiedliche Ursachen haben, z. B. Anfragen von Nutzern oder anderen Komponenten. Eine Komponentenplattform muss also die Möglichkeit bieten, Komponenten von außerhalb zu empfangen und lokal auszuführen. Die Verteilung von Komponenten zur Laufzeit ist notwendig, da nicht garantiert werden kann, dass eine Komponentenplattform alle möglichen Komponenten als lokale Kopie vorhalten kann.

KP-Anforderung 1 *Die Komponenten müssen zur Laufzeit auf die Komponentenplattformen* ***verteilt*** *und ausgeführt werden können.*

Die Robustheit aus IA-Anforderung 3 hat für die Komponentenplattform zur Folge, dass die Komponentenplattform Änderungen in der Verfügbarkeit einzelner Komponenten möglichst transparent für den Nutzer behandeln muss. Dafür sind Mechanismen notwendig, um die Verfügbarkeit einzelner Komponenten zu erhöhen. Eine Möglichkeit ist der dynamische Wechsel des Ausführungsorts einer Komponente zur Laufzeit. Das bedeutet, dass die Komponente auf eine neue Komponentenplattform migriert. Das kommt einem Neuplatzieren gleich, wobei aber zusätzlich der interne Zustand (die Sitzungsinformation) übernommen wird.

Die Komponentenplattform muss also die Möglichkeit bieten, empfangene Komponenten auszuführen und mit einem ebenfalls empfangenen Initialzustand zu versehen.

KP-Anforderung 2 *Die Komponentenplattform muss durch besondere Mechanismen die* ***Verfügbarkeit*** *von beliebigen Komponenten erhöhen. Beispielsweise müssen die Komponenten zwischen Komponentenplattformen unter Erhaltung der Sitzungsinformation migrieren können.*

Zur Laufzeit kann die Situation eintreten, dass eine Komponente die gewünschten QoS (Quality-of-Service)-Eigenschaften nicht mehr erfüllt, beispielsweise wenn die Integration von Eingaben auf einer höheren semantischen Ebene erfolgen soll, als es bis dahin der Fall war. Dann muss diese Komponente durch eine neue ausgetauscht werden können, ohne die Nutzerinteraktion zu unterbrechen. Es ist Aufgabe der Komponentenplattform, die auszutauschende Komponente zu ermitteln und den Austauschvorgang durchzuführen. Diese Komponenten müssen platziert und sowohl die ein- als auch die ausgehenden Verbindungen aktualisiert werden.

KP-Anforderung 3 *Die Komponentenplattform muss den* ***Austausch*** *von Komponenten zur Laufzeit untersützen.*

Aus den Anwendungsfällen und Kernfrage 1 ergibt sich zudem für die Komponentenplattform die Anforderung, dass diese auf unterschiedlichen Geräten verfügbar sein muss. Da mobile und stationäre Endgeräte erwähnt werden, muss die Komponentenplattform insbesondere auf Desktop-PCs als auch auf PDAs und Laptops verfügbar sein.

KP-Anforderung 4 *Die Komponentenplattform muss verteilt auf* ***heterogenen Geräten*** *wie Desktop-PCs, Laptops und PDAs verfügbar sein.*

Mit den KP-Anforderungen 1 bis 4 werden die Randbedingungen für die Komponentenplattform definiert, welche u. a. eine Mobilität des Codes und eine breite Verfügbarkeit als Ziel haben.

2.5. Anforderungen an die Zielanwendung

Damit die Zielanwendung entsprechend eingebunden bzw. adressiert werden kann, muss diese bestimmte Anforderungen an die Nutzerschnittstelle erfüllen. Daher wird in diesem Abschnitt zunächst eine Klassifizierung von Anwendungen nach der Art ihrer Beschreibung der Nutzerschnittstelle durchgeführt und hinsichtlich ihrer Eignung für multimodale und verteilte Nutzerschnittstellen analysiert (Unterabschnitt 2.5.1). Daraus werden dann Anforderungen an potentielle Zielanwendungen der zu entwickelnden Integrationsschicht abgeleitet (Unterabschnitt 2.5.2).

2.5.1. Klassifizierung von Anwendungen

Anwendungen bzw. Softwareprogramme lassen sich in Bezug auf die Beschreibung der zugehörigen Nutzerschnittstelle in zwei Kategorien unterteilen:

Explizite Nutzerschnittstellenbeschreibungen werden meist von verteilten Anwendungen erzeugt, bei denen eine physische Trennung von Geschäftslogik (engl. Business Logic, BL) und Nutzerschnittstelle (engl. User Interface, UI) vorhanden ist. Ein typisches Beispiel hierfür sind Web-Anwendungen. Die Geschäftslogik der Anwendung befindet sich auf dem Web-Server, während die Nutzerschnittstelle vom Client meist mittels eines Web-Browsers dargestellt wird.

Implizite Nutzerschnittstellenbeschreibungen kommen in monolitischen Anwendungen vor, bei denen es keine physische Trennung zwischen Geschäftslogik und Nutzerschnittstelle gibt. Dies ist beispielsweise bei Desktop-Anwendungen der Fall. Der Programmcode von Nutzerschnittstelle und Geschäftslogik liegt meist als Aggregat vor, welches nicht in seine Bestandteile aufgeteilt werden kann.

Diese Unterscheidung beruht auf dem Ziel, welches mit den beiden Architekturen verfolgt werden soll. Bei Web-Anwendungen existiert die Geschäftslogik nur *einmal* auf dem Web-Server, welche aber von *beliebig vielen* Nutzern verwendet werden kann. Daher muss die Nutzerschnittstelle in einem Format vorliegen, das die Übertragung und Darstellung auf entfernten Endgeräten ermöglicht. Bei Desktop-Anwendungen ist die Geschäftslogik auf jedem Client repliziert, der die Anwendung nutzen will. Es gibt daher *genauso viele* Instanzen der Geschäftslogik wie Nutzer vorhanden sind. Es ist zwar prinzipiell denkbar aber nicht notwendig, dass die Nutzerschnittstelle einer Desktop-Anwendung erst in ein u. U. standardisiertes Format transferiert und dann ausgeben wird.
In diesem Abschnitt werden die Unterschiede dieser beiden Anwendungskategorien am Beispiel von Desktop- bzw. Web-Anwendungen in Bezug auf das MMI-F (MultiModal Interaction Framework) analysiert. Das MMI-F definiert die Strukturierung eines Systems durch verschiedene Rollen für Ein- und Ausgabeverarbeitung entsprechend der Modalitäten. Es wird in Unterabschnitt 3.2.1 näher erläutert.

Desktop-Anwendungen Bei Desktop-Anwendungen ist die Nutzerschnittstelle und die Geschäftslogik direkt integriert, weil die gesamte Anwendung lokal auf einem Rechner installiert ist. Dabei kann die Datenhaltung entweder ebenfalls integriert sein oder durch externe Quellen realisiert werden.
Bezogen auf das MMI-F ergibt sich für Desktop-Anwendungen, dass alle Rollen des MMI-F von der Desktop-Anwendung bzw. dem Betriebssystem und zugehörigen Treibern realisiert werden, wie in Abbildung 2.5 dargestellt ist. Die Erkennung von Tastatur- und Mauseingaben (Recognition Tastatur `Rec T` und Maus

`Rec Ma`) sowie deren partielle Integration `IG` wird vom Betriebssystem bzw. den entsprechenden Treibern übernommen. Das Anwendungsframework übernehmen die Interpretation der Eingaben (Interpretation Tastatur `IP T` und Maus `IP Ma`) und führt eine weitere Integration durch (`IG`). Darüber hinaus ist das Framework für die Generierung `Gen` und Aufbereitung (Styling Monitor `Sty Mo` und Lautsprecher `Sty L`) der Anwendungsausgabe zuständig. Die Anwendung übernimmt schließlich die Interaktionskontrolle `Int`, die Anwendungslogik `App`, die Sitzungsverwaltung `Ses` und Systeminformation `Sys`, wobei die letzten beiden Aufgaben auch teilweise durch das Framework realisiert werden können. Falls eine Desktop-Anwendung kein zugrundeliegendes Framework besitzt, dann ist die Anwendung selbst für die Umsetzung der entsprechenden MMI-F-Rollen zuständig. Schließlich übernimmt das Betriebssystem bzw. die Treiber die eigentliche Ausgabe der Anwendung (Rendering Monitor `Ren Mo` und Lautsprecher `Ren L`).
Auch wenn diese Rollen aus Sicht eines Softwareentwicklers in der Implementierung der Anwendung oder des Frameworks sauber getrennt sind, so ist es nur mit großem Aufwand möglich, die Anwendung um zusätzliche Modalitäten für Eingabe oder Ausgabe zu erweitern. Die Gründe dafür sind:

1. Die implizite und verteilte Realisierung der Integration-Komponente. In einer Java-Anwendung beispielsweise bekommt die Anwendung Mausereignisse, die bereits teilweise mit den Ereignissen der Tastatureingabe zusammengeführt worden sind (z. B. Alt-, Umschalt- oder Steuerungstaste). Hierbei erfolgt die syntaktische Fusion (Integration) der Eingaben durch die Java-Laufzeitumgebung, wobei die semantische Interpretation durch die Anwendung implizit realisiert wird. Soll die Fusion auf Laufzeitumgebungsebene rekonfiguriert werden (z. B. Deaktivieren der Integration von Maus- und Tastatureingaben), so ist dies nur durch Modifikationen an der Laufzeitumgebung möglich.

2. Die implizite Generierung von Nutzerschnittstellen durch die Anwendung oder das Anwendungsframework. Eine Desktop-Anwendung erzeugt keine explizite Nutzerschnittstellenbeschreibung, die dann an die jeweiligen Ausgabemodalitäten angepasst werden kann, sondern generiert die Nutzerschnittstelle direkt. Daher ist es nicht möglich, weitere Ausgabemodalitäten ohne Modifikation der Anwendung oder des Frameworks hinzuzufügen.

Daraus ergibt sich, dass es nur mit erheblichem Aufwand möglich ist, zusätzliche Modalitäten für Eingabe und Ausgabe zu bestehenden Anwendungen hinzuzufügen, weil entweder die Anwendung, deren Laufzeitumgebung oder beides modifiziert werden muss ([KM07, Mül06]). Eine Modifikation der Laufzeitumgebung ist schwierig, da hier, neben dem erheblichen Umfang, die Ereignisse zu wenig semantische Information besitzen und somit nur auf syntaktischer Ebene verarbeitet werden können.

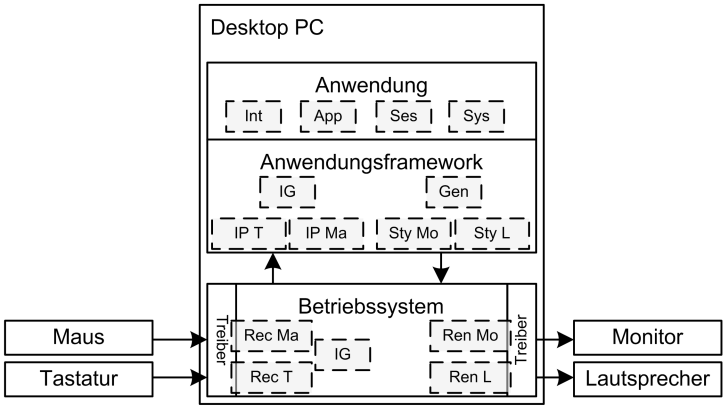

Abbildung 2.5.: Abbildung der Rollen des MMI-F auf Anwendungen mit Desktop-Architektur

Um dennoch beispielsweise multimodale Eingaben in bestehende Anwendungen zu ermöglichen, ohne die Anwendungen modifizieren zu müssen, gibt es verschiedene Ansätze. Die Idee von Wang et al. [WM03] sieht ein theoretisches Modell vor, mit dessen Hilfe die Wertebereiche von Eingabegeräten aufeinander abgebildet werden können. Damit ist es beispielsweise möglich, Eingaben eines zusätzlich zu unterstützenden Endgerätes auf Tastaturereignisse abzubilden. Durch entsprechende Abbildungsmodelle kann so jedes beliebige Eingabegerät durch bestehende, alternative Eingabegeräte realisiert werden. Ein alternativer Ansatz von Carter et al. [CHML06] unterstützt multimodale Eingabegeräte für Java Swing Anwendungen [LEW02], indem die Anwendung nach Interaktionselementen durchsucht wird, für die kein passendes Eingabegerät vorhanden ist. Diese Interaktionselemente werden dann durch andere Interaktionselemente ersetzt, die besser mit den verfügbaren Eingabegeräten verwendet werden können. Beide Ansätze sind Restriktionen unterworfen. Ersterer bezieht sich nur auf Eingaben, während der zweite Ansatz nur für Swing-Anwendungen geeignet ist.

Web-Anwendungen Bei Web-Anwendungen ist der Ausführungsort der Nutzerschnittstelle bereits per Definition von dem der Geschäftslogik und der Daten getrennt, was zu einer klaren Trennung der Rollen des MMI-F führt, sodass eine Erweiterung der Interaktion um zusätzliche Modalitäten einfacher erfolgen kann (Abbildung 2.6). Der Browser realisiert primär die Ein- und Ausgaberollen des MMI-F, was der View-Rolle des MVC (Model-View-Controller) entspricht. Die Geschäftslogik und die Datenhaltung werden durch den Server realisiert und entsprechen somit dem Controller und dem Model des MVC. Dabei wird von herkömmlichen Web-Anwendungen ausgegangen, die ohne client-seitige Verwendung von JavaScript auskommen. Bei Einsatz von AJAX (Asynchronous JavaScript and XML) bzw. RIA (Rich Internet Applications) allgemein befinden sich Teile

der Geschäftslogik auf dem Client und werden vom Browser realisiert, sodass hierbei die klare Trennung der MVC-Rollen nicht mehr gegeben ist. In diesen Fällen muss eine detailliertere Betrachtung der MVC-Rollen durchgeführt werden, um die dafür zusätzlich notwendigen Aspekte in Bezug auf verteilte Nutzerschnittstellen zu ermitteln.
Da der Geschäftslogikteil und der Nutzerschnittstellenteil über ein standardisiertes Protokoll wie beispielsweise HTTP (HyperText Transfer Protocol) verbunden sind und die Nutzerschnittstellen explizit in einer standardisierten Form wie beispielsweise HTML (HyperText Markup Language) vorliegen, ist es ohne großen Aufwand möglich, zu der existierenden Geschäftslogik alternative Nutzerschnittstellen (Views) zu erzeugen bzw. die Eingabe und Ausgabe entsprechend multimodaler Interaktionen zu manipulieren. So ist es beim Interagieren mit Web-Anwendungen leicht möglich, einen Proxy zwischen Nutzerschnittstelle und Anwendungslogik zu platzieren, der nicht nur wie bisher den indirekten Zugriff auf die Web-Anwendung ermöglicht, sondern auch eine explizite Unterstützung für die parallele Darstellung der Nutzerschnittstelle auf zusätzlichen Endgeräten ermöglicht. Dieser Proxy ist dann für die Interaktionskontrolle mit der Anwendung, die Generation der Ausgabe sowie teilweise für die Integration verantwortlich.

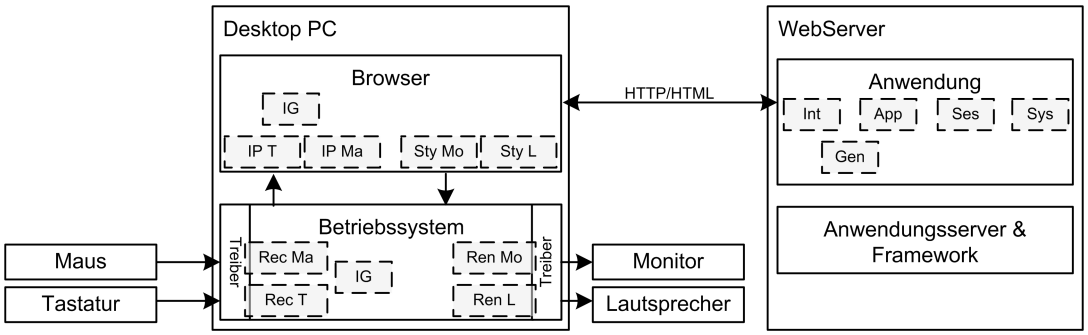

Abbildung 2.6.: Abbildung der Rollen des MMI-F auf Anwendungen mit Web-Architektur

2.5.2. Ergebnis des Vergleichs

Da Web-Anwendungen die Geschäftslogik und die Nutzerschnittstelle durch eine konkret definierte Schnittstelle voneinander trennen, bieten Sie umfangreiche Möglichkeiten zur Erweiterung um Ein- und Ausgabemodalitäten. Aus diesem Grund bezieht sich die weitere Arbeit auf Web-Anwendungen. Daraus folgt eine wesentliche Anforderung an Anwendungen, die multimodalen Zugriff ermöglichen sollen: Die Anwendung muss über eine explizite Nutzerschnittstellenbeschreibung verfügen, die von der Geschäftslogik getrennt ist und daher von dieser unabhängig zur Nutzerschnittstellendarstellung verwendet werden kann. Auf diese Weise las-

	Desktop-Anwendungen	Web-Anwendungen
Trennung von BL und UI	nein	ja
UI Beschreibung	implizit (Framework)	explizit (z. B. HTML)
Aufwand zur Erweiterung um weitere Modalitäten	hoch	niedrig

Tabelle 2.1.: Vergleich von Desktop-Anwendungen mit Web-Anwendungen

sen sich verschiedene Implementierungen der gleichen Nutzerschnittstelle auf die gleiche Anwendungslogik aufsetzen, wodurch eine flexible Erweiterung möglich wird.

ZA-Anforderung 1 *Eine Anwendung, die um multimodalen Zugriff durch föderierte Endgeräte erweitert werden soll, muss modular aufgebaut sein und eine* ***Trennung*** *von Geschäftslogik und Nutzerschnittstelle vorweisen. Diese Trennung muss durch klar definierte Protokolle und Inhaltsformate beschrieben sein.*

Für die Verwendung durch die Integrationsschicht ist nicht wichtig, ob die Zielanwendung als Komponente in der Integrationsschicht existiert oder auf einer externen Einheit (z. B. einem Web-Server) ausgeführt wird. Einer der wesentlichen Unterschiede ist, dass im Falle einer externen Anwendung die Föderation stets online sein muss, da die Anwendung nicht auf die Endgeräte verschoben werden kann.

2.6. Relation zwischen Kernfragen und Anforderungen

Die erste Kernfrage der Arbeit (siehe Abschnitt 1.3) bezieht sich auf benötigte Konzepte im Protokollbereich. Die Konzepte sollen zur nutzerfreundlichen Realisierung von Geräteföderationen zum gemeinsamen Zugriff auf Anwendungen beitragen. Ein wesentliches Konzept hierbei ist die Modularisierung und die beliebige Verteilung der Module in einem Netzwerk, wobei die Verteilung dynamisch zur Laufzeit angepasst werden kann (IA-Anforderung 1). Die Forderung nach Sicherung des Informationsaustausches ist ebenfalls als Konzept im Protokollbereich zu sehen (IA-Anforderung 4), wobei dadurch nur die Sicherheitsanforderungen abgedeckt werden, die jedoch nicht direkt zur Bereitstellung der Funktionalität des Gesamtsystems beitragen. Die Nutzerfreundlichkeit wird einerseits durch die nutzergesteuerte Föderationsverwaltung (IA-Anforderung 2), andererseits durch die Konsistenz (IA-Anforderung 5) und Synchronisierung der verteilten Nutzerschnittstelle realisiert (IA-Anforderung 6). Durch die Robustheit

Anforderung	Kernfrage 1	Kernfrage 2	Kernfrage 3
IA-Anf. 1 - Modularität	x		x
IA-Anf. 2 - Verwaltung	x		
IA-Anf. 3 - Robustheit	x		
IA-Anf. 4 - Sicherheit	x		
IA-Anf. 5 - Konsistenz	x		
IA-Anf. 6 - Synchronisierung	x		
IA-Anf. 7 - Transparenz	x		x
IA-Anf. 8 - Dynamik	x		
KP-Anf. 1 - Verteilung		x	
KP-Anf. 2 - Verfügbarkeit		x	
KP-Anf. 3 - Austausch		x	
KP-Anf. 4 - Heterogene Geräte		x	
ZA-Anf. 1 - Trennung BL & UI			x

Tabelle 2.2.: Gegenüberstellung der ermittelten Anforderungen zu den Kernfragen der Arbeit aus Abschnitt 1.3

(IA-Anforderung 3) und dynamische Änderbarkeit verwendeter Modalitäten (IA-Anforderung 8) soll ebenfalls sichergestellt werden, dass das System nutzerfreundlich verwendbar ist. IA-Anforderung 7 befasst sich mit der Nutzerfreundlichkeit für Anwendungsentwickler. Nichtsdestotrotz hat der Endnutzer bei Umsetzung dieser Anforderung den Vorteil, dass er bei der Auswahl von Modalitäten nicht durch die Zielanwendung eingeschränkt wird.

Die zweite Kernfrage adressiert Fragestellungen in Zusammenhang mit der Umsetzung des Systems mittels Komponenten in einer Komponentenplattform. Die Anforderungen aus Abschnitt 2.4 sehen vor, dass die einzelnen Komponenten nach Bedarf verteilt (KP-Anforderung 1) werden können. Außerdem muss die Komponentenplattform dafür sorgen, dass die Komponenten eine hohe Verfügbarkeit aufweisen, was z. B. durch Komponentenmigration realisiert werden kann (KP-Anforderung 2).

Entsprechend der in Abschnitt 1.3 aufgestellten dritten Kernfrage soll untersucht werden, welche Kriterien erfüllt sein müssen, damit eine Zielanwendung durch verteilte und multimodale Nutzerschnittstellen so verwendet werden kann, dass die verteilte Nutzerschnittstelle für die Zielanwendung transparent ist. Aus der Untersuchung in Unterabschnitt 2.5.1 geht hervor, dass die Zielanwendung eine Trennung zwischen Nutzerschnittstelle und Applikationslogik mit definierten Schnittstellen und Dokumentformaten aufweisen muss (ZA-Anforderung 1).

In Tabelle 2.2 werden die Kernfragen den in diesem Kapitel aufgestellten Anforderungen gegenübergestellt.

Aus diesen Anforderungen ergibt sich, dass eine Integrationsschicht benötigt wird, die auf flexible Weise Nutzerschnittstellen erzeugen kann, die sich über verschiedene Endgeräte erstrecken und in verschiedenen Modalitäten vorliegen. Dabei ist eine hohe Flexibilität erforderlich, weshalb die modalitätsspezifischen Aspekte auf möglichst wenige Komponenten beschränkt werden müssen. Aus der Verwendung als Nutzerschnittstellen-Integrationsschicht ergeben sich einige Anforderungen, die für herkömmliche Komponentenplattformen keine oder eine untergeordnete Rolle spielen.
Im folgenden Kapitel werden u. a. verwandte Arbeiten verglichen. Dabei wird analysiert, inwiefern diese die hier gestellten Anforderungen erfüllen.

Immer, wenn man die Meinung der Mehrheit teilt,
ist es Zeit, sich zu besinnen.

Mark Twain

3 Verwandte Arbeiten und Stand der Technik

In diesem Kapitel wird zunächst in Abschnitt 3.1 eine Bestimmung von wichtigen Begriffen vorgenommen. In Abschnitt 3.2 wird vorgestellt, wie das W3C (World Wide Web Consortium) mit der Problematik der multimodalen Interaktion im Rahmen der Multimodal Interaction Working Group umgeht. Im Anschluss daran werden in Abschnitt 3.3 vergleichbare Ansätze vorgestellt und bewertet, die ebenfalls eine multimodale Interaktion durch gleichzeitige Verwendung verschiedener Endgeräte zum Ziel haben. In Abschnitt 3.4 wird ein verwandter Ansatz bezüglich der Synchronisierung verteilter Nutzerschnittstellen erläutert. In Abschnitt 3.5 werden Komponentenplattformen vorgestellt und in Beziehung zu den Anforderungen gesetzt.

3.1. Begriffsbestimmung

Für das korrekte Verständnis der weiteren Arbeit ist es zunächst notwendig, einige zentrale Begriffe genau zu bestimmen. Im Folgenden wird daher definiert, was im Rahmen dieser Arbeit unter *Modalität* und *Multimodalität* verstanden wird, was der Unterschied zu *Multimedia* ist und wie der Term *multi-device* bisweilen sehr unterschiedlich und daher ggf. missverständlich in der Literatur gebraucht wird.

3.1.1. Modalität

Bernsen [BD03, Ber02] unterscheidet zwischen Medium und Modalität, wobei Medium die physische Realisierung von Informationsrepräsentationen an der Schnitt-

stelle zwischen Mensch und System ist und sich auf die Vorstellung von sensorischen Modalitäten des Menschen wie fühlen, sehen, hören, riechen, schmecken bezieht. Als begriffliche Modalität sieht Bernsen die Art und Weise, wie über ein Medium Informationen zwischen Mensch und Mensch bzw. System ausgetauscht werden können.
Dix et al. [DFAB04] versteht die menschlichen Sinne als Modalitäten, welche auch als Kommunikationskanäle bezeichnet werden können. Dabei unterscheiden Dix et al. zusätzlich Kommunikationskanäle des Systems und des Menschen, wobei sie sich auf Systemseite auf die jeweiligen Geräte beziehen und auf Seite des Menschen die Sinne als Kommunikationskanal betrachtet werden. Durch das Zusammenspiel zweier korrespondierender Kommunikationskanäle auf Seiten des Sytems und des Menschen wird die Kommunikation durchgeführt.
Coutaz [CC91] definiert Modalität als die genaue Form wie ein Gedanke wiedergegeben wird bzw. die Art und Weise, wie eine Aktion ausgeführt wird. Aufbauend auf dieser Definition bezeichnet Nigay [NC93] Modalität als die Art des Kommunikationskanals, über den Information transportiert wird. Zusätzlich unterscheidet Nigay zwischen Modalität (engl. 'modality') und Modus (engl. 'mode'), wobei durch den Modus der Zustand bestimmt wird, wie Information interpretiert wird, um eine Bedeutung zu extrahieren. In anderen Worten ausgedrückt bezeichnet Modalität die Art der ausgetauschten Daten und Modus den Interpretationskontext.
Maybury [May03] unterscheidet ebenfalls zwischen Medium (engl. 'medium') und Modalität, verwendet die englischen Begriffe 'mode' und 'modality' aber äquivalent. Ein Medium ist für Maybury ein Informationsträger wie Text, Grafiken oder Video. Er bezieht sich bei Modalität auf die verarbeitenden Sinne des Empfängers, welche durch optische, auditive oder haptische Transformation die Informationen des Mediums interpretieren. Die Verbindung von Modalität und menschlichem Sinn lehnt sich an das 1824 von Johannes Peter Müller definierte Gesetz der spezifischen Sinnesenergien. Hier hat Müller definiert, dass nicht der äußere Reiz die Qualität der Wahrnehmung bestimmt, sondern die Eigenart des gereizten Sinnesorgans [Her87, Aut87].
In dieser Arbeit orientiert sich die Definition für Modalität an den Definitionen von Bernsen und Coutaz/Nigay:

> Eine **Modalität** ist die Art und Weise, wie Informationen zwischen zwei Menschen bzw. zwischen Mensch und Maschine durch Interaktion übertragen wird.

Daher wird jede Modalität durch ein Interaktionsschema repräsentiert, welches die zum Informationsaustausch notwendige Interaktion zwischen zwei Interaktionspartnern beschreibt. Diese Definition von Modalität bezieht sich insbesondere im Kontext der Mensch-Maschine-Interaktion sehr stark auf die Interaktion mit Ein- und Ausgabegeräten, da die Art und Weise des Informationsaustausches stets durch die verwendeten Endgeräte definiert wird.

Wendet man diese Definitionen für Modalität auf einen herkömmlichen PDA an, bei dem Ausgabe per Display und Eingabe per Stift möglich ist, dann ergeben sich unterschiedliche Klassifikationen. Über Stift und Display werden auf unterschiedliche Weise Informationen in unterschiedlichen Medien ausgetauscht (Bernsen). Die Art der ausgetauschten Informationen bei Stift und Display ist ebenfalls unterschiedlich (Coutaz, Nigay). Die haptische Eingabe und optische Ausgabe gehören zu unterschiedlichen Sinnen (Maybury, Dix et al.). Demnach unterstützt ein PDA unterschiedliche Modalitäten entsprechend der vorgenannten Definitionen.

3.1.2. Multimodalität

Die vorgenannten Autoren haben neben Modalität auch Definitionen für Multimodalität erstellt, die im Folgenden näher betrachtet werden.
Nach Bernsen stellt ein multimodales System zwei oder mehr Modalitäten zur Verfügung [BD03, Ber02].
Dix et al. definieren ein multimodales interaktives System als ein System, dass auf der Verwendung multipler menschlicher Kommunikationskanäle beruht. In [DFAB04] schreiben die Autoren explizit, dass jedes interaktive System multimodal ist, da man immer haptische und visuelle Kanäle benutzt, um einen Computer zu bedienen.
Coutaz und Nigay definieren ein multimodales System als ein System, das mit dem Nutzer über verschiedene Kanäle kommunizieren und daraus automatisch eine Bedeutung ableiten kann [CC91, NC93].
Die Definition von Maybury sieht nur multiple Modalitäten vor, wobei jedoch aus [May03] nicht genau hervorgeht, ob ein multimodales System noch an weitere Anforderungen gebunden ist.
Auf das zuvor beschriebene Beispiel eines PDAs angewendet, ergibt sich laut Bernsen und Maybury, dass es sich bei einer Anwendung, die alle Möglichkeiten des PDAs ausnutzt, um eine multimodale Anwendung bzw. ein multimodales System handelt. Nach Dix et al. muss ein multimodales System multiple menschliche Kommunikationskanäle verwenden, was der PDA durch optische Ausgabe und haptische Eingabe erfüllt. Die systemzentrische Definition von Coutaz und Nigay setzt verschiedenartige Kommunikationskanäle in Richtung des Gerätes voraus, wobei der PDA nur Stifteingabe zur Verfügung stellt, sodass es sich unter dieser Betrachtung nicht um ein multimodales System handelt. Da Maybury Multimodalität durch Vorhandensein multipler Modalitäten definiert, handelt es sich bei dem PDA um ein multimodales System.
Oviatts Definition [Ovi02] eines multimodalen Systems ist vergleichbar mit der von Coutaz/Nigay und unterscheidet zwischen Systemeingabe und -ausgabe. Oviatt definiert ein System nur dann als multimodal, wenn es zwei oder mehr kombinierte Eingabemodalitäten (engl. 'mode', z. B. Sprache, Stift, Berührung, Geste, Blick, Kopf- und Körperbewegungen) verarbeitet und die Eingaben mit

multimedialer Ausgabe koordiniert. Der eben beschriebene PDA würde daher nicht als multimodales System klassifiziert werden. Diese Definition stimmt mit der intuitiven Auffassung von multimodalen Systemen überein. Ein PDA wird nicht dadurch multimodal, dass zusätzlich zur optischen Ausgabe noch akustische Ausgabe ermöglicht wird. Erst durch Unterstützung multimodaler Eingaben wird ein PDA als multimodal angesehen, ungeachtet seiner verschiedenen Ausgabemöglichkeiten.

Das Problem der Definitionen für Multimodalität von Bernsen, Dix et al., Coutaz und Maybury ist, dass sie nicht zwischen Eingabe und Ausgabe unterscheiden, denn laut der Definition von Oviatt wird ein System erst durch Unterstützung von multimodalen Eingaben des Nutzers multimodal. Demnach muss die Definition für Multimodalität einschließen, dass ein multimodales System unabhängig von der Ausgabe zumindest multimodale Eingabe unterstützen muss.

Beispielsweise kann die Positionierung eines Cursors mittels Maus, Tastatur, Trackball und Touchpad erfolgen. Die unterschiedlichen Geräte stellen dem Nutzer jedoch ein unterschiedliches Interaktionsschema zu Verfügung, woraus folgt, dass sie unterschiedliche Modalitäten verkörpern, obwohl bei der Interaktion das selbe Ziel verfolgt wird. Als Beispiel für unterschiedliche Interaktionsschemata der Informationsausgabe dienen ein Monitor und Drucker, die beide graphische Ausgabe ermöglichen, jedoch sich hinsichtlich ihrer Interaktionsschemata unterscheiden.

Die in der Arbeit verwendete Definition für Multimodalität orientiert sich an der Definition von Oviatt und dem allgemeinen Verständnis von Multimodalität:

> Ein System ist **multimodal**, wenn es dem Nutzer mehr als eine Modalität zur Eingabe und mindestens eine Modalität zur Ausgabe von Informationen anbietet.

Entsprechend der Definitionen für Modalität und Multimodalität sind alle PCs mit Tastatur, Maus und Bildschirm bereits multimodale Systeme, obwohl in der Literatur meist ein PC erst nach Einbeziehung akustischer Ein- und/oder Ausgabemöglichkeiten (zusätzlich zur optischen Ausgabe und taktilen Eingabe) als multimodal bezeichnet wird. Die Definition ist unabhängig von hierarchisch gegliederten Modalitäten und dem Einfluss des Kontextes auf die Multimodalität, in dem die Modalitäten verwendet werden. Jedoch können Kontextinformationen über die Modalitäten verwendet werden, um beispielsweise die Synchronisierung zu kontrollieren. Unimodal sind demnach alle Systeme, die nur eine Eingabemodalität unterstützen, ungeachtet der Anzahl an Ausgabemodalitäten. Beispiele für unimodale Systeme sind Sprachanwendungen mit Sprachein- und -ausgabe und PDAs, die Eingaben nur per Stift ermöglichen.

3.1.3. Abgrenzung zu Multimedia

Sowohl multimodale als auch multimediale Systeme sind dadurch charakterisiert, dass sie Ein- und Ausgabe in mehreren Modalitäten zulassen. Ein multimodales System führt allerdings eine Interpretation aller Eingabemodalitäten durch, während ein multimediales System nur den Datenstrom als solchen transportiert bzw. verarbeitet, ohne eine Semantik zu extrahieren [NC93]. Beispielsweise wird ein VoIP-Client nicht dadurch multimodal, dass er Sprache aufnehmen und versenden kann. Wird die Sprache aber interpretiert und kann auch zur Steuerung der Anwendung bzw. des Systems benutzt werden, spricht man von einem multimodalen System.

3.1.4. Multi-Device

Dieser Begriff wird in der Literatur in unterschiedlichen Bedeutungen verwendet, die im Folgenden gegenüber gestellt werden.
Einerseits versteht man darunter das Single-Authoring-Konzept, wobei eine abstrakte Nutzerschnittstelle definiert wird, die als Grundlage für die Transformation zu Nutzerschnittstellen verschiedener Endgeräte dient [MPS04, GZ02, GY03, Ric05, SM03, SMB05]. Der Vorteil ist ein geringerer Entwicklungsaufwand, da nicht für jedes Endgerät eine spezielle Anwendung entwickelt werden muss. Die gemeinsame Nutzung multipler Endgeräte wird damit nicht bezeichnet. Folglich bedeutet in diesem Zusammenhang *multi-device* so viel wie *geräteunabhängig*.
Andererseits wird beispielsweise eine Nutzerschnittstelle als *multi-device user interface* bezeichnet, wenn sie sich über mehrere Endgeräte erstreckt [HPN00, BM04, DRONP01, Sin03]. In diesem Fall wird die gemeinsame Nutzung einer Vielzahl von Endgeräten gleichzeitig bezeichnet, sodass *multi-device* im Sinne von *federated* oder *collaborative devices* verwendet wird.
Da sich die vorliegende Arbeit mit föderierten Endgeräten und auf selbige verteilte Nutzerschnittstellen befasst, bezieht sie sich auf die letztgenannte Bedeutung.

3.1.5. Föderierte Endgeräte

Eine Menge von Enderäten, die kooperativ und gleichzeitig die Nutzerschnittstelle einer Anwendung darstellen, werden als föderierte Endgeräte bezeichnet [BM05]. Genauer gesagt, sind föderierte Endgeräte jeweils voneinander unabhängig, werden jedoch zur Erfüllung eines bestimmten Zwecks gemeinsam benutzt bzw. kombiniert. Dazu teilen sie sich die Sitzung, die den Nutzer bei der Anwendung identifiziert. Durch Hinzunahme zu einer Föderation wird ein Gerät in die Sitzung aufgenommen, beim Verlassen der Föderation wird auch die Sitzung des Geräts beendet. Somit ist es auch möglich, eine Sitzung an ein anderes Gerät zu übertragen. Sind zwei Endgeräte föderiert, beispielsweise ein PDA und ein Laptop, kann wahlweise eines der beiden oder beide Geräte gleichzeitig zur

Interaktion benutzt werden. Prinzipiell lassen sich beliebig viele Endgeräte zu einer Föderation zusammenschließen.

3.1.6. Synchronisierung

Laut [SSW96] bezeichnet der Begriff „Synchronismus“ die Gleichzeitigkeit, Gleichlauf bzw. zeitliche Übereinstimmung zweier oder mehrerer Ereignisse. „Synchronisierung“ ist der Vorgang, wodurch der Synchronismus erreicht wird.
Insbesondere in der Informatik können jedoch unterschiedliche Bedeutungen hinter dem Begriff Synchronisierung verborgen sein, die sich jedoch auf abstrakter Ebene der oben angeführten Definition zuordnen lassen: die *inhaltliche* und die *zeitliche* Synchronisierung (auch wenn „zeitliche Synchronisierung“ wie ein Pleonasmus erscheint, da es das griechische Wort für Zeit *chrónos* enthält).

Inhaltliche Synchronisierung Diese Bedeutung wird bevorzugt im Datenbankbereich verwendet. Es handelt sich um inhaltliche Synchronisierung, wenn replizierte Datenbestände abgeglichen werden, sodass sie einen konsistenten Zustand einnehmen. Hierbei ist Synchronismus im Sinne von Gleichlauf gemeint, da nach Abschluss der Synchronisierung die replizierten Datenbestände den gleichen Inhalt ohne direkten Bezug zu einer bestimmten Zeit haben.

Zeitliche Synchronisierung Hiermit wird das gleichzeitige Eintreten von Ereignissen bezeichnet, wofür die zu synchronisierenden, unabhängigen Vorgänge entsprechend angeglichen werden müssen. Entsprechend obiger Definition sind die Begriffe Gleichzeitigkeit und zeitliche Übereinstimmung am Besten für die Charakterisierung zeitlicher Synchronisierung geeignet.

Darüber hinaus werden in der Informatik auch Prozesse und Uhren synchronisiert, worauf sich die obige Klassifizierung z. T. anwenden lässt (Prozesse werden zeitlich synchronisiert; der Zähler der Uhren wird auf den gleichen Stand gebracht und somit inhaltlich synchronisiert). Sofern es nicht explizit anders bezeichnet wird, bezieht sich in der weiteren Arbeit die Synchronisierung verteilter Nutzerschnittstellen auf die zeitliche Synchronisierung. Hierbei ist insbesondere das gleichzeitige Erscheinen der einzelnen Teile der verteilten Nutzerschnittstelle das Ziel (siehe Abschnitt 4.6).

3.2. W3C Multimodal Interaction Activity

Die Multimodal Interaction Activity [Wor02a] strebt nach einer Erweiterung des Internets, die Nutzern die dynamische Wahl der jeweils angemessenen Modalitäten gemäß der momentanen Bedürfnisse ermöglicht, was auch Menschen mit

Behinderungen einschließt und den barrierefreien Zugriff auf Informationen ermöglichen soll. Dabei sollen die Entwickler möglichst effektiv Nutzerschnittstellen für jegliche vom Nutzer wählbare Modalitäten erzeugen können. Abhängig vom Endgerät wird der Nutzer Informationen mittels Sprache, handschriftlich oder durch Tasten eingeben können, während die Ausgabe über Displays, Sprache, Audio oder taktile Mechanismen wie Vibrationseinheiten von Mobiltelefonen oder Braille-Zeilen erfolgen kann.
Im Rahmen der Multimodal Interaction Activity sind verschiedene Dokumente und Teilergebnisse entstanden, die einerseits Entwicklern beim Umsetzen multimodaler Architekturen helfen sollen und andererseits Inhaltsformate definieren, die für die Kommunikation zwischen einzelnen Komponenten einer multimodalen Architektur Verwendung finden.
Angefangen hat die Tätigkeit der MMI Working Group mit Anwendungsfällen für und Anforderungen an multimodale Interaktion. Kurz darauf wurden Anforderungen an EMMA (Extensible Multimodal Annotation Markup Language) und InkML (Ink Markup Language) ebenfalls als *W3C Note* veröffentlicht (Januar 2002). InkML dient als Datenformat für die Beschreibung von Eingaben (z. B. Schreibspuren), die mittels elektronischer Tinte erzeugt wurden.
Neben den Auszeichnungssprachen hat die MMI Working Group auch verschiedene Dokumente als *Note* bzw. *Working Draft* veröffentlicht, die sich mit der Architektur von multimodalen Systemen befassen, wie z. B. das in Unterabschnitt 3.2.1 beschriebene „Multimodal Interaction Framework“ oder das Dokument zu „Multimodal Architecture and Interfaces“ (Unterabschnitt 3.2.2). Die Verwendung von EMMA wird in Unterabschnitt 3.2.3 näher erläutert.

3.2.1. MultiModal Interaction Framework

Das MMI-F [Wor02b] beschreibt auf abstrakter Ebene, welche Komponenten mit welchen Funktionen durch ein multimodales System zur Verfügung gestellt werden. Obwohl in der Beschreibung des MMI-F stets die Rede von Komponenten ist, wird im Weiteren der Begriff *Rolle* verwendet, wenn Komponenten des MMI-F gemeint sind, da es sich dabei eher um ein Rollenmodell als ein Komponentenmodell handelt. Der Begriff *Komponente* wird verwendet, wenn sich auf die konkrete Umsetzung als Softwarekomponente bezogen wird.
Neben den Rollen beschreibt das MMI-F, welche Auszeichnungssprachen für ein multimodales System wichtig sind und welche Rollen davon beeinflusst werden. In Abbildung 3.1 sind die Rollen des MMI-F und ihre Beziehungen zueinander dargestellt.
Die einzelnen Rollen sind in drei Kategorien für Eingabe-, Ausgabe- und Anwendungsverarbeitung gegliedert. Eingabeverarbeitung umfasst die Erkennung, Interpretation und Integration von Nutzereingaben, während Ausgabeverarbeitung den entgegengesetzten Weg mit Generation, Styling und Rendering von Nutzerschnittstellen abdeckt. Die Anwendungsfunktionen unterteilen sich in den In-

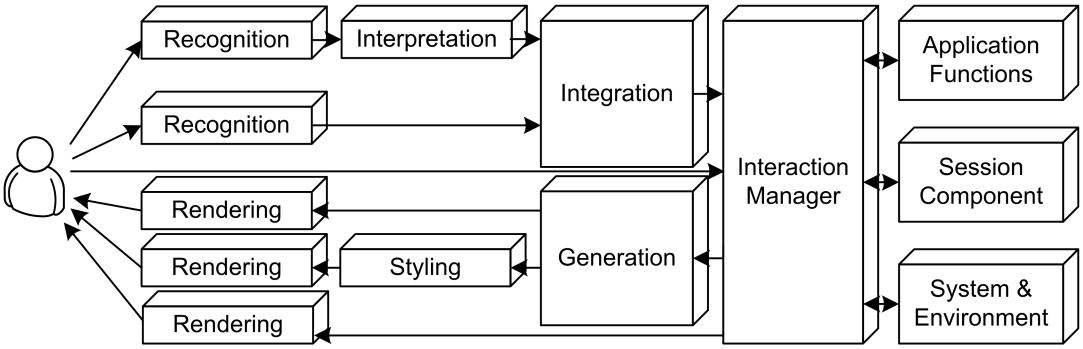

Abbildung 3.1.: Vereinfachte Darstellung des Multimodal Interaction Frameworks (nach [Wor02b])

teraktionsmanager, die Geschäftslogik, die Sitzungsverwaltung und System- und Umgebungsinformationen. Im Einzelnen haben die unterschiedlichen Rollen folgende Aufgaben:

Recognition erfasst natürliche Eingaben des Nutzers und transformiert sie in eine Form, in der die Eingaben weiterverarbeitet werden. Dazu kann die Recognition-Komponente z. B. eine Grammatik verwenden. Beispiele für Recognition-Komponenten sind Spracherkennung oder Handschrifterkennung zur Transformation von Sprache bzw. Schriftzügen in Text.

Interpretation kann die Ergebnisse der Recognition weiter verarbeiten, identifiziert die Semantik der Eingabe und kann so semantisch gleiche Eingaben auf eine gemeinsame syntaktische Repräsentation reduzieren. Beispielsweise können so die syntaktisch unterschiedlichen Wörter „ja“, „ok“ und „in Ordnung“ auf ein „Ok“ reduziert werden.

Integration kombiniert das Ergebnis von Interpretationskomponenten unterschiedlicher Modalitäten zu einer komplexen, multimodalen Eingabe. Auf diese Weise kann das bekannte Beispiel „put that there“ von Bolt ([Bol80]) realisiert werden, in dem per Spracheingabe definiert wird, wohin der Gegenstand bewegt wird, wobei der Nutzer auf den Gegenstand und das Ziel zeigt (Gestenerkennung).

Die Integration-Rolle muss nicht als dedizierte Komponente am Ende der Eingabeverarbeitung realisiert werden, sondern kann vielmehr auch als Teil der Recognition, Interpretation oder Interaction-Komponente umgesetzt werden.

Generation ermittelt die zur Ausgabe benötigten Modalitäten und versorgt die benötigten Ausgabe-Komponenten mit einer Nutzerschnittstellenbeschrei-

bung. Diese Beschreibung liegt in einem internen, geräte- und modalitätsunabhängigen Format vor. Das W3C äußert sich nicht weiter zu Details des Formats.

Styling ist für eine bestimmte Ausgabemodalität zuständig und fügt Layoutinformationen zur Nutzerschnittstellenbeschreibung hinzu, um beispielsweise zu definieren, wie grafische Elemente angeordnet werden oder um in eine Sprachausgabe noch Pausen einzufügen. Das Ergebnis ist eine Auszeichnungssprache wie z. B. SSML (Speech Synthesis Markup Language), SVG (Scalable Vector Graphics) oder XHTML (Extensible HyperText Markup Language).

Rendering wandelt die modalitätsspezifische Nutzerschnittstellenbeschreibung der Styling-Komponente in ein Format um, das durch den Nutzer verstanden werden kann. Die Komponente stellt beispielsweise grafische Ausgaben auf einem Bildschirm dar oder gibt Sprache oder sonstige Audio-Informationen über Lautsprecher aus.

Interaction Manager ist eine logische Rolle, die den Dialogfluss verwaltet, indem sie entgegengenommene Eingaben durch die Application Functions bearbeiten lässt und anschließend das Ergebnis zurückliefert. Der Interaction Manager kann als einzelne Komponente realisiert sein oder aus einer Komposition von Teilkomponenten bestehen, die über verschiedene Prozesse und Geräte verteilt sind.

Application Functions sind die eigentlichen Geschäftslogikteile der multimodalen Anwendung.

Session Component ist für die Verwaltung des Anwendungszustandes sowie der temporären und permanenten Sitzungen zuständig.

System & Environment ermöglicht dem Interaction Manager Eigenschaften und Änderungen des Systems in Bezug auf Geräteeigenschaften, Nutzereinstellungen oder Umgebungsbedingungen zu ermitteln und darauf zu reagieren.

3.2.2. Multimodal Architecture and Interfaces

Der W3C Working Draft „Multimodal Architecture and Interfaces“ [Wor06] beinhaltet die Beschreibung einer lose gekoppelten Architektur, die sowohl zentrale als auch verteilte Implementierungen zulässt. Das Ziel des Dokuments ist die Spezifikation einer Architektur des MMI-F und der Schnittstellen zwischen den einzelnen Bestandteilen. Dadurch soll erreicht werden, dass die Interoperabilität zwischen unterschiedlichen Anbietern durch ein flexibles Framework unterstützt wird, weshalb die Komponenten und deren Interaktion auch nur geringfügig durch

das Framework eingeschränkt werden. Der Fokus liegt auf der Bereitstellung sowohl von generellen Hilfsmitteln zur Kommunikation zwischen Komponenten als auch von grundlegenden Infrastruktur- und Plattformdiensten sowie einer Anwendungskontrolle.
Für die Entwicklung des Frameworks lagen folgende Entwurfsziele zu Grunde: Kapselbarkeit (Behandlung aller Komponenten als Black Box und keine Annahmen über deren Interna), Verteilung (Unterstützung von verteilten und zentralen Implementierungen), Erweiterbarkeit (durch neue Modalitätskomponenten), Rekursivität (Verpacken von einer Architekturinstanz in eine andere Instanz) und Modularität (Trennung von Daten, Kontrolle und Präsentation, vergleichbar mit dem Architekturmuster MVC der Softwareentwicklung).
Weiterhin unterscheiden die Autoren zwischen Aspekten der Entwurfszeit und Laufzeit von multimodalen Architekturen. Die Entwurfszeitsüberlegungen beschäftigen sich mit Beschreibungsdokumenten, die die Anwendung als solche repräsentieren. Hierbei wird die Art der Dokumente nicht eingeschränkt, sondern explizit die Möglichkeit zusammengesetzter Dokumente mit verschiedenen Namensräumen erwähnt. Es wird darauf hingewiesen, dass sowohl *Controller markup* als auch *Presentation markup* benötigt wird, wobei für den Controller-Teil CCXML (Call Control XML) bzw. SCXML (State Chart XML) als mögliche Sprachen genannt werden [W3C07, W3C06].

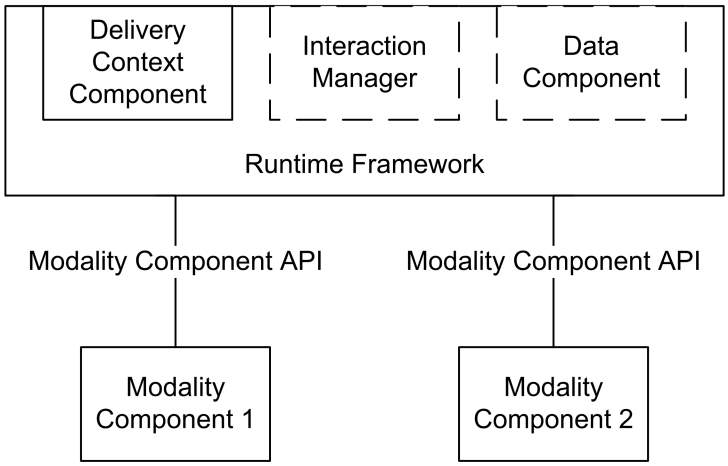

Abbildung 3.2.: Komponenten der multimodalen Architektur des MMI (nach [Wor06])

Eine Übersicht über die Architektur ist in Abbildung 3.2 dargestellt. Die Komponenten unterteilen sich in das *Runtime Framework* und die *Modality Components*. Das *Runtime Framework* ist in drei Subkomponenten unterteilt: Die *Delivery*

Context Component verfügt über Kontextinformationen zu Nutzereinstellungen, Umgebungsbedingungen und Gerätekonfigurationen, welche für eine Adaption der Inhalte abgefragt werden können. Der *Interaction Manager* ist für die Verarbeitung von Ereignissen zuständig, die die *Modality components* erzeugen. Die *Data Component* speichert anwendungsbezogene Daten, die vom *Interaction Manager* entsprechend der Kontrollflusslogik verändert werden können. Insgesamt stellt das *Runtime Framework* die Infrastruktur zur Verfügung, die einerseits das aus dem Anwendungsentwurf resultierende Kontrolldokument interpretiert und andererseits von verschiedenen *Modality components* benutzt werden kann, um eine komponentenbasierte, multimodale Interaktion zu ermöglichen.
Die Beschreibung der „Multimodal Architecture and Interfaces“ bewegt sich nur auf einer logischen bzw. abstrakten Ebene und macht keine Annahmen darüber, wie die Elemente in einer tatsächlichen Implementierung realisiert werden. Die Autoren definieren nur, welche Komponenten prinzipiell vorhanden sein müssen, ohne jedoch auf die Verteilung der Komponenten in einem verteilten System einzugehen. Es wird explizit erwähnt, dass das Dokument auf die Verwendung von Markup, Scripting und Schnittstellen zwischen den Bestandteilen fokussiert ist. Daher werden Aspekte bezüglich des Laufzeitverhaltens von Komponenten bzw. des Systems insgesamt nicht berücksichtigt, die aber für die Umsetzung eines verteilten Systems mit einer Komponentenplattform mindestens genauso wichtig sind. Die Laufzeitaspekte und notwendigen Kriterien einer Komponentenplattform werden in Kapitel 4 erarbeitet.

3.2.3. Extensible MultiModal Annotation Markup Language

EMMA ist als Austauschformat zwischen Komponenten eines multimodalen Systems entwickelt worden. Mit EMMA ist es möglich, die aus multimodalen Nutzereingaben entstandenen Interpretationsergebnisse zu beschreiben und durch Annotationen anzureichern, sodass eine möglichst exakte Weiterverarbeitung unterstützt wird. Im Rahmen des MMI-F ist EMMA als Schnittstelle zwischen der Interpretations- und Integrations- bzw. weiter zur Interaktionsrolle gedacht, sodass in der Regel die EMMA-Dokumente automatisch vom System erzeugt und nicht manuell durch den Nutzer angelegt werden müssen. Durch EMMA lassen sich sowohl einfache als auch zusammengesetzte Eingaben beschreiben und um entsprechende Annotationen anreichern. In Listing 3.1 ist ein Beispiel für ein EMMA-Dokument dargestellt.
Das Beispiel enthält die Interpretationsergebnisse einer Flugbuchungsanwendung, in die per Spracherkennung der Start- und Zielort eingegeben werden kann. Die Attribute des Elements `emma:one-of` deklarieren die Zeitstempel von Beginn und Ende der Aufnahme, während das Element die Bedeutung hat, dass nur ein Interpretationsergebnis richtig ist, auch wenn mehrere Ergebnisse angegeben wurden. Hier wurden zwei mögliche Varianten der Äußerung des Nutzers gefunden. Beide haben durch die Interpretationskomponente einen Vertrauenswert bekom-

```
<emma:emma version="1.0"
    xmlns:emma:="http://www.w3.org/2003/04/emma"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.w3.org/2003/04/emma
    http://www.w3.org/TR/emma/emma10.xsd"
    xmlns="http://www.example.com/example">
  <emma:one-of id="r1" emma:start="1000" emma:end="1003">
    <emma:interpretation id="int1" emma:confidence="0.75"
      emma:tokens="flights from boston to denver">
      <origin>Boston</origin>
      <destination>Denver</destination>
    </emma:interpretation>

    <emma:interpretation id="int2" emma:confidence="0.68"
      emma:tokens="flights from austin to denver">
      <origin>Austin</origin>
      <destination>Denver</destination>
    </emma:interpretation>
  </emma:one-of>
</emma:emma>
```

Listing 3.1: EMMA-Beispiel (nach [Wor05a])

men, der angibt, wie sicher das Ergebnis aus Sicht des Systems ist. Der Inhalt des `emma:interpretation`-Elements ist anwendungsabhängig und in diesem Beispiel der Start- und Zielort (`origin`, `destination`). Die Aufgabe der Nachfolgekomponente ist dann, aus den beiden Interpretationsvorschlägen mit Hilfe weiterer Vorschläge von anderen Kanälen oder Kontextinformation der Anwendung (mögliche Start- und Zielorte) die richtige Interpretation auszuwählen.

3.2.4. Synchronisierung von Ausgabeströmen mittels SMIL

Zur Synchronisierung empfiehlt das MMI-Framework die Verwendung von SMIL (Synchronized Multimedia Integration Language) [Wor05b]. SMIL synchronisiert mehrere Ausgabeströme durch Zeitpunkte für Beginn und Ende bzw. Zeitdauern von einzelnen Elementen. Diese Zeitpunkte sind relativ zu anderen, wie z. B. zum Start der SMIL-Präsentation. Beispielsweise kann ein Video mit `<video begin="1s"dur="5s"...>` definiert werden, was zur Folge hat, dass dieses Video eine Sekunde nach dem Start des darüberliegenden Elements abgespielt wird. Dieser Mechanismus der Ausgabesynchronisation funktioniert nur, wenn sich alle Ausgabe-Renderer auf dem selben Gerät befinden. Wenn die Ausgabe-Renderer auf verschiedenen, über Netzwerke verbundenen Geräten platziert sind (z. B. eine Präsentation, die auf Monitoren zweier unterschiedlicher PCs gezeigt werden

soll), ist die Synchronisation mit SMIL nicht ausreichend, da die Übertragung der Kommandos über heterogene Netzwerke und Geräte unvorhersehbare Verzögerungen zwischen Absenden und der eigentlichen Ausführung verursachen kann, sodass die Ausgabe u. U. nicht synchronisiert erfolgt. Dies kann nur durch synchronisierte Uhren auf allen beteiligten Geräten realisiert werden, was aber im mobilen Umfeld dynamischer Geräteföderationen nicht immer gegeben ist.

3.2.5. Fazit

Die Spezifikationen des W3C lassen an verschiedenen Punkten Raum für Interpretationen. So geht z. B. aus der Spezifikation des MMI-F nicht eindeutig hervor, ob die Generation-Komponente für jede Modalität eine modalitätsabhängige oder eine modalitätsunabhängige Nutzerschnittstellenbeschreibung für alle Modalitäten erzeugt. Laut West hat diese Beschreibung ein modalitätsunabhängiges Format [WAQ04], was aber nichts über den Ort der tatsächlichen Aufspaltung aussagt. Für den Anwendungsfall, dass die Nutzerschnittstelle auf verschiedene Endgeräte verteilt ist und auf diesen repliziert werden soll, ist der tatsächliche Aufspaltungsort nicht von Belang. Wenn jedoch einzelne Teile der Nutzerschnittstelle auf bestimmten Geräten bzw. Modalitäten ausgegeben werden sollen, muss die Generation-Komponente die Aufspaltung in die Teilbeschreibungen durchführen.
Ein Ergebnis der Anwendung des MMI-F im Projekt SNOW (Services for NOmadic Workers) [The05b] war u. a., dass eine Denormalisierung zwischen dem Framework und der physischen Architektur [PHK07b, PHK07a]. Dies hat sich darin geäußert, dass nicht alle logischen Verbindungen auf physische abgebildet wurden. Im konkreten Fall war das eine Verbindung zwischen der Generation-Komponente und dem Styling für Sprachausgabe. Im MMI-F ist diese Verbindung direkt, während in der SNOW-Architektur die Kommunikation über das Endgerät und die zugehörige Rendering-Komponente geroutet wurde. Der Grund wurde als die im MMI-F fehlende Berücksichtung von verteilten Implementierungen identifiziert, wodurch sich Teile der Funktionlität auf mobilen Endgeräten befinden. Da diese oft über beschränkte Ressourcen verfügen, liegen lokale Optimierungen zur Erhöhung der Performanz oder Verfügbarmachung von Funktionalität nahe. Als Lösung für die Denormalisierung wurde die Einführung einer weiteren Rolle in das MMI-F vorgeschlagen, die sich insbesondere mit der Aufteilung der Nutzerschnittstellenbeschreibung auf Geräte und Modalitäten befasst, außerdem aber auch die Synchronisierung der u. U. verteilten Nutzerschnittstelle realisiert. Da die Rolle aber eine direkte Assoziation zur Generation hat und auch nur einmal vorkommt, kann die Funktionalität logisch gesehen von der Generation abgedeckt werden, allerdings sollte dies genauer im MMI-F berücksichtigt werden. Im weiteren Verlauf der Arbeit ist die Generation für diese Funktionalität verantwortlich. Ob tatsächlich eine zusätzliche Rolle benötigt wird, setzt eine grundlegende Untersuchung des Modells und der gefundenen Defizite voraus.

Diese Information sollte in der Spezifikation des MMI-F expliziter gemacht werden. Zwar unterstützt die MMI-F prinzipiell die Idee föderierter Endgeräte, jedoch werden Detailaspekte wie verteilte Synchronisierung oder verteilte Realisierung in dem W3C-Dokument nicht in Betracht gezogen.
Da das MMI-F nur ein Rahmenwerk für multimodale Anwendungen darstellt, macht es keine Angaben über die physische Verteilung der Komponenten. Es spezifiziert nur, welche Rollen vorhanden sein müssen und welche Aufgaben diese übernehmen. Sowohl das MMI-F als auch die zugehörige Architekturdefinition suggerieren auf Komponenten aufbauende Lösungen, wobei die Architekturdefinition sogar direkt von Komponenten spricht. Beide Dokumente adressieren jedoch nicht die notwendigen Konzepte der Komponentenplattformen, die für die Realisierung einer komponentenbasierten, multimodalen Nutzerschnittstelle notwendig sind. Das MMI-F legt weiterhin nicht fest, wie die Rollen etwa in einem komponentenbasierten System auf Komponenten verteilt werden. Darüber hinaus ist der Zusammenhang zwischen dem MMI-F und der zugehörigen Architektur nicht offensichtlich und wird in der Architekturbeschreibung auch nicht explizit diskutiert.
Weiterhin geht die W3C Arbeitsgruppe davon aus, dass die Spezifikationen stets für Web-Anwendungen eingesetzt werden, d. h. Anwendungen, die auf einem Server laufen und ihre Nutzerschnittstelle über Auszeichnungssprachen wie HTML zum Gerät des Nutzers transportieren. Das ist jedoch nicht immer der Fall, da die Nutzer auch herkömmliche Desktop- bzw. Nicht-Web-basierte Anwendungen multimodal einsetzen möchten.

3.3. Multimodale Systeme mit föderierten Endgeräten

In diesem Abschnitt werden bestehende Systeme vorgestellt, die Anwendungen mit föderierten Endgeräten unterstützen bzw. eine Kombination von Endgeräten vorsehen, um gemeinsam eine Nutzerschnittstelle zu realisieren. Hierbei liegt der Fokus auf Ansätzen, die für die Verwendung durch einen Nutzer gedacht sind. Arbeiten aus dem Groupware-Bereich, bei dem mehrere Nutzer über jeweils ein Endgerät kollaborativ mit der selben Anwendung interagieren [KWK04], werden nicht betrachtet, weil sich für diese Systeme andere Anforderungen bezüglich der zeitlichen Synchronisierung und Mobilitätsunterstützung ergeben, da jeder Nutzer für sich nur ein Endgerät in Verwendung hat. Durch die zu entwickelnde Integrationsschicht sollen aber gerade die Interaktionsmöglichkeiten des Nutzers um die Verwendung beliebig vieler Endgeräte erweitert werden. Ebenso werden Lösungen zum Weiterleiten von Nutzerschnittstellen auf andere Geräte nicht näher untersucht (z. B. X-Window-System[1], [DRONP01]).

[1] http://www.x.org

3.3.1. Mundo

Mundo [HAH+02, AKM07] ist ein Konzept für ubiquitäres Computing basierend auf der Infrastruktur des Internets. Mundo abstrahiert von der unterliegenden Netzwerkinfrastruktur [BA03] und stellt den Anwendungen eine gemeinsame Middleware zur Verfügung. Es werden sogenannte *overlay cells* als Backbone definiert, die durch vier unterschiedliche Arten von Entities genutzt werden.
In Abbildung 3.3 ist die horizontale Architektur von Mundo dargestellt. Das ME (minimal entity) ist das persönliche Gerät eines Nutzers und damit seine digitale Repräsentation. Das ME kann sich zu USs (ubiquitous associable objects) in Reichweite verbinden und Informationen austauschen. Beispiele für USs sind Speicher, Prozessoren, Displays oder Netzwerkgeräte. Nichtassoziierbare Smart Items werden ITs genannt. Eine Gruppe von zwei oder mehr persönlichen Umgebungen (die Umgebung um ein ME), wird als WE bezeichnet (wireless group environment). WEs können sich mit THEYs verbinden, die die nicht-lokale Welt repräsentieren und in der Regel nicht für ad-hoc-Kommunikation geeignet sind. Dienste in Mundo werden durch zusätzliche Informationen über die Dienstqualität beschrieben.

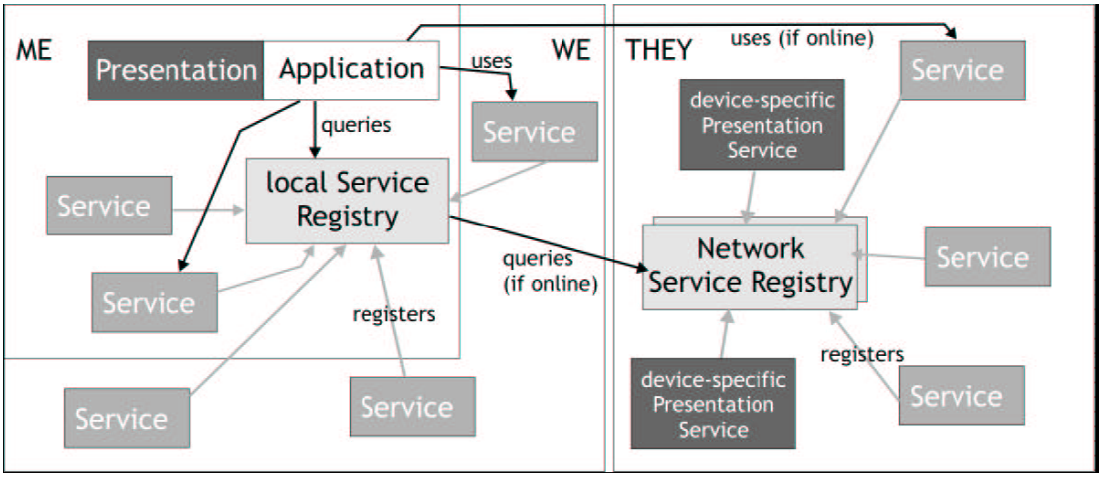

Abbildung 3.3.: Horizontale Architektur von Mundo (aus [HAH+02])

Obwohl Mundo u. a. das Konzept von *appearances* für adaptive Nutzerschnittstellen umfasst und über Assoziationen zu USs Geräteföderationen aufbauen kann, werden wichtige Aspekte wie die zeitliche Synchronisierung der verteilten Nutzerschnittstellen nicht explizit behandelt. Eine adaptive Nutzerschnittstelle besteht bei Mundo aus einer abstrakten Beschreibung, die zur Laufzeit entsprechend der verfügbaren Geräte in eine konkrete Beschreibung umgewandelt und ausgegeben wird. Mundo adressiert die Integration von in der Umgebung des Nutzers befindlichen Ressourcen auf Netzwerkebene, um Prozessorkapazität oder Speicher dynamisch nutzen zu können. Mundo-Anwendungen müssen explizit für die Be-

nutzung der Kommunikationsmiddleware entwickelt werden, wofür entsprechende Tools bereitgestellt wurden [AKM05, Ait06, AM07].

3.3.2. QuickSet

Das QuickSet-System [CJM+97a, CJM+97b] verwendet eine verteilte Multiagentenplattform (die Open Agent Architecture™) um verschiedene Komponenten für Nutzerschnittstellen und verteilte Anwendungen zu integrieren. Das Hauptentwurfskriterium von QuickSet war die Verfügbarkeit gleicher Interaktionsmöglichkeiten für Handheld und Desktop PCs sowie Terminals mit wandmontierten Displays. Die Autoren betrachteten Sprache und gestenbasierte Interaktion (Geste bezieht sich auf 2D-Zeichnungen auf einem Bildschirm) als den kleinsten gemeinsamen Nenner. Hauptanwendungsfall des QuickSet-Systems (siehe Abbildung 3.4) ist das Simulationstraining des US Militärs. Die Agentenarchitektur gruppiert alle Komponenten um einen Facilitator, der für das entsprechende Routing, Dispatching und Triggering zuständig ist. Dieses System berücksichtigt keine flexible Kombination heterogener Endgeräte, sondern beschränkt sich vielmehr auf eine statische Konfiguration von vorher festgelegten Geräten. Eine Erweiterung des Ansatzes um beliebig viele Interaktionskomponenten insbesondere für den mobilen Bereich scheitert an der Skalierbarkeit, da jede Kommunikation zwischen den Komponenten über den zentralen Facilitator läuft, der somit einen Flaschenhals bezüglich CPU-Leistung und Datenrate darstellt.

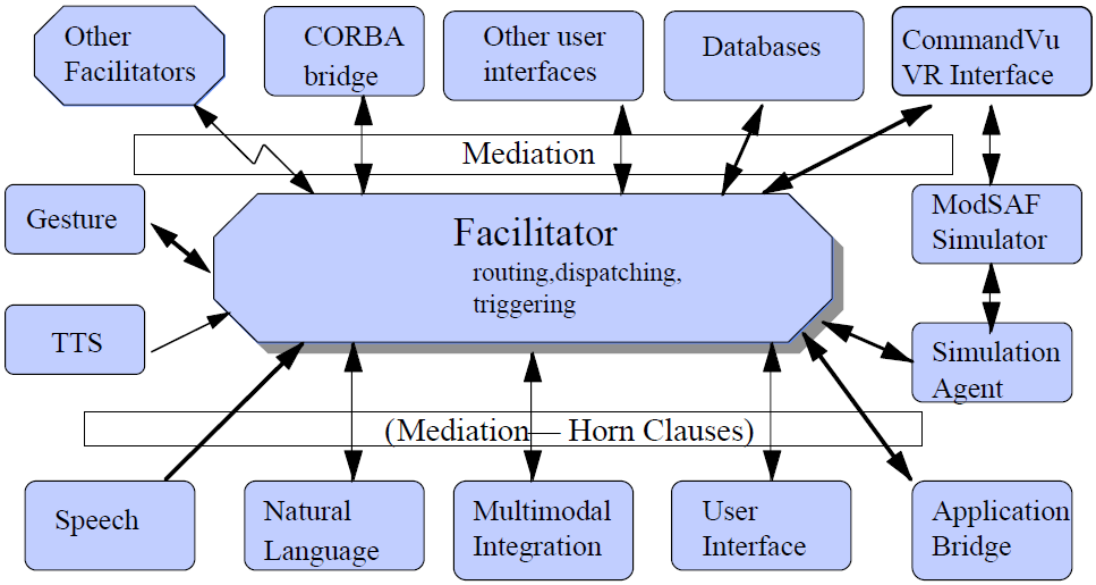

Abbildung 3.4.: Die Agentenarchitektur von QuickSet (aus [CJM+97a])

3.3.3. Nightingale

Nightingale [WAQ04] ist eine ubiquitäre Systemarchitektur, die multimodale, kontextsensitive Anwendungen unterstützt, die auf verschiedene Endgeräte verteilt sind. Das Kontextmanagement beinhaltet einen auf Ontologien basierenden Schlussfolgerungsansatz. Ziel von Nightingale ist das Entkoppeln multimodaler Anwendungen von speziellen Eingabegeräten. Daher erlaubt das System dem Nutzer, sich einem beliebigen Gerät in der Umgebung zu nähern und dieses für den multimodalen Dialog zu benutzen.

Die Architektur basiert auf einem Multiagentensystem mit Agenten für Eingabeerkennung, Anwendungen und Ausgabegenerierung. Die Eingaben sind in EMMA kodiert, was neben der Agentenaufteilung dem MMI-F entspricht. Der Anwendungsentwickler muss neben anwendungs- bzw. modalitätsabhängigen Grammatiken und Interpretern für Eingabeagenten auch entsprechende Stylesheets für Ausgabeagenten zur Verfügung stellen. Die Eingabeagenten erhalten die Grammatiken von der Anwendung, zu der sie verbunden sind. Die Fusion und Integration von Eingaben wird dann vom Anwendungsagenten durchgeführt. Das Nightingale-System geht von einer statischen Verteilung der benötigten Geschäftslogik aus, sodass die Mobilität des Nutzers nur eingeschränkt unterstützt wird. Nutzerzentrische Aspekte wie das Synchronisierungsverhalten der Nutzerschnittstelle werden nicht berücksichtigt.

3.3.4. Embassi

Der Fokus von Embassi [ERS03] liegt darauf, Menschen mit wenig Erfahrung im Umgang mit Computern die Interaktion mit Heimelektronik durch natürliche Schnittstellen zu erleichtern. Dazu unterstützt das System eine dynamische Anzahl von Modalitätsanalysekomponenten und -renderern. Die Architektur von Embassi sieht ähnlich wie die des MMI-F Komponentenklassen für die Erkennung und Interpretation von Eingaben bzw. Generierung und Rendering von Ausgaben vor. Ebenso wie QuickSet basiert Embassi auf einer Agentenarchitektur, bei der aber eine Änderung der verwendeten Agenten zur Laufzeit berücksichtigt wurde. Da das Embassi-System für die Steuerung von Heimelektronik (Videorekorder, Radio, Licht, etc.) konzipiert wurde, ist die Nutzung beliebiger Anwendungen nicht in Erwägung gezogen worden.

Die für Embassi geeigneten Anwendungen müssen speziell dafür entwickelt werden. Es ist nicht möglich, bestehende Anwendungen einzubeziehen. Außerdem berücksichtigt es die Synchronisationsprobleme zwischen den verschiedenen Endgeräten nicht.

3.3.5. Pebbles

Das Pebbles-Projekt untersucht, wie mobile Endgeräte (z. B. PDA) für die Kommunikation mit anderen mobilen Endgeräten, PCs oder computerisierten Haushaltsgeräten verwendet werden können [Mye01]. Im Rahmen des Projektes wurden einige Prototypen entwickelt, mit denen verschiedene Anwendungen auf einem Desktop-PC mittels eines PDAs verwendet werden konnten. Da der Fokus auf der „Fernsteuerung“ existierender Anwendungen lag, war eine Entwicklung von entsprechenden Steuerkomponenten auf Serverseite notwendig, die die vom PDA empfangenen Kommandos an die Zielanwendung weiterreichen. Dazu wurden dementsprechende Clientanwendungen für die PDA-Seite entwickelt. Die als *PebblesPC* bezeichnete Komponente läuft auf dem PC und agiert als Vermittler zwischen den Client- und Serverkomponenten. Das Konzept ist auf PCs und PDAs ausgerichtet und sieht keine wahlfreie Erweiterung durch beliebige Endgeräte vor. Die Kontrollanwendungen sind statisch verteilt und explizit für eine jeweilige Zielanwendung entwickelt worden.
Als Erweiterung stellen sich die Autoren einen *personal universal controller* (PUC) vor [NM06, NMH+02], der als Fernbedienung für beliebige Haushaltsgeräte dienen kann und die Nutzerschnittstelle aus einer abstrakten Spezifikation der Fähigkeiten des Gerätes ableitet.

3.3.6. OpenInterface

Die OpenInterface-Architektur[2] ist im Rahmen des SIMILAR[3] NoE (Network of Excellence) entstanden und hat das Ziel, die Ergebnisse aus den Bereichen Signalverarbeitung und HCI miteinander zu kombinieren, sodass zusammen multimodale Anwendungen unterstützt werden können. Hierfür wird besonderer Wert auf Redundanzfreiheit, Einfachheit der Komponentenintegration, hochperformante Kommunikation zwischen heterogenen Komponenten und Trennung der Verantwortlichkeiten in Bezug auf die Komponentenentwicklung einzelner Partner gelegt [GLM+05]. Die OpenInterface-Plattform basiert auf C++ und Linux, weshalb die Plattformunabhängigkeit eingeschränkt ist. Weiterhin addressiert die OpenInterface-Plattform keine dynamische Verteilung der Komponenten, Mechanismen zur dynamischen Kopplung oder Austausch von Komponenten zur Adaption an geänderte QoS-Anforderungen.

3.3.7. Oxygen

Die Vision des Oxygen-Projekts[4] ist eine Welt, in der die Nutzer jederzeit und überall Zugang zu Computer-gestützten Ressourcen durch deren allgegenwärti-

[2] http://www.oi-project.org
[3] http://www.similar.cc
[4] http://www.oxygen.lcs.mit.edu/Software.html

ge Verfügbarkeit haben (vergleichbar mit Luft zum Atmen). Das Projekt ist in Geräte-, Netzwerk-, Software-, Wahrnehmungs- und Nutzertechnologien unterteilt, die spezifische Problemstellungen bearbeiten. Entsprechend der Softwarearchitektur werden die Anwendungen komponentenbasiert sein. Die Komponenten können ihr Verhalten ändern und auf verschiedenen Granularitätsebenen ersetzt werden. Außerdem unterstützt das System personenzentrierte Sicherheitsmechanismen und abgekoppelte Operationen sowie Komponentenmigration. Leider wird durch die Webseite nicht klar, welche konkreten Ergebnisse aus Oxygen entstanden und in welchen Publikationen diese zu finden sind. Zudem wurde die Projektseite zuletzt 2004 aktualisiert.

3.3.8. Virtual device service gateway

In [FSF+04] wird ein Framework beschrieben, mit dem die Fähigkeiten mobiler Endgeräte durch Endgeräte aus der Umgebung erweitert werden können. Die Autoren bezeichnen ein erweitertes Endgerät als „virtual device“, da es als ein Gerät erscheint, dessen Fähigkeiten jedoch aus verschiedenen physischen Geräten abgeleitet werden. Dafür ist ein neues Programmiermodell erforderlich, mit dem Anwendungen ohne direkte Kopplung zu Endgeräten entwickelt werden, wobei die Zuordnung durch das Framework zur Laufzeit gemacht wird. Das Framework sieht einen *Virtual device service gateway* zwischen Endgeräten und Anwendung vor, dessen Teilkomponenten für das Finden von Geräten und der Zuordnung der Geräte zu Nutzerschnittstellenteilen sowie deren Adaption verantwortlich ist. Für die Zuordnung werden die durch die Anwendung nachgefragten Ressourcen dem Angebot durch die Geräte gegenübergestellt, wobei möglichst so zugeordnet wird, dass keine Adaption des Inhalts erforderlich ist. Zur Präsentationssteuerung kommt SMIL zum Einsatz, das die in Unterabschnitt 3.2.4 beschriebenen Nachteile beim Einsatz in heterogenen Netzwerken hat. Zudem ist die Geschäftslogik der Nutzerschnittstellen statisch verteilt und eine synchrone Präsentation der verteilten Nutzerschnittstellen wird nicht explizit adressiert.

3.3.9. ASAP

Das in [HXLM05, HL06] beschriebene System ASAP (A Synchronous Approach for Photo sharing) ermöglicht es mehreren Nutzern, Fotos auf ihrem jeweiligen Endgerät mit den anderen Teilnehmern zu teilen, um die Bilder gemeinsam zu betrachten, ohne durch das u. U. kleine Endgerät eingeschränkt zu werden. Es erfolgt nicht nur eine Präsentation durch einen Nutzer, sondern jeder Nutzer hat die Möglichkeit, mit der Anwendung und den Fotos zu interagieren, wobei die Modifikationen auf die anderen Endgeräte synchronisiert werden. Das ASAP bietet weiterhin die Möglichkeit, dass auf einem PDA eine Übersicht von Bildern dargestellt wird (Thumbnails), während auf dem anderen die Volldarstellung des jeweils ausgewählten Bildes erfolgt. Das entspricht in etwa einer Web-Anwendung mit

zwei Frames, die auf zwei verschiedene Geräte verteilt werden. Anwendungsseitig ist mit dem System auch eine Suche nach und Darstellung von inhaltsähnlichen Bildern möglich. Die Konzeption des Systems hat sich auf die verteilte Realisierung dieser eben beschriebenen Anwendung spezialisiert, ohne einen generellen Ansatz für verteilte Nutzerschnittstellen zu berücksichtigen. Daher sind die jeweiligen Anwendungsteile auch statisch verteilt, können nicht flexibel um andere Funktionalität oder Modalitäten bzw. Geräte erweitert werden. Die in [HXLM05] vorgestellten Synchronisierungsansätze beziehen sich auf die inhaltliche Synchronisierung zwischen den Endgeräten. Darüber hinaus werden von den Autoren implizit Endgeräte mit visueller Ausgabe vorausgesetzt und keine multimodale Interaktion berücksichtigt.

3.3.10. Multimodal-Browsing-Framework

In [CDML03] beschreiben Coles et al. ein Framework, bei dem mehrere Endgeräte zum gemeinsamen koordinierten Darstellen von Web-Seiten verwendet werden können. Dabei können die Endgeräte auch unterschiedliche Modalitäten und damit Repräsentationen der jeweiligen Inhalte verwenden, wobei diese Diversität vor der Anwendung durch Verwendung eines konsolidierenden Proxys verborgen bleibt. Wenn durch einen Browser eine neue Seite angefordert wird, werden die anderen Browser von diesem benachrichtigt und stellen ebenfalls eine Anfrage nach dieser Seite. Durch erweiterte HTTP-Header konsolidiert der Proxy diese Anfragen, sodass nur eine aus den Teilanfragen zusammengesetzte Anfrage an den Anwendungsserver gestellt wird. Voraussetzung ist aber, dass der Browser auf dem Endgerät einerseits alle anderen Browser kennt und diese auch direkt erreichen kann. Darüber hinaus werden in der Arbeit noch weitere Konzepte wie XForms [Gro06a] vorgestellt, das zur Umsetzung der koordinierten Anfragen und inhaltlichen Synchronisierung verwendet werden kann. Die Eingaben in Form-Elementen werden über ein spezielles *inter-client messaging subsystem* direkt zwischen den Browsern übertragen. Auf diese Weise ist zusammengesetzte Modalität nicht möglich. Zeitlich synchronisierte Präsentationen der Nutzerschnittstelleninhalte sind ebenfalls nicht vorgesehen. Auch ist die allgemeine Architektur des Frameworks nur teilweise modular und die Browser statisch verteilt.

3.3.11. Weitere Ansätze

Im Folgenden werden weitere verwandte Arbeiten vorgestellt. Aufgrund ihrer großen Anzahl und teilweise geringeren Relevanz werden diese nur kurz beschrieben werden.

Der UbicompBrowser [BSLG98] ersetzt die Standard-Web-Nutzerschnittstelle eines monolithischen Browsers durch eine aus PDA und Umgebungsgeräten bestehende zusammengesetzte Schnittstelle, wobei die Umgebungsgeräte automatisch

anhand der Position des PDAs ermittelt werden. Neben herkömmlichen Web-Anwendungen soll mit dem System auch die Kontrolle über TV oder Lichtschalter möglich sein, indem das aus dem Web bekannte Konzept der URI (Uniform Resource Identifier) auf diese Geräte erweitert wird. Das System beschränkt sich auf die Nutzung eines PDAs ohne Möglichkeiten zur Erweiterung.

In [BAKM04] wird eine Architektur beschrieben, mit deren Hilfe der Inhalt einer Webseite auf mehrere Geräte aufgeteilt werden kann, sodass der Nutzer diese Geräte gemeinsam nutzen kann. Die Architektur kann dabei die Auswahl der föderierten Endgeräte treffen, die einzelnen Teilsichten synchronisieren und die zur Nutzerschnittstelle gehörenden Dokumente umwandeln. Dazu wurde UIML (User Interface Markup Language)[5] um Unterstützung für föderierte Endgeräte erweitert [BM04]. Im Gegensatz zu anderen Ansätzen werden bei diesem keine bestimmten Geräte vorausgesetzt, sondern eine beliebige Anzahl zugelassen. Nichtsdestotrotz sind alle beteiligten Komponenten statisch verteilt und nicht für die Berücksichtigung unterschiedlicher Nutzeranforderungen in Bezug auf QoS-Eigenschaften der Verarbeitung ausgelegt. Darüber hinaus wurden nur „unimodale“ Browser als Nutzerschnittstelle verwendet. Weitere Ansätze, in denen der Inhalt von Webseiten auf mehrere Endgeräte verteilt wird, befinden sich in [JPSF01, HPN00].

Die Arbeitsgruppe WiDex (Widget Description Exchange Service) der IETF (Internet Engineering Task Force) hat das Ziel, einen leichtgewichtigen Mechanismus in IP (Internet Protocol)-Netzen zu entwickeln, mit dem Nutzerschnittstellen auf entfernten Geräten darstellbar sind [Gro06b]. Da es sich dabei um Web-Anwendungen handelt, geht die Arbeitsgruppe davon aus, dass Technologien wie XML (eXtensible Markup Language) und DOM zum Einsatz kommen, um die verteilten DOM-Objekte zu synchronisieren. Außerdem sollen Parameter für Service Discovery und ein realisierendes Framework spezifiziert werden. Das einzige Ergebnis der Arbeitsgruppe ist jedoch bislang ein Anforderungsdokument [Gro07].

Das MONA-Projekt[6] (Mobile multimOdal Next generation Applications) hatte das Ziel, die Nutzung multimodaler Anwendungen auf mobilen Endgeräten zu ermöglichen. Dabei kommt UIML zum Einsatz, welches von einem Presentation Server zu endgerätespezifischen Inhaltsbeschreibungen wie Java AWT (Abstract Windows Toolkit)/Swing, HTML, WML (Wireless Markup Language) und VoiceXML verarbeitet wird. Weitere Modalitäten können hinzugefügt werden, ohne dass die Anwendung selbst aktualisiert werden muss [ADJ+04, WDJS04]. Allerdings ist nicht vorgesehen, dass der Nutzer zwei oder mehr Geräte gleichzeitig zur Interaktion benutzt. Weiterhin ist der MONA-Ansatz serverzentriert, weshalb eine dynamische Verteilung von Geschäftslogikkomponenten nicht vorgesehen ist.

[5] http://www.uiml.org
[6] http://mona.ftw.at

Das Hauptziel des CDCE-Frameworks (Composite Device Computing Environment) [PSG00, SDPG01] ist die Nutzbarmachung von Geräten aus der Umgebung zur Darstellung von Informationen in Zusammenarbeit mit dem lokalen Endgerät des Nutzers (meist ein PDA oder SmartPhone). Nachdem Geräte zur Komposition gefunden wurden, teilt der Nutzer seine gewünschten Dienste dem CDCE-Gateway mit, der diese dann den Teilen des zusammengesetzten Endgeräts zuordnet. Hierbei ist die Geschäftslogik der Nutzerschnittstellen und Framework-Komponenten statisch verteilt. Synchronisierungsaspekte von Nutzerschnittstellen auf zusammengesetzten Endgeräten wurden nicht betrachtet.

Rigole et al. beschreiben in [Rig05] eine verteilte und komponentenbasierte Middleware. Von den auf dieser Middleware lauffähigen Anwendungen wird vorausgesetzt, dass sie sich an geänderte Systembedingungen anpassen können. Die Middleware soll auf möglichst vielen Hardware-Plattformen verfügbar sein und ist daher als Microkernel mit erweiterbarem Funktionsumfang konzipiert. Die Komponenten der Middleware werden nach der *SEESCOA*-Methode (Software Engineering for Embedded Systems using a Component-Oriented Approach, [UvBTHB01]) entwickelt und kommen auf der *DRACO*-Laufzeitumgebung (DistriNet Reliable and Adaptive COmponents) zur Ausführung. Die unterstützten Anwendungen sind aus Komponenten aufgebaut und kommunizieren über asynchrone Nachrichten. Um auf allen Endgeräten eine dem jeweiligen Endgerät entsprechende Nutzerschnittstelle anbieten zu können, verfügen die Endgeräte über eine sogenannte *Interaction Component*, die als Vermittler zwischen Nutzer und Anwendungskomponenten fungiert, welche die Nutzerschnittstellenbeschreibung ihrerseits in einer geräteunabhängigen Nutzerschnittstellenbeschreibungssprache anbieten. Die verteilte Nutzung von entfernten Komponenten wird durch transparente Proxies realisiert, die lokale Komponenten vortäuschen. Das System ist dafür geeignet, um von unterschiedlichen Geräten aus auf Anwendungen zuzugreifen, sodass eine dem jeweiligen Endgerät angepasste Nutzerschnittstelle erzeugt wird. Die gleichzeitige Verwendung von mehreren Geräten zur Erzeugung einer verteilten Nutzerschnittstelle bzw. die Synchronisierung der Inhalte zwischen diesen wurde nicht betrachtet.

In [MMB+02] beschreiben Myers et al. ein System, das die Zusammenarbeit von unterschiedlichen Modalitäten und Endgeräten unterstützt, sodass räumlich verteilte Personen an einer gemeinsamen Sitzung teilnehmen können. Daraus ergibt sich die Hauptanwendung des Systems im Bereich Groupware (Stichworte CSCW (Computer Supported Cooperative Work, auch Computer Supported Collaborative Work)), in diesem speziellen Fall ein militärischer Kommandoposten. Die Personen können Modalitäten entsprechend der verwendeten Endgeräte (z. B. PDA) nutzen, wobei die öffentlichen Informationen des großen Displays für den einzelnen Nutzer auch erweitert und aufbereitet werden können. Die Architektur ist zwar auch in einzelne Komponenten strukturiert, die verteilt zur Ausführung kommen, allerdings sind diese fest auf die jeweiligen Geräte verteilt

und haben nicht die Verwendung durch einen Nutzer zum Ziel, sodass keine Mechanismen zur nutzerfreundlichen Präsentation des Inhalts entwickelt wurden.
Ferda the Ant ist im Rahmen des CATCH-2004 Projekts entstanden und stellt eine multimodale Browser-Architektur dar, die unimodale Browser-Technologien wie HTML, VoiceXML oder WML unterstützt, um multimodalen Zugriff auf Anwendungen zu gewähren [KSKB03]. Dafür kommunizieren die einzelnen Browser mit einem virtuellen Proxy, der die inhaltliche Synchronisierung realisiert. Hierbei wird von den Autoren explizit offen gelassen, ob sich die einzelnen Browser auf einem Endgerät befinden oder in einem Netzwerk verteilt sind. Besonderer Fokus liegt hierbei auf der Verwendung von bereits existierenden Browser-Lösungen, die lediglich um die Synchronisierungsschnittstelle erweitert werden, damit asynchrone Synchronisierungsnachrichten gesendet und empfangen werden können. Über das entwickelte Synchronisierungsprotokoll werden z. B. die übrigen Browser veranlasst, eine bestimmte Seite anzufordern, die vom auslösenden Browser angefordert wurde, sodass auf diese Weise der jeweilige Inhalt der Seite an den Browser übermittelt werden kann, ohne dessen interne Implementierung zu ändern. Bei der nur rein inhaltlichen Synchronisierung auf *page*-Ebene wurden aber insbesondere die Auswirkungen von heterogenen Netzwerken und Geräten auf die Reaktionszeiten nicht in Betracht gezogen. Darüber hinaus sind die Browser-Komponenten statisch verteilt und unterstützen keine dynamischen Umgebungen mit mobilen Endgeräten.
Die MICA-Architektur ist eine Middleware-Schicht, die auf der Blackboard-Idee basiert, welche einen globalen, geteilten Speicher vorsieht, der allen Geräten sowohl für die Kommunikation als auch als persistenter Speicher dient [KS04]. Weiterhin wurde bei der Konzeption des Systems besonderer Wert auf die Trennung von Anwendungslogik und Nutzerschnittstelle, Einsatz in heterogenen Umgebungen und Unterstützung von multiplen Eingabe- und Ausgabemodalitäten gelegt. Die Nutzerschnittstellen werden von Agenten realisiert, die sich die darzustellenden Informationen von MICA Objects holen, welche sich auf dem Blackboard befinden. Hierbei arbeiten alle Agenten autonom, es gibt keine Unterstützung beispielsweise für synchrone Darstellung durch unterschiedliche Agenten. Des Weiteren ist durch die Architektur keine zusammengesetzte Multimodalität möglich, d. h. es können nicht verschiedene Agenten verwendet werden, um eine gemeinsame Eingabe an eine bestimmte Anwendung zu senden, ohne dass diese das explizit unterstützt.
Für die Migration von Nutzerschnittstellen existieren vielzählige Ansätze in der Literatur. Berti et al. beispielsweise entwickelten ein System, welches migrierende multimodale Nutzerschnittstellen in „multi-device environments“ unterstützt [BP05]. Damit ist gemeint, dass ein Nutzer über ein Endgerät mit einer Anwendung interagiert, diese Interaktion unterbrechen kann, um sie dann von einem anderen Endgerät aus fortzusetzen. Das resultiert in einer Nutzerschnittstelle, die von einem Endgerät zum anderen migriert ist. Dazu wird ein modellgetriebener Ansatz (oder Reverse Engineering) zugrunde gelegt, um aus der graphi-

schen Nutzerschnittstelle einer existierenden Anwendung auf einer abstrakteren Ebene eine Abbildung auf eine multimodale Nutzerschnittstelle zu ermöglichen. Hierbei wird jedoch durch die „multi-devices“ nur sequentielle Multimodalität unterstützt, da der Nutzer immer nur ein Endgerät gleichzeitig verwenden kann. Ein weiterer Ansatz ist der von Salminen et al. [SHR07], wobei die Autoren einen middleware-basierten Ansatz verfolgen, der zudem auf einer Orientierung an Komponenten aufbaut. Es wird eine explizite Trennung von Anwendungslogik und Nutzerschnittstelle vorgesehen, sodass die Nutzerschnittstelle auch von einem entfernten Gerät dargestellt werden kann. Auf Kommando, beispielsweise das Lesen eines RFID-Tags (Radio-Frequency Identification), wird die Nutzerschnittstelle vom Handy auf ein Gerät in der Umgebung migriert, d. h. die darstellende UI-Komponente wird ausgetauscht. Eine zustandsbehaftete Migration, bei der die Interaktion auf dem Handy begonnen und auf dem PC beendet wird, unterstützt die Middleware laut [SHR07] nicht. Die parallele Verwendung von mehreren Geräten oder Modalitäten ist nicht vorgesehen.

Mrohs et al. beschreiben ein Framework, welches verschiedene Anpassungen an Informationen durchführt, die in intelligenten Umgebungen über u. U. mehrere Endgeräte zum Nutzer übertragen werden [MR03]. Dazu gehören Personalisierung und Nutzerschnittstellenadaption. Das Framework unterstützt weiterhin ein Umgebungsbewusstsein, ein Sitzungsmodell und verschiedene Interaktionsmodi. Alle Ergebnisse der Arbeit beschränken sich jedoch auf abstrakte Aspekte eines solchen Frameworks, z. B. die Verwendung von Kontext. Die Autoren betrachten die physische Aufteilung und Verteilung von Funktionalität in einem verteilten System nicht. Demzufolge adressieren sie auch nicht, inwiefern beispielsweise zusammengesetzte Multimodalität unterstützt werden kann oder welche Geräte bzw. Modalitäten überhaupt in Betracht kommen.

3.3.12. Fazit und Vergleich

In Tabelle 3.1 werden die vorgestellten verwandten Arbeiten in Bezug auf die in Abschnitt 2.3 ermittelten Anforderungen an die Interaktionsanwendung gegenübergestellt, da es sich bei den betrachteten Arbeiten um das Äquivalent zur hier eingeführten Interaktionsanwendung handelt. Die Anforderungen an die Komponentenplattform werden in Abschnitt 3.5 zum Vergleich der Komponentenplattformen verwendet.

Hieraus wird ersichtlich, dass die verglichenen Systeme zwar meist modular aufgebaut sind, jedoch nur teilweise eine aktive Verwaltung der Föderation unterstützen. Besondere Anstrengungen zur Erhöhung der Robustheit bzw. Unterstützung von Sicherheit, Synchronisierung auf verschiedenen Ebenen sind kaum gemacht worden. Der Grund hierfür liegt vermutlich darin, dass die Entwicklung der Systeme nicht auf komponentenbasierten Ansätzen beruhen.

	IA-Anforderung							
	1 Modularität	2 Verwaltung	3 Robustheit	4 Sicherheit	5 Konsistenz	6 Synchronis.	7 Transparenz	8 Dynamik
Mundo (Integration von ubiquitären Ressourcen)	x	o	-	-	-	-	-	x
QuickSet (PDA, PC und Wanddisplay für Militärsimulation)	x	-	-	-	-	-	-	-
Nightingale (Entkopplung der Anwendung von Endgeräten)	x	-	-	-	-	-	o	-
Embassi (Heimelektronik für unerfahrene Nutzer)	x	o	-	-	-	-	-	x
Pebbles (PDAs zur Erweiterung von PC-Anwendungen)	x	x	-	-	-	-	-	-
OpenInterface (Integration von HCI und Signalverarbeitung)	x	-	-	-	-	-	-	-
Oxygen (Allgegenwärtiger Computerzugang)	x	-	x	x	-	-	-	x
Virtual device services gateway (Fähigkeitserweiterung von Geräten)	x	o	-	-	-	-	-	x
ASAP (Fotoverteilanwendung)	x	x	-	-	-	o	-	x
Multimodal Browser	o	-	-	-	x	o	x	x

Tabelle 3.1.: Tabellarischer Vergleich der verwandten Arbeiten und der Anforderungen aus Abschnitt 2.3 (x = wird unterstützt, o = wird teilweise unterstützt, - = wird nicht unterstützt)

3.4. Synchronisierung verteilter Nutzerschnittstellen

In [FCFG99] wird ein Visualisierungssystem vorgestellt, welches aus mehreren, zu unterschiedlichen PCs gehörenden Monitoren besteht. Die heterogenen und unabhängigen PCs haben möglicherweise nichtsynchrone Uhren, was ein Problem für den Anwendungsfall der Visualisierung darstellt. Als ersten Ansatz schlagen die Autoren die Verwendung der Technik *dead reckoning* vor [IEE93], bei der statt rein syntaktischer Information zur direkten Objektmanipulation (z. B. Attribut x bekommt Wert y, entspricht Level 0 in [Roe95]) semantisch höherwertige Information übertragen wird (z. B. Attribut x ändert sich um z pro Sekunde, ent-

spricht Level 1 in [Roe95]). Mit diesem Ansatz lässt sich aber lediglich die benötigte Nachrichtenanzahl zwischen den verteilten Anzeigen reduzieren, die Latenz in heterogenen Netzwerken jenseits des LAN (Local Area Network) wird damit nicht behandelt. Dafür kommt SNTP (Simple NTP) entsprechend RFC (Request for Comments) 2030 [Mil96] zum Einsatz, mit dessen Hilfe auf den Visualisierungsteilnehmern eine simulierte Uhr erzeugt und synchronisiert wird. Damit ist eine synchrone Visualisierung auf den unabhängigen Geräten basierend auf einem Zeitstempelmechanismus möglich.
Darüber hinaus wurde die Uhrensynchronisation in verschiedenen Einsatzszenarien wie beispielsweise ad-hoc Netzwerke untersucht [Röm01, Mil95]. Bei verteilten Systemen kommt es oft nicht auf die exakte zeitliche Synchronisierung an, sondern eine logische Ordnung von Ereignissen ist ausreichen. Dafür wurden logische Uhren entwickelt [Lam78, Mat88].
Die vorgestellen Ansätze ermöglichen z. T. eine zeitliche Synchronisierung verteilter Systeme durch logische oder virtuelle Uhren. Doch auch wenn die Uhren der verteilten Systeme exakt gleich laufen, so ist dadurch die Ermittlung des frühestmöglichen Zeitpunkts einer synchronen Aktualisierung der verwendeten Endgeräte nicht inherent möglich. Für die synchrone Präsentation der verteilten Nutzerschnittstelle sind die Ansätze daher nur bedingt geeignet.

3.5. Vergleich von Komponentenplattformen

Aus den Anforderungen in sowohl Abschnitt 2.3 als auch Abschnitt 2.4 geht hervor, dass für die Realisierung des Systems ein komponentenbasierter Ansatz in Frage kommt. Aus diesem Grund werden im folgenden Unterabschnitt die Grundlagen von Komponenten und Komponentenplattformen dargelegt. Anschließend werden zwei Vertreter bestehender Komponentenplattformen untersucht, inwiefern sie die geforderten Randbedingungen erfüllen und somit für die Umsetzung geeignet sind.

3.5.1. Komponenten und Komponentenplattformen

Laut [CD00] werden Komponenten als Einheiten von Software bezeichnet, die nach bestimmten Prinzipien strukturiert sind, die hauptsächlich aus der Objekttechnologie stammen. Die *Vereinigung von Daten und Funktionen* definiert, dass ein Objekt aus Daten und Funktionen besteht, wobei die Funktionen die Daten verarbeiten und somit eine hohe Kohäsion zwischen Daten und Funktionen entsteht. Durch das Prinzip der *Kapselung* wird ausgedrückt, dass ein aufrufendes Objekt nur von der Spezifikation des aufgerufenen Objekts abhängt, aber nicht von dessen Implementierung. Details über den Speicherort der Daten oder die Implementierung der Funktionen des aufgerufenen Objekts sind für das aufrufende Objekt transparent. Schließlich wird durch das Prinzip der *Identität* festgelegt,

dass jedes Objekt eine eindeutige Identität unabhängig von dessen Zustand aufweisen muss.
Hauptziel der Verwendung von komponentenbasierten Ansätzen ist laut [CD00] nicht die durch Kapselung erhöhte Wiederverwendbarkeit, sondern vielmehr die Möglichkeit, dynamisch Komponenten ändern zu können, um Implementierungen zu verbessern oder zusätzliche Funktionalität durch Anpassen der Schnittstellen verfügbar zu machen. Daher werden die o. g. Prinzipien bei Komponenten erweitert, indem die Objektspezifikation durch eine explizite Repräsentation der Spezifikationsabhängigkeiten erweitert wird. Diese explizite Repräsentation wird *Schnittstelle* genannt und hat die Besonderheit, dass ein Objekt mehrere Schnittstellen aufweisen kann. Die Schnittstellen können unabhängig voneinander benutzt und geändert werden, sodass eine Erweiterung der Schnittstellen möglich ist, wobei die Auswirkung auf das Objekt referenzierende Objekte minimiert werden kann, solange die jeweilige Schnittstelle erhalten bleibt.
Komponenten können verschiedene Formen einnehmen: Zunächst existieren Komponenten als *Komponentenstandard* wie EJB (Enterprise Java Beans) oder Microsofts COM+ (Weiterentwicklung des Component Object Model). Damit eine bestimmte Komponente verwendet werden kann, muss eine *Komponentenspezifikation* existieren. Wesentlicher Teil der Komponentenspezifikation ist die *Komponentenschnittstelle*, die von einer *Komponentenimplementierung* realisiert wird. Wenn eine Komponente installiert wird, nimmt sie die Form einer *installierten Komponente* an, die wiederum ein oder mehrere *Komponentenobjekte* instanzieren kann, welche die eigentliche Funktionalität ausführen und die eigentliche Identität besitzen.
Die Ausführung von Komponenten wird von einer Komponentenumgebung (auch als Komponentenplattform bezeichnet) realisiert. Dazu müssen die Komponenten bestimmte Regeln einhalten bzw. konform zu dem jeweiligen Komponentenstandard sein. Die Komponentenumgebung verfügt meist über Infrastrukturdienste wie Transaktionsunterstützung und Sicherheit, auf die die Anwendungskomponenten zurückgreifen können. Beispiele für Komponentenumgebungen sind die Umsetzungen der Komponentenstandards COM+ und EJB aber auch CCM (CORBA Component Model).
Der Fokus von EJB, COM+ und CCM liegt vorwiegend in der Unterstützung von Anwendungen auf Serverseite, die durch einen Anwendungsserver bereitgestellt werden. Der Anwendungsserver realisiert dabei die Komponentenumgebung und ist somit für die Bereitstellung der Infrastrukturdienste verantwortlich. Bei COM+ sind das beispielsweise Transaktionen, Objektsynchronisation und Objekt-Pooling. EJBs legen u. a. großen Wert auf einfache Mechanismen für Persistenz und Transaktionssicherheit. Das CCM ist konzeptionell an EJB angelehnt.
Für die Anwendung in der vorliegenden Arbeit sind diese Technologien jedoch weniger geeignet, da es hier mehr auf die Dynamik und Flexibilität der Komponente an sich ankommt, als auf den Schutz der Persistenz und die Sicherstellung

von Transaktionen, welche vorwiegend bei Mehrnutzerbetrieb benötigt werden. Zudem benötigen die eben genannten Technologien Laufzeitumgebungen, die als schwergewichtige Anwendungsserver vorliegen und daher nicht für den Einsatz auf mobilen und ressourcenbeschränkten Endgeräten geeignet sind. Zwei Technologien, die für die prototypische Umsetzung eher in Frage kommen, sind Softwareagenten und OSGi (Open Services Gateway Initiative), welche im Folgenden bezüglich der in Abschnitt 2.4 aufgestellten Anforderungen verglichen werden.

3.5.2. Softwareagenten

Laut der Komponentendefinition von Cheesman et al. [CD00] können Softwareagenten als Komponenten betrachtet werden, da sie die eingangs in Unterabschnitt 3.5.1 erwähnten Kriterien erfüllen. Softwareagenten unterliegen ebenfalls einer Kapselung, vereinen Funktion und Daten, verfügen über eine Identität und können über Schnittstellen benutzt werden.
Softwareagenten sind ein Programmierparadigma, welches dabei hilft, einem Stück Software (dem Agenten) bestimmte Fähigkeiten und Eigenschaften zuzuordnen. In der Literatur existieren verschiedene Definitionen für Softwareagenten. Maes definiert beispielsweise, dass Agenten in einer komplexen Umgebung existieren, in dieser Umgebung autonom beobachten und agieren, um dadurch eine Menge von vordefinierten Zielen zu erreichen [Mae95]. Laut Hayes-Roth nehmen Agenten ihre dynamische Umgebung wahr, beeinflussen die Umgebung und können Schlussfolgerungen ziehen, um Beobachtungen zu interpretieren, Probleme zu lösen und Aktionen zu ermitteln [HR95]. Aus diesen und anderen Definitionen lassen sich Eigenschaften ableiten, durch welche Softwareagenten charakterisiert werden können. Hierzu eignet sich u. a. die von Wooldridge et al. erstellte schwache Notation, die Agenten über ihre Eigenschaften charakterisierbar macht [WJ95]:

Autonomie Die Agenten arbeiten ohne direkten Einfluss von anderen Agenten oder Menschen und haben in gewisser Weise Kontrolle über ihre Aktionen.

Soziale Fähigkeiten Agenten interagieren mit anderen Agenten oder Menschen mittels einer Agentensprache.

Reaktivität Agenten beobachten ihre Umgebung und reagieren auf Änderungen.

Proaktivität Agenten können zielorientiert agieren und die Initiative ergreifen.

Darüber hinaus sind Agenten meist *mobil*, sodass sie ihren Ausführungsort z. B. entsprechend Angebot und Nachfrage bestimmter Ressourcen wechseln können. Ein Agent ist auch *aufrichtig*, d. h. er kommuniziert wissentlich keine falschen Informationen. Weiterhin wird von Agenten angenommen, dass die Ziele nicht

widersprüchlich sind und möglichst erfüllt werden. Zudem wird von Agenten erwartet, dass sie derart agieren, dass die gesetzten Ziele erreicht werden können statt das Erreichen zu verhindern.
In Multi-Agenten-Systemen sind die kommunikativen Eigenschaften von Agenten wesentlich, damit Agenten miteinander kooperieren können, um so schneller, einfacher oder überhaupt ihre Ziele zu erreichen. Dabei ist es wesentlich, dass die Agenten über eine gemeinsame Sprache Informationen austauschen. Beispiele für Agentensprachen sind die von der FIPA (Foundation of Intelligent Physical Agents) standardisierte ACL (Agent Communication Language) [Fou] bzw. deren Vorgänger KQML (Knowledge Query and Manipulation Language) [UMB].
Softwareagenten können nach dem BDI-Prinzip implementiert werden. BDI (Belief-Desire-Intention) bedeutet, dass die Agenten Informationen über den derzeitigen Status von sich und der Umwelt haben (*belief*), bestimmte Ziele haben (*desires*) und bestimmte Aktionen durchführen bzw. sich zu deren Durchführung entschlossen haben (*intentions*) [Bra87].
FIPA-konforme Implementierungen von Agentenplattformen sind z. B. FIPA-OS[7] oder JADE (Java AGent Development Framework)[8]. Hierbei ist zu bemerken, dass FIPA-OS seit 2001 nicht mehr aktualisiert wurde, während die letzte Aktualisierung von JADE am 25. Juli 2007 erfolgte.
Der Lebenszyklus von FIPA-Agenten ist in Abbildung 3.5 dargestellt [Fou04]. Neben den Zuständen *active*, *suspended* und *waiting* fällt der Zustand *transit* auf. Dieser ist nur für mobile Agenten verfügbar und kennzeichnet den Zustand, in dem der Agent auf eine andere Plattform migrieren kann.

3.5.3. OSGi

Die OSGi Service Platform (Open Services Gateway Initiative, [The05a]) ist eine offene und allgemeine Architektur, die zum flexiblen Verteilen von Diensten und deren Verwaltung entwickelt wurde. Zielgeräte für die OSGi-Plattform sind neben PCs, PDAs, und Industriecomputern auch Unterhaltungselektronik wie Kabelmodems oder Draufstellkästen (Set-top Boxen).
Kern der OSGi Service Platform ist das OSGi Framework. Das Framework basiert auf Java und stellt den Anwendungen, auch *Bundle*s genannt, eine Laufzeitumgebung zur Verfügung, die verschiedene Aufgaben der Bundle-Verwaltung wie z. B. Lebenszykluskontrolle und Bundle-Abhängigkeiten übernimmt. Somit ist es einem Gerät mit OSGi Framework möglich, neue Bundles herunterzuladen und zu installieren und diese auch wieder zu entfernen, wenn sie nicht länger benötigt werden. Ein OSGi Framework besteht aus den Hauptbestandteilen *Security Layer*, *Module Layer*, *Life Cycle Layer* und *Service Layer*. Besonders wichtig sind der Life Cycle Layer für die Kontrolle des Lebenszyklus' der Bundles und der

[7] http://sourceforge.net/projects/fipa-os/
[8] http://jade.tilab.com/

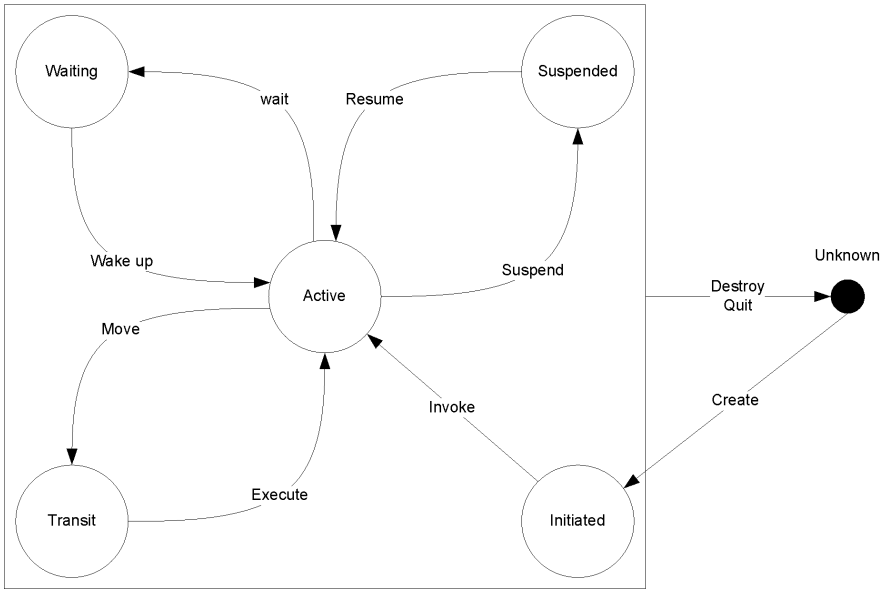

Abbildung 3.5.: Lebenszyklus von Agenten laut FIPA (nach [Fou04])

Service Layer, der ein einfaches Programmiermodell zum Binden von Diensten ermöglicht, wobei erst zur Laufzeit eine bestimmte Implementierung einer zuvor definierten Schnittstelle ausgewählt werden kann. Dafür kommt die Framework-interne Service Registry zum Einsatz, über die lokale Dienste registriert und gefunden werden können. Außerdem dient sie der Benachrichtigung über Zustandsänderungen der registrierten Dienste. Der Zustandsgraph eines OSGi-Bundles ist in Abbildung 3.6 dargestellt. Die Übergänge ohne Aktionsbezeichnung werden automatisch durchgeführt, wenn die notwendigen Prozesse des vorhergehenden Zustandes abgearbeitet wurden. Zu den bekanntesten OSGi-Implementierungen gehören Knopflerfish[9], Oscar[10] und Eclipse Equinox[11].

3.5.4. Verwandte Ansätze mit Komponentenplattformen

Es gibt in der Literatur einige Arbeiten, die Komponentenplattformen einsetzen, um in einem verteilten Netzwerk Geschäftslogik zu platzieren. Ein Beispiel hierfür ist [Yan03]. Das dort vorgestellte System basiert auf OSGi, verwendet das Konzept von intelligenten Agenten und unterstützt eine Netzwerkkommunikation auf Basis von Jini [SO00]. Wenn ein Nutzer einen bestimmten Dienst benötigt, wird der zugehörige Agent heruntergeladen und ausgeführt, sodass er den Dienst anbieten kann. Das System unterstützt auch hierarchisch strukturierte Zusammenschlüsse (Föderationen) von Dienstanbietern, wobei aus [Yan03] nicht klar

[9] http://www.knopflerfish.org
[10] http://oscar.objectweb.org
[11] http://www.eclipse.org/equinox/

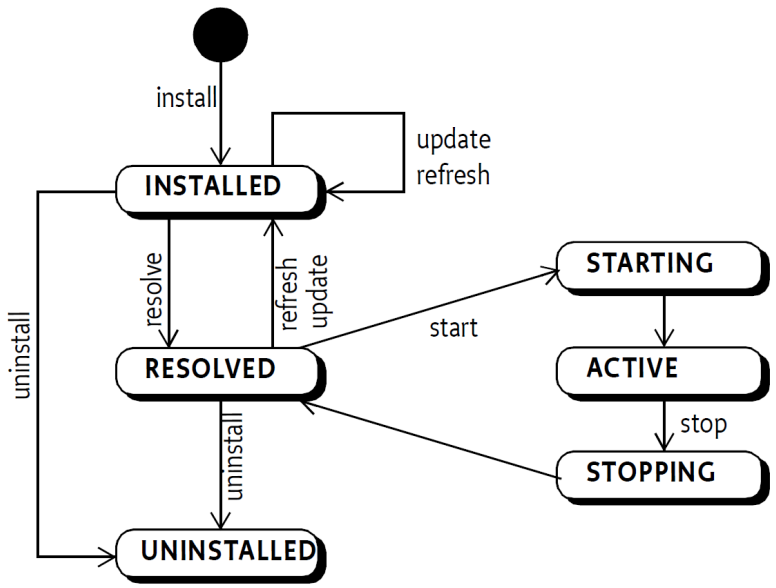

Abbildung 3.6.: Zustandsgraph eines OSGi Bundles (aus [The05a])

wird, wozu diese Zusammenschlüsse benötigt werden. Darüber hinaus bezieht sich die Arbeit auf die Heimvernetzung, ohne jedoch deutlich zu machen, welche Dienste genau angeboten werden sollen.
Ein ähnliches System wird in [HZ05] vorgestellt, wobei hier UPnP (Universal Plug and Play) als Netzwerkverbindungstechnologie eingesetzt wird. Die angestrebte Verwendung ist ebenfalls in der Heimautomatisierung angesiedelt, konkrete Konzepte sind dynamische Dienstversorgung, Fernsteuerung und Fernkontrolle von Geräten, Fehleranalyse und Kompatibilität. Hierbei wird jedoch die Geschäftslogik nicht dynamisch verteilt, sondern nur verteilt integriert. Zudem werden besondere UI-spezifische Konzepte nicht berücksichtigt, wie z. B. das Synchronisierungsverhalten.

3.5.5. Fazit und Vergleich

Eine zusammenfassende Gegenüberstellung der beiden vorgestellten Komponentenplattformen befindet sich in Tabelle 3.2. Hieraus wird ersichtlich, dass beide Varianten die Verteilung von Komponenten auf verschiedene Plattformen zur Laufzeit unterstützen.
Die Migration einer Komponente wird von bestimmten Agentenplattformen unterstützt, sofern es sich um mobile Agenten handelt. Die Migrationsfähigkeit ist allerdings bei Jade beispielsweise auf Intraplattform-Migration eingeschränkt, weshalb eine Agentenmigration zwischen unterschiedlichen Plattformen nicht möglich ist. In OSGi gibt es kein Konzept zur Migration, obwohl ein Bundle theoretisch auf unterschiedlichen Plattformarten ausgeführt werden kann. Weder

Plattform	KP-Anforderung				
	1 Verteilung	2 Migration	3 Verfügbarkeit	4 Austausch	5 Plattformunabhängigkeit
Agenten	x	x	-	-	(x)
OSGi	x	-	-	-	x

Tabelle 3.2.: Vergleich der Komponentenplattformen mit den Anforderungen aus Abschnitt 2.4

Agentenplattformen noch OSGi unterstützen Konzepte zur expliziten Erhöhung der Komponentenverfügbarkeit bzw. zum Austausch von Komponenten zur Laufzeit.
Die Plattformunabhängigkeit von Agentenplattformen ist nur eingeschränkt gegeben. Zwar kann sowohl FIPA-OS als auch Jade auf unterschiedlichen Java-Versionen ausgeführt werden (z. B. JSE (Java Standard Edition) und JME (Java Micro Edition)), allerdings ist das den Agenten verfügbare API (Application Programmer Interface) eingeschränkt. So können beispielsweise die Agenten nur die von der Plattform implementierte Kommunikationstechnologie benutzen. Bei Jade ist das RMI (Remote Method Invocation). Erweiterungen um beliebige, auch plattformspezifische, Bibliotheken sind nicht vorgesehen, sodass mit Agenten keine beliebigen Kommunikationsprotokolle verwendet oder auf systemnahe Funktionen wie Audioein- und -ausgabe zugegriffen werden kann.
OSGi ist mit den Implementierungen Knopflerfish und Oscar sowohl auf Desktop-PCs als auch auf PDAs vertreten. Für Knopflerfish beispielsweise ist die einzige Voraussetzung eine Java-Runtime mindestens der Version 1.2.2, was bedeutet, dass der CDC (Connected Device Configuration)-Teil der JME unterstützt wird, der CLDC (Connected Limited Device Configuration)-Teil aber unberücksichtigt bleibt.
Aus diesem Vergleich geht hervor, dass sich OSGi besser für die weitere Konzeption der Arbeit eignet. Zwar verfügen Agenten bereits über Migrationsfähigkeiten, allerdings verhindert das eingeschränkte API die Nutzung von dem jeweiligen Zweck angepassten Kommunikationstechnologien. Der fehlende Zugriff auf beliebige Bibliotheken bei Jade verhindert außerdem, dass je nach Endgerät dessen Besonderheiten in Bezug auf Aus- und Eingabe genutzt werden können.

3.6. Zusammenfassung

In diesem Kapitel wurden die Grundlagen für die weitere Konzeption gelegt, indem zunächst die verwendeten Begriffe eindeutig erläutert wurden. Anschließend wurde das MMI-F beschrieben und dessen Unzulänglichkeiten für den angestreb-

ten Verwendungszweck erläutert, bevor verwandte Arbeiten bezüglich Synchronisierung und vergleichbaren Ansätzen multimodaler Systeme analysiert und verglichen wurden. Zuletzt wurden zwei Komponentenplattformen verglichen, inwiefern sie für eine Verwendung als Grundlage der prototypischen Implementierung in Frage kommen.

Nichts in der Welt ist so mächtig, wie eine Idee, deren Zeit gekommen ist.

Victor Hugo

4 Konzeption der Integrationsschicht

In diesem Kapitel werden die verschiedenen Konzepte erläutert, die für die Umsetzung der Anforderungen aus den Abschnitten 2.3, 2.4 und 2.5 mit Hilfe einer Komponentenplattform notwendig sind.
In Abschnitt 4.1 wird zunächst ein Überblick über die Integrationsschicht erläutert und dargestellt, wie Komponenten, Komponentenplattform, Zielanwendung und Interaktionsanwendung zusammenhängen. Die Spezifikation des Komponentenmodells erfolgt in Abschnitt 4.2. Anschließend wird in Abschnitt 4.3 die Komponentenplattform und deren Basisdienste definiert. Ein wichtiger Teil der Integrationsschicht sind die Verzeichnisdienste, die in Abschnitt 4.4 erarbeitet werden. In Abschnitt 4.5 werden die Konzepte vorgestellt, welche von der Integrationsschicht bzw. der Komponentenplattform unterstützt werden müssen. Am Ende des Kapitels wird in Abschnitt 4.6 ein Konzept zur synchronen Darstellung von verteilten Nutzerschnittstellen erläutert.

4.1. Überblick über die Integrationsschicht

Die Integrationsschicht besteht aus Komponenten, einer Komponentenplattform und Verzeichnisdiensten.
Komponenten beinhalten einzelne Teile der Geschäftslogik, die über bestimmte Schnittstellen zugreifbar sind. In einer service-orientierten Architektur werden die Schnittstellen als Dienste bezeichnet, d. h. eine Komponente bietet einen oder mehrere (Anwendungs-)Dienste an. Das Komponentenmodell und weitere Besonderheiten werden in Abschnitt 4.2 beschrieben.

Die Komponenten werden von einer Laufzeitumgebung ausgeführt, der *Komponentenplattform*. Ohne Komponentenplattform kann ein Gerät keine Komponenten ausführen und somit auch nicht zu der Integrationsschicht beitragen. Die Eigenschaften einer Komponentenplattform werden in Abschnitt 4.3 erläutert.

Um die verteilten Komponenten nutzen und verwalten zu können, werden *Verzeichnisdienste* benötigt. Diese dienen u. a. der Zuordnung von Komponenten zu Komponentenplattformen. In Abschnitt 4.4 werden die Verzeichnisdienste spezifiziert.

In Abbildung 4.1 ist die vertikale Architektur der Integrationsschicht dargestellt. Die aus Komponenten bestehende Interaktionsanwendung verwendet Java, die Komponentenplattform OSGi und die in Abschnitt 4.3 näher definierten Basisdienste.

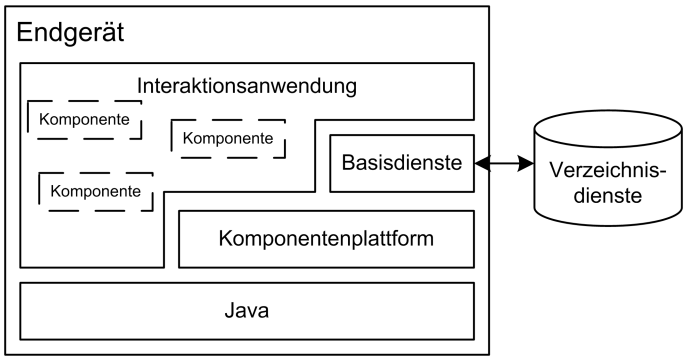

Abbildung 4.1.: Vertikale Architektur der Integrationsschicht

Ein Beispiel für eine Ausprägung der Integrationsschicht, wie sie für die multimodale Interaktion mit föderierten Endgeräten aussehen könnte, ist in Abbildung 4.2 dargestellt. Der Nutzer interagiert mit zwei unterschiedlichen Endgeräten, auf denen unterschiedliche Nutzerschnittstellenkomponenten zum Einsatz kommen. Die Komponenten der Interaktionsanwendung (IA) sind in Komponenten für die Nutzerschnittstelle (UI) und für die Unterstützung der verteilten, multimodalen Nutzerschnittstellen (MM) unterteilt. Die multimodalen Unterstützungskomponenten interagieren kanalisiert mit der Zielanwendung (ZA), welche nicht auf einem der Endgeräte ausgeführt wird, sondern sich auf einem Server befindet. Die Verzeichnisdienste werden den Komponenten bzw. der Komponentenplattform indirekt durch die Basisdienste zur Verfügung gestellt. Damit ist ein uniformer Zugriff auf die Verzeichnisse möglich, wobei bestimmte Informationen auch von den Basisdiensten zwischengespeichert werden können.

In dem Beispiel kommunizieren die Endgeräte direkt miteinander. Das muss jedoch nicht immer der Fall sein. Es ist auch möglich, dass diese aufgrund ihrer Hardwareausstattung nur mit dem Server kommunizieren können.

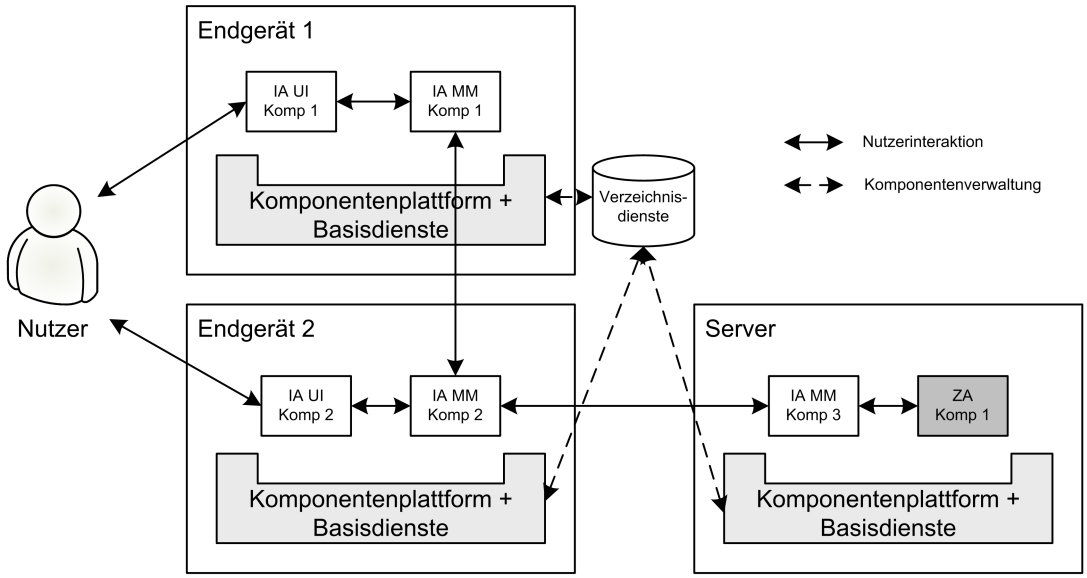

Abbildung 4.2.: Horizontale Architektur am Beispiel für eine Ausprägung der Integrationsschicht mit föderierten Endgeräten

Die Darstellung in Abbildung 4.2 kann auch als horizontale Architektur aufgefasst werden. Ein konkretes Zusammenspiel bestimmter Komponenten kann an dieser Stelle jedoch nicht aufgezeigt werden. Die Integrationsschicht stellt nur die Grundlage für das Zusammenspiel beliebiger Komponenten dar, die vom Nutzer bzw. Anwendungsentwickler erstellt werden.
Im nächsten Abschnitt wird erläutert, welche Auswirkungen die Anforderungen aus Abschnitt 2.3 sowie Abschnitt 2.4 auf die Konzeption der Integrationsschicht haben bzw. wie die Anforderungen auf Konzepte abgebildet werden.

4.1.1. Abbildung von Anforderungen auf Konzepte

In Abschnitt 2.3 wird mit IA-Anforderung 1 gefordert, dass die Interaktionsanwendung aus verteilt ausgeführten Komponenten besteht, sodass die Komponentenplattform verteilt vorliegen muss. Die Integrationsschicht ist demnach ein verteiltes System bestehend aus einzelnen Komponentenplattformen, die die gemeinsame Grundlage für die verteilte und multimodale Nutzerschnittstelle in Form der Interaktionsanwendung bieten. Das schließt ein, dass die benötigten Komponenten auf den jeweiligen Plattformen verfügbar sein müssen. Hierfür gibt es einen statischen und einen dynamischen Ansatz. Der statische Ansatz sieht vor, dass die Komponenten bei Initialisierung der Integrationsschicht auf die Plattformen verteilt werden, ohne dass diese Verteilung zur Laufzeit änderbar ist. Entsprechend des dynamischen Ansatzes werden die Komponenten zur Laufzeit auf verfügbare Plattformen verteilt, wodurch das System einerseits flexibler ist,

sich andererseits aber auch Wartezeiten ergeben können, die dadurch verursacht werden, dass angeforderte Komponenten erst übertragen und gestartet werden müssen.
Die dynamische Komponentenverteilung wird teilweise durch IA-Anforderung 3 und KP-Anforderung 1 motiviert, da dadurch Komponenten so verteilt werden können, dass ein robusteres Gesamtsystem entsteht bzw. auf Veränderungen in der Plattformverfügbarkeit reagiert werden kann, ohne dass jede erdenkliche Komponente auf jeder Plattform bereits verfügbar ist. Die begrenzte Ressourcenverfügbarkeit und mögliche Erweiterbarkeit durch aktualisierte oder eigene Komponenten sind zusätzlich Motivation für eine dynamische Verteilung. In Unterabschnitt 4.5.2 wird der verwendete Verteilungsmechanismus genauer spezifiziert. Dieser unterstützt die Platzierung nach unterschiedlichen Verteilungsstrategien (zentral vs. verteilt) und Zeitpunkten (bei Anwendungsinstallation vs. bei Bedarf). Dabei handelt es sich um nutzerspezifische Konfigurationen des Systems.
Aus der Tatsache, dass die Interaktionsanwendung aus verteilten und lose gekoppelten Komponenten besteht, folgt die Notwendigkeit von Mechanismen, mit denen sich die Komponenten finden und verbinden können. Hierfür sind verschiedene Verzeichnisdienste notwendig, in denen die Nutzer, Geräte und verfügbare Komponenten registriert werden. Mit Hilfe dieser Verzeichnisse können dann bestimmte Komponenten gefunden und benutzt werden. Die Verzeichnisdienste werden in Abschnitt 4.4 spezifiziert. Die Verbindung der Komponenten basiert auf einer Choreographie ohne zentrale Verwaltungseinheit, die einen Single-Point-of-Failure darstellen würde. Der Choreographiemechanismus wird in Unterabschnitt 4.5.1 erläutert. Daraus wird ersichtlich, dass die Kommunikation in der Integrationsschicht einem service-orientierten Konzept mit Dienstanbietern, Verzeichnissen und Dienstkonsumenten folgt.
Ebenfalls aus IA-Anforderung 3 (Robustheit) aber zusätzlich auch aus KP-Anforderung 2 (Migration) geht hervor, dass verfügbarkeitserhöhende Mechanismen erforderlich sind. Beispielsweise können die Komponenten zur Sicherstellung ihrer Erreichbarkeit von einer Komponentenplattform zu einer anderen migrieren, falls die Ursprungsplattform den Anforderungen der Komponente nicht mehr genügt. Dazu wird in Unterabschnitt 4.5.3 ein Konzept zur Migration von Komponenten beschrieben. Das Migrationskonzept wird stellvertretend für die Kategorie der Verfügbarkeitsmechanismen betrachtet.
Die Komponentenplattform kommt auf heterogenen Geräten zum Einsatz (siehe KP-Anforderung 4), was sich unter anderem in der Verfügbarkeit von unterschiedlichen Hardware- und Softwareressourcen bei jeder Plattforminstanz äußert. Aus diesem Grund werden die Fähigkeiten der Geräte bzw. Komponentenplattformen in einem Geräteprofil abgelegt (siehe Unterabschnitt 4.3.2). Zusammen mit dem Komponentenprofil aus Abschnitt 4.2 kann die Platzierung von Komponenten auf das jeweilige Gerät abgestimmt werden. Die KP-Anforderung 4 (Plattformunabhängigkeit) hat auf die Konzeption weniger Einfluss. Vielmehr wird diese Anforderung bei der Realisierung bzw. Implementierung des Systems interessant.

Die IA-Anforderungen 5 und 6 haben eine Nutzerschnittstelle zum Ziel, die eine möglichst nutzerfreundliche Verwendung der Anwendung ermöglicht. Dabei zielen die erwähnten Anforderungen auf die gleichzeitige Präsentation des Inhalts bzw. die Synchronisierung von Eingaben ab. Aus diesem Grund wird in Abschnitt 4.6 ein Konzept zur Synchronisierung verteilter Nutzerschnittstellen erarbeitet. Hierbei kann der Nutzer wählen, ob die Synchronisierungsstrategie auf *event*-Ebene vorläufige oder kontrollierte Aktualisierungen realisieren soll.
Die IA-Anforderung 7 (Transparenz) wird nicht durch ein explizites Konzept repräsentiert. Die Auswirkung auf die Konzeption ist vielmehr die verstärkte Forderung nach Einhaltung des Modulkonzeptes und der Kapselung in einzelne Funktionseinheiten, wie es u. a. durch das MMI-F vorgeschlagen wird. Dieses geht mit ZA-Anforderung 1 einher, wonach die multimodale Zielanwendung eine Trennung zwischen Geschäftslogik und Nutzerschnittstelle aufweisen muss. Das Ergebnis dieser beiden Anforderungen ist die Notwendigkeit einer geräteunabhängigen und multimodalen Nutzerschnittstellenbeschreibungssprache, wie z. B. D3ML (Device-independent MultiModal Mark-up Language) [GHKP06].
Eine flexible Anpassung der Komponenten sowie deren Verteilung ermöglicht eine den aktuellen Systemanforderungen entsprechende Interaktionsanwendung. Um die Interaktionsanwendung auch an unterschiedliche Nutzeranforderungen anpassen zu können, ist es notwendig, dass sich die Komponenten austauschen lassen (siehe KP-Anforderung 3). Hierzu wird in Unterabschnitt 4.5.4 ein Mechanismus für den flexiblen Austausch von Komponenten unter Aufrechterhaltung der Choreographie erläutert.
Die IA-Anforderung 4 (Sicherheit) wird nicht betrachtet sondern auf bestehende Technologien im Bereich Web Service Sicherheit zurückgeführt. Weitere Informationen dazu befinden sich in Unterabschnitt 4.3.3.

4.1.2. Annahmen

An die zu entwickelnde Integrationsschicht werden zusätzlich zu den Annahmen über die Konzeption des Systems an sich (siehe Abschnitt 2.2) weitere Annahmen getroffen, die die technischen Aspekte der Konzeption betreffen:

- Die Kommunikation zwischen Komponentenplattformen und Verzeichnisdiensten sowie der Komponentenplattformen untereinander basiert auf IP. Die Berücksichtigung anderer Protokolle und Technologien bleibt außen vor, da dies nicht im Fokus der Arbeit liegt und sich daraus weiterführende Forschungsfragen nach Protokolladaptern ergeben.

- Jede Komponentenplattform und die darauf befindlichen Komponenten können von jeder anderen Komponentenplattform erreicht werden. Zwei Komponenten können demnach nicht so platziert werden, dass sie nicht direkt miteinander kommunizieren können.

- Die Komponentenplattformen bieten eine homogene Ausführungsumgebung an, da sie auf OSGi und Java basieren. Daher sind auch die Komponenten in Java implementiert und nur für die Ausführung auf diesen Plattformen geeignet.

- Die Föderationsverwaltung erfolgt zentral über die Verzeichnisdienste. Prinzipiell ist auch eine dezentrale bzw. peer-to-peer-Verwaltung denkbar, bei der die einzelnen Komponentenplattformen u. a. über Zwischenspeicher für die Verzeichnisinhalte verfügen. Allerdings entstehen dadurch weitere erhebliche Anforderungen an die Integrationsschicht, die zur Realisierung der Kernidee nicht beitragen.

4.1.3. Zusammenfassung

In diesem Abschnitt wurden die Anforderungen auf konkrete Konzepte abgebildet und Beispielausprägungen der Integrationsschicht dargestellt. In Tabelle 4.1 befindet sich eine Gegenüberstellung der Anforderungen aus Kapitel 2 mit den Konzepten, die die Erweiterung der Integrationsschicht widerspiegeln. Für die Anforderung der Sicherheit wird, wie schon erwähnt, kein eigenes Konzept entwickelt.
Die in Abbildung 4.2 erwähnten Basisdienste entsprechen den im vorigen Kapitel aufgezählten Konzepten, die von der Integrationsschicht zu unterstützen sind, wie z. B. die Platzierung und das Auffinden von Komponenten, die Migration, die Verwaltung der Geräteföderation etc. Zusammengefasst haben die einzelnen Teile der Integrationsschicht folgende Aufgaben:

Verzeichnisdienste Registrieren und Finden von Komponenten, Diensten, Geräten mit Plattformen und Nutzern. Platzierung von Komponenten in Zusammenarbeit mit den Basisdiensten und Komponentenplattformen.

Komponentenplattform Bereitstellen einer allgemeinen Laufzeitumgebung für die Komponenten und Basisdienste, insbesondere Starten und Stoppen von Komponenten.

Basisdienste Platzierung, Austausch und Ausfallsicherung einzelner Komponenten, zustandsbehaftete Migration von Komponenteninstanzen zwischen Plattformen.

Komponente Bereitstellen von Nutzerschnittstellen und sonstiger Geschäftslogik mittels Diensten, Unterstützen von nutzerschnittstellenspezifischen Aspekten wie synchronisierte Verarbeitung auf unterschiedlichen Plattformen.

Anforderung	Konzepte						
	Komponenten-orientierung	Verwaltungs-anwendung	Platzierung	Migration	Choreographie	Austausch	Synchronisierung
IA-Anf. 1 - Modularität	x		x				
IA-Anf. 2 - Verwaltung		x					
IA-Anf. 3 - Robustheit				x			
IA-Anf. 4 - Sicherheit							
IA-Anf. 5 - Konsistenz							x
IA-Anf. 6 - Synchronisierung							x
IA-Anf. 7 - Transparenz	x						
IA-Anf. 8 - Dynamik		x	x		x	x	
KP-Anf. 1 - Verteilung	x		x				
KP-Anf. 2 - Verfügbarkeit				x			
KP-Anf. 3 - Austausch	x					x	
KP-Anf. 4 - Heterogene Geräte			x				
ZA-Anf. 1 - Trennung BL & UI	x						

Tabelle 4.1.: Gegenüberstellung der Anforderungen und Konzepte der Integrationsschicht

Die allgemeine Geräte- bzw. Föderationsverwaltung ist eine eigene Anwendung parallel zur Interaktionsanwendung und von dieser unabhängig. Zur Verwaltung wird u. a. von den Basisdiensten Gebrauch gemacht.

Zwischen physischen Geräten (Server und Endgeräte) und Komponentenplattformen besteht eine 1:1-Relation, d. h. auf jedem Gerät befindet sich eine Instanz der Komponentenplattform. Daher werden die Begriffe „Gerät“, „Endgerät“ und „Komponentenplattform“ als Synonyme für die Komponentenplattform auf dem Endgerät verwendet.

4.2. Spezifikation des Komponentenmodells

Die Komponenten repräsentieren einen bestimmten Teil der Interaktionsanwendung und realisieren dabei eine bestimmte Funktion. Durch das Zusammenspiel von mehreren Komponenten wird die Interaktionsanwendung realisiert. Da sich die Komponenten durch ihre Funktionsbeschreibung finden und koppeln (Unterabschnitt 4.5.1), ist eine genaue Spezifikation der Komponentenfunktion notwendig. In Unterabschnitt 4.2.1 wird erläutert, wie die Eigenschaften einer Komponente beschrieben werden. In Unterabschnitt 4.2.2 wird erläutert, wie der aktuelle Verarbeitungsstatus einer Komponente beschrieben wird. Anschließend wird in Unterabschnitt 4.2.3 eine Erweiterung des Standardlebenszyklus von Komponenten beschrieben, die für verschiedene Konzepte der Integrationsschicht notwendig ist. Die Schnittstellen einer Komponente werden in Unterabschnitt 4.2.4 beschrieben.

4.2.1. Komponentenprofil zur Eigenschaftsbeschreibung

Die Beschreibung einer Komponente besteht aus verschiedenen Teilen wie der Funktionsbeschreibung, der Schnittstellendefinition, dem Abhängigkeitsplan und der Anforderungen an die Ausführungsumgebung, die im Folgenden näher erläutert werden.

Funktionsbeschreibung

Eine Komponente benötigt eine abstrakte Funktionsbeschreibung, deren Elemente aus dem MMI-F hervorgehen. Zu den Funktionen müssen QoS-Parameter angegeben werden, die eine qualitative Unterscheidung von gleichen Funktionen ermöglichen. Die QoS-Parameter werden verwendet, wenn Dienste mit einer bestimmten Funktionalität benötigt werden, die Qualität dieser Funktionalität aber je nach Nutzerpräferenzen oder Systemzustand variieren kann.

Angebotene und benötigte Schnittstellen

Zur vollständigen Beschreibung der Komponente gehört auch eine Liste der von der Komponente angebotenen Schnittstellen (*provided ports*) bzw. die von der Komponente benötigten Komponenten (*required ports*). Die angebotenen Schnittstellen lassen sich in zwei Kategorien einteilen:

Standardschnittstellen sind die für die allgemeine Komponentenfunktionalität benötigten Schnittstellen wie start, stop etc. Diese werden in Unterabschnitt 4.2.4 genauer beschrieben.

Anwendungsschnittstellen werden je nach Verwendungszweck bzw. Funktion der Komponente definiert und sind daher spezifisch für die jeweilige Anwendung.

Zwischen der Anwendungsschnittstelle und der Komponentenfunktion besteht ein gewisser Zusammenhang, da die Schnittstellen von der Funktion abhängen. Dieser Zusammenhang garantiert aber nicht, dass für jede Funktion der Komponente auch eine Schnittstelle vorhanden sein muss. Wenn eine Komponente aus mehreren mit Einzelfunktionen versehenen Komponenten zusammengesetzt wurde, ist es möglich, dass nur die aus der Zusammensetzung entstandenen angebotenen und benötigten Ports nach außen sichtbar sind.
In Abbildung 4.3 sind zwei Komponenten dargestellt, die jeweils eine bestimmte Funktion realisieren (Spracherkennung in Abbildung 4.3a und Sprachinterpretation in Abbildung 4.3b). Aus den Funktionen lassen sich die benötigten und angebotenen Schnittstellen ableiten. Die Spracherkennung verarbeitet rohe Audiosignale vom Mikrofon zu Text und die Sprachinterpretation verarbeitet Text zu Kommandos.

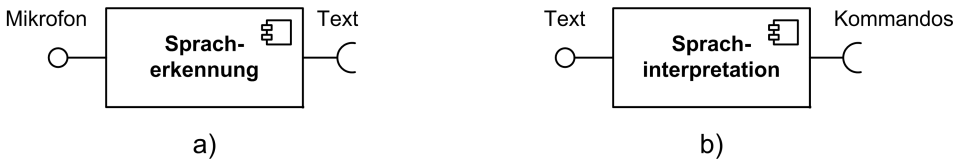

Abbildung 4.3.: Beispiel für einzelne Komponenten

Eine aus den beiden Komponenten aus Abbildung 4.3 zusammengesetzte Komponente ist in Abbildung 4.4 dargestellt. Diese Komponente führt eine Sprachverarbeitung durch, was sowohl Spracherkennung als auch Sprachinterpretation einschließt. Sie verfügt über einen Audiosignal-Eingangsport und einen Kommando-Ausgangsport. Obwohl Sprachinterpretation ein Teil der Komponente ist, ist dieser Teil in diesem Beispiel nicht für andere Komponenten nutzbar. Eine andere Implementierung kann diese Schnittstelle ebenfalls anbieten. Aus diesem Grund ist keine generelle Ableitung von Schnittstellen aus Funktionsbeschreibungen möglich.

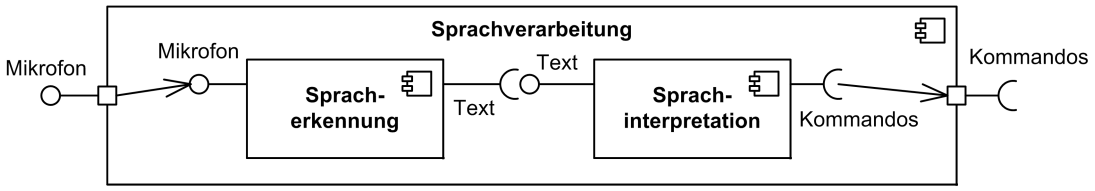

Abbildung 4.4.: Beispiel für eine zusammengesetzte Komponente

Weitere Informationen zu Schnittstellen und Funktionsbeschreibung befinden sich in Unterabschnitt 4.5.1.

Identifikation von Komponenten

Alle Komponenten benötigen eine eindeutige Identifikation, anhand derer die Informationen über Funktionalität und QoS bestimmt werden können. Diese ID definiert den Komponententyp und befindet sich somit auf Klassenebene (`cID`). Um mehrere Komponenten des gleichen Typs unterscheiden zu können, benötigen die Instanzen einer Komponente zur Laufzeit eine zusätzliche ID (`sID`).

Abhängigkeitsplan

Eine Komponente kann optional beschreiben, von welchen anderen Komponenten sie abhängig ist und somit welche Komponenten sie zur Realisierung ihrer Funktionalität zur Laufzeit benötigt. Dabei können die benötigten Komponenten entweder durch die Komponenten-ID referenziert oder durch erforderliche Eigenschaften beschrieben werden:

Komponenten-ID Durch Referenzierung mittels Komponenten-IDs legt der Autor der Komponente auf exakte Weise die benötigten Komponenten fest. Das ist nur dann möglich, wenn diese bereits zur Entwicklungszeit feststehen. Einerseits lässt sich damit das gewünschte Gesamtverhalten der Choreographie am Besten vorherbestimmen, da sich die einzelnen Komponenten genau aufeinander abstimmen lassen. Andererseits ist diese Variante unflexibel, beispielsweise in Bezug auf alternative Implementierungen der gleichen Funktionalität.

Komponentenbeschreibung Durch Referenzierung mittels Komponentenbeschreibung wird vom Autor nur festgelegt, welche Eigenschaften die referenzierte Komponente haben muss, jedoch nicht eine exakte Implementierung. Damit wird die eigentliche Implementierung erst zur Laufzeit ausgewählt und ist somit zum Entwicklungszeitpunkt nicht vorhersehbar. Trotz festgelegter Schnittstellen und Spezifikation der Komponenten kann es zu unterschiedlichem, u. U. nicht absehbaren Verhalten zur Laufzeit kommen. Allerdings ist somit auch eine größere Flexibilität in Bezug auf aktuell verfügbare Implementierungen gewährleistet.

Die Definition benötigter Komponenten ist nicht auf *direkte* Abhängigkeiten beschränkt. Es besteht eine direkte Abhängigkeitsbeziehung zwischen zwei Komponenten, wenn eine Komponente A zur Erfüllung ihrer Aufgaben eine Komponente B benötigt. Wenn Komponente B direkt Komponente C benötigt, dann wird Komponente C von Komponente A *indirekt* benötigt, da die Abhängigkeitsrelation transitiv ist. Indirekt benötigte Komponenten können im Abhängigkeitsplan ebenfalls definiert werden.
Auf diese Weise kann beispielsweise eine Nutzerschnittstellenkomponente die übrigen direkt und indirekt benötigten Unterstützungskomponenten definieren, die für deren Benutzung zum Zugriff auf eine Zielanwendung notwendig sind.

Der Abhängigkeitsplan kann beim Platzieren von Komponenten berücksichtigt werden (siehe Unterabschnitt 4.5.2). Letztlich legt der Nutzer seine Präferenz diesbezüglich durch das Nutzerprofil fest (siehe `DEPLOY_TYPE` in Unterabschnitt 4.4.1). Im Komponentenprofil wird der Abhängigkeitsplan durch die benötigten Schnittstellen repräsentiert.

Anforderungen an die Ausführungsumgebung

Damit eine Komponente sinnvoll platziert werden kann, muss sie definieren, welche Anforderungen sie an ihre Ausführungsumgebung stellt. Trotz einer einheitlichen Basis durch die Komponentenplattformen, kommen sie doch auf unterschiedlichen Geräten und Java-Versionen zur Ausführung. Daher lassen sich die Anforderungen in Hardware- und Software-Anforderungen unterteilen. Hardware-Anforderungen beziehen sich auf die Hardware-Eigenschaften eines Gerätes, wie verfügbare Anzeigemerkmale (z. B. Display-Auflösung), während sich die Software-Anforderungen auf verfügbare Bibliotheken beziehen, die zur Realisierung der Komponentenfunktion erforderlich sind (z. B. Multimediabibliotheken für Audioausgabe).

Repräsentation des Komponentenprofils

Um eine maschinelle Verarbeitung des Komponentenprofils zu ermöglichen, bietet sich eine Repräsentation mittels XML an. In Listing 4.1 ist ein Grundgerüst für die Beschreibung einer Komponente dargestellt. Unter `functions` werden die von der Komponente realisierten Funktionen bzw. Rollen des MMI-F abgelegt, welche als Grundlage für die Komposition fungieren (siehe Unterabschnitt 4.5.1). Die genaue Beschreibung der jeweiligen Funktionen befindet sich im Koppelprofil `couplingprofile`. Das Koppelprofil besteht aus den angebotenen Ports, wodurch die angebotenen Schnittstellen der Komponente für andere zur Verwendung beschrieben werden. Ebenfalls Teil des Kopplungsprofils sind die erforderlichen Ports, welche benötigte Komponenten bzw. Schnittstellen definieren. Diese Information wird für die Platzierung benötigter Komponenten verwendet (siehe Unterabschnitt 4.5.2). Die Anforderungen an `hardware` und `software` sind relevant für die kontextsensitive Platzierung entsprechend der Systemrandbedingungen.

4.2.2. Verarbeitungsstatus von Komponenten

Für Migration und Austausch von Komponenten ist es notwendig, dass der Verarbeitungsstatus der Komponenten (die Sitzungsinformation) ermittelt werden kann. Insbesondere für den Komponentenaustausch ist es notwendig, dass die Sitzungsinformation in einem Format exportiert wird, das von anderen Komponenten importiert und zur Weiterverarbeitung genutzt werden kann.

```
<componentprofile cID="...">
  <functions />
  <couplingprofile />
  <requirements>
    <hardware />
    <software />
  </requirements>
</componentprofile>
```

Listing 4.1: Grundgerüst des Komponentenprofils

Eine Möglichkeit ist die semantische Beschreibung in Form einer Ontologie, die alle Aspekte der Beschreibung von Komponentenzuständen umfasst und u. U. auch offen und flexibel erweiterbar ist. Da die Entwicklung einer Ontologie nicht im Fokus der Arbeit liegt, wird auf eine pragmatischere Variante ausgewichen. Die Komponenten orientieren sich in ihrer Funktionalität an den Rollen des MMI-F. Die Rollen stellen den kleinsten gemeinsamen Nenner dar, d. h. dass keine Komponenten vorgesehen sind, die einer feineren Granularität entsprechen als den Rollen des MMI-F.
Aus diesem Grund wird die Speicherung des Komponentenverarbeitungsstatus ebenfalls an den MMI-F-Rollen ausgerichtet. Jede Komponente, insbesondere auch zusammengesetzte Komponenten, müssen beim Exportieren ihres Zustandes die Informationen aufteilen und in getrennten Bereichen entsprechend der MMI-F-Rollen ablegen. In Listing 4.2 ist das Prinzip beispielhaft dargestellt. Zur Kodierung wird XML verwendet.

```
<componentstate>
  <stateinformation role='integration'>
    <!-- ... -->
  </stateinformation>
  <stateinformation role='styling' modality='visual'>
    <!-- ... -->
  </stateinformation>
</componentstate>
```

Listing 4.2: Beispieldokument zum Speichern der Sitzungsinformationen einer Komponente mit zwei Rollen

Entsprechend des MMI-F gibt es insgesamt 10 verschiedene Rollen. Die Rollen Recognition, Interpretation, Styling und Rendering sind modalitätsspezifisch, sodass diese mit dem `modality`-Attribut genauer spezifiziert werden müssen. Die Menge der möglichen Modalitäten ist nicht begrenzt und wird durch den jeweiligen Anwendungsfall bestimmt. Denkbare Modalitäten sind *visual* für visuelle Ein- und Ausgabe (z. B. Text- oder Bilderdarstellung), *voice* für Sprachein- und -ausgabe oder *mouse* für Eingaben mit der Maus.

Die genauen Inhalte der `<stateinformation>`-Elemente ergibt sich aus der MMI-F-Rolle bzw. aus deren Kombination mit der Modalität.

4.2.3. Erweiterter Lebenszyklus von Komponenten

Der Lebenszyklus von Komponenten definiert unterschiedliche Zustände, in denen sich die Komponenten befinden können. Die zwei wichtigsten Zustände sind *aktiv* und *inaktiv*. Eine *aktive* Komponente wird gerade ausgeführt, sodass sie ihre Funktionalität realisieren kann. Im Gegensatz dazu ist eine *inaktive* Komponente nicht aktiv und kann nicht zur Zusammenarbeit benutzt werden. Der Übergang zwischen den beiden Zuständen wird in der Regel von der Ausführungsumgebung bzw. der Komponentenplattform gesteuert.
Eine *aktive* Komponente ist u. a. durch Sitzungsinformationen gekennzeichnet, die durch Verarbeitung von Eingabedaten entstehen und für die Verarbeitung von weiteren Daten zwingend notwendig sind. Hierbei bezieht sich der Begriff Sitzungsinformation auf den inneren Zustand der Komponente, wie z. B. Variablenbelegungen. Diese Sitzungsinformation ist zeitlich veränderlich entsprechend der Verwendung der Komponente durch andere Komponenten und der damit einhergehenden Verarbeitungsschritte. Außerdem ist der Zustand flüchtig, d. h. beim Übergang der Komponente in den inaktiven Zustand geht die Sitzungsinformation verloren, es sei denn es erfolgt eine persistente Sicherung. Die Sicherung der Sitzungsinformation ist aber nicht in jedem Fall erwünscht, da der Verlust dem Standardverhalten entspricht. Daher muss zwischen dem normalen Stoppen einer Komponente mit Sitzungsinformationsverlust und dem Anhalten ohne Verlust unterschieden werden.
Für die Komponentenmigration ist es wichtig, dass die neue Instanz der migrierenden Komponente mit den Sitzungsinformationen der Originalinstanz initialisiert wird. Damit diese zur Laufzeit aus der Komponente extrahiert werden können, muss die Komponente einen Zustand einnehmen, in dem die Sitzungsinformation so ausgelesen werden kann, dass sie sich zum Initialisieren der neuen Instanz eignet. In diesem Zustand dürfen keine Methodenaufrufe aktiv sein bzw. generell keine Operationen der Komponente auf den Daten durchgeführt werden, damit sich die Sitzungsinformation nicht ändert.
Dazu ist neben aktiv und inaktiv ein weiterer Zustand notwendig, in dem die Komponente zwar ausgeführt wird, sie aber noch nicht für ihren Verwendungszweck benutzt werden kann (siehe Tabelle 4.2). Man kann die Komponente dann als *pausiert* betrachten (Abbildung 4.5). In diesem Zustand können die Sitzungsinformationen konsistent extrahiert werden. Vom *pausiert*-Zustand kann die Komponente entweder in *aktiv* oder *inaktiv* wechseln.
Der *pausiert*-Zustand ist als Zusatz zu den ohnehin vorhandenen Zuständen der jeweiligen Komponentenplattform zu betrachten. Die weiteren Zustände bleiben davon unberührt. Bei OSGi sind das *installed* und *uninstalled*, wobei *resolved* mit *inaktiv* gleichzusetzen ist (vgl. Abbildung 3.6).

	inaktiv	*pausiert*	*aktiv*
Komponente wird ausgeführt	nein	ja	ja
Komponente kann genutzt werden	nein	nein	ja

Tabelle 4.2.: Eigenschaften der drei Zustände einer Komponente

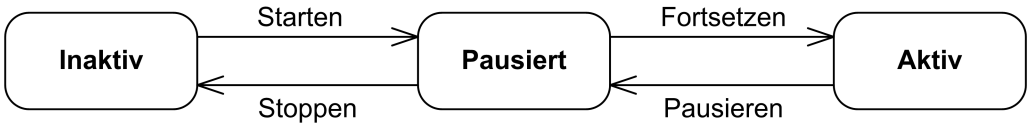

Abbildung 4.5.: Hauptzustände einer Komponente

Aus Abbildung 4.5 geht hervor, dass eine Komponente die folgenden Aktionen unterstützen muss:

Starten Damit wird die Ausführung einer Komponente veranlasst. Dafür muss die Komponente zuvor inaktiv gewesen sein. Nach Abarbeitung des Start-Kommandos ist die Komponente pausiert.

Fortsetzen Durch das Fortsetzen-Kommando wird die Komponente vom pausiert-Zustand in den aktiven Zustand versetzt, nachdem sie pausiert war.

Pausieren Eine aktive Komponente kann durch ein Pause-Kommando angehalten bzw. pausiert werden.

Stoppen Das Stopp-Kommando versetzt eine pausierte Komponente in den inaktiven Zustand. Soll eine aktive Komponente in den inaktiven Zustand versetzt werden, muss sie erst pausiert und dann gestoppt werden.

Der Übergang von *inaktiv* zu *pausiert* ist zunächst nur beim Starten der Komponente nach Migration notwendig. Im pausiert-Zustand kann die Komponente mit den empfangenen Sitzungsdaten initialisiert werden. Soll ein normaler Komponentenstart durchgeführt werden, muss die Komponentenplattform zusätzlich zum Start-Kommando noch das Resume-Kommando an die Komponente schicken, damit diese aktiviert wird, was in dem Fall ohne Initialisierungsdaten geschieht und einem herkömmlichen Start entspricht.

4.2.4. Schnittstellen einer Komponente

Die Kompatibilität einer Komponente mit der Integrationsschicht erfordert einige festgelegte Schnittstellen. Die Schnittstellen lassen sich in drei Gruppen entspre-

chend der Verwendung unterteilen: Konzeptunterstützungs-, Lebenszyklus-, und Anwendungsschnittstellen, welche im Folgenden näher beschrieben werden.

Konzeptunterstützung

Neben den allgemeinen Lebenszyklusschnittstellen müssen weitere Schnittstellen von den Komponenten implementiert werden, da die in Abschnitt 4.5 beschriebenen Konzepte auf eine aktive Unterstützung durch die Komponente angewiesen sind.

`Map extractSessionData()`
: Extrahiert die Sitzungsinformationen aus der Komponente.

`void insertSessionData(Map data)`
: Fügt die Sitzungsinformation in die Komponenten ein.

`void updatePort(String requiredPortID, String providedPort)`
: Aktualisiert die Verbindung zwischen zwei Komponenten, indem dem benötigten Port `requiredPortID` einer Komponente ein angebotener Port `providedPortID` einer anderen Komponente zugewiesen wird.

`String getComponentProfile()`
: Liefert das Komponentenprofil dieser Komponente zurück.

Der Aufruf der beiden Methoden zur Extraktion und Einfügen der Sitzungsinformation ist nur sinnvoll, wenn die Komponente angehalten ist bzw. sich im pausierten Zustand befindet. Andernfalls sollte eine entsprechende Fehlermeldung erzeugt werden.

Lebenszyklus

Die Lebenszyklusschnittstellen sind für die Kontrolle des Lebenszyklus der Komponente erforderlich, wie zuvor in Unterabschnitt 4.2.3 erläutert wurde.

`void start()` Starten der Komponente.

`void stop()` Stoppen der Komponente.

`void resume()` Fortsetzen der Komponente.

`void pause()` Anhalten der Komponente.

Die Methoden `start()` und `stop()` sind die einzigen Methoden, die OSGi für den aktiven Zugriff auf den Lebenszyklus vorsieht. Zusätzlich sind jedoch die `pause()` und `resume()`-Methoden für manche der in Abschnitt 4.5 beschriebenen Konzepte erforderlich.

Anwendung

Damit eine Komponente verwendet werden kann, muss sie ihre Geschäftslogik durch entsprechende Schnittstellen anderen Komponenten zur Verfügung stellen. Diese sind jedoch je nach Implementierung bzw. Verwendungszweck unterschiedlich und können daher hier nicht genauer spezifiziert werden.

4.3. Spezifikation der Komponentenplattform

Wie in Abschnitt 3.5 festgelegt wurde, erfolgt die prototypische Umsetzung des Systems mit OSGi. OSGi stellt bereits einige notwendige Dienste für die Kontrolle des Lebenszyklus der Komponenten bzw. Bundles bereit. Für die Unterstützung der zusätzlichen Konzepte sind die bereits erwähnten Basisdienste notwendig, die in diesem Abschnitt spezifiziert werden. Außerdem wird das Geräteprofil erläutert und auf die Sicherheit eingegangen.

4.3.1. Basisdienste

Die Basisdienste sind u. a. für die Funktionsfähigkeit der Integrationsschicht im Allgemeinen und die Unterstützung der Komponentenkonzepte im Speziellen verantwortlich. Im Einzelnen sind das die Komponentenverwaltung (Komposition, Installation, Migration, Austausch), die Nutzerverwaltung und allgemeine Funktionen.
Da die Basisdienste teilweise den Zugriff auf die Verzeichnisdienste kapseln, sind auch die Methodensignaturen ähnlich. In Abbildung 4.6 ist die Architektur der Komponentenplattform mit zugehörigen Basisdienstkomponenten dargestellt. Die Föderationsverwaltungsanwendung existiert gleichberechtigt neben den Komponenten für die Interaktionsanwendung, die durch die Komponentenplattform flexibel und dynamisch verfügbar gemacht werden können.

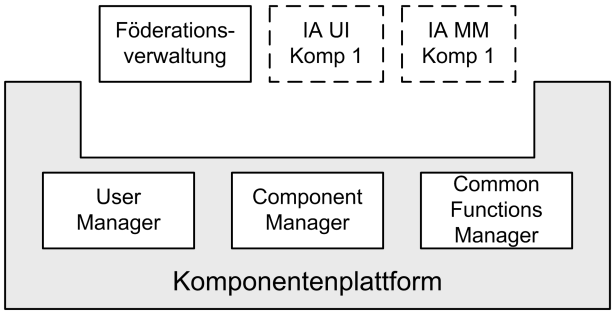

Abbildung 4.6.: Architektur der Komponentenplattform mit Basisdienstkomponenten und Föderationsverwaltungsanwendung

Komponentenverwaltung

Die Schnittstellen der Dienste zur Komponentenverwaltung sind öffentlich und somit nicht nur von der lokalen Plattform aus verwendbar, sondern stehen auch anderen Plattformen bzw. den Verzeichnisdiensten zur Verfügung. Die Funktionen werden vom Component Manager bereitstellt. Zur Komponentenverwaltung gehört die Installation, Deinstallation, Migration und der Austausch von Komponenten. Die einzelnen Dienste der Komponentenverwaltung werden im Folgenden näher definiert.

`boolean requestLocalComponent(int cID, int uID)`
Veranlasst die Installation der durch `cID` definierten Komponente für den Nutzer `uID` auf der lokalen Plattform.

`boolean removeLocalComponent(int sID)`
Entfernt die durch `sID` definierte Komponenteninstanz von der lokalen Komponentenplattform.

`boolean requestRemoteComponent(int cID, int uID, int dID)`
Veranlasst die Installation der Komponente `cID` des Nutzers `uID` auf der Komponentenplattform, die sich auf dem durch `dID` definierten Gerät befindet.

`boolean removeRemoteComponent(int dID, int sID)`
Entfernt die durch `sID` definierte Komponenteninstanz von der Komponentenplattform, die sich auf dem durch `dID` definierten Gerät befindet.

`boolean installComponent(String bin, String s, int uID, int sID)`
Installiert eine Komponente, deren Binärdaten durch `bin` übergeben werden. Durch `s` können Sitzungsdaten übergeben werden, was für die Migration benötigt wird (siehe Unterabschnitt 4.5.3). Ohne Sitzungsinformation entspricht das Ergebnis einer einfachen Komponentenplatzierung. Die Komponente gehört dem Nutzer `uID` und deren Dienst ist bereits unter der ID `sID` registriert.

`boolean migrateComponent(int sourceID, int targetID, int sID)`
Veranlasst die Migration der Komponente `sID` von der Plattform auf Gerät `sourceID` auf die Plattform des Gerätes `targetID`.

`boolean replaceComponent(ReplaceConfig replaceConfig)`
Veranlasst den Austausch von Komponenten entsprechend der übergebenen Austauschkonfiguration `replaceConfig`.

`String suspendComponent(int sID)`
Überführt die durch `sID` spezifizierte Komponente in den *pausiert*-Zustand.

`void resumeComponent(int sID)`
Setzt die durch `sID` spezifizierte Komponente fort.

Nutzerverwaltung

Zur Nutzerverwaltung müssen Methoden für das An- bzw. Abmelden des Nutzers verfügbar sein, da das System prinzipiell Mehrnutzerbetrieb zulässt. Jeder Nutzer bekommt aber seine eigenen Komponenten zugewiesen, damit es nicht zu ungewollten Interferenzen zwischen verschiedenen Nutzern kommt. Die Nutzerverwaltung ist nur auf Endgeräten erforderlich und wird vom User Manager bereitgestellt.

`int login(String userName)`
Meldet den Nutzer anhand seines Nutzernamens `userName` an. Liefert die Nutzer-ID zurück.

`boolean logout(String userName)`
Meldet den Nutzer anhand seines Nutzernamens `userName` ab. Liefert true zurück, wenn der Nutzer endgültig abgemeldet ist oder false, wenn er noch auf anderen Geräten angemeldet ist.

`int addUser(String userName)`
Fügt einen neuen Nutzer anhand seines Nutzernamens `userName` in das Nutzerverzeichnis ein und liefert die dabei erzeugte Nutzer-ID zurück.

`boolean removeUser(String userName)`
Entfernt den Nutzer anhand des Nutzernamens `userName` aus dem Nutzerverzeichnis. Liefert false zurück, wenn das Entfernen fehlgeschlagen ist.

`UserProfile getUserProfile(int uID)`
Liefert das Nutzerprofil des Nutzers `uID` zurück. Der Inhalt des Nutzerprofils ist in Unterabschnitt 4.4.1 beschrieben.

`void updateUserProfile(int uID, UserProfile uProfile)`
Aktualisiert das Nutzerprofil des Nutzers `uID` mit dem neuen Nutzerprofil `uProfile`.

Allgemeines

Die hier beschriebenen Schnittstellen werden vom Common Functions Manager bereitgestellt und sind nur von der lokalen Plattform aus zugreifbar. Dabei handelt es sich um Funktionen, die von der Föderationsverwaltungsanwendung benötigt werden, um dem Nutzer Informationen über Komponenten und Geräte zur Verfügung zu stellen.

`ServiceList getServices(int cID, int uID)`
Liefert eine Liste von Diensten, die durch Komponenten des Typs `cID` bereitgestellt und zur Weiterverarbeitung von der anfragenden Komponente genutzt werden können und dem Nutzer `uID` gehören.

`boolean updateServiceURL(int sID, String url, int dID)`
Aktualisiert die URL (Uniform Resource Locator) `url` und die Geräte-ID `dID` des Dienstes `sID`.

`boolean removeService(int sID)`
Entfernt den Dienst `sID` aus dem Dienstverzeichnis.

`DeviceMap getDevices()`
Liefert die Liste aller Geräte zurück, die im System vorhanden sind.

`ComponentMap getComponents(Properties criteria)`
Liefert eine `ComponentMap` mit allen Komponenten zurück, die den gewünschten Eigenschaften in `criteria` entsprechen.

`DeviceMap getDevicesForComponent(int cID)`
Liefert eine `DeviceMap` zurück, die die Geräte enthält, die für die Platzierung der Komponente `cID` geeignet sind.

Zusammenfassung

In Tabelle 4.3 ist eine Übersicht der Methoden der Basisdienste und deren Sichtbarkeit in der Infrastruktur dargestellt. Globale Sichtbarkeit bedeutet, dass jeder andere Teilnehmer der Infrastruktur diese Methode aufrufen kann. Lokale Sichtbarkeit stellt Methoden nur für die lokale Plattform zur Verfügung. Damit eine Methode von allen Infrastrukturteilnehmern verwendet werden kann (also lokale und entfernte), muss sie sowohl global als auch lokal zugreifbar sein, globale Sichtbarkeit reicht hierfür nicht aus. Im Prinzip ist dies mit den Zugriffsmodifikatoren von Java vergleichbar. Eine als *public* markierte Methode einer Klasse in Java kann von jeder Klasse aus verwendet werden (Instanziierung vorausgesetzt), was globalem und lokalem Zugriff entspricht. Als *private* bzw. *protected* markierte Methoden einer Klasse erlauben nur lokalen Zugriff der Klasse selbst bzw. des jeweiligen Packages. Für globalen ohne lokalen Zugriff gibt es in Java allerdings keinen Zugriffsmodifikator.

4.3.2. Geräteprofil

Das Geräteprofil dient der formalen Beschreibung der an der Integrationsschicht teilnehmenden Geräte. Zusammen mit dem Komponentenprofil (Unterabschnitt 4.2.1) kann bei der Installation von Komponenten vermieden werden,

Basisdienst	Dienste im Einzelnen	Sichtbarkeit global	Sichtbarkeit lokal
Component Manager	`requestLocalComponent`	x	x
	`removeLocalComponent`	x	x
	`requestRemoteComponent`	-	x
	`removeRemoteComponent`	-	x
	`installComponent`	x	-
	`migrateComponent`	x	x
	`replaceComponent`	x	x
	`suspendComponent`	x	-
	`resumeComponent`	x	-
User Manager	`login`	-	x
	`logout`	-	x
	`addUser`	-	x
	`removeUser`	-	x
	`getUserProfile`	-	x
	`updateUserProfile`	-	x
Common Functions Manager	`getDevices`	-	x
	`getDevicesForComponent`	-	x
	`getServices`	-	x
	`updateServiceURL`	-	x
	`removeService`	-	x
	`getComponents`	-	x

Tabelle 4.3.: Basisdienste der Komponentenplattform und deren Sichtbarkeit

dass Komponenten auf Geräten installiert werden, die die erforderlichen Randbedingungen nicht erfüllen (siehe Unterabschnitt 4.5.2). Dabei ist das Geräteprofil als „Angebot“ und das Komponentenprofil als „Nachfrage“ zu verstehen.
Das Geräteprofil besteht aus zwei Teilen: Hardware und Software. Die Hardwarebeschreibung beinhaltet Informationen zu den Hardwareeigenschaften des Gerätes. Hierzu gehören persistenter Speicher (insgesamt und verfügbar), Leistung der CPU (Central Processing Unit), Netzwerkfähigkeiten und RAM (Random Access Memory). Außer den „internen“ Kriterien sind Informationen über Interaktionsmöglichkeiten und die dafür vorhandenen Geräte von Interesse. Diese Geräte können in Geräte für Ein- und Ausgabe unterteilt werden. Eingabegeräte können für die Eingabe von Text (Tastatur) und Koordinaten (Maus, Pen) verwendet werden. Als Ausgabegeräte kommen Monitor (für Text und Grafiken), Lautsprecher (für Text) etc. in Frage.
Die Kategorie Software dient der Beschreibung aller software-bezogenen Eigenschaften, die das Gerät und die darauf befindliche Integrationsschicht charakteri-

siert. Dazu gehört die verfügbare Java-Version, die OSGi-Laufzeitumgebung und die Betriebssystemversion.

4.3.3. Sicherheit

Die Kommunikation zwischen einzelnen Komponenten der Architektur muss so gesichert werden, dass die Vertraulichkeit und Integrität gewahrt ist. Das bedeutet, dass Kommunikationsnachrichten weder abgehört, hinzugefügt oder modifiziert werden dürfen. Dafür können Standardtechnologien wie beispielsweise HTTPS (HyperText Transfer Protocol Secure) eingesetzt werden.

4.3.4. Anwendung zur Föderationsverwaltung

Die Anwendung zur Föderationsverwaltung gibt dem Nutzer die volle Kontrolle über die an der Föderation teilnehmenden Geräte und den darauf verwendeten Modalitäten bzw. Interaktionsanwendungsteilen. Sie stellt für den Nutzer den Einstiegspunkt in das System föderierter Endgeräte und verteilter Nutzerschnittstellen dar. Hierfür ist mindestens erforderlich, dass sich der Nutzer am System an- und abmelden kann.
Nach erfolgreicher Anmeldung kann sich der Nutzer die Liste verfügbarer Endgeräte anzeigen lassen und diese ggf. in die Föderation aufnehmen. Die Aufnahme geschieht implizit dadurch, dass der Nutzer einem Gerät aus der Liste eine Nutzerschnittstellenkomponente zuweist, wodurch dieses zu einem Teil der verteilten Nutzerschnittstelle wird.
Weiterhin kann der Nutzer Informationen über die aktuelle Föderation bekommen, d. h. welche Endgeräte sind beteiligt, welche Modalitäten werden genutzt und welche Komponenten kommen auf welcher Plattform zur Ausführung. Je nach Bedarf des Nutzers ist es auch möglich, bestimmte UI (User Interface)-Komponenten wieder zu entfernen. Damit geht implizit einher, dass auch das Gerät nicht mehr zur Föderation gehört, wenn es über keine Nutzerschnittstellenkomponenten mehr verfügt.
Die Funktionen der Verwaltungsanwendung im Einzelnen:

- Komponenten auf Geräte verteilen, die Verteilung ändern (Komponenten migrieren oder austauschen) und die Verteilung aufheben (Komponenten deinstallieren
- Nutzer an- und abmelden
- Nutzerprofil bearbeiten

In fortgeschrittenen Realisierungen kann die Föderationsverwaltungsanwendung auch dazu benutzt werden, die Verteilung der gesamten verwendeten Komponenten anzupassen. Dies ist beispielsweise dann interessant, wenn der Nutzer vorhersehen kann, dass er sich aus der Erreichbarkeit der Infrastruktur entfernt, sodass

Name	Typ	Beschreibung
id	int	Nutzer-ID
name	String	Nutzername
profile	String	Nutzerprofil (z. B. Art der Eingabefusion, Verteilung, Synchronisierung)
loggedin	int	Anzahl der Endgeräte, auf denen der Nutzer angemeldet ist

Tabelle 4.4.: Datenbankschema des Nutzerverzeichnisses

die sich dort befindlichen Komponenten nicht mehr verwendet werden können. In diesem Fall kann der Nutzer veranlassen, dass die benötigten Komponenten auf den Geräten verteilt werden, die der Nutzer immer bei sich führt und die somit stets verfügbar sind. Dabei muss es sich nicht zwangsläufig um Endgeräte handeln. Im Umkehrschluss kann der Nutzer auch die explizite Verwendung von Infrastrukturgeräten fordern, wenn sie sich in Reichweite befinden. Für die Realisierung dieser Funktionalität sind weitere Konzepte beispielsweise aus dem Bereich der Kontextsensitivität erforderlich, um die vom Nutzer bei sich geführten Geräte zu ermitteln. Das impliziert, dass es sich zwangsläufig um mobile Geräte handelt, aber nicht jedes dem Nutzer zugeordnete mobile Gerät wird von ihm auch mitgeführt.

4.4. Spezifikation der Verzeichnisdienste

Die folgenden Verzeichnisdienste werden benötigt, um die Verwaltung der Komponenten und Geräteföderation zu ermöglichen. Zur Verwaltung gehört u. a. das Ermitteln von Referenzen auf Komponenten, was zur Realisierung der Plattformkonzepte wie in Abschnitt 4.5 beschrieben notwendig ist.

4.4.1. Nutzerverzeichnis

Das Nutzerverzeichnis dient der Verwaltung aller am System angemeldeten Nutzer. Ein Nutzer ist durch eine eindeutige Nutzer-ID und einen Nutzernamen gekennzeichnet. Die Nutzereinstellungen werden ebenfalls im Nutzerverzeichnis gespeichert. Außerdem wird im Nutzerverzeichnis die Anzahl der Endgeräte festgehalten, von denen aus der Nutzer angemeldet ist. In Tabelle 4.4 befindet sich das Datenbankschema für das Nutzerverzeichnis.
Das Nutzerverzeichnis kann über die folgenden Schnittstellen benutzt werden. Zunächst muss es möglich sein, neue Nutzer anzulegen bzw. diese wieder zu löschen, wenn sie nicht gebraucht werden. Nach der Registrierung kann sich ein Nutzer am System an- und abmelden.

`int addUser(String userName)`
Mit dieser Methode kann ein neuer Nutzer angelegt werden, der den Namen `userName` trägt. Zurückgeliefert wird die Nutzer-ID oder -1, wenn der Nutzer schon vorhanden ist.

`boolean removeUser(String userName)`
Diese Methode dient dem Löschen von Nutzern entsprechend dem übergebenen Nutzernamen `userName`.

`int login(String userName)`
Über diese Methode kann ein Nutzer mit dem Nutzernamen `userName` angemeldet werden. Es wird die Nutzer-ID zurückgeliefert oder -1, wenn der Nutzer nicht existiert.

`boolean logout(String userName)`
Über diese Methode kann ein Nutzer mit dem Nutzernamen `userName` abgemeldet werden.

Das Nutzerverzeichniss enthält weiterhin das Nutzerprofil, über die der Nutzer die Integrationsschicht entsprechend seiner Wünsche konfigurieren kann. Für das Nutzerprofil wurde ein Objekt geschaffen, welches die Einstellungen beinhaltet. In der Datenbank wird eine XML-Repräsentation des Objektes in Form eines Strings abgelegt. Daher gibt es die zwei folgenden Methoden zum Zugriff auf die Nutzereinstellungen:

`UserProfile getUserProfile(int uID)`
Liefert die Einstellungen des Nutzers `uID` zurück.

`void updateUserProfile(int uID, String uProfile)`
Aktualisiert die Einstellungen eines Nutzers `uID` mit dem neuen Nutzerprofil `uProfile`.

Das Nutzerprofil dient der Konfiguration der Integrationsschicht entsprechend den Wünschen des Nutzers. Es wird im Feld `profile` im Nutzerverzeichnis abgelegt. Folgende Aspekte werden vom Nutzerprofil unterstützt:

`FUSION_TYPE` Diese Einstellung definiert, welche Art der Eingabefusion durchgeführt werden soll. Mit Eingabefusion ist die Zusammenführung von verschiedenen Eingabekanälen gemeint, um z. B. zusammengesetzte Multimodalität zu realisieren. Mögliche Werte sind:

- **`NONE`** Es wird keine Eingabefusion gewünscht (default).
- **`SIMPLE_FUSION`** Die Eingaben sollen fusioniert werden.

SYNCHRONIZATION_TYPE Diese Nutzereinstellung definiert, welche Art Algorithmus zur synchronen Aktualisierung von Nutzerschnittstellen verwendet werden soll. Genaue Informationen zu den einzelnen Sychronisierungsalgorithmen befinden sich in Abschnitt 4.6. Mögliche Werte sind:

- **NONE** Es wird keine besondere Behandlung zur synchronen Aktualisierung von Nutzerschnittstellen gewünscht (default).
- **SERVER_DELAY_INSERTION** Es soll der Algorithmus mit server-basierter Verzögerung schnellerer Nutzerschnittstellen verwendet werden.
- **CLIENT_ADAPTIVE_DELAY_INSERTION** Es soll der adaptive Algorithmus mit client-basierter Verzögerung und Adaption an den aktuellen Systemzustand verwendet werden.

DEPLOY_TYPE Diese Nutzereinstellung definiert, wann die Komponenten beim Start der Interaktionsanwendung platziert werden sollen. Mögliche Werte sind:

- **PREDEPLOY** Es sollen alle notwendigen Komponenten durch den Deployment Manager anhand einer Konfigurationsdatei vorplatziert werden, obwohl sie in dem Augenblick noch gar nicht benötigt werden.
- **ON_DEMAND** Neue Komponenten werden nur bei Bedarf bzw. Anforderung platziert (default).

Genauere Informationen zu den Platzierungsstrategien befinden sich in Unterabschnitt 4.5.2.

DISTRIBUTION_TYPE Diese Nutzereinstellung gibt an, wie die Komponenten beim Platzieren verteilt werden sollen. Mögliche Werte sind:

- **CENTRAL** Damit werden neu zu platzierende Komponenten möglichst so platziert, dass sie wenige Geräte benötigen.
- **DECENTRAL** Damit werden neu zu platzierende Komponenten möglichst so platziert, dass sie gut verteilt sind (default).

Genaue Informationen zu den Verteilungsstrategien befinden sich in Unterabschnitt 4.5.2.

SYNC_PAGE Gibt an, ob eine Synchronisierung auf *page*-Ebene durchgeführt werden soll (default: true).

SYNC_EVENT Gibt an, ob eine Synchronisierung auf *event*-Ebene durchgeführt werden soll (default: false).

Name	Typ	Beschreibung
componentID	int	eindeutige ID der Komponente
componentName	String	Komponentenname
componentProfile	Properties	Komponentenprofil
binaries	byte[]	Binärdaten der Komponente

Tabelle 4.5.: Datenbankschema des Komponentenverzeichnisses

4.4.2. Komponentenverzeichnis

Das Komponentenverzeichnis beinhaltet die verfügbaren Komponenten. Damit wird das Komponentenverzeichnis u. a. für die Anforderung von neuen Instanzen der Komponenten und somit deren Dienste benötigt. Nachdem eine Komponente angefordert und instanziiert wurde, erfolgt die Registrierung des Dienstes im Dienstverzeichnis. Das Datenbankschema des Komponentenverzeichnisses ist in Tabelle 4.5 dargestellt. Es kann über die folgenden Schnittstellen verwendet werden.

`int addComponent(String name, ComponentProfile p, String b)`
Mit dieser Methode kann eine neue Komponente in das Komponentenverzeichnis aufgenommen werden. Dafür muss ein Name `name`, das Komponentenprofil `p` und die Binärdaten `b` übergeben werden, die eindeutige Komponenten-ID `cID` wird zurückgeliefert.

`void removeComponent(int cID)`
Entfernt die durch `cID` definierte Komponente aus dem Komponentenverzeichnis.

`ComponentList getComponents(Properties criteria)`
Liefert eine Liste mit allen Komponenten, die den in `criteria` definierten Kriterien entsprechen. Die Kriterien werden gegen das Komponentenprofil abgeprüft und nur bei positivem Ergebnis der `ComponentList` hinzugefügt. Die `ComponentList` beinhaltet eine Menge von `Component`s, die jeweils aus ihrem Namen, ID und Profil bestehen. Die Binärdaten werden gesondert abgerufen.

`byte[] getBinaries(int cID)`
Liefert die Binärdaten der Komponente `cID` zurück.

Die Binärdaten beinhalten das JAR (Java ARchive)-File des OSGi-Bundles, was die Komponentenfunktionalität realisiert und auf den Plattformen ausgeführt werden kann. Diese müssen einem einheitlichen OSGi-Standard entsprechen bzw. die gleichen Bibliotheken zur Verfügung stellen.

4.4.3. Dienstverzeichnis

Das Dienstverzeichnis dient der Registrierung aller Dienste, die in der Integrationsschicht vorhanden sind. Diese werden von anderen Diensten gefunden und verwendet. Das Datenbankschema des Dienstverzeichnisses ist in Tabelle 4.6 dargestellt. Es kann über die folgenden Schnittstellen verwendet werden.

Name	Typ	Beschreibung
serviceID	int	eindeutige ID der Dienstes
componentID	int	eindeutige ID der Komponente
userID	int	eindeutige ID des Nutzers
deviceID	int	eindeutige ID des Gerätes, wo der Dienst angeboten wird
url	String	URL, mit der der Dienst erreicht werden kann
rip	boolean	Indikator für „replacement in progress“

Tabelle 4.6.: Datenbankschema des Dienstverzeichnisses

`String requestService(int cID, int uID, int dID)`
Fordert einen Dienst der Komponente vom Typ `cID` an, die dem Nutzer `uID` gehört und auf Gerät `dID` platziert werden soll. Es wird die `url` des Dienstes zurückgeliefert, mit der der Dienst aufgerufen werden kann. Wenn kein Gerät spezifiziert wird, wird eins vom Dienstverzeichnis bestimmt.

`boolean removeService(int sID)`
Entfernt den Eintrag des Dienstes `seID` aus dem Dienstverzeichnis.

`ServiceList getServices(int cID, int uID)`
Liefert Einträge des Dienstverzeichnisses des Nutzers `uID` zurück, wobei die Dienste von Komponenten des Typs `cID` bereitgestellt werden. Die `ServiceList` enthält `Services`, welche aus `cID`, `sID`, `dID`, `url` und `uID` bestehen. Wird die `cID` *-1* übergeben, so werden alle dem Nutzer zugeordneten Dienste zurückgeliefert.

`boolean updateServiceURL(int sID, String url, int dID)`
Mit dieser Methode wird die `url` des Dienstes `sID` und dessen Gerät `dID` aktualisiert.

`boolean requestMigratedService(int sID, int targetID, String s)`
Mit dieser Methode wird die Migration des Dienstes `sID` auf das Gerät `targetID` ausgelöst. Die Sitzungsinformationen befinden sich in `s` und sind XML-kodiert.

Name	Typ	Beschreibung
deviceID	int	eindeutige ID des Gerätes
ipAddress	String	IP-Adresse des Gerätes
port	int	Port des Gerätes
deviceProfile	Map	Eigenschaften des Gerätes

Tabelle 4.7.: Datenbankschema des Geräteverzeichnisses

`boolean replaceComponents(ReplaceConfig config)`
Mit dieser Methode wird der Austausch von Komponenten veranlasst. Die Konfiguration `config` enthält die Quell-Dienst-IDs und die Ziel-Komponenten-IDs.

4.4.4. Geräteverzeichnis

Das Geräteverzeichnis dient der Erfassung aller Geräte, auf denen Plattforminstanzen der Integrationsschicht verfügbar sind. Damit beinhaltet das Geräteverzeichnis eine Menge der Geräte, die durch den Nutzer für eine Föderation oder durch das System für die Platzierung von Komponenten ausgewählt und genutzt werden können. Neben der ID des Gerätes muss auch die IP-Adresse gespeichert sein. Mit Hilfe der IP-Adresse kann auf das Gerät zu allgemeinen Verwaltungszwecken zugegriffen werden, da die Verwaltungsschnittstelle auf allen Geräten gleich ist. Darüber hinaus muss jedes Gerät seine Eigenschaften in Form eines Geräteprofils spezifizieren (Unterabschnitt 4.3.2), die für eine sinnvolle Platzierung von Komponenten wichtig sind (siehe Unterabschnitt 4.5.2). Das Datenbankschema für das Geräteverzeichnis ist in Tabelle 4.7 dargestellt. Es kann über folgende Schnittstellen verwendet werden.

`int addDevice(String ip, int port, DeviceProfile profile)`
Fügt ein Gerät mit IP-Adresse `ip`, Port `port` und Geräteprofil `profile` in die Gerätedatenbank ein. Liefert die `dID` des Gerätes zurück.

`void removeDevice(int dID)`
Entfernt das Gerät `dID` aus dem Geräteverzeichnis.

`DeviceMap getDevices(Properties preferences)`
Liefert eine Map mit allen Geräten zurück, die die gewünschten Eigenschaften `preferences` aufweisen. Die Rückgabe-Map enthält die `dID` als Schlüssel und das `Device` als Wert, welches die charakteristischen Werte `dID`, `ipAddress`, `port` und `deviceProfile` beinhaltet.

`void setDeviceProfile(int dID, DeviceProfile profile)`
Aktualisiert das Geräteprofil des Gerätes `dID` mit dem neuen Geräteprofil `profile`.

4.4.5. Synchronisierungsdatenbank

Für das in Abschnitt 4.6 beschriebene Synchronisierungskonzept ist die Verarbeitungsrate der Komponenten und Netzwerkverbindungen zwischen Aussenden und Ausgabe der Nutzerschnittstellenbeschreibungen wichtig. Die Synchronisierungsdatenbank dient somit zur Datenhaltung der benötigten Verarbeitungsraten. Das Datenbankschema befindet sich in Tabelle 4.8. Die Synchronisierungsdatenbank kann über folgende Schnittstellen verwendet werden.

Name	Typ	Beschreibung
serviceID	int	eindeutige ID des Dienstes
targetServiceID	int	eindeutige ID des Dienstes, der Ziel der Kommunikation ist
userID	int	eindeutige ID des Nutzers
timestamp	long	Zeitpunkt der Verarbeitungszeitermittlung
rate	double	ermittelte Verarbeitungsrate

Tabelle 4.8.: Datenbankschema der Synchronisierungsdatenbank

`void addValue(int sID, int dID, int uID, long time, double rate)`
Fügt eine gemessene Verarbeitungsrate `speed` zum Zeitpunkt `time` für die Komponenten-ID `sID` des Nutzers `uID` in die Synchronisierungsdatenbank ein. Wenn die Komponenten-ID `dID` ungleich -1 ist, dann handelt es sich um die Netzwerkübertragungsrate zwischen `sID` und `dID`.

`MeasurementList getValues(int sID)`
Liefert eine Liste mit allen gemessenen Verarbeitungsraten des Dienstes `sID` und deren Zeitstempel zurück.

4.5. Konzeptunterstützung der Basisdienste

Die Konzeptunterstützung durch die Basisdienste umfasst verschiedene Phasen von der Installation über Ausführung und Kopplung bis zu Migration und Austausch von Komponenten. Hierbei sind für manche Konzepte unterschiedliche Realisierungsvarianten möglich, deren Vor- und Nachteile nicht pauschal die Verwendung einer Variante motivieren. Daher werden in Kapitel 5 diese Varianten gegenübergestellt und hinsichtlich ihrer Verwendbarkeit eingeschätzt.
In Unterabschnitt 4.5.1 wird zunächst erläutert, wie die Kopplung von bereits installierten Komponenten abläuft. Anschließend wird in Unterabschnitt 4.5.2 erläutert, wie neue Komponenten auf die Plattformen verteilt und installiert werden. Unterabschnitt 4.5.3 beinhaltet das Migrationskonzept zur Sicherstellung der Verfügbarkeit einer Komponente. Am Ende wird in Unterabschnitt 4.5.4 der Austausch von Komponenten zur Laufzeit beschrieben.

4.5.1. Kopplung der Komponenten

Damit Komponenten miteinander kommunizieren können, müssen sie gekoppelt werden. Die Kopplung von Komponenten erfolgt in einer service-orientierten Architektur auf Basis der Dienste, welche von den Komponenten angeboten werden. Dafür gibt es zwei Möglichkeiten: Orchestrierung und Choreographie [MtB07].

Die *Orchestrierung* setzt eine zentrale Kontrolleinheit voraus, die die Orchestrierung durchführt und koordiniert. Die Koordination erfolgt auf Basis von definierter Prozesslogik, durch die zwei oder mehr Dienste miteinander verbunden werden können. Auf diese Weise können einzelne Dienste zusammengesetzt und ein höherwertiger Dienst angeboten werden. Der Vorteil dabei ist die Verwendung von bestehenden Diensten, die für die Realisierung des jeweiligen Prozesses nicht modifiziert werden müssen. Eine übliche Implementierung der Orchestrierung folgt dem *hub-and-spoke*-Prinzip (dt. „Nabe-Speiche-Prinzip“), welches die Kommunikation zwischen zwei Knoten über einen zentralen Vermittler vorsieht. Dabei bezeichnet man den Vermittler als *hub*, die Kommunikationsverbindungen als *spoke*.

Zur Beschreibung von Orchestrierungen, die mehrere Web Services miteinander verbinden, wurde BPEL (Business Process Execution Language) entwickelt [ACD+03], was auch unter den Namen BPEL4WS (BPEL for Web Services) oder WS-BPEL (Web Services BPEL) bekannt ist. Mit BPEL wird ein Prozess inklusive der daran beteiligten Kommunikationspartner, die ausgetauschten Nachrichten und benötigten Informationsflüsse definert. Zusätzlich sind unter anderem bedingte Verzweigungen und Thread-Kontrolle möglich.

Im Gegensatz dazu basiert die *Choreographie* auf lose gekoppelten Diensten, die sich dynamisch zu Kollaborationen zusammenfinden. Hierfür ist keine zentrale Kontrolleinheit notwendig. Bei einer Choreographie verfügt jeder teilnehmende Dienst über bestimmte Funktionenen, die den Dienst definieren. Zur Spezifikation von Choreographien von Web Services existieren Choreographiesprachen, die jedoch im Allgemeinen bisher eher unausgereift sind [MtB07] und wenig Einsatz finden. Ein Beispiel ist WS-CDL (Web Service Choreography Description Language) [Wor05c]. Mit Hilfe von WS-CDL werden komplexe Aufgaben durch benötigte Kommunikationsvorgänge spezifiziert. Die gesamte Funktionalität einer Choreographie erfolgt mittels peer-to-peer Interaktionen zwischen den einzelnen Diensten.

Laut [Erl05] besteht einer der Hauptunterschiede zwischen Orchestrierung und Choreographie in der möglichen Integration von Diensten bezüglich der Grenzen von Unternehmen. Demnach sind Orchestrierungen zur Prozessbeschreibung innerhalb eines Unternehmens geeignet, während mittels Choreographie insbesondere die unternehmensübergreifende Kommunikation unterstützt wird. Technisch gesehen besteht hierfür allerdings kein Grund, es kann auch unternehmensübergreifend eine Orchestrierung eingesetzt werden, solange die entsprechenden Dienste über das Internet erreichbar sind.

Da die zu entwickelnde Integrationsschicht möglichst flexibel in Bezug auf die Kopplung der einzelnen Komponenten bzw. deren Dienste sein soll, kommt nur eine Komposition mittels Choreographie in Frage. Die Orchestrierung setzt einen zentralen Koordinator voraus, welcher die einzelnen Dienste verbindet und deren zusammengesetzte Funktion gekapselt weitergibt. Dies ist für eine dynamische Kopplung von unterschiedlichen Diensten nicht geeignet, zumal bei Ausfall des Koordinators oder Verbindungsabbruch zwischen Koordinator und Dienst die ganze Komposition beeinträchtigt wird.

Kompositionsdefinition mittels WS-CDL

Eine Kompositionsdefinition auf Basis von WS-CDL wird mit dem Ziel erstellt, die Kollaborationen zwischen beliebigen Teilnehmern auf eine präzise Weise zu spezifizieren [Wor05c]. Hierbei wird eine globale Sicht der Komposition erzeugt, die Voraussetzungen und Einschränkungen des Nachrichtenaustauschs beinhaltet und somit deren Verhalten beschreibt. Damit ist es möglich, Implementierungen entsprechend eines WS-CDL-Dokuments zu erstellen, wobei die Logik innerhalb der einzelnen Endpunkte beliebig geändert werden kann, solange die globalen Regeln bzw. das vereinbarte Verhalten der Choreographie eingehalten werden. Eine so definierte Choreographie entspricht einer vertraglichen Vereinbarung zwischen den Teilnehmern über die auszutauschenden Nachrichten.
In Listing 4.3 ist die Syntax des package-Elements einer mit WS-CDL definierten Choreographie dargestellt. Zunächst werden die benötigten Rollen mittels `roleType` definiert, die dann u. U. an verschiedenen Relationen `relationshipType` teilnehmen können. Durch `participantType` als Typ von Choreographieteilnehmern wird festgelegt, welche Rollen von einer Instanz dieses Typs realisiert werden müssen. Mit `channelType` werden Kollaborationspunkte zwischen `participantTypes` definiert, indem u. a. Art und Ort des Informationsaustauschs festgelegt werden. Durch das Element `informationType` wird von konkreten Datentypen aus WSDL (Web Services Description Language) oder XML Schema abstrahiert. Die Elemente `token` und `tokenLocator` werden für die Verwaltung von Daten verwendet, die in Variablen oder Nachrichten von der Choreographie verwendet werden.
Eine wesentliche Eigenschaft von WS-CDL ist, dass es keine „executable business process description language“ ist. WS-CDL-Dokumente sind also nicht zum Ausführen der durch sie definierten Prozesse geeignet. Die Sprache ist nur zur Entwurfszeit einer Choreographie interessant und dient als Richtlinie für die Implementierung der einzelnen Teilnehmer. Zur Laufzeit hat das Choreographiedokument keine Bedeutung mehr. Somit kann auch auf die explizite WS-CDL-Definition einer Choreographie verzichtet werden, wenn die jeweiligen Dienste bestimmte Regeln und Formate implizit definieren und sich die zugehörigen Kommunikationspartner daran halten. Auch wenn die Choreographiespezifikation flexibel angepasst werden kann, so ist deren Umsetzung auf konkrete Implementie-

```
<package
   name="NCName"
   author="xsd:string"?
   version="xsd:string"?
   targetNamespace="uri"
   xmlns="http://www.w3.org/2005/10/cdl">

   <informationType/>*
   <token/>*
   <tokenLocator/>*
   <roleType/>*
   <relationshipType/>*
   <participantType/>*
   <channelType/>*

   Choreography-Notation*
</package>
```

Listing 4.3: Syntax eines Choreographiepackages in WS-CDL (aus [Wor05c])

rungen stets mit einem gewissen Aufwand verbunden, sodass es praktikabler ist, wenn jede Komponente selbst ihre Funktionen, Schnittstellen und Verhaltensweisen definiert und diese von den Kollaborationspartnern berücksichtigt werden. So können sich die Komponenten flexibler zusammenfinden, als als es bei strikter Einhaltung der Choreographiespezifikationen der Fall wäre.

Spezifikation des Kopplungsprofils

Das Kopplungsprofil beinhaltet die Funktion und deren Eigenschaften einer Komponente, inklusive der Schnittstellen und Verhaltensweisen. Während ein WS-CDL-Dokument eine Choreographie-zentrische Sicht auf die Dienste und deren Komposition darstellt, ist das Kopplungsprofil eine Dienst-zentrische Teilmenge des Choreographiedokuments, wobei nur die Aspekte betrachtet werden, die die jeweilige Komponente betreffen.
In Listing 4.4 ist ein einfaches Beispiel des Kopplungsprofils dargestellt. Das Kopplungsprofil enthält zunächst eine Menge von `functions`, die der Spezifikation der abstrakten Funktionsarten dienen, welche von der Komponente realisiert werden. Hierbei handelt es sich im Wesentlichen um die durch das MMI-F definierten Rollen eines multimodalen Systems.
Jede Funktion (und damit jede Rolle) kann durch die `providedports` eine beliebige Anzahl von Ports bereitstellen, über die die Komponente von anderen Komponenten verwendet werden kann. Jeder angebotene Port `providedport` entspricht einer Methode mit zugehörigem Namen `methodname` und Methoden-

```
<couplingprofile>
  <function id="R1">
    <providedports>
      <providedport portid="P1">
        <methodname>Methodname</methodname>
        <methodsignature>
          <param name="param1" type="int" />
          <returnvalue type="int" />
        </methodsignature>
        <description />
      <providedport>
    </providedports>
    <requiredports>
      <requiredport cID="C2" portid="P1" />
      <requiredport>
        <description />
      </requiredport>
    </requiredports>
  </function>
</couplingprofile>
```

Listing 4.4: Einfaches Beispiel des Kopplungsprofils

signatur `methodsignature`, welche aus beliebig vielen Parametern `param` und einem Rückgabewert `returnvalue` besteht. Außerdem muss für jeden Port eine Beschreibung erzeugt werden, welche u. a. Informationen über die QoS-Eigenschaften des Ports beinhaltet, aber auch die Semantik der Portfunktion beschreiben kann.

Neben den angebotenen Ports kann durch das Kopplungsprofil auch definiert werden, welche Ports von der jeweiligen Funktion benötigt werden `requiredports`. Ein solcher benötigter Port `requiredport` ist durch eine Komponenten-ID und die Port-ID gekennzeichnet. Alternativ kann ein benötigter Port auch über seine semantische Beschreibung spezifiziert werden. Dadurch wird keine exakte sondern eine beliebige Komponente referenziert, solange diese die beschriebene Funktionalität erfüllt.

Für die Spezifikation des Kopplungsprofils könnte WS-CDL prinzipiell als Grundlage dienen. Allerdings gibt es für die Verarbeitung von WS-CDL-Dokumenten kaum Werkzeugunterstützung und WS-CDL müsste so modifiziert werden, dass aus der Choreographie-zentrischen Sicht eine Komponenten-zentrische Sicht abgeleitet werden kann, die darüber hinaus u. U. auf demselben Grunddokument aufbaut. Da für das Kopplungsprofil nur ein geringer Teil des Sprachumfangs von WS-CDL benötigt wird, kommt für die Spezifikation ein selbst-definierter XML-Dialekt zum Einsatz. Ein weiterer Grund hierfür ist die

dynamische Wiederverwendbarkeit der Komponenten, die somit nicht nur Teil einer Komposition sein müssen, sondern mit u. U. unterschiedlichen Diensten an unterschiedlichen Kompositionen teilnehmen können. Weiterhin werden in einer mit WS-CDL spezifizierten Choreographie nur die Typen von Rollen, Kanälen, Verbindungen usw. beschrieben, ohne jedoch auf konkrete Kardinalitäten der jeweiligen Umsetzung einzugehen. Daher ist es mit WS-CDL nicht möglich, Choreographien auf Instanzebene so zu definieren, dass beispielsweise zwei Instanzen des gleichen Typs benötigt werden, die jedoch die gleichen Kanäle und Interaktionen verwenden.

Ablauf der Kopplung

Die Integrationsschicht basiert auf einer losen Kopplung von Komponenten. Die Komponenten sind eigenständig und können unabhängig von anderen Komponenten ausgeführt werden. Zur Erfüllung der Gesamtfunktionalität der Komposition ist es jedoch erforderlich, dass die Komponenten miteinander kommunizieren. Das setzt voraus, dass sich die Komponenten suchen und finden können. Demzufolge ist das Ergebnis eines Suchvorgangs eine (oder mehrere) Referenzen auf Komponenten bzw. auf die angebotenen Dienste.

Damit eine Suche durchgeführt werden kann, müssen Kriterien definiert werden, nach denen gesucht werden kann. Diese Kriterien werden durch das Kopplungsprofil als Teil des Komponentenprofils definiert. Darüber hinaus wurde entweder durch eine explizite Choreographiespezifikation oder implizit durch entsprechende Implementierung festgelegt, welche verschiedenen Funktionen im System existieren, welche Komponenten miteinander kommunizieren und wie die Kommunikation erfolgt. Daher verfügt eine Komponente stets über die Information, was mit ihren Ergebnissen zu tun ist bzw. welche Funktion für die Weiterverarbeitung zuständig ist.

In Abbildung 4.7 ist die Interaktion zwischen Integrationsschicht-Elementen zur Ermittlung einer Komponentenreferenz dargestellt. Die gewünschte Komponente kann auf zwei Arten spezifiziert werden, die unabhängig von der Interaktion in Abbildung 4.7 sind. In beiden Fällen wird eine Komponente angefragt (Schritt 1 in Abbildung 4.7). Einerseits kann entweder nach einer bestimmten `cID` gesucht werden (Fall 1), wodurch die gewünschte Komponente exakt bestimmt wird. Andererseits kann die Anfrage an das Dienstverzeichnis eine funktionale Beschreibung der gewünschten Komponente beinhalten (Fall2), wodurch die konkrete Auswahl dem Dienstverzeichnis überlassen wird (2). In jedem Fall antwortet das Dienstverzeichnis entweder mit einer entsprechenden URI zur Komponente oder mit einer Fehlermeldung (3). Im positiven Falle kann die ermittelte URI zur Kommunikation mit der gewünschten Komponente verwendet werden (4). Ansonsten ist es Aufgabe des Anfragers, eine geeignete Fehlerbehandlung durchzuführen (z. B. die Restriktionen an QoS-Eigenschaften lockern).

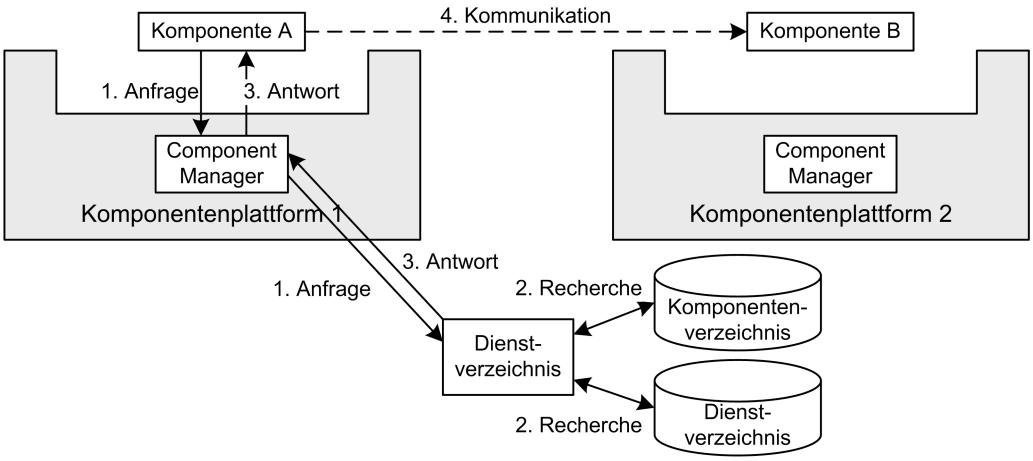

Abbildung 4.7.: Interaktion der Integrationsschicht-Elemente beim Ermitteln einer Komponentenreferenz

Der prinzipielle Ablauf der Ermittlung einer Komponentenreferenz im Dienstverzeichnis ist in Abbildung 4.8 dargestellt. Je nach Anforderungsart wird das Dienstverzeichnis und das Komponentenverzeichnis benutzt, um geeignete Komponenten zu finden. Ein mögliches Ergebnis des Ablaufdiagramms ist das Fehlen einer entsprechenden Referenz, sodass eine Fehlermeldung zurückgegeben werden muss. Alternativ kann die Komponente auch neu platziert werden, was im folgenden Unterabschnitt 4.5.2 beschrieben wird.

Registrierung von Verbindungen

Nachdem eine Komponente die Referenz auf eine andere Komponente ermittelt hat, kann sie mit dieser kommunizieren. Bei der ersten Interaktion wird die benutzte Komponente darüber informiert, von welcher Komponente sie benutzt wird. Genauer gesagt wird die implizit vorhandene Kopplungsinformation explizit gemacht, indem der verwendete angebotene Port darüber benachrichtigt wird, mit welchem benötigten Port er verbunden ist. Somit weiß eine Komponente stets, von welchen anderen Komponenten sie verwendet wird, was für die Aktualisierung der Referenzen beim Austausch von Komponenten notwendig ist. Diese Informationen werden vom lokalen *Component Manager* in einem Laufzeitprofil abgelegt, welches die aktiven Verbindungen beinhaltet. Somit kann beim Austausch von Komponenten aus diesem Laufzeitprofil abgelesen werden, welche Verbindungen bestehen und somit nach dem vollzogenen Austausch aktualisiert werden müssen.

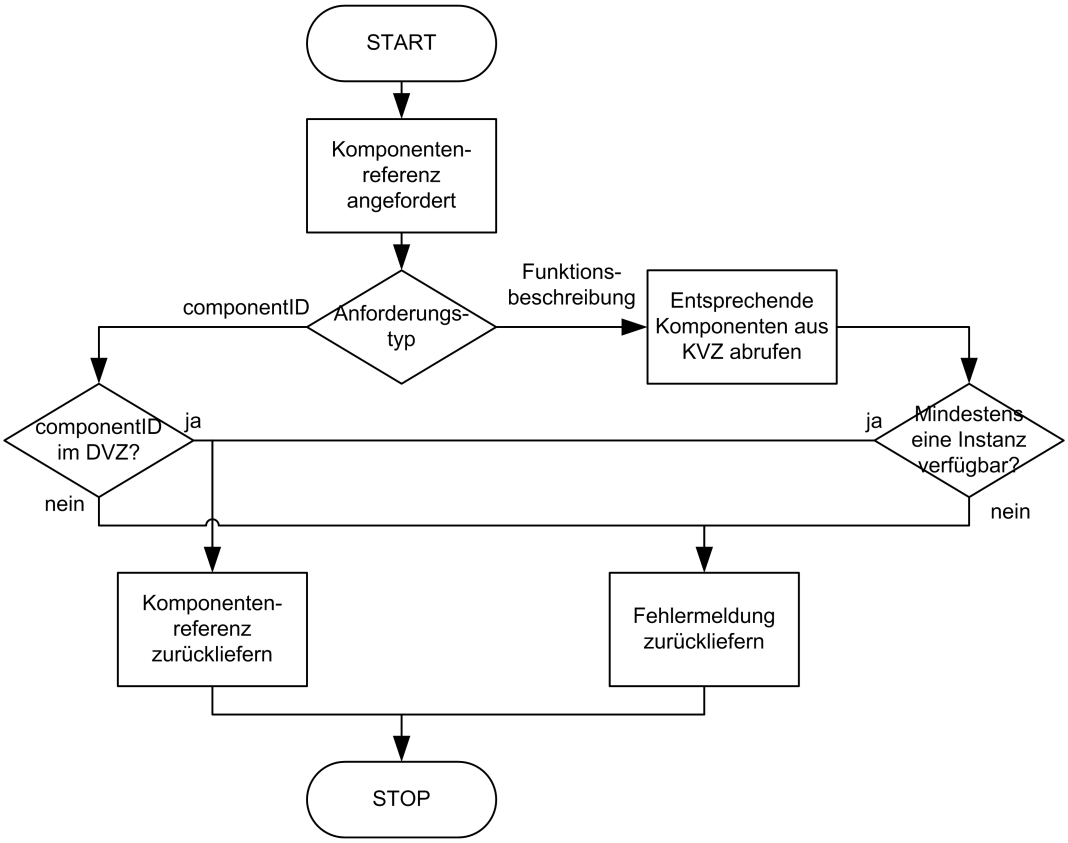

Abbildung 4.8.: Ablauf der Ermittlung einer Komponentenreferenz

4.5.2. Platzierung von Komponenten

Damit Komponenten gefunden und verwendet werden können, müssen sie in das System eingebracht werden. Das wird durch die Zuordnung einer Komponente zu einer Plattform erreicht. Hierfür spezifizieren Hostrechner und Komponenten jeweils ihre angebotenen und benötigten Ressourcen in ihren Profilen. Bei der Platzierung wird sichergestellt, dass alle Komponenten auf Geräten platziert werden, auf denen die Aufgaben erledigt werden können.

In diesem Abschnitt werden die verschiedenen Aspekte beleuchtet, die beim Platzieren von Komponenten beachtet werden müssen. So wird zunächst analysiert, welche Teile der Integrationsschicht eine Platzierung auslösen können und welche Auswirkung diese auf die Wahl der Geräte haben. Im Anschluss wird verglichen, welche Vorgehensvarianten bei der Platzierung von Anwendungen bestehen, die aus mehreren Komponenten aufgebaut sind. Abschließend werden zwei Strategien für die Komponentenverteilung vorgestellt.

Auslöser	gezielte Platzierung	wahlfreie Platzierung
Dienstverzeichnis	x	-
Anwendungskomponente	-	x
Verwaltungsanwendung	x	-

Tabelle 4.9.: Verfügbare Platzierungsstrategien in Abhängigkeit von den Platzierungsauslösern

Platzierungsstrategie

Die Platzierung kann danach unterschieden werden, welcher Teil der Integrationsschicht die Platzierung initiiert. Die Initiierung kann durch das Dienstverzeichnis, eine Anwendungskomponente und durch die Verwaltungsanwendung ausgelöst werden.

Die Platzierung durch das Dienstverzeichnis ist beispielsweise für das initiale Platzieren aller Komponenten einer Anwendung notwendig. Dabei wird die Platzierung vom Dienstverzeichnis aus kontrolliert, welches dann anhand des Anwendungsprofils, sowie der Komponenten- und der Geräteprofile eine akzeptable Verteilung planen kann. Somit handelt es sich hierbei um eine *gezielte Platzierung*.

Ist eine beliebige Anwendungskomponente der Auslöser einer Komponentenplatzierung, dann liegt dies in der Regel daran, dass diese Komponente Ergebnisse produziert hat, die von einer weiteren Komponente bearbeitet werden müssen. Die Anwendungskomponente stellt lediglich eine Anfrage nach einem bestimmten Komponententyp, ohne sich um dessen tatsächlichen Ausführungsort kümmern zu müssen. Außerdem für eine Netzwerkkommunikation notwendigen Mehraufwand besteht auch kein Unterschied zwischen einer lokalen und einer entfernten Komponente. Eine Anwendungskomponente muss somit nicht entscheiden, wohin die neue Komponente platziert werden soll. Aus diesen Gründen wird von einer Anwendungskomponente stets eine *wahlfreie Platzierung* ausgelöst.

Wenn die Platzierung durch die Verwaltungskomponente verursacht wird, dann handelt sich bei der platzierten Komponente in der Regel um Komponenten zur Darstellung der Nutzerschnittstelle. Dies liegt am hauptsächlichen Verwendungszweck der Verwaltungskomponente, dem Verwalten der Föderation durch Auswählen von Endgeräten und den dazugehörigen Modalitäten, die durch entsprechende Komponenten realisiert werden. Hierbei erfolgt eine *gezielte Platzierung*, da zuvor genau angegeben wurde, welche Komponente auf welchem Gerät verfügbar sein soll.

In Tabelle 4.9 werden die eben beschriebenen Platzierungsstrategien zusammengefasst.

Gezielte Platzierung Wie eben beschrieben, wird die Verwaltungsanwendung zur gezielten Platzierung verwendet. Dies ist unabhängig davon, ob das Ziel die lokale oder eine entfernte Plattform ist. Theoretisch entspricht die gezielte Platzierung auf der lokalen Plattform dem *pull*-Prinzip, weil die neue Komponente auf die lokale Plattform „gezogen“ wird. Diese *pull*-Platzierung wird aber ausschließlich für diesen Fall benötigt (Auslöser: Verwaltungsanwendung, Ziel: lokale Plattform), sodass dieser Fall auf die *push*-Platzierung zurückgeführt wird.
Bei der gezielten Platzierung handelt es sich meist um die Platzierung von Nutzerschnittstellenkomponenten, da diese nicht auf einem beliebigen Gerät ausgeführt werden können, sondern auf vom Nutzer ausgewählten Endgeräten erwartet werden.

Wahlfreie Platzierung Im Gegensatz zu Nutzerschnittstellenkomponenten sind die Komponenten zur Multimodalitätsunterstützung nicht direkt an bestimmte Geräte oder Endgeräte gebunden und können daher wahlfrei platziert werden. Hierbei müssen nur die technischen Randbedingungen der Geräte beachtet werden, sodass die neue Komponente auf einem Gerät platziert wird, auf dem sie von den anderen Komponenten erreicht werden kann. Diese Informationen müssen in den Auswahlalgorithmus einfließen, der für die Verteilungsplanung benutzt wird. Ein solcher Algorithmus wird hier jedoch nicht entwickelt, sondern vielmehr wird dessen Existenz angenommen.

Platzierungszeitpunkt

Im Komponentenprofil wird definiert, welche anderen Komponenten für die Durchführung der Funktionalität benötigt werden. Auf Basis dieser Information können verschiedene Mechanismen definiert werden, wann diese Komponenten tatsächlich auf die möglicherweise verschiedenen Plattformen platziert werden können. Die beiden Extrema sind die Platzierung zum Zeitpunkt der Anwendungsinstallation und die Platzierung bei Anfrage. Dazwischen sind Abstufungen denkbar, die z. B. eine Vorverteilung der Komponenten ohne deren Installation vorsehen oder zunächst die direkt benötigten Komponenten vorinstallieren. Diese Abstufungen werden jedoch nicht näher betrachtet.

Anwendungsinstallation Die Platzierung aller benötigten Komponenten bei Anwendungsinstallation sieht vor, dass zu dem Zeitpunkt der Komponentenanforderung alle von dieser Komponente benötigten Komponenten ebenfalls verfügbar gemacht werden. Zu diesem Zweck wird aus dem Komponentenprofil die Liste der benötigten Komponenten ermittelt. Das erfolgt auch bei der Installation der benötigten Komponenten, sodass indirekt benötigte Komponenten durch die Rekursion ebenfalls installiert werden. Vorteilig an dieser Variante ist, dass alle Komponenten von Beginn an installiert sind und sofort benutzt werden können.

Platzierungs-zeitpunkt	Vorteile	Nachteile
Anwendungs-installation	Benötigte Komponenten von Beginn an verfügbar	Längere Wartezeit am Anfang
Bei Anfrage	Minimale Wartezeit am Anfang	Wartezeiten für Platzierung während Benutzung

Tabelle 4.10.: Vergleich von Alternativen des Platzierungszeitpunktes

Nachteilig ist die Wartezeit, die der Nutzer in Kauf nehmen muss, ehe die u. U. große Menge an Komponenten installiert ist.

Bei Anfrage Die Platzierung bei Anfrage sieht vor, dass stets nur die angeforderte Komponente platziert wird, ohne Abhängigkeiten zu Nachfolgekomponenten zu berücksichtigen. Die Nachfolgekomponenten werden erst dann platziert, wenn sie auch tatsächlich benötigt werden. Dabei wird die ggf. vorhandene Information über Komponentenabhängigkeiten im Komponentenprofil ignoriert. Der Vorteil hierbei ist die kurze Verzögerungszeit bis zur Bereitschaft der angeforderten Komponente, wobei aber im weiteren Verlauf mit zusätzlichen Wartezeiten durch weitere Platzierungen zu rechnen ist.

Fazit In Tabelle 4.10 werden die Vor- und Nachteile der beiden Varianten tabellarisch gegenübergestellt. Die Vorteile der einen sind die Nachteile der anderen Variante. Letztlich bleibt dem Nutzer überlassen, welcher Platzierungszeitpunkt gewählt wird. Diese Entscheidung ist Teil des Nutzerprofils (siehe `DEPLOY_TYPE` in Unterabschnitt 4.4.1).

Verteilungsstrategie

Für die Verteilung der Komponenten in der aus einer Vielzahl an Komponentenplattformen bestehenden Infrastruktur können verschiedene Strategien angewandt werden. Hierbei gibt es einerseits die Strategie, die Komponenten möglichst stark verteilt zu platzieren. Auf der anderen Seite kann auch die Strategie angewandt werden, die Komponenten möglichst wenig zu verteilen. Dazwischen sind Abstufungen denkbar, sodass beispielsweise stark zusammenhängende Komponenten in Gruppen zusammengefasst, während unabhängige Komponenten möglichst stark verteilt werden. Im Folgenden werden die beiden Extremfälle erläutert. Auf die Abstufungen wird aus Platzgründen nicht näher eingegangen.

Möglichst stark verteilt Bei dieser Verteilungsstrategie ist das Ziel, die Komponenten möglichst stark zu verteilen, d. h. dass im Idealfall jede Komponente auf

Verteilungs-strategie	Vorteile	Nachteile
Stark verteilt	kein Single-Point-of-Failure, maximale Ressourcen für jede Komponente	Mehraufwand für Kommunikation
Wenig verteilt	kein Mehraufwand für Kommunikation	Komponenten teilen sich Ressourcen, Single-Point-of-Failure

Tabelle 4.11.: Vergleich von Alternativen der Verteilungsstrategie

einer separaten Kompontenplattform ausgeführt wird. Das hat den Vorteil, dass der Ausfall einer Komponentenplattform leichter kompensiert werden kann, da er nicht das ganze System beeinträchtigt. Außerdem kann jede Komponente über die gesamten Ressourcen ihrer Plattform exklusiv verfügen (im Einzelnutzerbetrieb). Nachteilig ist jedoch, dass dadurch ein Maximum an Netzwerkkommunikation verursacht wird.

Möglichst wenig verteilt Ziel dieser Verteilungsstrategie ist eine möglichst niedrige Verteilung der Komponenten auf verschiedenen Plattformen. Im Grenzfall befinden sich alle Komponenten auf einer Komponentenplattform, nämlich auf dem Endgerät, das der Nutzer zur Interaktion ausgewählt hat. Vorteil hierbei ist der minimale Kommunikationsmehraufwand, da quasi keine Netzwerkkommunikation stattfinden muss, falls sich die Zielanwendung ebenfalls auf dem Endgerät befindet. Dieser Vorteil hat allerdings den Nachteil zur Folge, dass sich die Komponenten die Ressourcen des Gerätes teilen müssen und sich somit u. U. gegenseitig beeinflussen. Zudem wird durch die Zentralität ein Single-Point-of-Failure geschaffen, d. h. bei Ausfall des Endgerätes wird die gesamte Anwendung unverfügbar.

Fazit In Tabelle 4.11 werden die beiden alternativen Verteilungsstrategien tabellarisch gegenübergestellt. Keine der beiden Varianten ist ideal, da sie beide Vor- und Nachteile haben. Um eine sinnvolle Entscheidung treffen zu können, ist Wissen über das aktuelle Einsatzszenario und die Netzwerkeigenschaften notwendig. Daher ist es dem Nutzer überlassen, für welche Strategie er sich entscheidet (siehe `DISTRIBUTION_TYPE` in Unterabschnitt 4.4.1).

Durchführung der Platzierung

Die Platzierung kann sowohl explizit als auch implizit von verschiedenen Teilen der Integrationsschicht ausgelöst werden. Bei einer expliziten Platzierung wird

von vornherein festgelegt, dass die angeforderte Komponente platziert werden soll, ungeachtet der Tatsache, ob diese schon vorhanden ist oder nicht (z. B. durch Nutzeranforderung). Die implizite Platzierung wird verwendet, wenn eine Referenz angefordert wird, dem Anforderer aber letztlich egal ist, ob es sich dabei um eine neue oder eine bereits existierende Komponente handelt. Jegliche Platzierung wird vom Dienstverzeichnis durchgeführt, d. h. Komponenten werden stets entsprechend des *push*-Prinzips auf die Komponentenplattformen transportiert. Für die Durchführung der Platzierung sind folgende Informationen notwendig:

- Die `uID` des Nutzers, der die Komponente verwenden will.
- Die `cID` zur Definition des Komponententyps.
- Ob eine Referenz auf eine beliebige Instanz ausreicht, oder ob eine neue Instanz erzeugt werden soll.
- Die optionale `dID` des Gerätes, worauf die Komponente platziert werden soll.

Hierbei wird angenommen, dass die `cID` der gewünschten Komponente bereits bestimmt wurde. Zu diesem Zweck kann das KVZ (Komponentenverzeichnis) nach Komponenten durchsucht werden, die vorgegebenen Kriterien entsprechen. Die Platzierung stellt hierbei eine Erweiterung des Kopplungsverfahrens aus Abbildung 4.8 dar, indem die Fehlerbehandlung durch die Platzierung erfolgt.
Der interne Ablauf bei Anforderung einer neuen Komponente im Dienstverzeichnis ist in Abbildung 4.9 dargestellt. Die Erweiterung des Kopplungsverfahrens ist grau hinterlegt. Die Fehlerbehandlung bei Fehlen einer entsprechenden Instanz sieht vor, dass eine entsprechende Komponente platziert, also eine neue Instanz erzeugt wird. Wenn per Nutzereinstellungen festgelegt wurde, dass alle benötigten Komponenten ebenfalls platziert werden sollen und dies noch nicht abgeschlossen ist, wird diese Prozedur solange wiederholt, bis die benötigten Komponenten platziert wurden. Anschließend wird die Referenz auf die ursprünglich angeforderte Komponente zurückgeliefert. Wenn kein passender Typ im KVZ vorhanden ist, wird eine Fehlermeldung zurückgeliefert.
Die eigentliche Platzierung ist in Abbildung 4.10 dargestellt. Hierfür wird die `dID` des Gerätes benötigt, worauf die Komponente platziert werden soll. Diese ist optional und nur für den Fall vorgesehen, dass eine gezielte Platzierung vorgenommen werden soll. Die automatische Ermittlung einer `dID` entspricht der Platzierung auf einem beliebigen Gerät. Hierbei wird entsprechend des Komponentenprofils ein Gerät ausgewählt, das den Anforderungen der Komponente entspricht. Wenn das Zielgerät bestimmt wurde, wird eine Verbindung zum Component Manager aufgebaut und die Komponente übertragen. Der Component Manager installiert und startet die Komponente, welche sich zunächst in den *pausiert*-Zustand begibt und auf Sitzungsinformationen zur Initialisierung wartet. Da es im Falle einer Neuinstallation keine Sitzungsinformationen gibt, wird

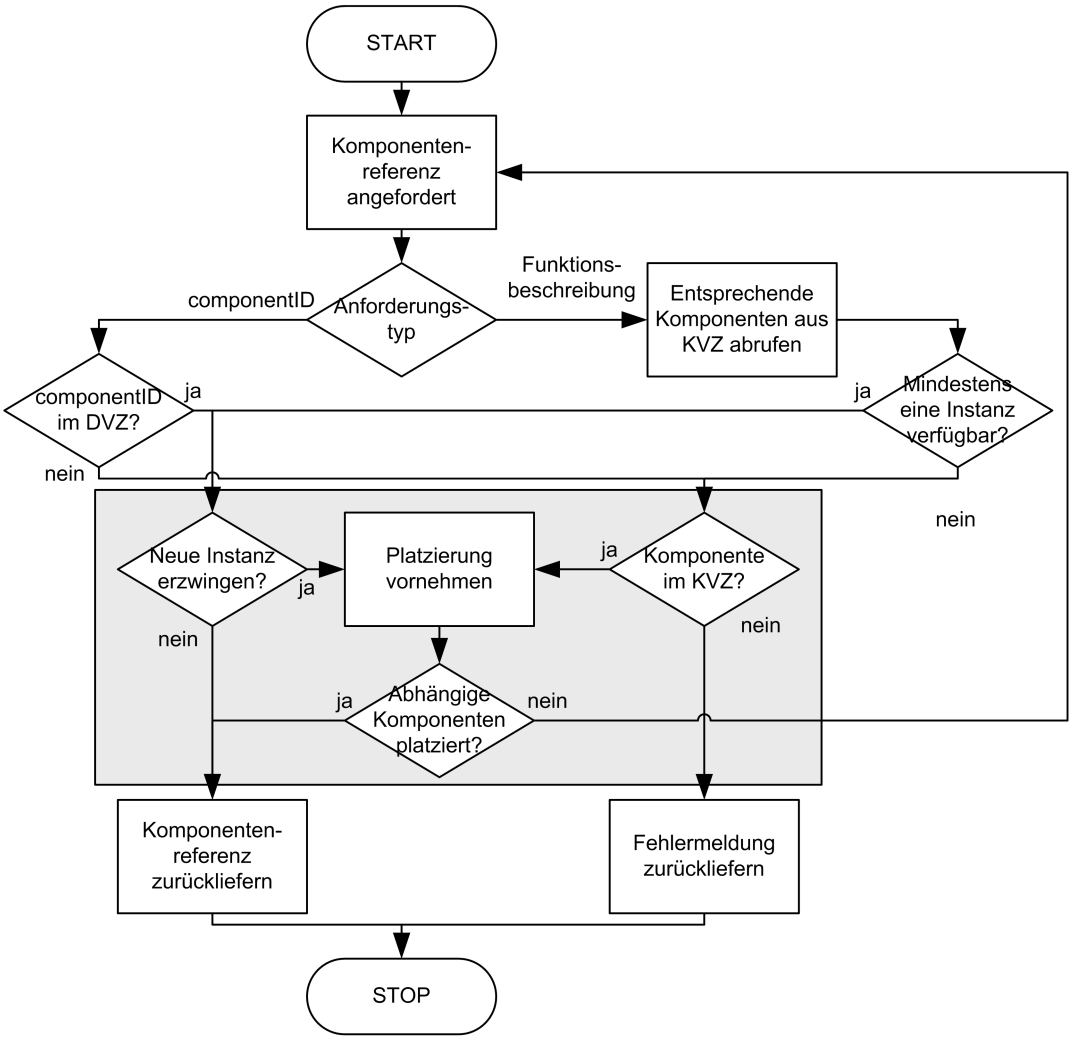

Abbildung 4.9.: Ablaufdiagramm bei Anforderung einer neuen Komponente (der Unterschied zu Abbildung 4.8 ist grau umrahmt)

die Komponente sofort vom Component Manager fortgesetzt, woraufhin sie sich im Dienstverzeichnis als verfügbar registriert. Danach wird die dabei mitübermittelte Referenz in Form einer URI an den Auslöser der Komponentenplattform zurückgeliefert.

Auswahl von Geräten für Platzierung

Bei der Geräteauswahl für die Platzierung soll ein Gerät identifiziert werden, was den Anforderungen der Komponente am Besten entspricht und somit am geeignetsten für deren Ausführung ist. Wenn mehrere abhängige Komponenten platziert werden sollen, spricht man auch von einer Verteilungsplanung. Für die Verteilungsplanung existieren in der Literatur verschiedene Algorithmen, die zu

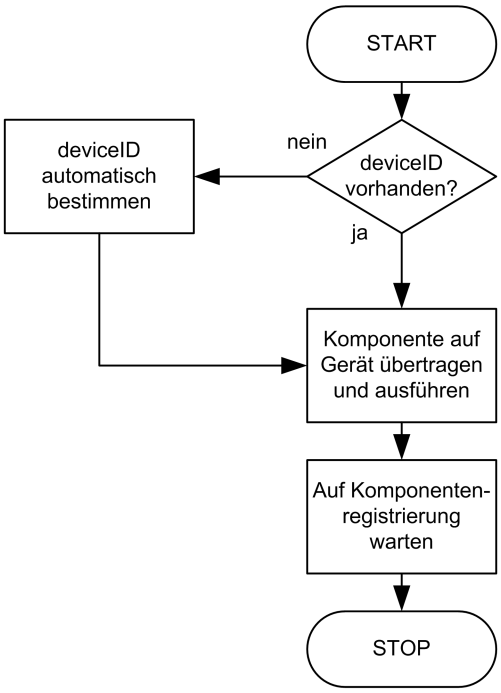

Abbildung 4.10.: Ablaufplan der Platzierung von Komponenten

unterschiedlichen Stadien in der Systementwicklung (z. B. Installationszeit, Laufzeit) angewendet werden oder nach unterschiedlichen Kriterien (z. B. Leistung, Kosten) optimieren [AK06, MMRM05, SSD+04, WW05].

Der Auslöser für die Platzierung von Komponenten ist immer eine Interaktion des Nutzers. Die Platzierung von Komponenten inklusive deren Registrierung dauert eine gewisse Zeit, die der Nutzer als Verzögerungszeit empfindet und die somit möglichst gering gehalten werden soll. Komplexe Verteilungsplanungsalgorithmen können eine Vielzahl von Kriterien berücksichtigen, um eine gute oder sogar optimale Platzierung zu berechnen. Allerdings wird dafür auch je nach Anzahl möglicher Varianten eine Zeit benötigt, die sich zu der vom Nutzer empfundenen Verzögerungszeit addiert.

Daher muss die Auswahl des Gerätes, auf das die jeweilige Komponente platziert werden soll, in einem zeitlich vertretbaren Rahmen erfolgen. Aus der Annahme, dass jedes Gerät mit jedem anderen kommunizieren kann, ergibt sich die Tatsache, dass keine Abhängigkeiten bezüglich der Erreichbarkeit innerhalb der Netzwerkinfrastruktur existieren. Daraus folgt, dass jede Komponente auf einem beliebigen Gerät platziert werden kann, solange es die Anforderungen bezüglich verfügbarer Hardware und Software erfüllt. Darüber hinaus soll der Platzierungsmechanismus keine optimale Lösung erzeugen, sondern lediglich Fehlplatzierungen verhindern, wie z. B. eine Sprachein- und -ausgabekomponente auf einem

OPTIMAL	Geschwindigkeit (Komplexität)	Qualität des Ergebnisses
ja	niedrig	optimal
nein	hoch	möglicherweise nicht optimal

Tabelle 4.12.: Gegenüberstellung der Vor- und Nachteile bei optimaler bzw. nicht-optimaler Geräteauswahl

Gerät ohne Mikrofon und/oder Lautsprecher oder eine Komponente zur Verarbeitung natürlicher Sprache auf einem PDA.
Aus diesem Grund werden als Resultat des Auswahlalgorithmus' nur „erfüllt" und „nicht erfüllt" als Bewertung zu Grunde gelegt. Prinzipiell wäre es auch möglich, eine Abstufung der Erfüllung von Anforderungen zu berücksichtigen (z. B. „sehr gut erfüllt", „gerade so erfüllt", etc.), was allerdings eine höhere Komplexität des Algorithmus zur Folge hat.
Der Auswahlalgorithmus ist in Abbildung 4.11 dargestellt. Die Ausgangssituation ist ein Komponentenprofil und beliebig viele Geräteprofile. Der Algorithmus durchläuft die Geräteprofile und prüft sie auf Erfüllung der obligatorischen Anforderungen. Wenn ein Gerät diese Anforderungen erfüllt und der Nutzer eine *schnelle* Auswahl möchte, dann wird dieses Gerät ausgewählt. Wünscht der Nutzer eine optimale Auswahl, so wird das Gerät in die Menge *OBLIG* hinzugefügt, die alle Geräte beinhaltet, die die obligatorischen Anforderungen erfüllen. Wenn alle Geräteprofile geprüft und die Menge *OBLIG* nicht leer ist, wird die Menge bezüglich der optionalen Anforderungen überprüft und das Gerät mit den meisten erfüllten optionalen Anforderungen ausgewählt. Dieses Gerät ist bezüglich der gestellten optionalen und obligatorischen Anforderungen die optimale Lösung für die zu platzierende Komponente. Wenn die *OBLIG*-Liste leer ist, kann kein Gerät alle obligatorischen Anforderungen erfüllen, weshalb eine Fehlermeldung ausgegeben wird.
In Tabelle 4.12 sind die Vor- und Nachteile der beiden Auswahlstrategien gegenübergestellt. Die schnellere Strategie führt zu Ergebnissen, die möglicherweise nicht optimal sind, da sofort nach Finden eines passenden Gerätes die Suche abgeschlossen wird. Bei entsprechendem zeitlichen Aufwand kann das für die Platzierung optimale Gerät ermittelt werden kann. Dafür ist jedoch eine vollständige Suche in allen Profilen erforderlich, wobei aus den möglichen Geräten auch noch das optimale Gerät herausgefunden werden muss.
Es ist denkbar, die Nutzerpräferenzen bezüglich der Geräteauswahl für die Platzierung weiter zu verfeinern, indem beispielsweise nach Art der Komponente unterschieden wird. Damit ließe sich z. B. definieren, dass für die Nutzerschnittstellenkomponenten optimale Geräte ausgewählt werden müssen, die multimodalen Unterstützungskomponenten hingegen einer schnellen Auswahl unterliegen. Da Nutzerschnittstellenkomponenten jedoch meist direkt platziert werden und selten dem Auswahlprozess unterliegen, wurde auf diese Verfeinerung verzichtet.

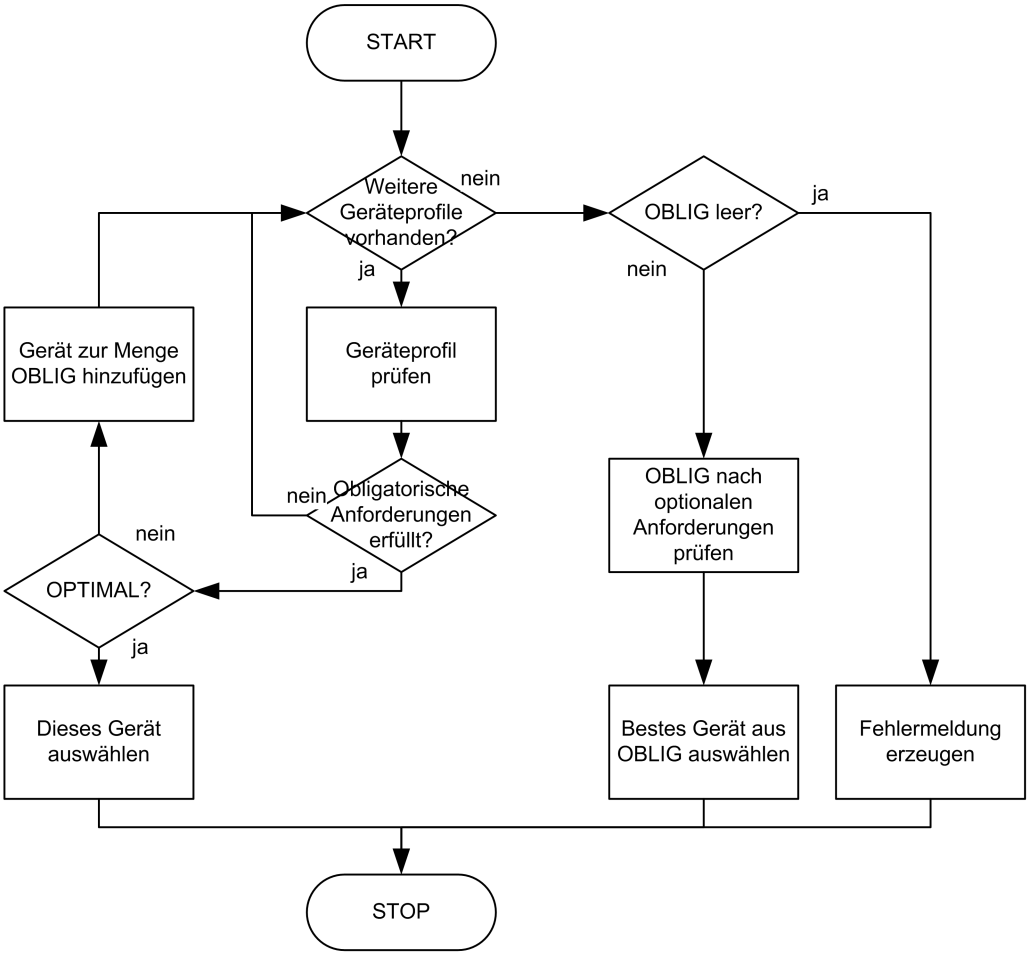

Abbildung 4.11.: Auswahl eines Gerätes für eine Komponente

Fazit

Der Platzierungsvorgang bzw. dessen Ergebnis kann durch verschiedene Kriterien an die Bedürfnisse des Nutzers und die Eigenschaften der Systemumgebung angepasst werden. Die Platzierung kann zum Zeitpunkt der Anwendungsinstallation oder bei Anforderung einer Komponente erfolgen, je nachdem, ob der Nutzer eine möglichst schnelle Reaktion vom System oder möglichst wenig Verzögerung beim Beginn der Interaktion möchte. Die unterschiedlichen Verteilungsstrategien ermöglichen zentrale bzw. dezentrale Verteilungen, während die Auswahl eines Gerätes in Bezug auf die Anforderungen der Komponente einerseits schnell, andererseits optimal, erfolgen kann. Die Platzierungsstrategie unterscheidet zwischen gezielter und wahlfreier Platzierung.

Durch die Platzierung von Komponenten kann sich der Nutzer seine gewünschten Modalitäten zusammenstellen und somit ein Modell der Nutzerschnittstelle

erschaffen. Es ist Aufgabe der Generation der einzelnen Nutzerschnittstellenbeschreibungen, dieses Modell effektiv zu nutzen, d. h. die Modalitäten sinnvoll in die Interaktion einzubeziehen. Die Komplexität dieser Aufgabe, die in anderen Arbeiten wegen der Adaption der Nutzerschnittstelle an die Endgeräte bzw. Modalitäten auch Adaptation Manager genannt wird, ist enorm und kein Ziel dieser Arbeit.

4.5.3. Migration zur Erhöhung der Verfügbarkeit

Die Integrationsschicht macht keine Annahmen über die Ausführungsumgebungen, wobei insbesondere die Zuverlässigkeit von Netzwerkverbindungen bzw. die Erreichbarkeit der Geräte entsprechend der heterogenen Hardware sehr unterschiedlich sein kann. Gerade mobile Nutzer laufen schnell Gefahr, sich aus dem Empfangsbereich bestimmter Geräte und somit Komponenten zu entfernen, was eine Nichtverfügbarkeit einzelner Komponenten verursachen kann. Bei Komponenten, die die Interaktion mit Anwendungen realisieren, kann die Nichtverfügbarkeit von einzelnen Komponenten u. U. zu einem Verlust der Kontrolle über die Anwendung führen. Aus diesem Grund müssen Mechanismen zur Behandlung von Ausfall bzw. temporärer Nichtverfügbarkeit einzelner Komponenten vorgesehen werden.
Ein solcher Mechanismus ist die Migration von Komponenten. Migration bezeichnet den Wechsel der Ausführungsumgebung einer Komponente inklusive des aktuellen Zustandes. Laut [FPV98] wird die Migration von Code und Ausführungszustand als starke Migration bezeichnet, während die schwache Migration nur den Code und ggf. Initialisierungsdaten berücksichtigt. Demnach handelt es sich bei dem im Folgenden beschriebenen Migrationsalgorithmus um schwache Migration, da kein direkter Ausführungszustand migriert wird sondern nur eine Menge an Initialisierungsinformationen.

Ziel und Voraussetzungen für Komponentenmigration

Das Ziel der Migration ist der Transfer der Binärdaten und Initialisierungsinformationen einer Komponente von einer Quellplattform auf eine Zielplattform. Daraus ergibt sich das Fortbestehen einer bestimmten Komponenteninstanz, sodass diese Komponente weiterhin genutzt werden kann. Gründe für eine Migration sind:

- Sicherung der Verfügbarkeit einer Komponente, wenn eine Nichtverfügbarkeit der Quellplattform und damit der dort ausgeführten Komponenten absehbar ist. Daraus ergibt sich, dass die Verfügbarkeit einer Kompenente gesichert wird. In diesem Zusammenhang bezieht sich Verfügbarkeit auf die Erreichbarkeit der Komponente. Dies kann der Fall sein, wenn das Netzwerk überlastet ist, der Hostrechner ausgeschaltet werden soll oder ein Endgerät die Föderation verlässt.

- Adaption der Komponentenverteilung, sodass zwischen serverunterstütztem und peer-to-peer-Betrieb der Föderation gewechselt werden kann. Dabei werden alle Komponenten der Interaktionsanwendung auf die Endgeräte verteilt, sodass die Föderation ohne Serverunterstützung weiter existieren und sich der mobile Nutzer daher aus dem Bereich der Server-Erreichbarkeit herausbewegen kann.

In Abbildung 4.12 ist die Komponentenmigration dargestellt. Komponente A und B kommunizieren miteinander über verschiedene Komponentenplattformen hinweg. Komponente B migriert von der Quellplattform zur Zielplattform, wobei der Bearbeitungszustand erhalten bleibt.

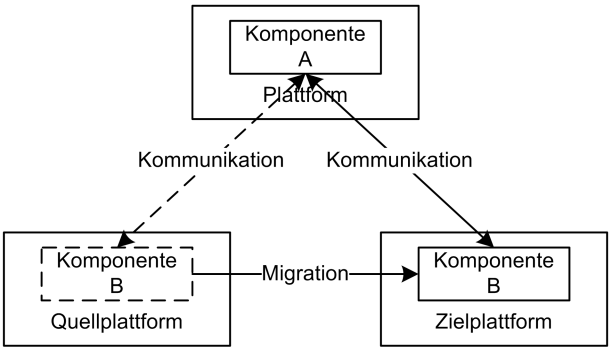

Abbildung 4.12.: Komponentenmigration

Die Komponentenmigration zur Verfügbarkeitssicherung basiert auf der Annahme, dass die Nichtverfügbarkeit der Quellplattform vorhergesehen werden kann. Dies kann z. B. durch kontrolliertes Herunterfahren des Hostrechners verursacht werden. Auch kann ein kontinuierliches Beobachten der Netzwerkverbindung die Migration auslösen, sodass die auf dem Host befindlichen Prozesse die Möglichkeit einer letzten Aktion haben.
Generell kann die Migration durch die Komponente selbst oder durch ein externes Ereignis ausgelöst werden. Das externe Ereignis kann z. B. der Nutzer oder eine Monitorkomponente verursachen. Für den zweiten, oben genannten Anwendungsfall, bei dem sich der Nutzer aus der Server-Reichweite entfernt, wird angenommen, dass die Migrationsanforderung durch den Nutzer ausgelöst wird.
Für alle im folgenden beschriebenen Migrationssituationen und -varianten gilt, dass ein Zeitraum entsteht, in dem die migrierende Komponente nicht erreichbar ist. In diesem Zeitraum müssen Vorkehrungen getroffen werden, damit die an die nicht verfügbare Komponente gesendeten Informationen nicht verloren gehen.
Es gibt verschiedene Situationen, in denen sich eine Komponente befinden kann, wenn ihre Migration ausgelöst bzw. angefordert wird. Im einfachsten Fall wird die Komponente gerade nicht verwendet und befindet sich somit im „Leerlauf“, weshalb eine Migration ohne große Komplikationen durchgeführt werden kann.

Komponente verwendet	nein	normaler Funktionsaufruf	Datenstrom-empfänger
Migrations-voraussetzung	-	aktiven Aufruf behandeln	Strom umleiten

Tabelle 4.13.: Übersicht der Kategorisierung von Migrationssituationen und deren Migrationsvoraussetzungen

Besondere Beachtung muss jedoch dem Fall geschenkt werden, wenn die Komponente gerade verwendet wird, die Komponente also gerade eine aktive Verbindung zu einer anderen Komponente hat. Hierbei kann weiter zwischen „normalen“ Funktionsaufrufen und Übertragung von Datenströmen unterschieden werden. Je nach Situation der Komponente sind verschiedene Voraussetzungen zu erfüllen, bevor die Migration durchgeführt werden kann. In Tabelle 4.13 sind die Voraussetzungen in Abhängigkeit der Verwendung einer Komponente dargestellt.
Im Folgenden werden die einzelnen Situationen genauer beschrieben und es werden Konzepte zur Migration vorgestellt.

Migration einer Komponente ohne Verwendung

Die Migration einer Komponente ohne Verwendung bedeutet, dass die Komponente im Augenblick der Migration zwar aktiv ist, jedoch von keiner anderen Komponente verwendet wird. Daraus ergeben sich keine zusätzlichen Anforderungen an die Migration. Dennoch kann zwischen zwei Varianten dieser Situationskategorie unterschieden werden, die sich sich in der Dauer der Nichtverfügbarkeit der jeweiligen Komponente unterscheiden.

Gemeinsame Migration von Code und Daten Hierbei werden die Binärdaten und der Zustand der Komponente gleichzeitig auf die neue Plattform übertragen. Das hat den Vorteil der gebündelten Übertragung der beiden Informationsteile, jedoch ist die Zeit der Unverfügbarkeit relativ groß. Der Ablauf der Migration geschieht in den folgenden Schritten:

SUSPEND Die Komponente muss einen Zustand einnehmen, in dem ihre Sitzungsinformationen konsistent extrahiert werden können, sodass die Komponente damit auf der Zielplattform initialisiert werden kann. Verarbeitungsprozesse wie Schleifen o.ä. müssen zunächst beendet werden.

EXTRACT Im angehaltenen Zustand wird die Sitzungsinformation aus der Komponente extrahiert.

MIGRATE Die Sitzungsinformation wird zusammen mit den Binärdaten der Komponente auf die Zielplattform transferiert.

INSERT Auf der Zielplattform wird die Komponente installiert und deren Zustand mit den ebenfalls empfangenen Sitzungsinformationen initialisiert.

RESUME Anschließend wird die Komponente und somit deren Verarbeitung fortgesetzt.

UPDATE Schließlich müssen alle Referenzen auf die Komponente aktualisiert werden. Das kann über das Dienstverzeichnis erfolgen, die Komponenten bzw. die Komponentenplattformen können sich zudem aber auch direkt verständigen, sodass die Aktualisierungsinformation schneller verteilt wird.

Probleme ergeben sich, wenn die Migrationskomponente in dem Zeitraum zwischen Pausieren und Aktualisierung der Referenz durch andere Komponenten verwendet werden soll. In diesem Zeitraum ist die Komponente nicht verfügbar, sodass Informationen verloren gehen können. Die Aufrufbehandlung während der Nichtverfügbarkeit der Komponente wird am Ende dieses Abschnitts behandelt. In Abbildung 4.13 sind die Schritte zur Migration zusammengefasst, wobei hervorgehoben wird, dass die Komponente in einem bestimmten Zeitraum nicht erreichbar ist.

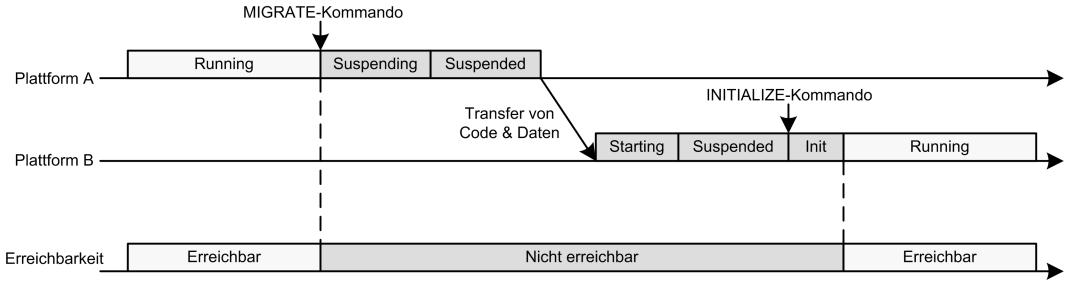

Abbildung 4.13.: Gemeinsame Migration von Code und Daten

Getrennte Migration von Code und Daten Um die tatsächliche Zeit der Nichtverfügbarkeit einer Komponente zu verringern, ist es möglich, den Code und die Sitzungsinformationen einer Komponente getrennt voneinander zu transportieren. Somit kann der Code direkt nach dem MIGRATE-Kommando zur Zielplattform transportiert und die neue Komponenteninstanz angelegt werden. Sobald die Migrationskomponente im *pausiert*-Zustand ist, kann die Sitzungsinformation ausgelesen werden und ebenfalls zur Zielplattform transportiert werden, wo dann die Initialisierung der neuen Instanz erfolgt. Der Ablauf der Migration ist der gleiche wie bei der gemeinsamen Migration von Code und Daten, nur wird sie in zwei Teile für Code und Daten aufgeteilt. Der Ablauf ist in Abbildung 4.14 dargestellt. Durch diese Variante wird die gesamte Zeit der Nichtverfügbarkeit um die Zeit verringert, die zum Transfer und Starten der Komponente auf der Zielplattform benötigt wird.

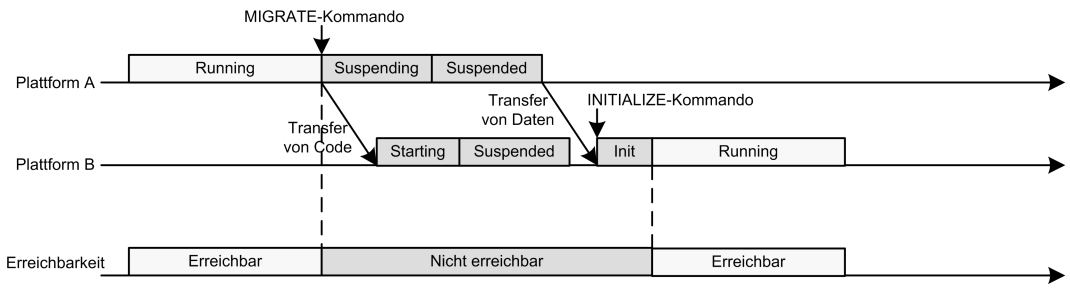

Abbildung 4.14.: Getrennte Migration von Code und Daten

Migration einer Komponente während aktivem Funktionsaufruf

Nicht in jedem Fall befindet sich die zu migrierende Komponente im Leerlauf und kann problemlos migriert werden. Da die Komponenten Dienste anbieten, ist ein aktiver Funktionsaufruf eines Dienstes während der Migrationsanforderung möglich, sodass eine einfache bzw. sofortige Migration ausgeschlossen ist. Beispielsweise kann die Aufforderung zur Migration eintreffen, während eine Nutzerschnittstellenkomponente ihre erkannten Eingaben an die Integration-Komponente übermittelt. Um diese Situtation zu behandeln, gibt es verschiedene Möglichkeiten:

1. Der aktuelle Aufruf wird sofort abgebrochen und eine Fehlermeldung zurückgeliefert.

2. Der aktuelle Aufruf wird abgearbeitet und das Ergebnis ordnungsgemäß zurückgeliefert.

3. Der aktuelle Aufruf wird an einer geeigneten Stelle unterbrochen, von der migrierten Komponente fertig abgearbeitet und von dieser das Ergebnis zurückgeliefert.

Bei Möglichkeit 1 geht eine gewisse Zeit verloren, die für die Bearbeitung des Aufrufs bereits investiert wurde. Zudem muss hierbei explizit sichergestellt werden, dass Transaktionseigenschaften wie Atomarität und Konsistenz gewahrt werden [HR83], da die teilweise Verarbeitung des Aufrufs Auswirkungen auf weitere Komponenten und/oder Datenbestände hat. Diese Auswirkungen müssen bei einem Abbruch des Aufrufs rückgängig gemacht werden, wobei aber nicht jede Komponente die Transaktionseigenschaften bereits unterstützt.
Möglichkeit 2 benötigt keine explizite Beachtung von Transaktionseigenschaften, da der Aufruf ohne gesonderte Behandlung abgeschlossen wird. Dadurch kann aber eine zusätzliche Verzögerung der Migration verursacht werden, deren exakte Länge nicht vorhersehbar ist und daher insbesondere auch länger dauern kann, als für die Migration überhaupt zur Verfügung steht. Aber auch wenn es keinen explizit begrenzten Zeitrahmen für die Migration gibt, so sollten zumindest die vom Nutzer initiierten Migrationen innerhalb einer gewissen Zeit abgeschlossen

Mögl.	Vorteile	Nachteile
1	Migration ohne Verzögerung	Transaktionalität muss gewahrt bleiben, Verlust der Teilergebnisse und der dafür benötigten Zeit
2	keine besondere Behandlung notwendig	Dauer u. U. länger als zur Verfügung steht
3	kein Verlust von Zeit oder Daten, zeitnaher Abschluss der Migration	Verbindungsabbruch und manuelle Reintegration der Antwort notwendig

Tabelle 4.14.: Vor- und Nachteile von Behandlungsmöglichkeiten von nichtverfügbaren Komponenten

sein, weil das System ohne explizites Feedback dem Nutzer sonst den Eindruck vermittelt, dass es nicht mehr reagiert.
Auch Möglichkeit 3 besitzt einige Nachteile. Da die Verarbeitung nicht abgeschlossen wird, wird die Verbindung zur aufrufenden Komponente abgebrochen, sodass die Antwort auf den Aufruf nicht direkt zurückgeliefert werden kann. Vorteil dieser Möglichkeit ist allerdings die schnelle Reaktionszeit, da die Migration in relativ kurzer Zeit und ohne den Verlust der bereits ausgeführten Verarbeitungen abläuft.
In Tabelle 4.14 sind die Vor- und Nachteile der drei genannten Möglichkeiten dargestellt.
Generell gilt, dass keine neuen Aufrufe zugelassen werden dürfen, während die Migration durchgeführt wird. In einem solchen Fall muss eine Fehlermeldung zurückgeliefert werden, die entweder bereits die neue Adresse der Komponente beinhaltet, oder aber der aufrufenden Komponente signalisiert, dass gerade eine Migration durchgeführt wird, was in einem erneuten Versuch des Aufrufs nach einer zufälligen Zeit resultiert.
Das Konzept zur Realisierung von Möglichkeit 3 wird *Late Response* genannt und ist in Abbildung 4.15 dargestellt. Die Ausgangssituation ist ein aktiver Funktionsaufruf einer Komponente (1) während die Migration ausgelöst wird (2). Damit die Verbindung nicht einfach nur abgebrochen wird, wird eine `Migration-in-progress`-Nachricht als Antwort des Aufrufs an die aufrufende Komponente zusammen mit einer beliebigen `requestID` geschickt (3). Die Plattform der aufrufenden Komponente reagiert auf die Nachricht mit Warten auf die asynchrone Antwort mit der entsprechenden `requestID` (4). Durch die Beendigung der Verbindungen ist die zu migrierende Komponente von anderen abgekoppelt und kann wie die Komponente im Leerlauf (siehe Unterabschnitt 4.5.3) migriert werden (5). Der einzige Unterschied ist, dass die Komponente hier einen

teilverarbeiteten Funktionsaufruf und die `requestID` beinhaltet. Wird die Komponente fortgesetzt, wird der ursprüngliche Aufruf weiter verarbeitet. Ist die Verarbeitung abgeschlossen, dann wird das Ergebnis zusammen mit der `requestID` an die aufrufende Komponente zurückgeschickt (6).

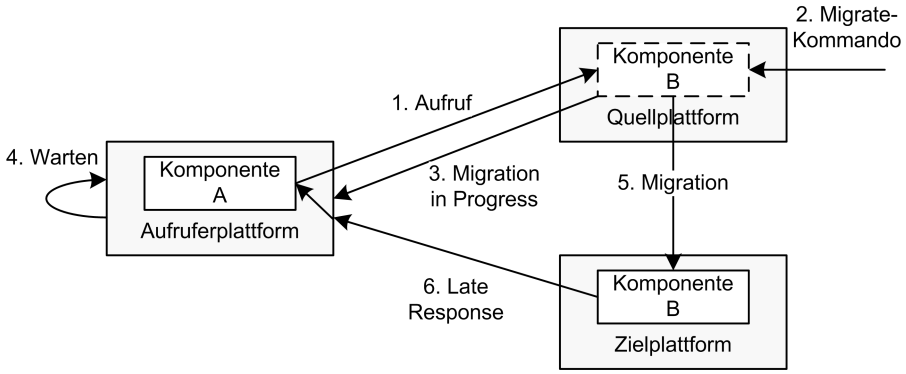

Abbildung 4.15.: Migration während eines aktiven Funktionsaufrufs: *Late Response*

Voraussetzung für den *Late Response*-Mechanismus ist die Unterstützung durch die Komponente selbst. Diese muss den Funktionsaufruf vorzeitig abbrechen und eine entsprechende ID zurücksenden. Ergebnis des Konzepts ist eine asynchrone Kommunikation zwischen den Komponenten.

Migration von Datenstromkomponenten

Besonders kritisch bei der Migration von Komponenten als Bestandteil einer Choreographie ist es, wenn die zu migrierende Komponente der Empfänger eines Datenstroms ist. Das ist z. B. der Fall, wenn die Komponente die Interpretation von Sprachsignalen durchführt, die vom Nutzer per Mikrofon eingegeben werden. Hierbei muss sichergestellt werden, dass der innere Zustand der Spracherkennungskomponente bei der Migration berücksichtigt wird, da er für die weitere Verarbeitung benötigt wird.
Aus Sicht des Empfängers ist ein Datenstrom durch folgende Kriterien gekennzeichnet [BBD+02]:

- Der Datenstrom besteht aus Datenelementen, die über eine Online-Verbindung ankommen.
- Die Datenelemente kommen in einer Reihenfolge an, die weder innerhalb eines Stroms noch über mehrere Ströme hinweg durch den Empfänger kontrolliert werden kann.
- Der Datenstrom ist prinzipiell unbeschränkt groß.

- Sobald ein Datenelement verarbeitet wurde, wird es entweder verworfen oder archiviert.

Daraus ergibt sich ein wesentliches Kriterium für das im Folgenden beschriebene Konzept: ein Datenstrom muss aus einer Reihe von geordneten, diskreten Datenelementen bestehen.
Das Problem bei der Migration von datenstromempfangenden Komponenten ist vergleichbar mit dem Handover in Mobilfunknetzen auf Basis von GSM (Global System for Mobile Communications) oder UMTS (Universal Mobile Telecommunications System). Ein Handover ist z. B. dann nötig, wenn der Nutzer sich während einer aktiven Sprachverbindung aus dem Einzugsgebiet der aktuellen Funkzelle herausbewegt, d. h. der Abstand zur momentanen BTS (Base Transceiver Station) wird zu hoch. Damit die Verbindung weiterhin aufrecht erhalten werden kann, muss die Verbindung an die nächste BTS mit besserer Signalstärke übergeben werden.

Vergleich mit GSM Der Hauptunterschied zwischen Handover in GSM (Global System for Mobile Communications) (siehe Abbildung 4.16) oder ähnlichen Netztechnologien und der Migration von datenstromempfangenden Komponenten ist die Änderung des direkten Kommunikationspartners. Beim Handover zwischen zwei GSM-Funkzellen ändert sich nur der unmittelbare Kommunikationspartner (also die BTS), aber nicht der mittelbare Kommunikationspartner bzw. der eigentliche Empfänger des Datenstroms (der BSC (Base Station Controller) oder höhere Elemente der Architektur). Der neue unmittelbare Kommunikationspartner leitet die als Datenstrom eintreffenden Informationen an den gleich gebliebenen Empfänger weiter. Im Gegensatz dazu wird bei der Migration sowohl der unmittelbare Kommunikationspartner als auch der mittelbare Empfänger geändert, welche in einer Komponente vereinigt sind. Die alten und neuen unmittelbaren Kommunikationspartner beim Handover arbeiten kontextfrei, d. h. sie besitzen keinen besonderen Zustand, der für den Empfang und die Verarbeitung des Datenstroms wesentlich ist. Im Gegensatz dazu beinhaltet die unmittelbar adressierte, migrierende Komponente teilverarbeitete Informationen, auf deren Basis die neue Komponente den ankommenden Datenstrom verarbeiten muss. Die Komponenten arbeiten somit kontextsensitiv. Dabei entsteht das Problem, dass ein Zeitfenster benötigt wird, in dem die eigentliche Migration durchgeführt wird und somit Inkonsistenzen entstehen können.

Vergleich mit herkömmlichen Komponenten Komponenten werden als *herkömmlich* bezeichnet, wenn die Kommunikation mit diesen Komponenten dem herkömmlichen Request-Response-Paradigma entspricht. Daher sind die bisher beschriebenen Migrationskonzepte für herkömmliche Komponenten geeignet. Der Hauptunterschied zwischen der Migration von datenstromemfangenden und herkömmlichen Komponenten besteht darin, dass Datenstromkomponenten von ei-

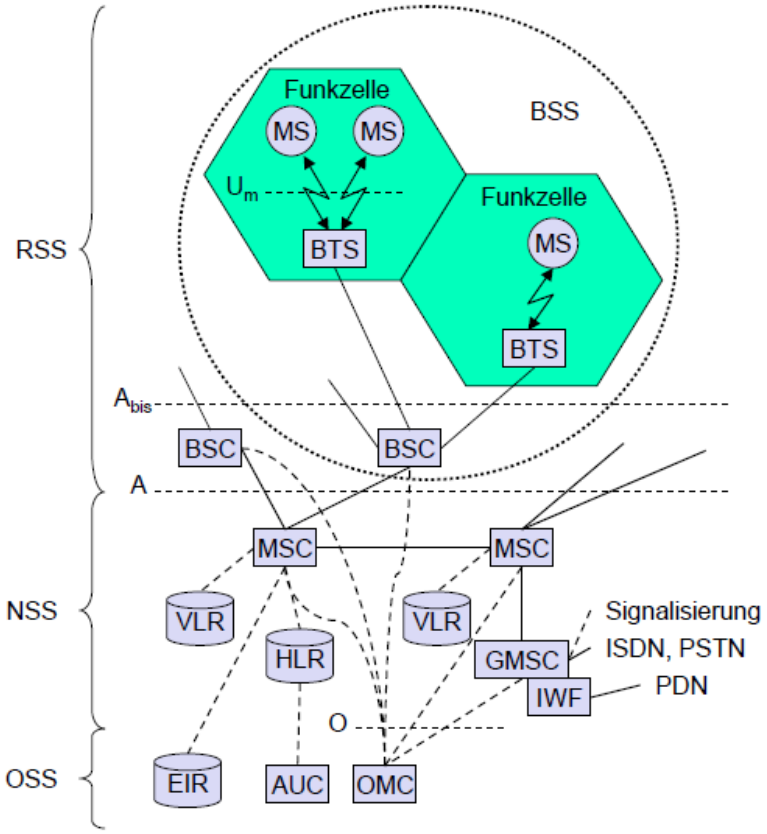

Abbildung 4.16.: GSM-Architektur (aus [Sch03a])

nem kontinuierlichen Datenstrom gefüllt werden, den sie zu bearbeiten haben. Eine direkte Antwort von der Datenstromkomponente wird in der Regel nicht erwartet, da sie die Ergebnisse der Stromverarbeitung an eine weitere Komponente überträgt, welche u. U. mit der Datenstromquelle identisch sein kann.

Catch-up Migration Das Konzept der *Catch-up Migration* besteht darin, dass der Datenstrom in einzelne Segmente (Datenelemente) gegliedert werden kann, die die mögliche Granularität von Kontrollpunkten (Checkpoints) vorgeben. Das Anlegen von Kontrollpunkten verschiedener Prozesse in einem verteilten System kann in drei verschiedene Arten unterteilt werden [EAWJ02]: koordiniert, unkoordiniert und Nachrichten-gesteuert. Es kommt vorwiegend zur Wiederherstellung eines bestimmten Zustandes nach einem Fehlerfall zum Einsatz, sodass das System die Arbeit nach Behebung des Fehlers an diesem Punkt fortsetzen kann, ohne die gesamte, bereits erledigte Arbeit zu verlieren. Die Verwendung von Checkpoints ist ein typischer Ansatz zur Wiederherstellung von Systemzuständen bei Fehlerfällen [KT87]. Bei der *catch-up Migration* werden die Checkpoints primär

zur Synchronisierung von migrierenden Komponenten gedacht, wie im Weiteren beschrieben wird.
Eine Migration ist an den jeweiligen Kontrollpunkten möglich. Ausgangssituation ist eine Zusammenarbeit von Komponente A und B, wobei A einen Datenstrom von B empfängt, die Ergebnisse an C sendet (1) und zu geeigneten Zeitpunkten einen Kontrollpunkt anlegt und A darüber informiert (siehe Abbildung 4.17). In dieser Situation erreicht die Migrationsaufforderung die Komponente B (2). Komponente B ermittelt den letzten Kontrollpunkt und legt diesen als inneren Zustand für die Migration fest, sodass die Migration durchgeführt werden kann (3). Anschließend arbeitet Komponente B zunächst weiter und teilt den Kontrollpunkt und das Migrationsziel der Komponente A mit (4). Anschließend wird der Datenstrom ab dem definierten Kontrollpunkt an die Komponente B' gesendet (5). Die Komponente B' verarbeitet die Daten weiter und sendet die Ergebnisse an C.
Da die zwei an der Migration beteiligten Komponenten A und B über Kontrollpunkte an den gleichen Stellen verfügen müssen, erzeugt die empfangende Komponente B die Kontrollpunkte an geeigneter Stelle und teilt dann der sendenden Komponente A mit, bei welchem Stand des Eingabestroms der Kontrollpunkt angelegt wurde. Damit handelt sich um ein koordiniertes Checkpointing. Das Ergebnis ist ein definierter Zustand in der Komponente B, der auf dem Datenstrom der Komponente A bis zum Zeitpunkt des Kontrollpunktes basiert und an diesem fortgesetzt werden kann. Komponente A kann nach Festlegen des Kontrollpunktes alle zwischengespeicherten Informationen verwerfen, die vor dem Anlegen des Kontrollpunktes an Komponente B gesendet wurden, da diese bereits zum Erzeugen des Zustandes in Komponente B' beigetragen haben.
Da zwischen dem Erreichen des Kontrollpunktes bei der Verarbeitung in der Komponente B und dem Fortsetzen an dieser Stelle in der Komponente B' eine gewisse Zeit vergangen ist, muss diese aufgeholt werden, damit die B durch die B' abgelöst werden kann. Daher werden die Inhalte des Datenstroms so schnell wie möglich übertragen, damit die neue Komponente den aktuellen Zustand aufholen kann. Komponente C empfängt in dieser Zeit Ergebnisse von beiden Komponenten, kann aber zwischen dem aktuellen und dem verspäteten Ergebnis unterscheiden. Wenn die Ergebnisse der beiden Komponenten gleichzeitig oder mit nur geringem zeitlichen Unterschied ankommen, sendet Komponente C eine Angleichungsbenachrichtigung an die beiden vorigen Komponenten. Diese wissen, welche das Original und welche die durch Migration entstandene Kopie ist. Auf dieser Basis kann die Originalkomponente die Arbeit einstellen und die migrierte Komponente setzt ihre Arbeit wie bisher fort.
Damit dieses Konzept funktionieren kann, müssen einige Annahmen getroffen werden. Die wichtigste Annahme bezieht sich auf die Charakteristik des Datenstroms. Damit der Datenstrom zeitlich komprimiert werden kann, dürfen entweder keine zeitlichen Informationen zur Weiterverarbeitung verwendet werden oder diese Informationen müssen sich entsprechend komprimieren lassen. Weiter-

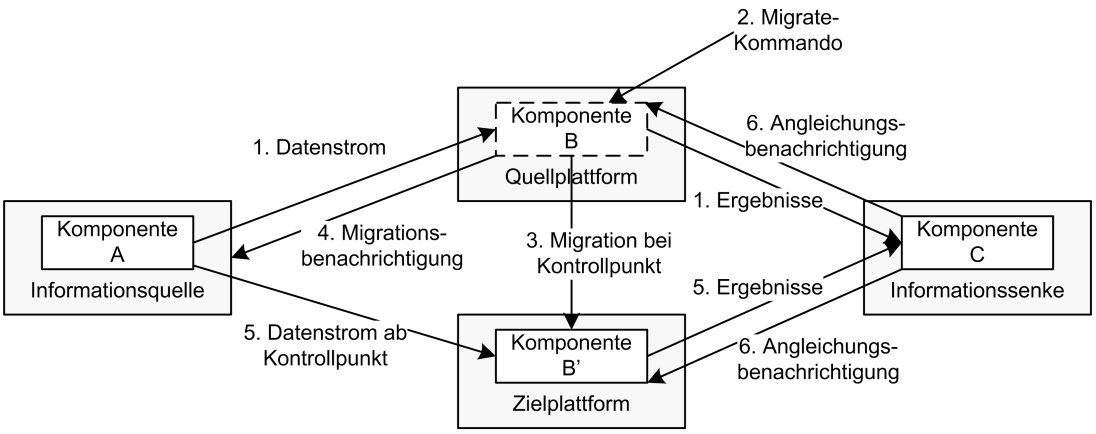

Abbildung 4.17.: Migration eines Datenstromempfängers: *Catch-up Migration*

hin ist es erforderlich, dass weder das Netzwerk noch die CPU der beteiligten neuen Komponenten eine geringere Übertragungs- bzw. Verarbeitungsrate als die alten Komponenten verfügbar haben, da sonst der verspätete Zustand nicht aufgeholt werden kann. Außerdem wird dem *Catch-up*-Prinzip zugrunde gelegt, dass die migrierende Komponente einen Nachfolger hat, der die Angleichung der beiden temporär parallel existierenden Komponenten feststellen und diese darüber benachrichtigen kann. Alternativ können sich die Komponenten auch untereinander über ihren aktuellen Verarbeitungszeitpunkt abstimmen, was jedoch eine erhöhte Netzwerklast zur Folge hat. Zudem ist für eine vollständige und korrekte Migration notwendig, dass die dafür zur Verfügung stehende Zeit größer ist, als für die Angleichung benötigt wird. Ist dies nicht gegeben, kann die Migration dennoch erfolgen, allerdings wird dies durch Verzögerungen des Endergebnisses beeinträchtigt.

Aufrufbehandlung während der Nichtverfügbarkeit von Komponenten

Das kann einerseits durch Zwischenspeichern und Weiterleiten der Anfragen geschehen, andererseits kann auch die Ursprungsschnittstelle bis zum Abschluss der Migration bestehen bleiben und eventuelle Anfragen mit einer Fehlermeldung quittieren. Die Fehlermeldung kann einerseits die neue Adresse beinhalten und somit von der anfragenden Komponente als Umleitung interpretiert werden, andererseits bei unbekannter neuer Adresse auch einfach eine Wiederholung der Anfrage nach einer gewissen Zeit beabsichtigen.

Mit impliziter Umleitung Damit die zu migrierende Komponente während der Migration für Anfragen verfügbar bleibt und somit zu jedem Zeitpunkt erreichbar ist, muss die Migration so erfolgen, dass die Übergabe zwischen alter und neuer Instanz der Komponente nahtlos funktioniert. Eine gewisse Zeit der Nicht-

verfügbarkeit lässt sich jedoch nicht vermeiden, weil die Komponente in jedem Fall in den *pausiert*-Zustand wechseln muss, um die konsistente Extraktion der Sitzungsinformationen zu ermöglichen. In diesem Zeitraum ist die Komponente für Aufrufe von anderen Komponenten nicht erreichbar.
Um zu verhindern, dass Aufrufe von anderen Komponenten in der unverfügbaren Zeit fehlschlagen, werden diese in der Quellplattform zwischengespeichert. Wenn die Komponente dann wieder verfügbar ist, können die Aufrufe abgearbeitet werden, woraus lediglich eine gewisse Zeitverzögerung der Verarbeitung resultiert. Im Falle der Migration ist jedoch nicht zu erwarten, dass die Komponente wieder in den *aktiv*-Zustand wechselt, weshalb die zwischengespeicherten Aufrufe nicht lokal verarbeitet werden können. Daher ist für die Migration notwendig, dass die Aufrufe an die neue Komponenteninstanz weitergeleitet werden, sobald diese aktiv ist. In diesem Fall agiert die Quellplattform als Proxy, bis die Aktualisierung der Referenz auf die Komponente im System abgeschlossen ist. In Abbildung 4.18 sind die drei Phasen der Migration mit Handover von Aufrufen dargestellt. Abbildung 4.18a zeigt die normale Kommunikation, wobei der Proxy die Anfragen einfach an die lokale Komponente weiterleitet. In Abbildung 4.18b wird die Komponente auf die Zielplattform migriert. Währenddessen speichert der Proxy der Quellplattform die Aufrufe und leitet sie an den Proxy der Zielplattform weiter, sobald die neue Komponenteninstanz existiert. Nach Abschluss der Referenzaktualisierung erfolgt die Kommunikation nur noch mit der neuen Instanz auf der Zielplattform (Abbildung 4.18c).
Beim Einsatz gesicherter Kommunikation (z. B. über HTTPS) werden dann zwei voneinander unabhängig gesicherte Verbindungen aufgebaut, sodass es sich nicht mehr um eine Ende-zu-Ende-Verschlüsselung handelt. Da aber die Plattform des Proxys die ursprünglich gewünschte Komponente beinhaltete und dieser Plattform vertraut wurde, stellt dies keine Sicherheitslücke (z. B. im Sinne der Man-in-the-Middle-Attack) dar.

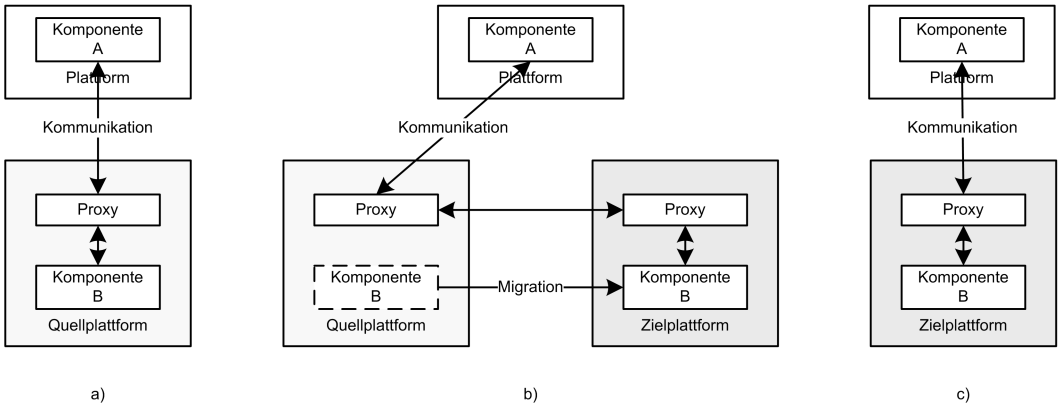

Abbildung 4.18.: Handover von Aufrufen bei Migration der Komponente

Mit expliziter Umleitung Das zuvor beschriebene Vorgehen der impliziten Umleitung sieht eine Zwischenspeicherung durch die Quellplattform vor, was nicht praktikabel ist, wenn sich die Plattform vor dem Herunterfahren befindet und deswegen die Komponente migriert werden soll. Daher ist es günstiger, wenn die für die Migrationskomponente ankommenden Aufrufe auf der Quellplattform nicht gespeichert, sondern zurückgewiesen werden. Da dieser Fall nur eintreten kann, nachdem die Migration bereits angestoßen wurde, ist bereits eine Zielplattform definiert. Somit wird der Aufruf nicht einfach zurückgewiesen, sondern mit einem Verweis auf die Zielplattform versehen, sodass die aufrufende Komponente darüber informiert wird, dass sich die Referenz zur gewünschten Komponente geändert hat. Mit dieser Information kann die aufrufende Komponente ihren Aufruf nochmal an die Zielplattform absenden. Wenn die neue Komponenteninstanz schon *aktiv* ist, kann der Aufruf bearbeitet werden, ansonsten ist sie entweder *pausiert* und wartet auf ihre Initialisierung oder noch *inaktiv* bzw. nicht existent. In letzterem Fall schlägt der Aufruf fehl.

Weitere Mechanismen zur Verfügbarkeitserhöhung

Im Allgemeinen ist es möglich, eine ausgefallene Komponente durch eine neue Instanz zu ersetzen. Hierbei ergibt sich allerdings das Problem, dass die Sitzungsinformationen der ausgefallenen Komponente verloren sind, sodass sich die Verfügbarkeitserhöhung auf die Verfügbarkeit eben dieser Sitzungsinformation bezieht.

Neuinstanziierung und Wiederholung der Übertragung Eine ausgefallene Komponente kann schnell durch eine neue Instanz des gleichen Typs ersetzt werden. Wenn die vorherigen Komponenten die an die ausgefallene Komponente gesendeten Informationen zwischenspeichern, dann ist durch eine Wiederholung der Übertragung an die neue Komponenteninstanz der interne Zustand der ausgefallenen Komponente wiederherstellbar. Hierbei ist es wichtig, dass die beiden Komponenten durch Synchronisierungspunkte festlegen, welche Information in der sendenden Komponente vorgehalten werden muss, sodass der Umfang der zwischengespeicherten Information auf einem verträglichen Niveau gehalten werden kann.
Ein Problem bei diesem Ansatz ist der Fall, wenn zwischen sendender und empfangender (ausfallender) Komponente eine n:1-Relation besteht, was bedeutet, dass sich die Sitzungsinformation der ausfallenden Komponente aus empfangenen Informationen mehrerer Komponenten zusammensetzt. In diesem Fall ist eine explizite Reintegration notwendig, wobei erst alle Teilinformationen erneut übertragen werden müssen, auf deren Basis die Sitzungsinformation rekonstruiert werden kann. Der Vorteil dieser Variante ist, dass der Netzwerkverkehr nicht permanent erhöht wird, sondern nur im Ausnahmefall eine zusätzliche Übertragung erforderlich ist. Nachteil bei dieser Variante ist die zeitlich aufwändige Rekon-

struktion der Sitzungsinformation für den Fall, dass mehrere Quellen die ausgefallene Komponente mit Eingabeinformationen versorgt haben.

Redundante Komponenten Der Einsatz redundanter Komponenten sieht vor, dass zwei (oder mehr) Instanzen einer Komponente in der Integrationsschicht existieren, die dieselbe Aufgabe haben [ZLWS06]. Alle zu verarbeitenden Informationen werden an beide Komponenten übertragen, sodass beide über die gleichen Sitzungsinformationen verfügen. Dabei ist festgelegt, welche Komponente die primäre ist und die Ergebnisse der Verarbeitung zur weiteren Verarbeitung an nachfolgende Komponenten übermitteln darf. Die sekundären Komponenten empfangen und verarbeiten die Informationen lediglich, ohne diese jedoch weiterzusenden. Wenn die primäre Komponente ausfällt, wird eine bis dahin redundante Komponente zur neuen primären und sendet die Ergebnisse der Verarbeitung an nachfolgende Komponenten. Die Aktualisierung des Dienstverzeichnisses ist nur eine Formsache, da sich für die vorherigen Komponenten nichts ändert, außer dass Übertragungen an die ehemals primäre Komponente fehlschlagen. Dieses Prinzip wird auch als *Hot Standby* bezeichnet. Vorteilig wirkt sich die schnelle und zuverlässige Ausfallbehandlung aus, da die redundante Komponente sofort einspringen kann. Der Nachteil ist jedoch die permanente Erhöhung des Netzwerkverkehrs, da alle Eingabeinformationen für die Primärkomponente auch an die Sekundärkomponente übertragen werden müssen.
Auf eine detailliertere Spezifikation der beiden geschilderten Verfahren wird verzichtet, weil es sich hierbei um bereits existierende Konzepte handelt und durch bestehende Lösungen aus dem Bereich der Zuverlässigkeit und Fehlertoleranz abgedeckt werden können [Wil00, Pra96]. Beispielsweise gibt es auch einen Ansatz, wonach verschiedene Implementierungen desselben Systems existieren, die theoretisch das gleiche Ergebnis produzieren und somit im Fehlerfall diesen zumindest aufdecken, u. U. aber auch auflösen können [Avi85].

Fazit

In diesem Abschnitt wurden verschiedene Mechanismen zur Migration vorgestellt, die für Komponenten geeignet sind, welche einerseits ohne Verwendung sind, andererseits aber zur Verwendung für herkömmliche Funktionsaufrufe und auch als Datenstromempfänger zum Zeitpunkt der Migration eingesetzt werden. Die erläuterten Konzepte stellen unterschiedliche Anforderungen an die Systemumgebung, welche in den jeweiligen Abschnitten beschrieben wurden. Für alle Migrationskonzepte wird angenommen, dass zwischen den migrierten Komponenten und den jeweiligen, im Datenfluss nachfolgenden Komponenten keine Abhängigkeiten bestehen, die gesondert betrachtet werden müssen. Die Annahme legt zugrunde, dass die migrierten Komponenten diese Abhängigkeiten als Teil der Zustandsicherung behandeln.

Insbesondere die Migration von Datenstromkomponenten und deren Anforderungen verdeutlichen, dass es sich hierbei um ein Konzept handelt, was durch die Konstellation von Komponenten-basierter und verteilter Nutzerschnittstelle verursacht wird. Durch den Komponentenansatz wird die Verteilung der Geschäftslogikteile und deren Anpassung an geänderte Systembedingungen motiviert. Die Verwendung für Nutzerschnittstellen erfordert eine schnelle und verlustlose Migration. Wie oben erläutert, ist der Handover beispielsweise in GSM-Netzen durch kontextfreie unmittelbare Kommunikationspartner gekennzeichnet. Migration von mobilen Softwareagenten ist nicht an Nutzerinteraktion gekoppelt und kann somit zu (fast) beliebigen Zeitpunkten durchgeführt werden.
Bei der eigentlichen Durchführung der Migration kann entweder eine gemeinsame Übertragung von Code und Daten erfolgen, oder dies zwecks kürzerer Ausfallzeit getrennt erfolgen. Die Aufrufe während der Migration können explizit und implizit umgeleitet werden.

4.5.4. Austausch von Komponenten

Insbesondere bei Kompositionen, die auf einer Vielzahl von Komponenten aufbauen, ist es möglich, dass unterschiedlich implementierte Komponenten existieren, die einerseits eine bestimmte Funktionalität mit unterschiedlicher Qualität erfüllen, andererseits eine unterschiedliche Anzahl an Funktionen beinhalten. Bezogen auf das MMI-F ist es beispielsweise möglich, dass eine Komponente sowohl Integration, Interaktion und Generation realisiert, während für die drei Rollen auch separate Komponenten existieren. Auch qualitative Unterschiede beim Realisieren bestimmter Funktionalität durch unterschiedliche Komponenten sind möglich, z. B. kann eine Komponente nur mit einfachem HTML Ausgaben erzeugen, während eine andere XHTML, PNG (Portable Network Graphics) und JPEG (Joint Photographic Experts Group) darstellen kann.
Da die Integrationsschicht eine hohe Flexibilität aufweisen soll, müssen Komponenten zur Laufzeit austauschbar sein. Die Flexibilität durch Komponentenaustausch kann sich in einem Anstieg der Performanz, Nutzerfreundlichkeit oder de Funktionsumfangs auswirken, da mit austauschbaren Komponenten auf die aktuelle Systemsituation reagiert werden kann. Beim Austausch von Komponenten sind folgende Fälle zu unterscheiden:

Einfacher Austausch Als einfacher Austausch wird der 1:1 Austausch bezeichnet, also eine Komponente wird durch eine andere ausgetauscht. Das setzt voraus, dass sich die beiden Komponenten nur in der Art der Funktionsrealisierung unterscheiden, nicht aber in der Funktion selbst.

Komplexer Austausch Der komplexe Austausch bezeichnet das Austauschen von beliebig vielen Komponenten durch beliebig viele Komponenten (n:m). Dabei kann entweder n oder m den Wert 1 haben, wobei eine zusammen-

gesetzte Komponente durch Teilkomponenten bzw. Teilkomponenten durch eine zusammengesetzte Komponente ausgetauscht werden.

Beispielsweise kann so eine einfache Spracherkennungskomponente durch eine komplexe Variante ausgetauscht werden, wenn ein Gerät mit ausreichend Rechenkapazität zur Verfügung steht, sodass Eingaben mittels NLP (Natural Language Processing) [BG04] oder andere aufwändige Mechanismen (z. B. erheblich größere Wörterbücher) durchgeführt werden können. Weniger Komponenten in der Komposition (Austausch n:1 mit n>1) haben den Vorteil, dass weniger Kommunikationsaufwand entsteht. Mit mehr Komponenten (Austausch 1:n mit n>1) kann gezielter auf einzelne QoS-Anforderungen eingegangen werden.
In beiden Fällen muss sichergestellt sein, dass die Sitzungsinformationen der alten Komponente(n) auf die neue(n) Komponente(n) übertragen werden, damit eine nahtlose Fortsetzung der Arbeit gewährleistet ist. Außerdem muss die Komposition aktualisiert werden, d. h. die übrigen, nicht ausgetauschten Komponenten der Komposition müssen über den Austausch benachrichtigt werden. Diese beiden Anforderungen sind auch schon bei der Komponentenmigration aufgetreten.
Damit ein Austausch von Komponenten vorgenommen werden kann, sind folgende Voraussetzungen zu erfüllen:

1. Die Summe der Funktionen der auszutauschenden Komponenten muss gleich der Summe der Funktionen der ausgetauschten Komponenten sein. Alle vorher existierenden Funktionen müssen auch nach dem Austausch vorhanden sein.

2. Wie auch schon bei der Migration ist für den Austausch von Komponenten die Einhaltung der Transaktionseigenschaften Atomarität und Konsistenz notwendig. Nach Modifikationen an der Komposition muss ein konsistenter Zustand erreicht sein muss. Der Austausch kann nur stattfinden, wenn alle erforderlichen neuen Komponenten verfügbar (d. h. installiert und gestartet) sind.

Damit der Austausch von Sitzungsinformationen zwischen unterschiedlichen Implementierungen funktioniert, muss ein entsprechendes Format zur Beschreibung der Sitzungsinformationen benutzt werden, welches von allen beteiligten Komponenten unterstützt wird. Wie in Listing 4.2 bereits angedeutet, wird die Sitzungsinformation entsprechend der Funktion(en) der Komponente strukturiert. Bei einem einfachen Austausch kommt diese Anforderung nicht zum Tragen, ist jedoch wesentlich für den komplexen Austausch.
Auf diese Weise ist eine Kapselung der Spezifikation von Sitzungsinformationen auf Granularitätsebene der Funktionsarten möglich. Daraus ergibt sich die Möglichkeit, die Spezifikation isoliert von anderen Funktionsarten zu erweitern. Für den komplexen Austausch essentiell ist jedoch die Tatsache, dass die Sitzungsinformationen den einzelnen Funktionsarten zugeordnet sind und somit den entsprechenden Komponenten übergeben werden können.

In Abbildung 4.19 ist die Übertragung der Sitzungsinformation beispielhaft dargestellt. Komponente A mit Funktionsarten 1 und 2 soll durch die Komponenten B mit Funktionsart 1 und C mit Funktionsart 2 ausgetauscht werden. Die Sitzungsinformation von A enthält die beiden Sektionen entsprechend der Funktionsarten, die dann auf zwei Sitzungsinformationsdokumente B und C aufgeteilt werden und zur Initialisierung der jeweiligen Komponenten dienen.

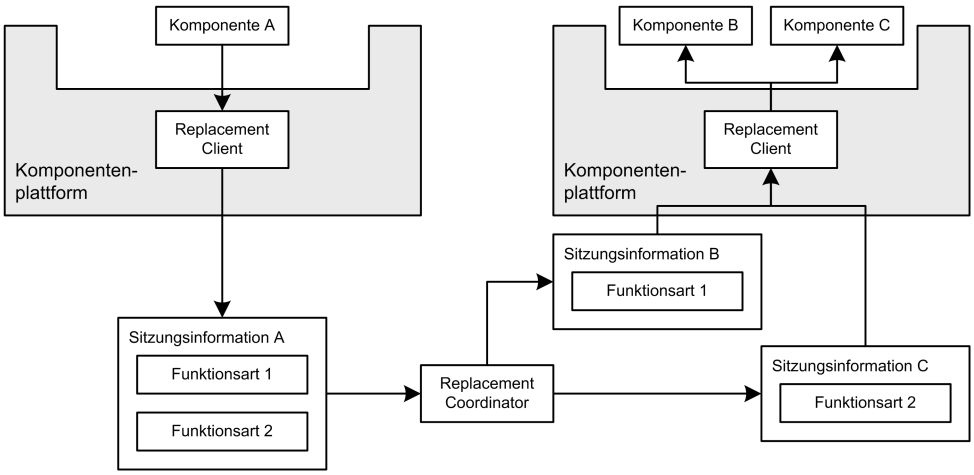

Abbildung 4.19.: Sitzungsinformationsfluss beim Austausch von Komponenten im Verhältnis 1 zu 2

Initiierung eines Austauschs und dessen Konfiguration

Der Austausch von Komponenten kann durch unterschiedliche Ereignisse ausgelöst werden. Dabei kann zwischen automatischen und manuellen Ereignissen unterschieden werden.
Das automatische Auslösen eines Austausches resultiert aus geänderten Systembedingungen, die von einer Erkennungseinheit automatisch erkannt und zu einem Komponentenaustausch verarbeitet werden. Die Erkennung kann im Dienstverzeichnis erfolgen, da dort alle Dienste mit ihrer URI registriert sind. Dafür ist erforderlich, dass bestimmte Austauschregeln definiert werden, die von der Erkennungseinheit angewendet werden können. Eine solche Regel kann beispielsweise beinhalten, dass zwei Komponenten durch eine zusammengesetzte ausgetauscht werden, sobald sich die beiden Komponenten auf der gleichen Komponentenplattform befinden, wobei verschiedene Randbedingungen erfüllt sein müssen: Einerseits muss die zusammengesetzte Komponente überhaupt vorhanden sein. Andererseits muss sichergestellt sein, dass die durch den Austausch eventuell entfernten Schnittstellen nicht verwendet wurden.
Im Gegensatz dazu wird der manuelle Komponentenaustausch vom Nutzer ausgelöst. Dieser kann mit Hilfe der Föderationsverwaltungsanwendung die Kompo-

```
<componentexchange>
  <sourcecomponent sID="12345" />
  <targetcomponent cID="98765" dID="ABCDEF" />
</componentexchange>
```

Listing 4.5: Beispiel eines Komponentenaustauschkonfigurationsdokuments

nenten auswählen, die ausgetauscht werden sollen. Auf dieser Basis werden von der Verwaltungsanwendung die möglichen Zielkomponenten ermittelt. Wenn der Nutzer alle den Quellkomponenten entsprechenden Zielkomponenten ausgewählt hat, kann er noch die Geräte auswählen, auf denen die Komponenten platziert werden sollen. Diese Informationen werden in einem Austauschkonfigurationsdokument zusammengefasst und an den Austauschkoordinator geschickt, der den Austausch dann durchführt. Ein Beispiel für ein solches Dokument ist in Listing 4.5 dargestellt. Hier soll die Komponenteninstanz mit der `sID` 12345 durch eine Komponente mit der `cID` 98765 ausgetauscht werden, die auf Gerät mit der `dID` ABCDEF platziert werden soll.

Ablauf des Komponentenaustauschs

Die Durchführung des Komponentenaustauschs wird von einem Teil des Dienstverzeichnisses (dem Austauschkoordinator) kontrolliert. Bei diesem trifft das fertige Komponentenaustauschkonfigurationsdokument ein und bildet die Grundlage für das weitere Vorgehen.
Als erstes werden alle Zielkomponenten platziert und auf deren Registrierung im DVZ (Dienstverzeichnis) gewartet. Wenn dieser Vorgang abgeschlossen ist, werden die Quellkomponenten im DVZ markiert, sodass sie bei Referenzanfrage nicht mehr berücksichtigt werden. Anschließend werden die ein- und ausgehenden Verbindungen der ausgetauschten Komponenten aktualisiert, sodass die Verbindungen an den Grenzen der Komposition und innerhalb der ausgetauschten Komponenten wieder funktionsfähig sind. Da die Quellkomponenten nun nicht mehr verwendet werden, kann aus ihnen die Sitzungsinformation extrahiert, vom Austauschkoordinator in die jeweiligen Sitzungsinformationen für die Zielkomponenten umgewandelt und zum Initialisieren der Zielkomponenten verwendet werden.
Der Austausch von datenstromempfangenden Komponenten erfolgt prinzipiell ähnlich wie deren Migration, nur dass es zwischenzeitlich nicht zwei Instanzen der gleichen Komponente gibt, sondern eine ausgetauschte und eine austauschende Komponente. Die austauschende Komponente bekommt den letzten Checkpoint der ausgetauschten Komponente zur Initialisierung, empfängt dann den Datenstrom ab dem Checkpoint und holt den Bearbeitungszustand auf, bis beide Komponenten gleich laufen und die ausgetauschte Komponente abgeschaltet werden kann. Das setzt voraus, dass die jeweiligen Verarbeitungsergebnisse zu-

einander kompatibel sind, damit die nachfolgende Komponente den Gleichlauf erkennen kann.

4.6. Synchronisierung der verteilten Nutzerschnittstellen

Die verteilten Nutzerschnittstellen müssen sowohl inhaltlich als auch zeitlich synchronisiert werden, wobei die inhaltliche Synchronisierung insbesondere dann zum Tragen kommt, wenn es sich um replizierte Nutzerschnittstellen handelt. Eine verteilte Nutzerschnittstelle sollte möglichst die gleichen Informationen anzeigen, was sich insbesondere auch auf Eingaben des Nutzers z. B. in Textfelder bezieht. Die zeitliche Synchronisierung ist in beiden Arten von verteilten Nutzerschnittstellen (aufgeteilt und repliziert) erforderlich, um dem Nutzer ein möglichst konsistentes Bild der Anwendung zu vermitteln. Die Anwendung verhält sich demnach schlüssig ohne den Nutzer zu überraschen, wie schon in [Jam86] als *Law of Least Astonishment* beschrieben wurde. Dies wird dadurch erreicht, dass die einzelnen Teile der verteilten Nutzerschnittstelle auf den verschiedenen Endgeräten möglichst gleichzeitig aktualisiert werden.
In Unterabschnitt 4.6.1 werden zunächst die möglichen Synchronisierungsarten ermittelt und gegenübergestellt. Anschließend wird in Unterabschnitt 4.6.2 die Kernidee des Synchronisierungsverfahrens vorgestellt.

4.6.1. Klassifizierung von Synchronisierungsarten

Eine Nutzerschnittstelle wird in verschiedene Ebenen unterteilt, auf denen eine Synchronisierung stattfinden kann. Die W3C MMI Working Group definiert in den Multimodal Interaction Requirements [Wor03] u. a. folgende fünf, hierarchisch gegliederte Kategorien: *event*, *field*, *form*, *page* und *session*. Synchronisierung auf *field*-Ebene bedeutet beispielsweise, dass die Eingaben eines Eingabefelds mit einer Kopie des Feldes synchronisiert werden, wenn die Eingabe abgeschlossen ist und das Feld gewechselt wird. Auf *form*-Ebene werden die Inhalte eines Formulars beim Abschicken dessen synchronisiert. Im Gegensatz dazu wird mit *session*-Synchronisierung das Verhalten bezeichnet, dass eine Anwendung in einer Modalität beendet und in einer anderen fortgesetzt werden kann. Die wichtigste Art für ein flüssiges Nutzer-Feedback ist jedoch die *event*-Ebene, während die konsistente Erscheinung der Anwendung durch Synchronisierung auf *page*-Ebene erreicht wird.
Auf beiden Ebenen gibt es zwei Aktualisierungsstrategien, nach denen die jeweiligen Elemente in der Nutzerschnittstelle reflektiert werden können: die asynchrone und die synchrone Strategie. Wie der Name schon sagt, wird bei der asynchronen Strategie jedes Element der Nutzerschnittstelle unabhängig von anderen zur Wiedergabe gebracht, was meist so schnell wie möglich erfolgt. Dies entspricht

einem Ansatz ohne besondere Berücksichtigung der Synchronisierung. Im Gegensatz dazu erfolgt die synchronisierte Strategie so, dass die Ereignisse aneinander ausgerichtet sind, sodass eine möglichst gleichzeitige Präsentation möglich ist.
Aus den zwei Synchronisierungsebenen und zwei Aktualisierungsstrategien ergeben sich vier Kombinationen, die unterschiedlich geeignet sind, die Anforderungen in Bezug auf nutzerfreundliche Interaktion mit der Anwendung zu erfüllen. Im Detail sind die vier Synchronisierungsarten wie folgt charakterisiert:

Asynchrone *event*-Synchronisierung Jedes erkannte Eingabeereignis wird auf dem erkennenden Endgerät sofort dargestellt sofern es angebracht ist und asynchron bzw. so schnell wie möglich an die anderen Nutzerschnittstellen über ggf. vorhandene Zwischenkomponenten wie Eingabeintegration weitergeleitet. Das hat zur Folge, dass sich bei simultaner Nutzung von verschiedenen Eingabegeräten die sofort dargestellte Eingabeinformation nach erfolgter Integration nochmal ändern kann. Daher handelt es sich hierbei um *vorläufige Aktualisierung*.

Synchrone *event*-Synchronisierung Jedes erkannte Eingabeereignis wird zunächst an den die verteilten Nutzerschnittstellen verwaltenden Controller geschickt, der die Integration gleichzeitiger Eingaben durchführt und anschließend die Ergebnisse auf den Endgeräten synchronisiert zur Wiedergabe bringt. Dadurch werden temporäre Inkonsistenten vermieden, allerdings ergibt sich eine Verzögerung zwischen dem Zeitpunkt der Eingabe und der Reaktion der Nutzerschnittstelle auf die Eingabe.

Asynchrone *page*-Synchronisierung Die asynchrone Synchronisierung auf *page*-Ebene aktualisiert die einzelnen Teile der verteilten Nutzerschnittstelle so schnell wie möglich, d. h. so schnell wie es die zwischenliegenden Netzwerke und Verarbeitungselemente erlauben. Dies führt bei deren Heterogenität zu Aktualisierungen an unterschiedlichen Zeitpunkten.

Synchrone *page*-Synchronisierung Die synchrone Synchronisierung auf *page*-Ebene sorgt dafür, dass die Aktualisierung der einzelnen Teile der verteilten Nutzerschnittstelle zum gleichen Zeitpunkt erfolgen, sodass für den Nutzer ein konsistenteres Bild der Anwendung entsteht.

In [Kne07] wurden diese vier Synchronisierungsarten u. a. prototypisch implementiert und in einer Nutzerstudie evaluiert. Die Nutzerstudie hat ergeben, dass auf *event*-Ebene asynchrone Aktualisierungen besser geeignet sind als synchrone Aktualisierungen, da die Verzögerung durch die Integration und kontrollierte Darstellung erheblichen Einfluss auf eine flüssige Interaktion mit der Anwendung hat. Die Nutzer gaben an, dass im Mittel 322ms die maximale Verzögerung für flüssiges Feedback ist, während dieser Wert für asynchrone bzw. vorläufige Aktualisierung das Dreifache beträgt (13 Befragte).

	event-Ebene	*page*-Ebene
asynchron	+	o
synchron	-	+

Tabelle 4.15.: Aktualisierungsarten für verteilte Nutzerschnittstellen und deren praktische Eignung (+ = gut geeignet, o = mäßig geeignet, - = nicht geeignet)

Auf *page*-Ebene hat eine zweite Nutzerstudie in [Kne07] ergeben, dass eine Nutzerschnittstelle mit Verwendung des Synchronisierungsverfahrens ein konsistenteres Bild der Anwendung vermittelt, was von 85.7% der Befragten (7 Befragte) bestätigt wurde. Die vier Synchronisierungsarten und deren praktische Eignung sind in Tabelle 4.15 dargestellt.

4.6.2. Kernidee des Synchronisierungsverfahrens

Das konzipierte Synchronisierungsverfahren basiert auf der Annahme, dass die Komponenten für die Ausgabeerzeugung entsprechend des MMI-F kategorisiert und als separate Softwarekomponenten realisiert werden, welche sich auf verschiedenen Rechnern befinden und über ein Netzwerk kommunizieren können. Daraus folgt, dass die Generation-Komponente die letzte Komponente ist, welche Kontrolle über alle unterschiedlichen Nutzerschnittstellenbeschreibungen hat, bevor diese über potentiell heterogene Netzwerke (z. B. WLAN, Bluetooth, Ethernet) zur Styling- und schließlich Rendering-Komponente übertragen werden. Wenn keine Synchronisierung stattfindet, ist es wahrscheinlich, dass die einzelnen Teile der Nutzerschnittstelle zu unterschiedlichen Zeiten auf den jeweiligen Endgeräten ausgegeben werden, weil die einzelnen Zeiten für Netzwerkübertragung und Verarbeitung auf den Geräten entsprechend der verwendeten Technologien möglichen Schwankungen unterliegen. Im Beispiel in Abbildung 4.20 wird deutlich, dass die Übertragung über Ethernet kürzer dauert als über Bluetooth und ebenso der Laptop mit der Verarbeitung eher fertig ist, als der PDA, sodass die Nutzerschnittstelle auf dem Laptop u. U. wesentlich eher aktualisiert wird als die des PDAs.
Die Kernidee des Synchronisierungsverfahrens besteht darin den frühestmöglichen Zeitpunkt zu ermitteln, an dem eine synchronisierte Aktualisierung der verteilten Nutzerschnittstellen möglich ist. Dieser Zeitpunkt ist abhängig von verschiedenen Faktoren, die für jede verwendete Modalität unterschiedlich sein können. Diese Faktoren sind hauptsächlich die Netzwerkeigenschaften, Verarbeitungskapazitäten der Zwischenkomponenten und die jeweilige Größe des Dokuments und dessen Inhalt. Für die Konzeption des Verfahrens wird der Inhalt bzw. die Struktur des Dokuments nicht berücksichtigt. Weiterhin setzt das Konzept keine synchronisierten Uhren voraus, weil diese nicht zur Ermittlung des frühestmöglichen Ak-

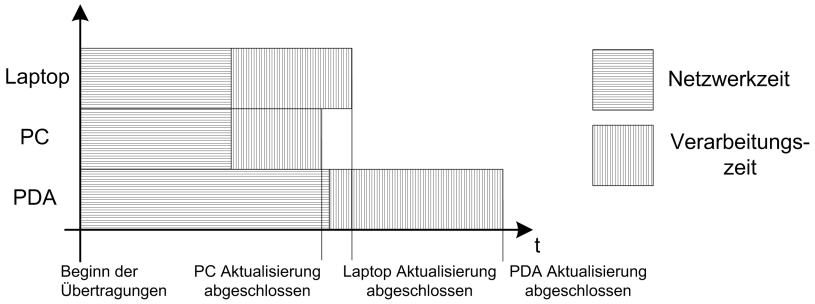

Abbildung 4.20.: Beispiel für die asynchrone Aktualisierung von Nutzerschnittstellen heterogener Endgeräte

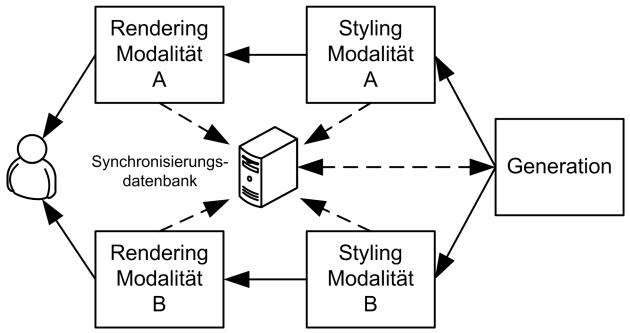

Abbildung 4.21.: Abstrakte Architektur der relevanten Komponenten

tualisierungszeitpunktes beitragen würden. Demzufolge werden Fähigkeiten wie NTP (Network Time Protocol) der Geräte nicht angenommen.
Wie in Abbildung 4.21 dargestellt ist, wirkt sich das Synchronisierungskonzept nur auf die Ausgaberollen des MMI-Frameworks (Generation, Styling, Rendering) aus. Für die Speicherung der erforderlichen Informationen hat das System zusätzlich eine Synchronisierungsdatenbank, in der die notwendigen Schlüsselattribute (Verarbeitungsraten von Netzwerken und Komponenten sowie den Inflationsfaktor) festgehalten werden. Die Ausgabekomponenten des MMI-F speichern ihre lokal ermittelten Schlüsselattribute in der Datenbank, welche dann von der Generation-Komponente ausgelesen und für die Synchronisierung verwendet werden.
Die durchgezogenen Linien in Abbildung 4.21 stellen den normalen Datenfluss von Nutzerschnittstellenbeschreibungen dar, wie er in der Implementierung des Systems auftritt. Alle den Synchronisierungsansatz aus Unterabschnitt 4.6.4 betreffenden Kommunikationen sind als gestrichelte Linien dargestellt, wobei die Rendering und Styling-Komponente nur in die Datenbank schreiben und die Generation sowohl Schreib- als auch Leseoperationen auf der Synchronisierungsda-

tenbank durchführt, um die durch die Komponenten eingetragenen Werte abzurufen.

4.6.3. Ermitteln des frühestmöglichen Synchronisierungszeitpunktes

Die Zeit für den Transport und die Zwischenverarbeitung der Nutzerschnittstellenbeschreibung von der Generation-Komponente bis zur tatsächlichen Ausgabe auf dem Endgerät wird durch verschiedene Faktoren beeinflusst, die sich in zwei Kategorien einteilen lassen: Zeit für Netzwerkkommunikation zwischen Komponenten und Zeit für Verarbeitung der empfangenen Daten und Absenden des Ergebnisses. Das Zeitintervall für die Übertragung einer Nutzerschnittstellenbeschreibung bis zur Ausgabe für eine Modalität ($T_{modality}$) berechnet sich demnach wie in Gleichung 4.1 dargestellt ist.

$$T_{modality} = T_{network} + T_{processing} \tag{4.1}$$

Da das System aus zwei Verarbeitungskomponenten und zwei potentiell heterogenen Netzwerktechnologien für jede verwendete Modalität zwischen Generation- und Rendering-Komponente besteht, lässt sich Gleichung 4.1 weiter zerlegen. Die Zeit für Netzwerkkommunikation besteht aus den Zeiten für Übertragung von Generation zu Styling ($T_{gen-sty}$) und Styling zu Rendering ($T_{sty-ren}$). Ebenso besteht die Verarbeitungszeit aus Zeit für Styling ($T_{styling}$) und Rendering ($T_{rendering}$). Folglich ergibt sich daraus Gleichung 4.2.

$$T_{modality} = T_{gen-sty} + T_{styling} + T_{sty-ren} + T_{rendering} \tag{4.2}$$

Mit Hilfe der Gleichung 4.2 wird das Zeitintervall zwischen Erzeugung einer UI-Beschreibung auf dem Server und Ausgabe an der Nutzerschnittstelle für eine Modalität ermittelt. Da die Generation-Komponente die letzte Komponente ist, die Kontrolle über alle Teile der verteilten Nutzerschnittstelle hat, ist sie verantwortlich für die Berechnung der Zeitintervalle für alle verwendeten Modalitäten und die anschließende Verwendung der Ergebnisse.
Um das Zeitintervall für die Übertragung und Verarbeitung der UI-Beschreibung einer bestimmten Modalität zu ermitteln, benötigt die Generation-Komponente Informationen über die zugehörigen Verarbeitungs- und Netzwerkraten der Komponenten und Netzwerke, die die Nutzerschnittstellenbeschreibung durchlaufen muss.
Die Netzwerkraten ($v_{gen-sty}$, $v_{sty-ren}$) des aktuell anstehenden Transports werden durch die jeweils sendende Komponente ermittelt, indem die Zeit für die Übertragung der Nutzerschnittstellenbeschreibung zur nächsten Komponente gemessen wird. Aus der gemessenen Zeit und der Datenmenge ermitteln die sendenden Komponenten, namentlich Styling und Generation, die Datenübertragungsrate und speichern diese in der Synchronisierungsdatenbank. Die Styling- und

Rendering-Komponenten messen die Verarbeitungsraten ($T_{styling}$, $T_{rendering}$) und speichern das Ergebnis ebenso in der Synchronisierungsdatenbank. Die Styling-Komponente ermittelt zusätzlich einen Inflationsfaktor α, der angibt, wie sehr die Nutzerschnittstellenbeschreibung durch die Styling-Komponente verändert wurde. Dieser Faktor wird ebenfalls in der Synchronisierungsdatenbank gespeichert. Die Verarbeitungsrate von Komponenten und der zugehörige Grad der Modifizierung eines Dokuments hängt stark von dessen Inhalt ab. Da für die Berechnung der beiden Größen nur die reine Datenmenge berücksichtigt wird, wird angenommen, dass die Dokumente stets eine charakteristische Struktur bezüglich des Inhalts haben, sodass dennoch die Berechnung der Werte auf Basis der Dokumentgröße als Indikator für die Verarbeitungsrate bzw. Modifikationsmenge gilt. Daraus ergibt sich, dass die Synchronisierungsdatenbank eine Liste mit den tatsächlichen Verarbeitungsraten jeder Netzwerkverbindung und jeder Verarbeitungskomponente sowie die aufgetretenen Inflationsfaktoren enthält. Da die einzelnen Messergebnisse natürlichen Schwankungen unterworfen sein können, die zu einer Verfälschung der weiteren Berechnung führen können, wird durch die Generation-Komponente nicht nur der jeweils letzte Messwert berücksichtigt. Zur Aufbereitung der Messwerte gibt es unterschiedliche Verfahren wie z. B. Mittelwertbildung oder Extrapolation, die je nach Eignung verwendet werden. Nach dem Einsetzen von Datenmenge ($data$), Verarbeitungsraten und Inflationsfaktor in Gleichung 4.2 ergibt sich Gleichung 4.3.

$$T_{modality} = \frac{data}{v_{gen-sty}} + \frac{\alpha * data}{v_{sty-ren}} + \frac{data}{v_{styling}} + \frac{\alpha * data}{v_{rendering}} \tag{4.3}$$

Die Synchronisierungsdatenbank wird von drei Komponenten aktualisiert: Rendering-, Styling- und Generation-Komponente. Die Aktualisierung der Werte erfolgt regelmäßig mit jedem Transport von Nutzerschnittstellenbeschreibungen durch die jeweilige Komponente. Wenn kein „natürlicher“ Verkehr über einen gewissen Zeitraum vorhanden war, kann das System „künstlichen“ Verkehr erzeugen, der am Ende nicht ausgegeben wird, aber zur Aktualisierung der Synchronisierungsdatenbank dient. Damit kann sichergestellt werden, dass die Synchronisierungsdatenbank gültige und nützliche Werte enthält, die einen relativ aktuellen Systemzustand repräsentieren. Die exakte Aktualisierungsrate hängt stark von der Verbindung zwischen Komponente und Datenbank ab, da beispielsweise mehrere Aktualisierungen pro Sekunde nicht sinnvoll sind, wenn es sich um eine langsame Verbindung über GSM handelt.
Gleichung 4.3 kann vereinfacht werden, wenn mehrere Rollen des MMI-Frameworks durch eine Komponente umgesetzt werden, z. B. wenn eine Komponente sowohl Styling als auch Rendering realisiert. Dadurch reduziert sich die Netzwerkzeit, aber insgesamt kann es zu einer Erhöhung der Gesamtverarbeitungszeit kommen, weil die Komponente dadurch auf einem Endgerät lokalisiert sein kann, welches zu keiner so leistungsfähigen Verarbeitung fähig ist, wie beispielsweise ein Server aus einer drahtgebundenen Infrastruktur.

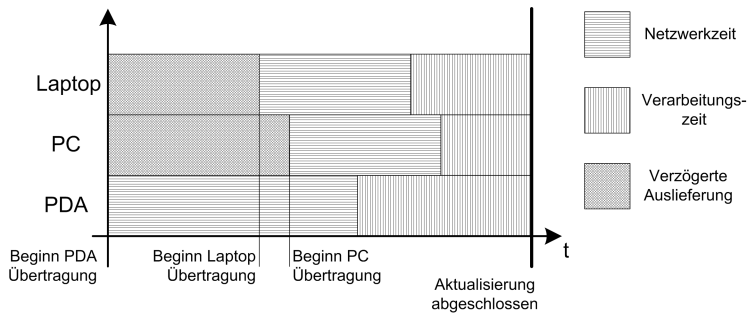

Abbildung 4.22.: Synchrone Aktualisierungen der verteilten Nutzerschnittstellen

Das längste Zeitintervall aller Modalitätszeiten ergibt die Referenzdauer (T_{ref}), welche den frühestmöglichen Zeitpunkt bestimmt, an dem alle Teile der Nutzerschnittstelle gleichzeitig aktualisiert werden können. Eine Modalität kann entsprechend der zu Grunde liegenden Hardware verschiedene Zeitintervalle haben, weshalb die Zeitintervalle für alle Instanzen einer Modalität bestimmt werden müssen, um die Referenzdauer zu ermitteln. Das alleinige Betrachten von unterschiedlichen Modalitätsklassen reicht nicht aus.
Mit dieser Referenzdauer lassen sich nun verschiedene Ansätze entwickeln, wie das gewünschte Ziel der synchronen Aktualisierung erreicht werden kann. Hierbei ist jedoch zu beachten, dass die berechneten Modalitäts-Zeitintervalle und somit die Referenzzeit einen vergangenen Systemzustand zugrundelegen, der bei der aktuell anstehenden Auslieferung eventuell nicht mehr gültig ist.

4.6.4. Verzögerte Auslieferung

Ein möglicher Ansatz für die synchronisierte Präsentation ist die verzögerte Auslieferung von Nutzerschnittstellenbeschreibungen durch die Generation-Komponente. Dazu werden die Zeitintervalle aller Modalitäten an der Referenzdauer ausgerichtet, indem die Generation-Komponente eine Korrekturzeit berechnet ($T_{modality,corr}$), die sich aus der Differenz zwischen Referenzdauer und Zeitintervalle der jeweiligen Modalität zusammensetzt (Gleichung 4.4).

$$T_{modality,corr} = T_{ref} - T_{modality} \tag{4.4}$$

Auf dieser Basis plant die Generation-Komponente das Senden der Nutzerschnittstellenbeschreibungen, wobei die mit dem längsten Zeitintervall sofort ausgesendet wird. Das Senden der übrigen Beschreibungen wird entsprechend der ermittelten modalitätsspezifischen Verzögerungszeit $T_{modality,corr}$ verzögert. In Abbildung 4.22 ist das Ergebnis nach Einführung der Verzögerungszeit vor dem Senden dargestellt.

Wie bei allen Ansätzen, die zukünftige Werte basierend auf vergangenen Werten berechnen, ist auch der beschriebene Ansatz nicht vor unvorhersehbaren Schwankungen des aktuellen Systemzustandes geschützt, die z. B. durch plötzlich hohen Netzwerkverkehr oder erhöhte CPU-Last verursacht werden. Darüber hinaus basieren alle Verarbeitungsraten zusätzlich auch auf Zeiten, die unabhängig von der tatsächlich verarbeiteten Datenmenge sind (z. B. Task Switching oder Verbindungsaufbau), was zu unterschiedlichen Verarbeitungsraten für unterschiedliche (insbesondere kleine) Datenmengen führen kann. Daher wird für diesen Ansatz angenommen, dass alle Datenmengen groß genug sind, dass die datenmengenunabhängigen Zeitanteile keine signifikante Auswirkung auf die berechneten Verarbeitungsraten haben und keine unvorhersehbaren Laständerungen auftreten.
Die verzögerte Auslieferung ist besonders in solchen Szenarien geeignet, wo die Kontrolle der Synchronisierung im Netzwerk liegen soll.
Einer der Nachteile ist der permanente Netzwerkverkehr durch die regelmäßigen Datenbankaktualisierungen. Außerdem ist eine nachträgliche Ergänzung eines Systems um diese Synchronisierungsfähigkeit mit hohem Aufwand verbunden, da Änderungen an allen Komponenten durchgeführt werden müssen. Des Weiteren ist die Extrapolation anfällig für unvorhergesehene Änderungen der Systemlast.

4.6.5. Verzögerte Ausgabe

Der eben beschriebene Ansatz Synchronisierung durch verzögerte Auslieferung kann modifiziert werden, sodass nicht mehr die Generation-Komponente die Verzögerung einfügt, sondern diese Aufgabe von den Rendering-Komponenten übernommen wird. Wenn jedoch einfach nur die modalitätsspezifische Verzögerungszeit ($T_{modality,corr}$) zum Renderer übertragen wird und dort vor Ausgabe abgewartet wird, ergibt sich für das Ergebnis keine Änderung. Für den Nutzer ist transparent, wo genau die Verzögerungen stattfinden, welche in beiden Fällen die selben Zeiten ergäbe.
Weiterhin wird durch diesen Ansatz die Verzögerungszeit im Voraus verbraucht und das Ergebnis durch die Generation-Komponente bestimmt, ohne dass die Möglichkeit der weiteren Verbesserung durch Anpassung an die bei der Auslieferung geltenden Systembedingungen besteht.
Daher wird bei dem Ansatz zur Synchronisierung durch verzögerte Ausgabe nicht die modalitätsspezifische Verzögerungszeit mit zur Renderer-Komponente übertragen, sondern die von der Generation-Komponente ermittelte Referenzzeit T_{ref}. Die einzelnen Komponenten subtrahieren nun die tatsächlich verbrauchte Zeit für Verarbeitung oder Netzwerkübertragung von der Referenzdauer, sodass am Ende eine modalitätsspezifische Korrekturzeit übrig bleibt, die im Idealfall dem normalerweise berechneten Wert $T_{modality,corr}$ entspricht aber praktisch den momentanen Systemzustand berücksichtigt und damit zu erwarten ist, dass dieser genauer ist als der vorher berechnete Wert. Sollte das Ergebnis negativ sein, hat die Verarbeitung länger gedauert als berechnet und die Aktualisierung muss ohne weitere

Pause sofort durchgeführt werden. Das Ergebnis der nachträglichen Verzögerung ist in Abbildung 4.23 dargestellt.

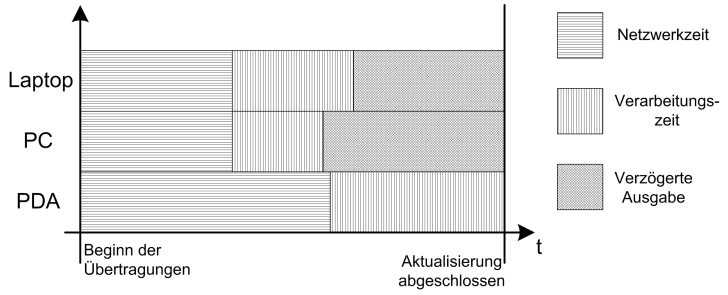

Abbildung 4.23.: Synchrone Aktualisierung mit Verzögerung durch das Endgerät

Dieser Ansatz hat den Vorteil, dass die effektive modalitätsspezifische Verzögerungszeit nicht wie bei der Berechnung durch die Generation-Komponente auf Basis der Vergangenheitswerte ermittelt wird, sondern die tatsächlichen Laufzeitwerte berücksichtigt werden. Die Vergangenheitswerte werden nur für die Bestimmung der Referenzzeit verwendet, welche in dem Fall nur als Anhaltspunkt dient, wann die frühestmögliche synchrone Aktualisierung möglich ist.

4.6.6. Weitere Adaptionen des Synchronisierungsverfahrens

Die vorgestellten Ansätze sind geeignet, um verteilte Nutzerschnittstellen möglichst synchronisiert zu aktualisieren, indem schnellere Modalitäten verzögert und damit an die Übertragungsdauer langsamerer Modalitäten angeglichen werden. Dabei werden aber zwei wesentliche Annahmen getroffen. Zum Einen wird angenommen, dass alle Modalitäten für den Nutzer gleichwichtig sind und somit die Synchronisierung überhaupt erforderlich ist. Zum Anderen sind die Ansätze nur geeignet, wenn die modalitätsspezifischen Verzögerungszeiten in etwa die gleiche Größenordnung besitzen. Andernfalls besteht die Möglichkeit, dass sehr langsame Modalitäten die gesamte Synchronisierung verzögern, was zu unakzeptablen Wartezeiten führt.
Aus diesen beiden Annahmen ergeben sich zwei Ansätze, wie die Synchronisierung weiter verfeinert werden kann, welche im Folgenden näher erläutert werden. Die Ansätze werden nur kurz beschrieben und fließen nicht in die Konzeption und Validierung ein.

Adaption durch Nutzerschnittstellenbeschreibung

Eine der beiden Annahmen war die für den Nutzer gleiche Priorisierung der Nutzerschnittstellenteile. Die automatische Ermittlung der Wichtigkeit einzelner Nutzerschnittstellenteile gestaltet sich schwierig, weil dies nicht pauschal geschehen

kann sondern vielmehr vom Inhalt der jeweiligen Nutzerschnittstellen abhängt. Wenn die Nutzerschnittstellenbeschreibung jedoch eine multimodale und verteilte Nutzerschnittstelle berücksichtigt, so kann diese priorisierende Informationen beinhalten, die als Grundlage für das exakte Planen der tatsächlichen Synchronisierung dienen.
Neben Prioritäten von Modalitäten kann die Nutzerschnittstellenbeschreibung auch konkreten Einfluss auf die Synchronisierung nehmen, indem sie bestimmte Modalitäten in eine zeitliche Relation zu anderen setzt. Das kann beispielsweise von Vorteil sein, wenn eine Sprachausgabe eher als die grafische Darstellung erfolgen soll, wodurch sich für den Nutzer eine geringere Verweildauer auf der jeweiligen Seite ergibt.
Diese Informationen der Nutzerschnittstellenbeschreibung werden von der die Synchronisierung kontrollierenden Komponente genutzt (z. B. die Generation-Komponente), um die tatsächliche Synchronisierung zu planen. Dafür werden die jeweiligen modalitätsspezifischen Verzögerungszeiten entsprechend angepasst, sodass sich die Änderungen auf die letztlich gültige Wartezeit auswirken.

Adaption durch flexible Synchronisierungsmodalitätsauswahl

Damit die Synchronisierung nicht durch sehr langsame Modalitäten negativ beeinflusst wird, kann nach Ermittlung der modalitätsspezifischen Verzögerungszeit eine Auswahl getroffen werden, welche Modalitäten tatsächlich synchron aktualisiert werden sollen. Dafür kann ein Schwellwert definiert werden, wobei die schnelleren Modalitäten synchronisiert werden, die langsameren jedoch bei der Synchronisierung vernachlässigt werden und asynchron, d. h. so schnell wie möglich, aktualisiert werden. Der Schwellwert kann dabei sowohl statisch durch einen absoluten Wert (z. B. 200ms) als auch dynamisch durch einen relativen Wert (z. B. dreifache minimale modalitätsspezifische Verzögerungszeit) definiert werden. Hierfür sind aber weitere Untersuchungen insbesondere in Kombination mit der Priorität notwendig, damit nicht gerade die für den Nutzer wichtigsten Modalitäten von der Synchronisierung vernachlässigt werden.

4.6.7. Berechnen statt Messen

Die in Unterabschnitt 4.6.4 und Unterabschnitt 4.6.5 beschriebenen Ansätze zur Verzögerung basieren auf einer kontinuierlichen Lastmessung des Systems, wodurch das Netzwerk zusätzlich belastet wird und auch eine zusätzliche Komponente (die Synchronisierungsdatenbank) erforderlich ist, welche regelmäßig mit Informationen versorgt werden muss. Diese Aspekte können sich negativ auf das System auswirken, wenn z. B. schmalbandige Netzwerkverbindungen zum Einsatz kommen.
Um diese Nachteile zu umgehen, können im Vorfeld die verwendeten Geräte, Technologien und Komponenten in Bezug auf ihre angebotenen bzw. benötigten Res-

sourcen (Hardwareausstattung, z. B. CPU, Speicher, Netzwerkverbindung) untersucht werden, welche in einer für die Generation-Komponente zugreifbaren Datenbank abgelegt sind. Auf dieser Basis lässt sich grob abschätzen, wie groß die Verarbeitungsrate der jeweiligen Komponenten ist. Somit kann die modalitätsspezifische Verzögerungszeit berechnet werden, ohne dass dafür das System permanente Messungen durchführen muss.

Die Berechnungsvariante basiert wie die Messvariante auf einer Abschätzung von zukünftigen Werten für die Verarbeitungsraten, sodass auch hier die Gefahr vor unvorsehbaren Änderungen des Systemverhaltens besteht. Beide Varianten können für die serverseitige als auch die clientseitige Verzögerung verwendet werden.

Der Vorteil der Berechnungsvariante gegenüber der Messvariante ist der geringe und isolierte Implementierungsaufwand, da für die Umsetzung nur die Generation-Komponente modifiziert werden muss, sodass eine nachträgliche Einführung des Berechnungsansatzes für die übrigen Komponenten transparent erfolgen kann. Daher ergibt eine Kombination von Berechnung und clientseitiger Verzögerung nur wenig Sinn, wenn ein bestehendes System um die Synchronisierungsfunktionalität erweitert werden soll.

Nachteil der Berechnungsvariante ist die Tatsache, dass im Gegensatz zur Messvariante die modalitätsspezifische Verzögerungszeit und damit die Referenzzeit nicht vom Systemzustand zur Laufzeit abhängt, sondern von einem idealen System ausgeht.

4.6.8. Zusammenfassung Synchronisierung

Der entwickelte Synchronisierungsmechanismus stellt sicher, dass Unterschiede der Verarbeitungsdauer einzelner Modalitäten durch heterogene Komponenten und Netzwerke kompensiert werden, sodass die Aktualisierung der verteilten Nutzerschnittstellen synchronisiert erfolgt. Dabei kann die tatsächliche Angleichung durch den Server oder den Client durchgeführt werden, wobei sich unterschiedliche Vor- und Nachteile ergeben, wie die genauere Angleichung durch den Client oder die aufwandsarme Ergänzung eines bestehenden Systems durch serverbasierte Angleichung. Dabei kann der Synchronisierungsmechanismus sowohl auf *event*-Ebene als auch auf *page*-Ebene eingesetzt werden, wobei sich aber der Einsatz auf *event*-Ebene als nicht empfehlenswert herausgestellt hat. Da AJAX-Anwendungen wie eingangs erwähnt aufgrund der Vermischung von Nutzerschnittstelle und Geschäftslogik gesondert betrachtet werden müssen, ist auch das Synchronisierungsverfahren nicht ohne Weiteres auf diese Anwendungen erweiterbar, sondern bedürfen einer tiefergehenden Untersuchung der getroffenen Annahmen z. B. über die Nachrichtengröße.

4.7. Fazit

In den vergangenen Abschnitten wurden alle notwendigen Aspekte der Integrationsschicht beschrieben, die für die Umsetzung der motivierenden Anwendungsfälle im Speziellen, aber im Allgemeinen auch zur Realisierung von beliebigen Anwendungen mit beliebigen Modalitäten und föderierten Endgeräten geeignet sind.
Die Integrationsschicht besteht aus Komponenten und Komponentenplattformen, die die Komponenten ausführen und verschiedene Basisdienste anbieten. Weiterhin sind die Verzeichnisdienste wesentlicher Bestandteil der Integrationsschicht, da diese zur Verwaltung von Nutzern, Geräten, Komponenten und Diensten benötigt werden.
Neben den technischen Elementen bzw. Struktureinheiten verfügt die Integrationsschicht auch über verschiedene Konzepte, die besonders für die Verwendung als UI-Plattform erforderlich sind. Grundlage dafür ist die service-orientierte Komposition von Komponenten in Form einer Choreographie. Aufbauend darauf können die Komponenten kontextsensitiv verteilt werden, von einer Plattform zur anderen migrieren und durch neue Komponenten ausgetauscht werden. Zur Erhöhung der Nutzerfreundlichkeit werden die verteilten Nutzerschnittstellen einer Synchronisierung unterworfen, sodass möglichst gleichzeitige Aktualisierungen mit dem Ziel erfolgen, eine konsistente Erscheinung der Zielanwendung zu erzeugen.
Einige der beschriebenen Konzepte unterstützen unterschiedliche Varianten der Realisierungen, die teilweise prinzipiell das gleiche Ergebnis haben, sich aber in Realisierungsdetails unterscheiden. Eine Übersicht der entwickelten Konzepte und deren Varianten ist zusammenfassend in Tabelle 4.16 dargestellt. In allen Fällen gilt, dass nicht pauschal festgelegt werden kann, welche Variante optimal ist, da dies stets vom jeweiligen Verwendungszweck bzw. der gewünschten Anwendung abhängt. In Kapitel 5 werden die Konzepte in Bezug auf ihre Eignung verglichen.

Konzept	Varianten
Platzierungsstrategie	gezielt, wahlfrei
Platzierungszeitpunkt	Anwendungsinstallation, bei Bedarf
Verteilungsstrategie	stark verteilen, wenig verteilen
Migration	ohne Verwendung, mit Verwendung, Datenstromempfänger
Aufrufbehandlung	implizite Umleitung, explizite Umleitung
Austausch	einfach, komplex
Synchr.-granularität	event, page
Synchr.-ausführung	Server, Client
Synchr.-strategie	kontrolliert, vorläufig

Tabelle 4.16.: Zusammenfassung der mit Varianten spezifizierten Konzepte

Denn es ist zuletzt doch nur der Geist, der jede Technik lebendig macht.

Johann Wolfgang von Goethe

5 Validierung

Dieses Kapitel dient der Validierung der Integrationsschicht und der zugehörigen Konzepte aus Kapitel 4. Dazu wird in Abschnitt 5.1 zunächst das Ziel und das Vorgehen der Validierung erläutert. In Abschnitt 5.2 wird das Wartungsszenario aus Kapitel 2 aufgegriffen und detailliert beschrieben. Die vom Wartungsarbeiter implizit verwendete Interaktionsanwendung und ihre Komponenten werden in Abschnitt 5.3 spezifiziert. Anschließend wird in Abschnitt 5.4 der entwickelte Prototyp vorgestellt und in Bezug auf Erfüllung der in Unterabschnitt 2.1.1 aufgestellten Testfälle untersucht. Die durchgeführten Performanztests und deren Ergebnisse werden in Abschnitt 5.5 dargestellt. Anschließend werden die verschiedenen Varianten der entwickelten Konzepte in Abschnitt 5.6 gegenübergestellt. Daraufhin wird die entwickelte Lösung bezüglich der aufgestellten Anforderungen in Abschnitt 5.7 bzw. den Kernfragen in Abschnitt 5.8 analysiert. Abschnitt 5.9 fasst das Validierungskapitel zusammen.

5.1. Ziel und Vorgehen der Validierung

Mit der Validierung soll festgestellt werden, inwiefern die entwickelten Konzepte den Nutzer beim Verwenden von föderierten Endgeräten zum multimodalen Zugriff auf Anwendungen unterstützen bzw. diese Art der Interaktion ermöglichen. Zur Validierung wird das in Unterabschnitt 2.1.1 eingeführte Anwendungsszenario aufgegriffen und einer quantitativen Untersuchung unterworfen. In dem Anwendungsszenario erledigt ein Wartungsarbeiter verschiedene Tätigkeiten an einer Maschine anhand elektronischer Dokumentation. Entsprechend seiner jeweiligen Aufgabe föderiert der Arbeiter dabei sein hauptsächlich genutztes Endgerät, den

PDA, mit anderen Endgeräten, um seine Interaktionsmöglichkeiten mit der Anwendung zu erweitern. Dazu wird das Anwendungsszenario in Abschnitt 5.2 mit einem großen Maß an Detail erneut beschrieben. Es wird in jedem Schritt dargestellt, welche Komponente der in Abschnitt 5.3 definierten Interaktionsanwendung sich auf welchem Endgerät befindet und welche Aktionen der Wartungsarbeiter bezüglich der Komponenten und Geräte durchführt. Für verschiedene Aktionen werden in Abschnitt 5.2 Testfälle definiert, die durch genaue Vor- und Nachbedingungen beschrieben werden. In Abschnitt 5.4 werden die Testfälle aufgegriffen, teilweise erläutert und auf ihre Erfüllung untersucht. Die komplette Übersicht der Testfälle inkl. deren Beschreibung befindet sich in Anhang A.
Anhand der Testfälle und deren Erfüllung kann geschlussfolgert werden, inwiefern die entwickelten Konzepte den Nutzer bei der multimodalen Interaktion unterstützen. Wie gut die Unterstützung erfolgt, wird teilweise durch die Performanzuntersuchungen in Abschnitt 5.5 belegt.

5.2. Anwendungsszenario - Wartungsarbeiter

Das Anwendungsszenario des Wartungsarbeiters wird in Unterabschnitt 2.1.1 noch unabhängig von der technischen Umsetzung beschrieben. Daher wird dieses Anwendungsszenario in diesem Abschnitt spezifisch zur entwickelten Lösung erneut und dementsprechend detaillierter erläutert.
Der erste Teil des Szenarios ist in Abbildung 5.1 dargestellt. Der Wartungsarbeiter beginnt seine Tätigkeit an der Maschine, indem er den Laptop benutzt, um sich bei der Anwendung zur Föderationsverwaltung und damit bei der Integrationsschicht anzumelden (TF01). Anschließend wählt er aus einer Liste möglicher Nutzerschnittstellenkomponenten den HTML-Browser aus (TF02) und lässt ihn auf dem Laptop platzieren (TF03). In den Browser gibt der Arbeiter die URL ein, welche die Zielanwendung identifiziert, d. h. wo die Zielanwendung zu finden ist. Bereits zu Beginn der Interaktion versucht der Browser, die einzelnen Eingaben integrieren zu lassen, wofür eine Komponente benötigt wird, die die Rolle der Integration erfüllt. Daher stellt der Browser eine entsprechende Anfrage an das Dienstverzeichnis und erhält als Antwort eine URI als Referenz auf die Eingaben-integrierende Komponente bzw. den von ihr angebotenen Dienst (TF04). Da zunächst keine semantische oder syntaktische Integration benötigt wird und außerdem noch keine Instanz einer solchen Komponente existiert hat, wird mit Hilfe des Komponentenverzeichnisses eine neue Instanz auf einem beliebigen Gerät platziert (TF05), die lediglich zur Kanalisierung der Ereignisse benötigt wird. Sobald die neue Integration-Komponente die ersten Eingaben empfängt, ermittelt sie eine Referenz auf den Interaktionsmanager, der die weitergeleiteten Eingaben entsprechend ihrer Art weiterverarbeitet, d. h. entweder an die Teile der verteilten Nutzerschnittstelle oder die Anwendung weiterleitet. Die Eingabe der URL wird abgeschlossen, woraufhin der Interaktionsmanager den angegebenen

Inhalt vom Anwendungsserver anfordert, die Ausgabe für den HTML-Browser erzeugt, an diesen weiterleitet (TF06) und darstellt.
Nachdem der Wartungsarbeiter mehrere Schritte der Wartungsanwendung durchlaufen hat, möchte er für die nachfolgenden Abschnitte die Anwendung zusätzlich per Sprache steuern, weil er die Aufgaben mit beiden Händen erledigen muss. Dazu nimmt er seinen PDA, meldet sich an der Föderationsanwendung an und wählt aus der Liste von Nutzerschnittstellenkomponenten einen Voice-Browser zur lokalen Installation aus (TF07), der gleichzeitig auch eine Sprachinterpretation durchführt. Zusätzlich installiert er aus Gründen des Komforts auch auf dem PDA den HTML-Browser, um schnell und einfach auch visuelles Feedback zu seinen Eingaben zu erhalten.
Die Infrastruktur besteht zu Beginn aus einem Integrationsschicht-Server für die Verzeichnisdienste, einem Anwendungsserver für die Zielanwendung und dem Laptop als Endgerät zur Nutzerinteraktion. Die Komponenten für Föderationsverwaltung und Verzeichnisdienste sind schon vorher installiert, während die anderen (HTML-Browser und Integration) erst durch den entsprechenden Testfall (TF03 bzw. TF05) auf den jeweiligen Plattformen platziert werden. Der Nutzer nimmt den PDA hinzu und installiert auf diesem in TF07 den Voice-Browser mit Integration.

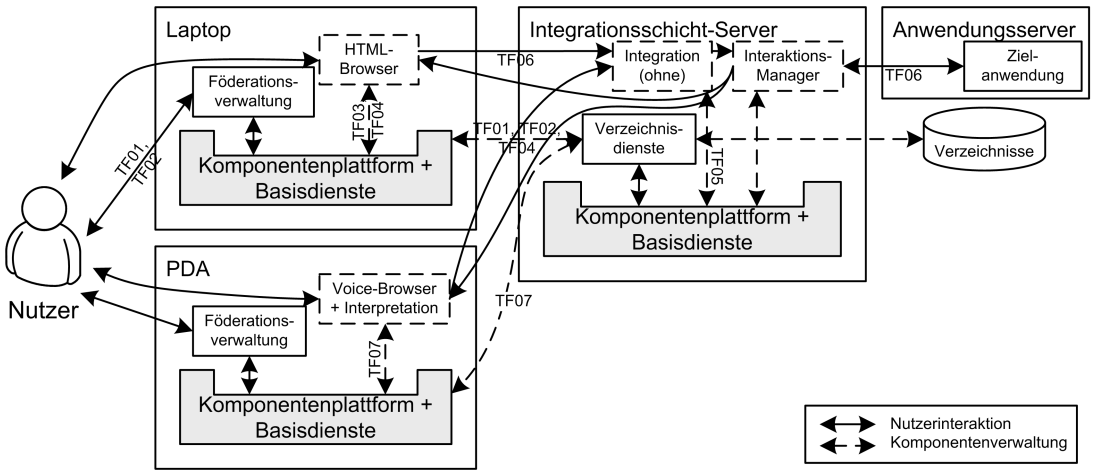

Abbildung 5.1.: Detailliertes Anwendungsszenario, Teil 1

Nach wenigen Interaktionen mit dem Voice-Browser stellt der Nutzer fest, dass dieser zu wenige Kommandos erkennt. Daher entscheidet sich der Nutzer, den alten Voice-Browser mit Spracherkennung gegen einen neuen Voice-Browser mit externer Spracherkennung zu installieren (TF08). Der Wartungsarbeiter möchte, dass alle vom Voice-Browser benötigten weiteren Komponenten zu dessen Installationszeit ebenfalls installiert werden. Dabei soll eine möglichst geringe Verteilung erfolgen, um die Latenz durch Netzwerkkommunikation so niedrig wie möglich zu halten (TF09). In diesem Fall wird noch eine Interpretationskom-

ponente auf dem PDA installiert, welche die Interpretation von Sprachsignalen realisiert und daraus Text erzeugt (TF10), der weiter verarbeitet werden kann. Erneut stellt der Wartungsarbeiter nach wenigen Interaktionen mit dem Voice-Browser fest, dass dieser unzureichend funktioniert. Diesmal ist der Grund jedoch die zu geringe Verarbeitungsleistung auf dem PDA. Deshalb weist er mittels der Föderationsverwaltungsanwendung des Laptops an, dass die bisherige Spracherkennungskomponente auf einen Infrastrukturserver migrieren soll, wo durch eine höhere Prozessorleistung eine bessere Verarbeitungsleistung erreicht werden soll (TF11).
In Abbildung 5.2 ist der zweite Teil des Wartungsszenarios anhand einer detaillierten Ansicht von PDA und Integrationsschicht-Server dargestellt. Die Kombination aus Voice-Browser und Voice-Interpretation wird aufgelöst (TF08 und TF09 in Abbildung 5.2a) während in Abbildung 5.2b die Voice-Interpretation auf den Integrationsschicht-Server verschoben wird.

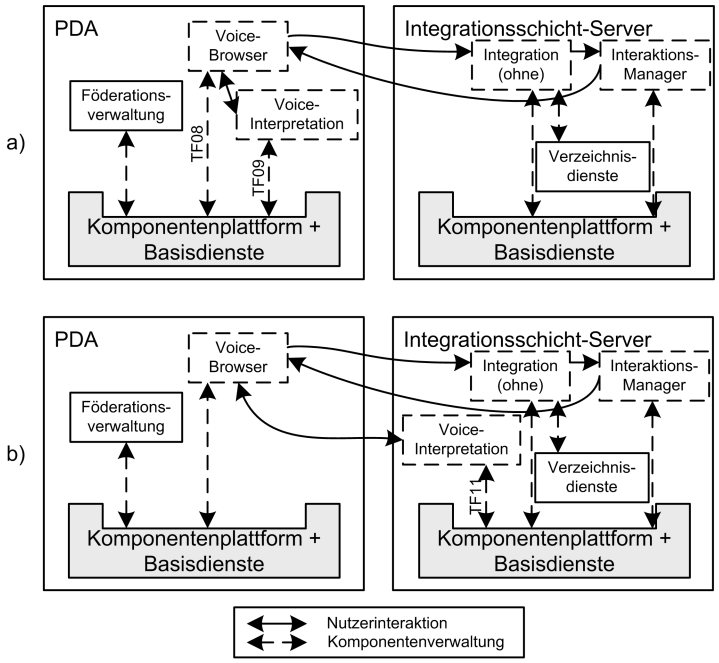

Abbildung 5.2.: Detailliertes Anwendungsszenario, Teil 2

Den Laptop hat der Wartungsarbeiter währenddessen so aufgestellt, dass auch aus größerer Entfernung die grobe Struktur der Wartungsaufgabe zu sehen ist, welche sich dem aktuellen Dialogzustand anpasst, den der Wartungsarbeiter nun durch den Voice-Browser kontrolliert (TF12). Beim Ausführen eines bestimmten Wartungsschritts fällt dem Wartungsarbeiter eine Ungereimtheit auf, die er in der Anwendung als Annotation hinterlegen will. Zur besseren Kontrolle begibt er sich zum Laptop und spricht die Annotation in den PDA. Auf dem Laptop erscheint der Text in Kleinbuchstaben (TF13). Um den wichtigen Charakter der

Annotation hervorzuheben, soll der Text in Großbuchstaben erscheinen. Daher ändert der Wartungsarbeiter seine Nutzereinstellung der Eingabeintegration von „ohne“ auf „einfach“, wodurch der Austausch der alten Integrationskomponente durch eine neue ausgelöst wird (TF14). Anschließend hält der Wartungsarbeiter die Umschalt-Taste gedrückt und spricht seine Annotation erneut, sodass der Text in Großbuchstaben erzeugt und kontrolliert auf den Endgeräten angezeigt wird (TF15).

Als die Wartungsaufgabe abgeschlossen ist, benötigt der Wartungsarbeiter die Sprachfunktionalität nicht mehr. Daher meldet er sich von der Föderationsverwaltung auf dem PDA ab, was die Deinstallation des Voice-Browsers zur Folge hat (TF16), da sich dieser auf dem PDA befand. Anschließend begibt sich der Wartungsarbeiter zum Manager der gewarteten Anlage, um ihm Bericht zu erstatten. Da sich die Wartungsergebnisse auf dem im Managerzimmer installierten Beamer besser darstellen lassen, wählt der Wartungsarbeiter erneut den HTML-Browser in der Föderationsverwaltungsanwendung aus und lässt ihn auf dem PC des Beamers installieren (TF17). Daraufhin wird die Wartungsanwendung auf beiden Endgeräten dargestellt, sodass die beteiligten Personen bequem den Inhalt sehen können. Damit der Wartungsarbeiter die Anmerkungen des Direktors in die Anwendung eingeben kann, ohne durch die Verzögerung einer kontrollierten Synchronisierungsstrategie behindert zu werden, modifiziert er seine Nutzereinstellungen dementsprechend, sodass ab sofort eine vorläufige Synchronisierung von Eingaben erfolgt (TF18).

Die Testfälle TF14, TF16 und TF17 sind in Abbildung 5.3 dargestellt. Die Integrationskomponente ist durch eine andere ausgetauscht worden. Weiterhin wurde der PDA aus der Föderation entfernt und der PC des Beamers hinzugefügt. Die Voice-Interpretation ist noch immer verfügbar, auch wenn sie keine Aufgaben mehr erledigt.

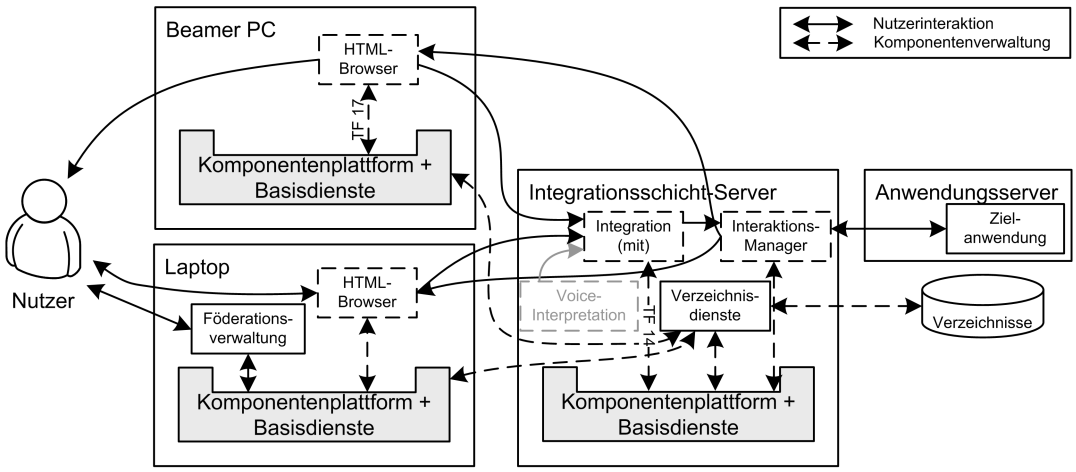

Abbildung 5.3.: Detailliertes Anwendungsszenario, Teil 3

Nach Abschluss der Präsentation benutzt der Wartungsarbeiter die Föderationsverwaltungsanwendung auf dem Laptop, um den HTML-Browser von Beamer-PC und Laptop zu deinstallieren (TF19), da die Aufgabe abgeschlossen ist. Daher meldet sich der Wartungsarbeiter von der Integrationsschicht ab, weshalb alle übrigen ihm zugeordneten Komponenten deinstalliert werden (TF20).

5.3. Die Interaktionsanwendung und ihre Komponenten

Die komponentenbasierte Interaktionsanwendung stellt dem Wartungsarbeiter die benötigten Modalitäten zur Verfügung und realisiert auch deren Koordinierung. Prinzipiell ist die Integrationsschicht und deren Konzepte für beliebige Anwendungen geeignet, die auf verteilten Komponenten basieren und eine serviceorientierte Architektur zur Komposition und Kommunikation unterstützen.
Die Komponenten der Interaktionsanwendung orientieren sich in Struktur und Funktionsweise am MMI-F. Damit die Flexibilität nicht einschränkt und ein performantes Laufzeitverhalten möglich ist, muss eine entsprechende Abbildung der Rollen des MMI-F auf Komponenten gefunden werden. Diese Abbildung wird durch das Rollenmodell des MMI-F absichtlich offen gelassen. Die beiden Forderungen nach Flexibilität und Performanz haben zur Folge, dass einigen Komponenten mehrere Rollen mit enger funktionaler Kopplung zugewiesen werden. In Unterabschnitt 5.3.1 wird das MMI-F bezüglich der funktionalen Kopplung untersucht.
Im Laufe der Tätigkeit verwendet der Wartungsarbeiter fünf Nutzerschnittstellenkomponenten auf Laptop, PDA und Beamer. Im Hintergrund existieren einige Komponenten, die für die Verwaltung und Integration der Informationen der Interaktionsanwendung verantwortlich sind. Nachdem in Unterabschnitt 5.3.1 das MMI-F in Bezug auf notwendige Komponentenarten untersucht wird, beinhaltet 5.3.2 die Spezifikation der für die prototypische Umsetzung benötigten Komponenten.

5.3.1. Untersuchung des MMI-Frameworks

In diesem Abschnitt wird ermittelt, welche Rollen des MMI-F eine enge funktionale Kopplung aufweisen und sich deswegen für eine gemeinsame Abbildung auf eine Komponente eignen.
In Abbildung 3.1 sind die Kardinalitäten der Rollen implizit dargestellt. Die Rollen lassen sich in modalitätsspezifische (Rendering, Styling, Recognition und Interpretation) und modalitätsunabhängige Rollen (Integration, Interaction, Generation, Application functions, System&Environment und Session Component) unterteilen. Die modalitätsunabhängigen Rollen werden nur einmal benötigt, während die modalitätsspezifischen Rollen für jede verwendete Modalität zum

Einsatz kommen. Daraus ergibt sich eine notwendige Trennung des Gesamtsystems an der Stelle zwischen modalitätsunabhängigen und modalitätsabhängigen Komponenten.
Die Recognition und Interpretation sind jeweils spezifisch für eine Modalität, sodass eine gewisse funktionale Kopplung vorhanden ist. Der Grad dieser Kopplung hängt jedoch von der jeweiligen Modalität ab. Bei Texteingabe ist es naheliegend, dass die gleiche Komponente sowohl Recognition als auch Interpretation durchführt. Dies wird am Beispiel eines Web-Browsers deutlich, der die Eingaben von Tastatur und Maus sowohl erkennt als auch interpretiert. Zusätzlich werden diese beiden Modalitäten bereits durch den Browser integriert. Gleichzeitig ist der Browser auch für die visuelle Ausgabe zuständig.
Für Spracheingabe jedoch sind getrennte Komponenten sehr gut geeignet, da je nach Systemumgebung eine unterschiedlich komplexe Interpretation des aufgenommenen Sprachsignals erfolgen kann. Dafür bieten sich unterschiedliche Komponenten an, die aber jeweils die Rolle der Sprach-Interpretation realisieren. Ein Voice-Browser beispielsweise kann nur für die direkte Ein- und Ausgabe verantwortlich sein, während die Umsetzung des Sprachsignals in konkreten Text oder Kommandos von einer entfernten Komponente erfolgen kann.
Auf Ausgabeseite werden die Nutzerschnittstellenbeschreibungen von Styling und Rendering bearbeitet, welche eine hohe Kopplung aufweisen und daher von einer Komponente realisiert werden sollten.
Der Interaction Manager nimmt eine zentrale Position ein, da er die ankommenden Informationen von der Integration ggf. an die Zielanwendung weiterleitet, von der Zielanwendung Resultate empfängt und diese an die Generation unter Einbeziehung von System&Environment und Session Component weiterleitet. Sowohl System&Environment als auch Session Component werden ausschließlich vom Interaction Manager verwendet und sollten daher von der selben Komponente realisiert werden.
Die Fusion von Integration und Generation mit dem Interaction Manager ist zwar möglich, da zwischen diesen jeweils auch nur eine 1:1-Relation besteht. Allerdings können diese beiden Rollen mit sehr unterschiedlichen QoS-Eigenschaften realisiert werden, sodass separate Komponenten hierfür sinnvoll sind.
Da die Application functions eine eigenständige Komponente bleiben, ist die Flexibilität der Anbindung verschiedener Zielanwendungen an die Interaktionsanwendung gegeben.
Zusammenfassend sind in Tabelle 5.1 die Komponenten dargestellt, die für die Realisierung der Nutzerinteraktion notwendig sind. Die für den Prototypen notwendigen Komponenten werden im folgenden Abschnitt 5.3.2 spezifiziert.

5.3.2. Spezifikation der Interaktionsanwendungskomponenten

Wie aus dem Anwendungsszenario in Abschnitt 5.2 hervorgeht, werden für den Prototypen verschiedene Komponenten benötigt, die sich sowohl auf dem Endge-

Anzahl	Komponentenart	MMI-F-Rollen
*	Nutzerschnittstelle	Recognition, Styling, Rendering, (Interpretation)
*	Interpretation	Interpretation
1	Eingabenverarbeitung	Integration
1	Ausgabeerzeugung	Generation
1	Interaktionssteuerung	Interaction Manager, Session Manager, System&Environment, (Generation)
1	Anwendung	Application functions

Tabelle 5.1.: Benötigte Komponenten und deren MMI-F-Rollen

rät und daher mit Nutzerschnittstelle, als auch zur Unterstützung ohne Nutzerschnittstelle auf einem beliebigen Gerät befinden.
Auf Laptop und Beamer benötigt der Wartungsarbeiter eine visuelle Ausgabe (TF03, TF17). Bei Verwendung des Laptops sind zusätzlich Eingabemöglichkeiten durch Tastatur und Maus notwendig. Daraus folgt, dass eine gemeinsame Komponente verwendet werden kann, um den Inhalt der Anwendung auf visuelle Weise auszugeben. Hierfür bietet sich ein HTML-Browser an. Der PDA wird für sprach-basierte Interaktion verwendet, weshalb ein Voice-Browser benötigt wird, der die in VoiceXML kodierte Nutzerschnittstelle zur Verfügung stellt.
Die Voice-Browser und der HTML-Browser werden im Folgenden spezifiziert. Außerdem werden nachfolgend die Unterstützungskomponenten spezifiziert, die beispielsweise für die Integration verschiedener Eingaben, Interaktionssteuerung oder Voice-Interpretation benötigt werden.

HTML-Browser

Der HTML-Browser (Komponenten-ID 2) realisiert die Interaktion des Nutzers mit der Zielanwendung durch visuelle Ausgabe und haptische Eingabe, wobei die Dialoge in HTML-kodiert sind. Er unterstützt eine visuelle Ausgabe der Nutzerschnittstelle auf Laptop und Beamer, während er auf dem Laptop ebenfalls die Eingabe von Informationen unterstützt. Da der Beamer an einem PC angeschlossen ist, hat er ebenfalls Eingabemöglichkeiten, auch wenn diese vom Nutzer u. U. nicht erreichbar sind, da er primär zur Anzeige genutzt werden soll. Damit ist der HTML-Browser auch für die Erkennung von Text- und Mauseingaben verantwortlich, welche bereits interpretiert und teilweise integriert werden. Demzufolge erfüllt der HTML-Browser die Funktionen, welche im `functions`-Teil des Komponentenprofils dargestellt sind. Das Komponentenprofil der HTML-Browser-Komponente befindet sich in Abschnitt B.1.

Schnittstelle Die HTML-Browser-Komponente muss eine Schnittstelle besitzen, über die der Browser Informationen zur Aktualisierung der verteilten Nutzerschnittstelle empfangen kann. Prinzipiell gibt es zwei verschiedene, durch den Browser verarbeitbare Informationsarten, die sich aus den Synchronisierungsgranularitäten ergeben: zum Einen werden feingranulare Ereignisse von den jeweiligen UI-Komponenten erzeugt und müssen zu den anderen repliziert werden (*event*-Ebene), zum Anderen kann der Dialogzustand von einer der UI-Komponenten geändert werden, was zu einem komplett neuen Dialogschritt führt und den anderen UI-Komponenten mitgeteilt werden muss (*page*-Ebene). Daher besitzt die Schnittstelle der HTML-Browser-Komponente zwei (angebotene) Methoden:

`void render(String url, String xhtmlDescription, String options)`
: Weist die HTML-Browserkomponente an, das übergebene HTML-Dokument darzustellen, welches zur Adresse `url` gehört. Die Optionen `options` sind für die Übergabe von Synchronisierungsinformationen erforderlich.

`void reactEvent(String clientID, String event)`
: Mit dieser Methode wird der Browser angewiesen, ein bestimmtes Ereignis `event` zu verarbeiten und somit das UI zu aktualisieren. Die `clientID` gibt an, auf welchem Client das Ereignis aufgetreten ist, damit dieser das Ereignis nicht erneut verarbeitet. Das Ereignis besteht aus Informationen über den XPath-Ausdruck `xpath` [XSL99], um das Ziel-Element zu identifizieren, die `eventID` zur Identifikation des Ereignisses, den `eventType` zur Bestimmung des Ereignistyps und den eigentlichen Inhalt `eventContent`. Zusätzlich wird durch die `pageID` die Seite bezeichnet, auf der das Ereignis aufgetreten ist.

Neben den angebotenen Methoden hat der Browser eine benötigte Schnittstelle, über die er die lokalen Ereignisse zur Integration und Weiterverteilung an die anderen UI-Teile übermitteln kann. Die entsprechende Funktion wird von einer Komponente bereitgestellt, die die Funktion Integration erfüllt. Die Repräsentation der Methoden im Kopplungsprofil ist im Komponentenprofil in Abschnitt B.1 dargestellt.

Abhängigkeitsplan Der Browser hat als benötigte Komponente die Integration (Komponenten-ID 3) definiert.

Anforderungen an die Ausführungsumgebung Damit der HTML-Browser verwendet werden kann, muss das Endgerät über bestimmte Eigenschaften verfügen. Dazu gehört, dass es die durch die Funktionen Rendering und Recognition spezifizierten Modalitäten unterstützt, was im `requirements`-Teil des Komponentenprofils festgelegt wird. Weitere Anforderungen an die Softwareumgebung

hat der HTML-Browser nicht, da er nur auf die Verfügbarkeit der minimalen Systemkonfiguration optimiert wurde.

Implementierungsdetails Der HTML-Browser basiert auf der frei verfügbaren HTML-Rendering-Engine Cobra, welche Teil des Lobo Projekts[1] ist. Die Cobra-Engine basiert auf Java Swing und wurde für den HTML-Browser so modifiziert, dass sie kompatibel mit Java AWT ist. Weiterhin wurden die für die Interaktion mit Web-Servern zuständigen Klassen modifiziert, sodass sie die Interaktion nicht selbst ausführen, sondern die jeweiligen Kommandos an die Integration-Komponente und letztlich die Interaktionssteuerung übermitteln. Darüber hinaus war es notwendig, dass alle Eingabeelemente in einer gesonderten Liste mit ihrem identifizierenden XPath [XSL99] gespeichert werden, damit empfangene Ereignisse dem entsprechenden Eingabeelement zugeordnet werden können. Es wurde außerdem ein Warteschlangenkonzept implementiert, das die Reihenfolge in der Verarbeitung ankommender Ereignisse sicherstellt. Diese werden nur verarbeitet, wenn deren Sequenznummer der erwarteten Sequenznummer enstpricht. Ist dies nicht der Fall, wird das Ereignis in die Warteschlange geschrieben und von dort ausgelesen, wenn das Vorgängerereignis verarbeitet wurde. Bezogen auf den Inhalt werden derzeit nur textuelle Änderungen synchronisiert. Änderungen am Fokus oder an Auswahlelementen bzw. Scrollen innerhalb einer Seite werden gegenwärtig nicht unterstützt. Weitere Details befinden sich in [Pen07].

Voice-Browser mit Interpretation

Der Voice-Browser mit Interpretation (m. I., Komponenten-ID 5) ist ein Voice-Browser, der eine Interaktion des Nutzers mit der Anwendung via Sprache ermöglicht. Dazu wird eine Dialogbeschreibung in VoiceXML benötigt, die von der Generation-Komponente entsprechend der Anwendung erzeugt werden muss oder von dieser bereits geliefert wird. Im Unterschied zum HTML-Browser wird jedoch vom Voice-Browser kein XHTML benötigt, sondern VoiceXML. Der Voice-Browser hat eine integrierte Interpretationskomponente und erfüllt die Funktionen, die im Komponentenprofil in Abschnitt B.5 aufgeführt sind.

Schnittstelle Ebenso wie der HTML-Browser verfügt der Voice-Browser über eine Schnittstelle zum Empfang von Dokumenten und einzelnen Ereignissen. Der einzige Unterschied ist das benötigte Inhaltsformat. Während der HTML-Browser HTML erwartet, verarbeitet der Voice-Browser den Dialog als VoiceXML-Dokument. Der Unterschied des akzeptierten Inhaltsformats wird durch einen QoS-Eintrag der jeweiligen Methode im Kopplungsprofil angegeben.

[1] http://html.xamjwg.org/index.jsp

Abhängigkeitsplan Der Voice-Browser benötigt ebenso wie der HTML-Browser die Integration-Komponente (Komponenten-ID 3).

Anforderungen an die Ausführungsumgebung Der Voice-Browser ist von Ein- und Ausgabegeräten für Sprache abhängig, die von einer ausführenden Plattform bereitgestellt werden müssen. Das wird durch den requirements-Teil des Kopplungsprofils verdeutlicht.

Implementierungsdetails Der Voice-Browser war eigentlich für den Einsatz auf dem PDA vorgesehen. Da aber keine frei verfügbaren Bibliotheken für Spracherkennung und -synthese gefunden werden konnten, die den restriktiven Anforderungen einer PDA-Laufzeitumgebung entsprechen, wurde entschieden, dass die Sprachfunktionalität nur simuliert wird. Daher basiert der Voice-Browser auf Ausgabe der Informationen durch ein Textfeld und Eingabe in ein anderes Textfeld. Die Eingabe kann nur sequentiell erfolgen und kann auch nicht zurückgenommen oder gelöscht werden, wie ein gesprochenes Wort nicht zurückgenommen werden kann. Im Ausgabetextfeld markiert ein Cursor das aktive Eingabeelement, dem eine getätigte Eingabe zugeordnet wird. Die Voice-Interpretation arbeitet grammatikbasiert, d. h. ein mit der Grammatik konformer Ausdruck wird zurückgeliefert, sobald die entsprechenden Eingabeereignisse erkannt wurden. Dieser Ausdruck ist die Grundlage für die Navigation bzw. das Füllen des Eingabefeldes, was stets auf Feldebene erfolgt und auf *event*-Ebene zwar möglich aber nicht konform mit der Interaktion bei einer Sprachanwendung ist.

Voice-Browser ohne Interpretation

Der Voice-Browser ohne Interpretation (o. I., Komponenten-ID 6) hat die gleiche Aufgabe wie der Voice-Browser mit integrierter Interpretation. Der Unterschied ist die fehlende Interpretation der rohen Sprachsignale, die von einer externen Komponente übernommen wird. Die Funktionsdefinition ist gleich und wird daher nicht wiederholt. Das vollständige Komponentenprofil der Voice-Browser-Komponente ohne integrierte Interpretation befindet sich in Abschnitt B.6.

Schnittstelle Die Schnittstelle in Bezug auf die Dialogdarstellung (`render`) und Ereignisverarbeitung (`reactEvent`) ist ebenfalls wie beim Voice-Browser mit Interpretation vorhanden. Zusätzlich hat die Schnittstelle des Voice-Browsers ohne Interpretation aber noch eine Methode zum Empfang der Interpretationsergebnisse von der Interpretation-Komponente.

`void resultReceived(String result)`
: Mittels dieser Methode empfängt der Voice-Browser Interpretationsergebnisse von der Interpretation-Komponente. Das Ergebnis in `result` ent-

spricht einem gültigen Wert der für die Interpretation-Komponente definierten Grammatik.

Abhängigkeitsplan Der Voice-Browser ohne Interpretation benötigt die Voice-Interpretation-Komponente (Komponenten-ID 7) und die Integration (Komponenten-ID 3).

Anforderungen an die Ausführungsumgebung Ebenso wie der Voice-Browser mit Interpretation ist der ohne Interpretation darauf angewiesen, dass die ausführende Plattform die Ein- und Ausgabe von Sprache ermöglicht.

Implementierungsdetails Für den Voice-Browser ohne Interpretation gilt im Großen und Ganzen das gleiche wie für den mit Interpretation. Der einzige Unterschied besteht darin, dass der Voice-Browser o. I. die erkannten Tastenereignisse in Form eines Datenstroms zur Voice-Interpretation sendet. Dieser besteht aus den Tastenereignissen des Eingabetextfeldes und, falls für 200ms keine Eingabe stattfand, aus Fülldaten in Form eines Leerzeichens. Somit kann es vorkommen, dass bei zu langem Zögern bei der Eingabe diese nicht erkannt wird, weil sich Leerzeichen zwischen die Buchstaben eingereiht haben und somit eine Übereinstimmung mit der Grammatik nicht erfolgt.

Interaktionssteuerung

Die Interaktionssteuerung (Komponenten-ID 4) umfasst verschiedene Rollen des MMI-F, da sie für die Kommunikation mit der Zielanwendung verantwortlich ist. Dazu empfängt die Interaktionssteuerung Ereignisse bzw. Nutzereingaben von der Integration-Komponente und verarbeitet diese entsprechend. Dabei wird in zwei verschiedene Ereignisarten unterschieden: Ereignisse wie Texteingaben, die in den anderen Teilen der Nutzerschnittstelle reflektiert werden müssen und Kommandos, die den Dialogzustand der Anwendung beeinflussen. Die Änderungen von Formularinhalten oder Navigationen innerhalb eines Dokuments wirken sich lediglich auf die Darstellung des aktuellen Dialogs aus, ohne den Dialog selbst zu verändern, sodass diese Ereignisse von der Interaktionssteuerung einfach zur Ausgabe weitergeleitet werden.
Im Gegensatz dazu gibt es Kommandos wie das Absenden eines Formulars oder das Anwählen eines Hyperlinks, die Einfluss auf den Dialogzustand der Anwendung haben. Diese Kommandos werden von der Interaktionssteuerung nicht an die übrigen Nutzerschnittstellenkomponenten weitergeleitet, sondern mit dem Ergebnis einer neuen Dialogbeschreibung ausgeführt, welche erst aufgeteilt und dann an die einzelnen Nutzerschnittstellenkomponenten ausgeliefert wird (ggf. mit Synchronisierung). Damit erfüllt die Interaktionssteuerung auch die MMI-F-Rolle Generation. Eine explizite Generation ist nur notwendig, wenn die Nutzerschnittstellenbeschreibung semantisch aufgeteilt werden muss und nicht schon

in einem Format vorliegt, das nur noch auf die Modalitäten aufgeteilt werden muss.
Aus der Funktionsdeklaration des Komponentenprofils der Interaktionssteuerung wird ersichtlich, dass alle Funktionen modalitätsunabhängig sind. Das Komponentenprofil der Interaktionssteuerungskomponente ist in Abschnitt B.4 dargestellt.

Schnittstelle Die Schnittstelle der Interaktionssteuerung verfügt über drei Methoden, mittels derer sie Ereignisse von der Interpretation empfangen kann. Diese unterscheiden sich in der Art der Auswirkung, wie in der Funktionsbeschreibung erläutert. Die Methoden sind im Detail:

`void reportEvent(String clientID, String event)`
: Mit dieser Methode wird ein Ereignis an die Integration-Komponente berichtet. Es besteht aus der `clientID`, die den berichtenden Client identifiziert und dem Ereignis `event`, welches aus den gleichen Teilen besteht, wie das vom Browser zu empfangende Ereignis der Methode `reactEvent`.

`void doGet(String clientID, String getEvent)`
: Mit dieser Methode wird ein HTTP GET Kommando an die Interaktionssteuerung übertragen, was den Abruf einer neuen Webseite auslöst. Der sendende Client identifiziert sich mit der `clientID`, die zum Ereignis `getEvent` gehörenden Informationen sind die `pageID` der aktuell angezeigten Seite, die `url` der angeforderten Seite und sonstige Optionen `options`.

`void doPut(String clientID, String putEvent)`
: Mit dieser Methode wird ein HTTP PUT Kommando an die Interaktionssteuerung übertragen, was den Abruf einer neuen Webseite mit Übertragung von Informationen (z. B. Formulardaten) zum Server auslöst. Der Inhalt des `putEvent`s ist der gleiche wie der des `getEvent`s, zusätzlich ist jedoch der zu übertragende Inhalt `content` vorhanden.

Abhängigkeitsplan Die Interaktionssteuerung benötigt keine weiteren Komponenten.

Anforderungen an die Ausführungsumgebung Die Interaktionssteuerung hat keine besonderen Anforderungen an die Ausführungsumgebung.

Implementierungsaspekte Die Interaktionssteuerung hat trotz ihrer rein unterstützenden Funktion eine Nutzerschnittstelle, in der die momentan registrierten Modalitäten aufgelistet werden. Nur die hier aufgeführten Modalitäten werden beim Ausliefern von Nutzerschnitstellenbeschreibungen berücksichtigt, was der Generation-Funktion der Interaktionssteuerung entspricht. Dies schließt die

Kontrolle über die Synchronisierung mit ein. Daher werden für jede Modalität die ermittelten Werte für Netzwerkübertragungs- und Verarbeitungsrate der UI-Komponenten, die Nachrichtengröße, die daraus resultierenden T_{mod} bzw. T_{ref} benötigt. Um die Werte aktuell zu halten, führt die Interaktionssteuerung alle 5 Sekunden zwei Pings mit jeder Modalität und unterschiedlichen Nachrichtengrößen durch. Auf diese Weise wird die Netzwerkübertragungsrate ermittelt (siehe dazu [Kne07]).
Die Registrierung der Modalitäten erfolgt nicht direkt bei der Interaktionssteuerung sondern indirekt über das Dienstverzeichnis. Die Interaktionssteuerung hat eine interne Modalitätenliste, die alle 5 Sekunden mit Hilfe des Dienstverzeichnisses aktualisiert wird, indem alle Dienste abgerufen werden, die zu UI-Komponenten gehören.

Integration mit und ohne Fusion

Die Integration-Komponente ist für die Zusammenführung (Fusion) von unterschiedlichen Eingabekanälen verantwortlich. Dafür können verschiedene Algorithmen angewendet werden, um die Eingaben der einzelnen Kanäle zu kombinieren. Im Prototyp kommen zwei verschiedene Integration-Komponenten zum Einsatz, wobei die eine keine Fusion (Komponenten-ID 3) und die andere eine einfache Kombination mehrerer Kanäle durchführt (Komponenten-ID 8). Das Ergebnis der jeweiligen Verarbeitung wird an die Interaktionssteuerung weitergeleitet. Daher erfüllen die Komponenten die gleiche Funktionsdefinition, wie in Abschnitt B.2 und Abschnitt B.3 dargestellt ist. Der Unterschied wird in der QoS-Beschreibung der Methoden sichtbar.

Schnittstelle Beide Integration-Komponenten verfügen über Methoden, über die Ereignisse berichtet werden können, sodass diese nach eventueller Integration an die Interaktionssteuerung übertragen und schließlich über die Nutzerschnittstellen reflektiert werden. Die Methoden der Integration-Komponenten entsprechen denen der Interaktionssteuerung mit dem Unterschied, dass die Integration-Methoden für die Rolle der Integration zuständig sind und somit durch die Beschreibung das QoS-Merkmal des Fusionstyps für jede Methode angeben, was im Kopplungsprofil-Teil der Komponentenprofile deutlich wird. Zudem haben die Integration-Komponenten die Interaktionssteuerung als benötigte Komponente.

Abhängigkeitsplan Die Integration-Komponenten benötigen, wie im Kopplungsprofil dargestellt, als Nachfolger eine Komponente mit der Komponenten-ID 4, womit die Interaktionssteuerung bezeichnet wird.

Anforderungen an die Ausführungsumgebung An die Ausführungsumgebung bestehen keine weiteren Anforderungen.

Implementierungsdetails Die Integration ohne Fusion hat die einzige Aufgabe, die ankommenden Ereignisse an die Interaktionssteuerung weiterzuleiten. Dies erfolgt direkt und ohne weitere Verarbeitung. Die Integration mit Fusion filtert die Ereignisse, die der Umschalt-Taste zuzuordnen sind (drücken, loslassen). Diese Information wird in einer lokalen Variable gespeichert. Je nach Wert der Variablen werden alle weiteren textuellen Ereignisse entsprechend in Großbuchstaben umgewandelt, sodass von einer einfachen Integration zweier Eingabekanäle gesprochen werden kann.

Voice-Interpretation

Die Voice-Interpretation (Komponenten-ID 7) ist für die Umsetzung von rohen Sprachsignalen in Text verantwortlich. Damit wird die Aufgabe der Interpretation auf die Voice-Interpretation und den Voice-Browser aufgeteilt. Die Interpretation interpretiert die eingehenden Signale syntaktisch und erzeugt daraus Text entsprechend einer definierten Grammatik, die z. B. in JSGF (Java Speech Grammar Specification) beschrieben ist. Der Voice-Browser verwendet den erkannten Text zum Füllen von Eingabefeldern (*event*- bzw. *form*-Ebene) oder Auslösen von Kommandos auf *page*-Ebene. Entsprechend dieser Funktionsaufteilung wird das Interpretationsergebnis von der Interpretation-Komponente an den Browser zurückgeliefert und von diesem dann an die Integration-Komponente geschickt. Das Komponentenprofil der Voice-Interpretation ist in Abschnitt B.7 dargestellt.

Schnittstelle Die Schnittstelle der Voice-Interpretation verfügt über zwei Methoden. Eine zum Empfang von uninterpretierten Sprachsignalen und eine zur Definition der Grammatik, die der Interpretation zu Grunde liegt. Daher ergeben sich die Methoden wie folgt:

`void reportEvent(String clienturl, String event)`
: Mit dieser Methode wird das Resultat der Voice-Erkennung an die Interpretation übertragen. Dabei ist `clienturl` die URL, an die ein mögliches Ergebnis zurückgeliefert werden soll. Gleichzeitig dient der Parameter auch zum Blocken anderer Voice-Browser, damit immer nur ein Voice-Browser eine Interpretation-Komponente verwendet. Der Parameter `event` beinhaltet das eigentliche Ereignis.

`void setGrammar(String grammar)`
: Diese Methode dient dem Definieren der Grammatik `grammar`, auf deren Basis die Interpretation erfolgt.

Abhängigkeitsplan Die Voice-Interpretation benötigt einen Voice-Browser (Komponenten-ID 6), der selbst keine Interpretation durchführen kann.

Anforderungen an die Ausführungsumgebung Die Voice-Interpretation hat keine besonderen Anforderungen an ihre Ausführungsumgebung.

Implementierungsdetails Wie beim Voice-Browser schon erwähnt wurde, arbeitet die Voice-Interpretation grammatikbasiert, d. h. es wird nach Eintreffen eines Ereignisses geprüft, ob die bereits erkannten Ereignisse zu der Grammatik passen. Ist dies der Fall, wird der jeweilige erkannte Ausdruck zurückgeliefert. Da die Voice-Interpretation die *Catch-up-Migration* unterstützt, erzeugt sie jeweils nach Erkennen eines Ausdrucks einen Checkpoint, der das letzte Ereignis im Verlauf der ankommenden Ereignisse angibt, welches zum Erkennen geführt hat. Bei der Migration wird dieser Checkpoint an die Datenquelle übergeben, die alle nachfolgenden Ereignisse erneut überträgt, um die migrierte Komponente auf den aktuellen Stand zu bringen.

5.3.3. Die Föderationsverwaltungsanwendung

Die Föderationsverwaltungsanwendung ermöglicht dem Nutzer eine feingranulare Verwaltung der Komponenten und Geräte in seiner Verwendung. Bevor der Nutzer verwaltende Aktionen durchführen kann, muss er sich anmelden. Danach steht ihm die gesamte Bandbreite der Funktionen zur Verfügung:

Installieren Mit der Installieren-Funktion bekommt der Nutzer zunächst eine Liste aller Komponenten angezeigt, die für das System zur Installation zur Verfügung stehen. Nachdem er eine beliebige Komponente ausgewählt hat, wählt er aus einer Liste von Geräten ein Gerät zur Installation aus. Die Komponente wird auf dem ausgewählten Gerät installiert und gestartet.

Deinstallieren Mit der Deinstallieren-Funktion werden dem Nutzer zunächst die installierten Komponenten auf einem vom Nutzer ausgewählten Gerät angezeigt. Nach Bestätigung der Auswahl von Gerät und Komponente wird diese deinstalliert.

Migration Mit der Migrations-Funktion wählt der Nutzer wie beim Deinstallieren eine Komponente eines Gerätes aus, die migriert werden soll. Anschließend wird ein Zielgerät ausgewählt und bestätigt, was das Stoppen der Komponente auf dem Quellgerät, die eventuelle Extraktion und den Transfer der Sitzungsinformationen zum ServiceDirectory und das Installieren der Komponente inkl. Sitzungsinformation auf dem Zielgerät zur Folge hat.

Austausch Zum Austauschen von Komponenten erstellt der Nutzer ein Austauschkonfigurationsdokument, das definiert, welche Dienste durch welche Komponenten ausgetauscht werden sollen. Dazu kann er über den Dialog IDs von Diensten und Komponenten dem Dokument hinzufügen. Nach

abschließender Bestätigung wird das Dokument an das ServiceDirectory gesendet, welches den Austausch kontrolliert.

Nutzereinstellungen Beim Anmelden des Nutzers wird ein Nutzerprofil mit Standardwerten angelegt. Die vom Nutzer definierbaren Optionen sind Platzierungszeitpunkt, Verteilungsstrategie, Synchronisierungsart und Fusionstyp. Außerdem kann die Synchronisierung auf *page*- und *event*-Ebene aktiviert und deaktiviert werden.

5.4. Prototypische Umsetzung der Integrationsschicht

In diesem Abschnitt wird die prototypische Umsetzung der Integrationsschicht im Allgemeinen und die Interaktionsanwendung im Speziellen beschrieben, wobei untersucht wird, inwiefern die in Abschnitt 5.2 erwähnten Testfälle realisiert werden.

5.4.1. Technische Eigenschaften der prototypischen Umsetzung

Hardware Der zum Testen verwendete PDA ist ein Qtek 9090 mit Windows Mobile 2003. Der Laptop ist ein IBM ThinkPad T41 mit Windows XP Professional. Sämtliche Kommunikation läuft über WebServices, daher ist für die prototypische Implementierung prinzipiell irrelevant, ob es sich um lokale oder Kommunikation im Netzwerk handelt. Die Performanzmessungen zu Unterschieden zwischen Aktionen verschiedener OSGi-Container werden auf einem Rechner durchgeführt, um Verfälschungen durch möglicherweise asynchrone Uhren der Geräte auszuschließen. Für einige Tests wurde WLAN 802.11b verwendet.

Software Die Java-Laufzeitumgebung auf dem PDA wird durch die JVM (Java Virtual Machine) von IBM namens J9 zur Verfügung gestellt, welche dem CDC Version 1.1 entspricht und somit zu großen Teilen kompatibel mit Java SE 1.4.2 ist. Auf dem Laptop kommt Java in der Version Java SE 1.4.2 bzw. kompatibel zum Einsatz. Auf beiden Geräten wird Knopflerfish Version 2.0.0 als OSGi-Laufzeitumgebung eingesetzt. Die Entwicklung der Komponenten bzw. Bundles erfolgte größtenteils mit Eclipse 3.3 und dem Knopflerfish OSGi IDE 1.0.16.

Minimale Systemkonfiguration Da alle Geräte möglichst die gleichen Voraussetzungen für die Komponenten bereitstellen sollen, ist die Definition eines kleinsten gemeinsamen Nenners in Bezug auf die Systemkonfiguration sinnvoll. Ein PDA besitzt die größten Einschränkungen und ist daher ausschlaggebend für

die minimale Systemkonfiguration. Die IBM J9 ist zwar teilweise kompatibel zu Java SE 1.4.2, dennoch fehlen Bibliotheken wie Swing. Daher sind alle Nutzerschnittstellenkomponenten als AWT-Anwendungen realisiert. Da WebServices zur Kommunikation zwischen Komponenten der Interaktionsanwendung und Verzeichnisdiensten verwendet werden sollen, wird eine entsprechende SOAP (Simple Object Access Protocol)-Engine benötigt. Die Gründe für den Einsatz von WebServices sind die prinzipielle plattformübergreifende Verfügbarkeit, leichtgewichtige Anforderungen an die Laufzeitumgebung und insbesondere die Dynamik und Flexibilität einer Service-orientierten Architektur, was mit Technologien wie Java RMI oder ähnlichen Mechanismen nur eingeschränkt möglich ist [SS07]. Als OSGi-Bundle gibt es hierfür die Bibliotheken Apache Axis[2] und kSOAP2[3]. Die Axis-Bibliothek ist jedoch für Java SE implementiert worden und nutzt Bibliotheken, die auf der IBM J9 nicht verfügbar sind wie z. B. `java.sql` und `java.beans`. Aus diesem Grund wird kSOAP2 als SOAP-Engine verwendet. Diese hat zwar wesentlich weniger Features als Axis (z. B. ist mit ksoap das Erzeugen einer WSDL-Datei nicht möglich), allerdings ist der Speicherplatz auch wesentlich geringer und es existieren keine wesentlichen Abhängigkeiten, die den Einsatz auf mobilen Endgeräten verhindern. Zur Verarbeitung von XML wird kXML2[4] eingesetzt, das einen leichtgewichtigen Zugriff auf XML-Dokumente ermöglicht.

Architektur einer Integrationsschicht-Instanz Wie in Abschnitt 4.3 spezifiziert wurde, bietet die Integrationsschicht verschiedene Basisdienste, die den Komponenten zur Verfügung stehen. Diese Basisdienste werden hauptsächlich von dem Bundle *BaseServices* realisiert. Dieses enthält für jeden der Verzeichnisdienste einen entsprechenden Client, über den auf das jeweilige Verzeichnis zugegriffen werden kann. Weiterhin gibt es das Bundle *CommonDatatypes*, welches in XML serialisierbare Klassen für die verschiedenen benötigten Datentypen wie z. B. `Component`, `Device`, `Service` und `User` beinhaltet. Diese beiden Bundles bilden die Grundlage für die Teilnahme der Plattform an der Integrationsschicht. In Abbildung 5.4 ist die Architektur einer Integrationsschicht-Instanz dargestellt. Die abgerundeten Rechtecke stellen OSGi-Bundles im technischen Sinne dar. Das *BaseServices*-Bundle besteht aus den ManagementClients wie in Abbildung 4.6 in Abschnitt 4.4 spezifiziert wurde. Die gestrichelt umrandeten Verzeichnisdienste sind nur auf einer Plattform vorhanden und realisieren den Zugriff auf die MySQL-Datenbank.
Je nach Verwendungszweck der Plattform gibt es weitere Bundles. Wird die Plattform als Endgerät eingesetzt, so ist das Bundle *FederationManagement* essentiell, welches dem Nutzer die Verwaltung der Föderation ermöglicht (siehe Unterabschnitt 5.3.3). Soll die Plattform jedoch als unterstützender Infrastruktur-Server

[2] http://ws.apache.org/axis/
[3] http://ksoap2.sourceforge.net/
[4] http://kxml.sourceforge.net/kxml2/

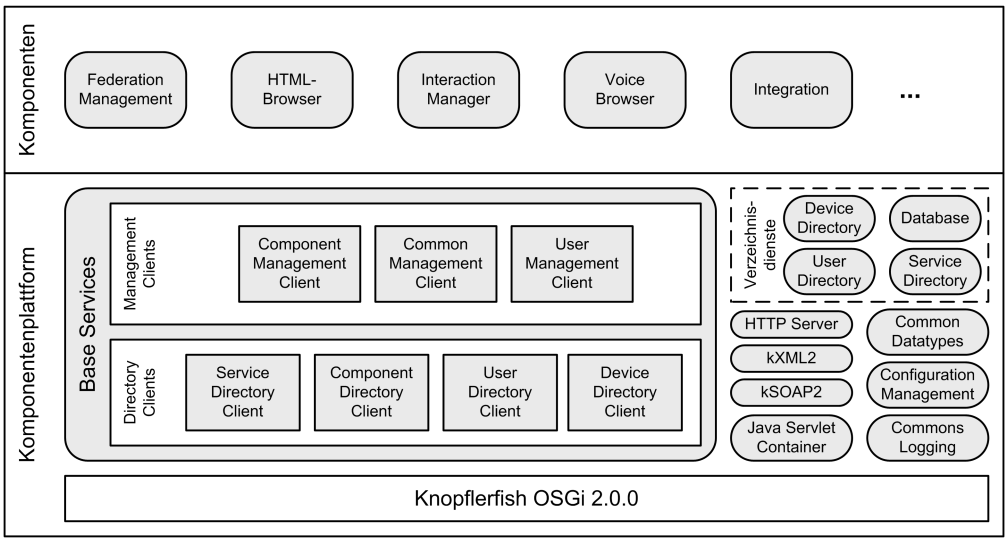

Abbildung 5.4.: Systemarchitektur der prototypischen Umsetzung

verwendet werden, so ist es möglich, dass diese die Verzeichnisdienste realisiert. Dafür werden die Bundles *Database*, *DeviceDirectory*, *UserDirectory* und *ServiceDirectory* benötigt. Das *Database*-Bundle kapselt den Zugriff auf die MySQL-Datenbank mittels des MySQL-Connectors 3.1.8. Die übrigen Bundles stellen ihre Schnittstellen als WebService bereit und nutzen das *Database*-Bundle zum Zugriff auf die MySQL-Datenbank, welche die in Abschnitt 4.4 spezifizierten Tabellen in einer Datenbank vereint vorhält.

Neben diesen selbstentwickelten Bundles werden von der OSGi-Plattform folgende Bibliotheken benötigt: `cm_all-2.0.0.jar` (Configuration Manager), `commons-logging_all-2.0.0.jar` (Logging), `http_all-2.0.0.jar` (HTTP-Server) und `jsdk-2.2.jar` (Java Servlet Container). Die von Knopflerfish bereitgestellten Bibliotheken für kSOAP2 und kXML2 wurden nicht direkt verwendet, sondern durch eigene Bundles ersetzt, die auf aktuelleren Versionen der beiden Bibliotheken basieren. Hierfür gibt es zwei wesentliche Gründe: Zum Einen verhindert das kSOAP2-Bundle die Verwendung von kXML2, da die Packages von kXML2 auch im kSOAP2-Bundle vorhanden sind, aber nicht exportiert und somit vom Zugriff ausgeschlossen werden. Der zweite Grund ist mindestes ein Bug im kSOAP2, der gelegentlich dazu führt, dass die Zuordnung der adressierten URL mit dem zugehörigen SoapObject und der darin definierten Methode verloren geht. Das resultiert darin, dass die Methode auf einem durch die URL definierten WebService aufgerufen werden soll, die diese Methode nicht implementiert. Das kSOAP2-Bundle entspricht der Version kSOAP 2.1.1. Das kXML-Bundle entspricht kXML2 Version 2.2.2 und XmlPull[5] Version 1.1.3.4c.

[5] http://www.xmlpull.org/

5.4.2. Untersuchung der Testfälle

In Abschnitt 5.2 wurden 20 Testfälle aus dem Anwendungsszenario extrahiert. In diesem Abschnitt werden diese auf ihren Erfüllungsgrad in der prototypischen Umsetzung untersucht. Da das Anwendungsszenario nutzergetrieben ausgerichtet ist, können nicht alle Konzepte und deren Variationen von diesem berücksichtigt werden. Der Grund hierfür ist der teilweise zu hohe Detailgrad, sodass die Variationen nicht durch Nutzereinstellungen kontrolliert werden können, sondern vielmehr einen Konfigurationsaspekt der Integrationsschicht repräsentieren. Daher wurden für die übrigen Konzeptvarianten Sonderfälle SF01 bis SF06 definiert, die trotz ihrer fehlenden Berücksichtigung im Anwendungsszenario relevante Funktionen der Integrationsschicht darstellen und somit untersucht werden müssen.

In Tabelle 5.2 sind die Testfälle dargestellt, die im ersten Teil des Wartungsszenarios auftreten. Die dabei verwendeten Konzepte sind vorwiegend die Platzierung von Komponenten, die entweder gezielt (TF03 und TF07) oder aber wahlfrei erfolgt, wobei die weiteren Faktoren bezüglich Platzierungszeitpunkt und -strategie berücksichtigt werden.

Der Platzierungsmechanismus wurde in Unterabschnitt 4.5.2 beschrieben. Die gezielte Platzierung erfolgt als Reaktion des Nutzers, der in der Föderationsverwaltungsanwendung den HTML-Browser bzw. den Voice-Browser auswählt und das gewünschte Gerät definiert, auf dem die jeweilige Komponente platziert werden soll. In beiden Fällen wird zur Anforderung der Komponente neben der obligatorischen Nutzer-ID die Komponenten-ID (1 bzw. 5) und die Geräte-ID zum ServiceDirectory übertragen, welches die Platzierung vornimmt.

Im Gegensatz dazu wird bei der wahlfreien Platzierung der Integrationskomponente keine Geräte-ID übertragen, sodass das ServiceDirectory anhand der Nutzer-ID das Nutzerprofil ermittelt und somit feststellt, dass der Nutzer Platzierung bei Bedarf und dezentrale Verteilung wünscht. Durch letzteres wird einfach das lokale Gerät zur Platzierung ausgewählt, welches u. a. die Verzeichnisdienste bereithält. Damit ist die Platzierung abgeschlossen, da alle weiteren Komponenten bei Bedarf platziert werden sollen.

Die wahlfreie Platzierung wurde vom HTML-Browser ausgelöst. Er überträgt erkannte Eingabeereignisse an eine Integration-Komponente. Zu Beginn hat er noch keine Information darüber, ob und wo sich eine solche Komponente befindet, weshalb er eine Anfrage nach Referenzen auf Komponenten mit der Komponenten-ID 3 an das ServiceDirectory stellt. Da es noch keine solche Referenz gibt, löst der HTML-Browser die wahlfreie Platzierung aus und stellt die Anfrage nach Referenzen von Integrations-Komponenten erneut. Nun bekommt der HTML-Browser eine gültige Referenz und nutzt diese zur Übermittlung der erkannten Nutzereingaben.

Die Testfälle des zweiten Abschnitts des Anwendungsfalles sind in Tabelle 5.3 dargestellt. Das Ergebnis der Testfälle TF08, TF09 und TF10 ist dabei der Austausch

TF-Nr.	Beschreibung	Zugehöriges Konzept
TF01	Anmelden des Nutzers	
TF02	Anzeigen der verfügbaren Komponenten (UI und sonstige)	
TF03	Platzieren des HTML-Browsers auf dem Laptop	Gezielte Platzierung
TF04	Komposition von HTML-Browser und Integration-Komponente durch Ermitteln der Referenz	
TF05	Automatisches Platzieren der Integration-Komponente	Wahlfreie Platzierung bei Bedarf und mit dezentraler Verteilung
TF06	Inhalt der Anwendung anfordern und auf dem HTML-Browser darstellen	
TF07	Platzieren des Voice-Browsers auf dem PDA	Gezielte Platzierung

Tabelle 5.2.: Testfälle des Anwendungsszenarios - Teil 1

des Voice-Browser mit integrierter Interpretation (m. I.) durch den Voice-Browser ohne integrierte Interpretation (o. I.) und einer externen Voice-Interpretation. Dabei kann dieses Ziel durch einen komplexen Austausch (1:2) oder durch einen einfachen Austausch mit Platzierung benötigter Komponenten erreicht werden. In den Testfällen wird die zweite Variante beschrieben.
Der Austausch erfolgt, nachdem der Nutzer das Austauschkonfigurationsdokument mit der Service-ID des Voice-Browsers m. I. und der Komponenten-ID des Voice-Browsers m. I. gefüllt und abgeschickt hat. Das ServiceDirectory erfüllt hierbei die Rolle des ReplacementCoordinators aus Abbildung 4.19. Dieser stoppt den Voice-Browser m. I. und extrahiert dessen Sitzungsinformation. Anschließend wird der Voice-Browser o. I. auf dem PDA installiert und mit den Sitzungsinformationen initialisiert. Dies entspricht einem normalen Installationsvorgang mit dem Unterschied, dass Sitzungsinformationen vorhanden sind. Da das Komponentenprofil des Voice-Browsers o. I. die Voice-Interpretation-Komponente als benötigten Port aufgeführt hat und der Nutzer seine Nutzereinstellungen auf Vorabinstallation und zentrale Verteilung geändert hat, wird die Voice-Interpretation-Komponente ebenfalls auf dem PDA installiert. Diese Komponente wird jedoch nicht initialisiert, da sie nicht Teil der Austauschkonfiguration war. Dies kann erreicht werden, wenn statt dem einfachen Austausch ein komplexer Austausch durchgeführt wird, d. h. der Austausch des Voice-Browsers m. I. durch den Voice-Browser o. I. und der Voice-Interpretation veranlasst wird.
Die Testfälle TF12, TF13 TF15 und TF18 (siehe Tabelle 5.4) befassen sich mit der Synchronisierung von Nutzerinteraktionen, wobei die verschiedenen Synchro-

TF-Nr.	Beschreibung	Zugehöriges Konzept
TF08	Austausch des Voice-Browsers mit integrierter Spracherkennung durch einen ohne integrierte Spracherkennung	einfacher Austausch
TF09	Installieren von benötigten Komponenten des neuen Voice-Browsers	Wahlfreie Platzierung und zentrale Verteilung, Vorabinstallation
TF10	Erzeugen von Text aus Sprache durch die Voice-Interpretation	
TF11	Migration der Voice-Interpretation vom PDA auf den Integrationsschicht-Server	Migration eines Datenstromempfängers

Tabelle 5.3.: Testfälle des Anwendungsszenarios - Teil 2

nisierungsebenen von *event*- und *page*-Ebene zum Einsatz kommen. Beide Ebenen werden jeweils asynchron und synchron verwendet. Asynchron bedeutet dabei eine schnellstmögliche Verarbeitung ohne künstliche Pausen. Einfacher Integration ist die Fusion einer gedrückten Umschalt-Taste mit den standardmäßig kleinbuchstabigen Spracherkennungsergebnissen. Weitere Fusionsmöglichkeiten sind nicht vorgesehen. Die weiteren Konzepte wie gezieltes Platzieren und Deinstallieren wurden bereits von früheren Testfällen abgedeckt. Schließlich meldet sich der Nutzer ab, wodurch die Integrationsschicht wieder in den Grundzustand versetzt wird.
In Tabelle 5.5 sind die bereits erwähnten Sonderfälle tabellarisch dargestellt. Diese haben keine explizite Berücksichtigung im Anwendungsfall. Dennoch finden sie in der prototypischen Umsetzung der Integrationsschicht teilweise Berücksichtigung. So ist beispielsweise die einfache Migration SF01 eine wesentliche Voraussetzung für komplexere Migrationen (SF02 und TF11). Die Sonderfälle SF03 und SF04 adressieren das Problem der Fehlerbehandlung, wenn ein Aufruf an eine Komponente während ihrer Migration gestellt wird. Da dies nur in sehr seltenen Fällen von langer Migrationsdauer und zwischenzeitlichen Aufrufen vorkommt und außerdem nicht im Anwendungsfall auftrat, wurde diese Funktionalität im aktuellen Prototyp nicht berücksichtigt. Die Unterscheidung der beiden Synchronisierungsausführungsorte Server und Client ist dem Nutzer über entsprechende Felder im Nutzerprofil möglich.
Eine detaillierte Beschreibung der Testfälle befindet sich in Anhang A.

5.4.3. Einschränkungen des Prototypen

Die gegenwärtige prototypische Implementierung ist verschiedenen Einschränkungen unterworfen. Dabei sind die Einschränkungen in der Regel darin motiviert, dass deren Behebung zwar für das gesamte System einen Mehrwert bringt, aber

TF-Nr.	Beschreibung	Zugehöriges Konzept
TF12	Navigation duch die Anwendung mit Synchronisierung auf *page*-Ebene	Synchronisierung auf *page*-Ebene
TF13	Ein- und Ausgabe von Formularinformationen mit Synchronisierung auf *event*-Ebene	Synchronisierung auf *event*-Ebene
TF14	Austausch der Integrationskomponente	einfacher Austausch
TF15	Einfache Integration von Eingaben und deren kontrollierte Repräsentation auf dem Laptop	kontrollierte Synchronisierung auf *event*-Ebene
TF16	Deinstallieren des Voice-Browser	
TF17	Gezieltes Platzieren des HTML-Browsers auf dem Beamer-PC	Gezieltes Platzieren
TF18	Umschalten von kontrollierter auf vorläufige Aktualisierung auf *event*-Ebene	Vorläufige Synchronisierung auf *event*-Ebene
TF19	Deinstallieren der HTML-Browser von Beamer-PC, PDA und Laptop	
TF20	Abmelden des Nutzers	

Tabelle 5.4.: Testfälle des Anwendungsszenarios - Teil 3

die Konzepte dadurch nicht besser oder schlechter getestet werden können oder aber der notwendige Aufwand zur Behebung enorm ist.

Eine der größten Einschränkungen wurde bereits bei der Beschreibung des Voice-Browsers erwähnt: die fehlende Unterstützung für tatsächliche Sprache. Für Spracherkennung und -synthese bieten sich die freiverfügbaren Bibliotheken Sphinx[6] und freeTTS[7] an. Beide stellen jedoch Voraussetzungen an ihre Laufzeitumgebung, die nur mit einem Laptop/PC und nicht mit einem PDA erfüllt werden können. Die Spracherkennung Sphinx benötigt beispielsweise über 96MB Heapgröße zum Laden der Engine. Auf dem PDA mit Windows Mobile 2003 bekommt die J9 maximal 32MB an Heap zugewiesen, größere Werte werden ignoriert. Dazu kommt, dass die Bibliotheken Java 5.0 kompatibel sind und von dieser Version auch verschiedene Funktionen (z. B. assertions) nutzen, die in Java 1.4 und somit in der J9 nicht verfügbar sind.

Die weiteren Einschränkungen beziehen sich auf Details der Implementierung. So ist z. B. die IP-Adresse für die Verzeichnisdienste statisch implementiert worden. Denkbar wäre ein Einsatz von Sun's Jini[8] zum dynamischen Finden des Hostrechners, auf dem die Verzeichnisdienste vorgehalten werden. Alternativ könnte

[6] http://www.speech.cs.cmu.edu/sphinx/
[7] freetts.sourceforge.net
[8] http://www.sun.com/software/jini/

SF-Nr.	Beschreibung & Zugehöriges Konzept
SF01	Migration einer nichtverwendeten Komponente zwischen zwei Plattformen
SF02	Migration einer verwendeten Komponente zwischen zwei Plattformen
SF03	Implizite Aufrufumleitung
SF04	Explizite Aufrufumleitung
SF05	Server-basierte Synchronisierungsausführung auf *page*-Ebene
SF06	Client-basierte Synchronisierungsausführung auf *page*-Ebene

Tabelle 5.5.: Sonderfälle

man diese Information auch in externen Konfigurationsdateien ablegen, deren Modifikation einfacher ist als eine Änderung am Quellcode.
Weiterhin wurden zwei Zwischenspeicher in der Integrationsschicht an Stellen eingebaut, wo eine unnötige Überlastung von Netzwerk und Komponenten befürchtet werden musste. Der erste Zwischenspeicher ist das UserProfile, welches durch die Föderationsanwendung geändert, durch das Nutzerverzeichnis gespeichert und von verschiedenen Komponenten in der Integrationsschicht verwendet wird. Damit nicht bei jeder Verwendung das Original aus dem Nutzerverzeichnis abgerufen wird, wurde eine Zeit von 5 Sekunden definiert, nach der die lokale Kopie ungültig und bei Bedarf vom Nutzerverzeichnis erneut abgerufen wird. Der zweite Zwischenspeicher wurde in der Interaktionssteuerung realisiert. Die Liste der verfügbaren Modalitäten wird bei jeder Interaktion benötigt (Textereignisse, Navigation), aber ebenfalls nur alle 5 Sekunden durch Anfragen beim Dienstverzeichnis aktualisiert.
Wie schon erwähnt wurde, sind Mechanismen zur Wahrung der Sicherheit auf Kommunikationsebene nicht implementiert worden. Auch auf Ebene der Komponentenverwaltung gibt es keine Sicherheitsmaßnahmen, die beispielsweise verhindern, dass ein nichtautorisierter Nutzer Änderungen an Komponenten, Geräten oder Profilen durchführt. Daher ist es möglich, dass mit modifizierten Föderationsverwaltungsanwendungen beliebige Änderungen am System durchgeführt werden können.
Die Synchronisierung der Nutzereingaben auf *event*-Ebene (z. B. Änderungen an Textfeldinhalten) benötigt eine Identifizierung der jeweiligen Felder. Im gegenwärtigen Prototyp geschieht das über den XPath-Ausdruck des jeweiligen Feldes im Dokument, weil dies bei der Konzeption des HTML-Browsers ausreichend war. Da andere Modalitäten andere Nutzerschnittstellenbeschreibungen haben können, sind semantisch zueinandergehörende Elemente nicht mehr über den XPath adressierbar. Als Lösung gibt es zwei Varianten: einerseits kann die originale Nutzerschnittstellenbeschreibung bereits mit eindeutigen IDs für solche Felder ausgestattet sein, andererseits kann die Interaktionssteuerung die Zuordnung von

XPath-Ausdruck zum jeweiligen Element in einer anderen Beschreibung durchführen. Das Konzept ist für die Zielanwendung transparent, bedeutet aber einigen Aufwand in Zusammenhang mit der eventuellen Entwicklung eines Adaptionsmechanismus für beliebige Nutzerschnittstellenbeschreibungen.

5.5. Performanzuntersuchung

In diesem Kapitel werden verschiedene Aspekte der Integrationsschicht auf ihre Performanz untersucht, was sich in der Regel auf die zeitliche Dauer bezieht, die für bestimmte Aktionen erforderlich ist. Aus den Ergebnissen werden Schlussfolgerungen in Bezug auf Nutzbarkeit und Nutzerfreundlichkeit gezogen und mögliche Verbesserungsansätze diskutiert.

5.5.1. Platzieren von Komponenten

Die Platzierung der Komponenten ist ein wichtiger Aspekt der Integrationsschicht, von welchem der Nutzer als erstes Gebrauch macht. Daher wird dies auch als Erstes näher untersucht. Dabei wird zwischen drei verschiedenen Arten unterschieden. Zunächst wird das normale Platzieren von Komponenten untersucht, ohne auf Abhängigkeiten zu anderen Komponenten einzugehen. Im Anschluss werden die Unterschiede im Platzierungszeitpunkt benötigter Komponenten anhand eines Beispiels untersucht.

Normales Platzieren

Beim normalen Platzieren wird eine neue Komponente angefordert wird, ohne Abhängigkeiten zu anderen Komponenten zu berücksichtigen. Die eigentliche Platzierung wird durch das Dienstverzeichnis koordiniert. Im Beispiel des HTML-Browsers ist der Auslöser wie bei jeder UI-Komponente die Föderationsverwaltungsanwendung. In Abbildung 5.5 ist der Ablauf bei der Platzierung als Sequenzdiagramm dargestellt. In der Abbildung wurde die Interaktion mit der eigentlichen Datenbank aus Platzgründen ausgelassen. Tatsächlich ist jede Informationsanforderung bei einem der Verzeichnisdienste mit einem Aufruf über WebServices an die Schnittstelle und von dieser an die MySQL-Datenbank verbunden.
Die Zeitmessung des normalen Platzierens wurde auf einem Rechner durchgeführt, um mögliche Differenzen der Systemuhren auszuschließen. Dennoch befanden sich auf dem Rechner zwei Knopflerfish-OSGi-Container, zwischen denen die Kommunikation stattfand, wie sie auch bei Verwendung von unterschiedlichen Geräten zum Einsatz kommt. Es wurde das 1.614.716 Byte große HTML-Browser-Bundle 10 mal auf der gleichen Plattform platziert, auf der auch die Föderationsverwaltungsanwendung vorhanden ist, wie in Abbildung 5.5 dargestellt.

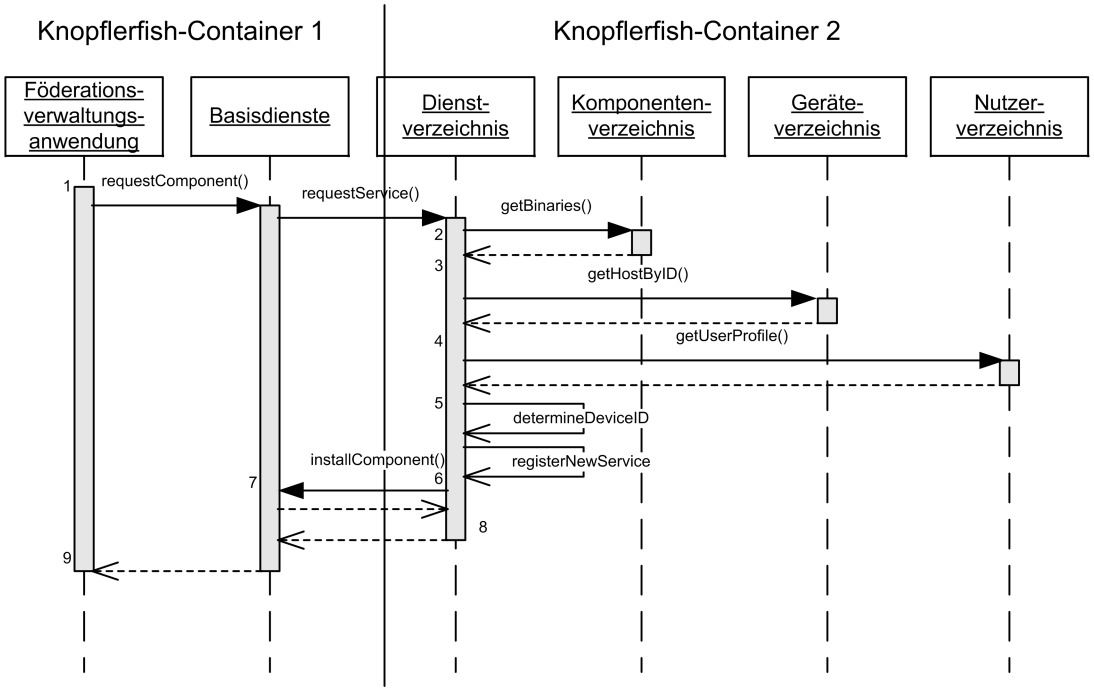

Abbildung 5.5.: Sequenzdiagramm der Platzierung einer Komponente ausgelöst durch die Föderationsverwaltungsanwendung

Abschnitt	1-2	2-3	3-4	4-5	5-6	6-7	7-8	8-9	Summe (1-9)
Mittelwert	6	872	12	12	61	1948	7716	2	10630

Tabelle 5.6.: Performanz der Platzierung des HTML-Browsers (in ms)

Vor jeder Messung wurden die Knopflerfish-OSGi-Container neu gestartet. In Abbildung 5.5 sind außerdem die Stellen eingetragen, an denen die Zeitstempel ermittelt wurden, auf deren Basis die Zeitdauer für die verschiedenen Phasen berechnet wurden.
In Tabelle 5.6 ist das Ergebnis der Untersuchung dargestellt. Der gesamte Vorgang der Bundle-Platzierung dauert durchschnittlich 10630ms. Hiervon entfallen die größten Anteile auf das Beschaffen der Binärdaten aus der Datenbank (872ms), die Übertragung der Binärdaten zum Client (1948ms) und das dortige Installieren und Starten des Bundles (7716ms).
Bemerkenswert ist der Zeitunterschied zwischen Beschaffen der Binärdaten aus der Datenbank und deren Übertragung zum Client. In beiden Fällen findet die Kommunikation über WebServices statt, bei ersterem ist sogar noch eine Datenbankabfrage notwendig. Der Grund für die etwa doppelt so große Übertragungszeit zum Client ist vermutlich die Tatsache, dass es sich hierbei um Kommunikation zwischen zwei Instanzen des Knopflerfish-OSGi-Frameworks und somit

zwei unterschiedlichen virtuellen Maschinen handelt, sodass in diesem Fall keine Optimierung von Seiten der JVM möglich ist.

Von der Zeitdauer für das Installieren (Phase 7-8) werden ca. 68% für das Installieren benötigt, während in der übrigen Zeit das Starten des Bundles erfolgt. Das Installieren beinhaltet die Deserialisierung des Strings in ein Bytearray, das Entpacken des Bundles auf Basis des Bytearrays durch das OSGi-Framework und der Speicherung des Inhalts im Framework-Verzeichnis.

Wie in Tabelle 5.6 deutlich wird, werden Aufrufe mit einem geringen Datenumfang im Bereich von weniger als 70ms Dauer abgearbeitet. Daher ergibt sich eine Optimierungsmöglichkeit für zukünftige Systeme. Mit der Realisierung von lokalen Caches auf den einzelnen Geräten lassen sich die Zeiten drastisch reduzieren, weil die Abfrage der Datenbank und das Übertragen des Bundles entfällt und somit ca. 2800ms gegen eine Anfrage von wenigen ms getauscht wird. Falls das Bundle schon installiert wäre und nur noch gestartet werden müsste, könnte die Gesamtdauer auf ca. 2700ms reduziert werden.

Platzierung bei Anwendungsinstallation

Beim Installieren bei Anwendungsinstallation werden alle von der zu platzierenden Komponente benötigten Komponenten ebenfalls platziert, ohne dass deren Vorhandensein zum Zeitpunkt der Installation benötigt wird. Im Gegensatz dazu wird beim Installieren bei Anfrage nur die direkt gewünschte Komponente installiert. Die benötigten Komponenten werden dann bei Anfrage an diese platziert.

Als Testbeispiel für das Installieren bei Anwendungsinstallation dient wiederum der HTML-Browser. Dieser hat als benötigte Komponente die Integration spezifiziert, welche wiederum die Interaktionssteuerung benötigt. Daher werden beim Installieren aller Abhängigkeiten für den HTML-Browser zwei weitere Komponenten (die Integration mit 11.424 Bytes, Interaktionssteuerung mit 869.175 Bytes) installiert, ehe der Nutzer mit dem HTML-Browser interagieren kann.

In Tabelle 5.7 sind die Messergebnisse der Ausführung des eben beschriebenen Testbeispiels dargestellt. Dabei werden für jede Komponente zwei Zeiten angegeben. Während *pre* die Zeit zur Vorbereitung der Installation bezeichnet (also Binärdaten beschaffen, Nutzerprofil abrufen etc.), wird mit *post* die Zeit für die eigentliche Installation angegeben. Es wurden wieder 10 Messungen durchgeführt. Die Werte des HTML-Browsers stimmen in etwa mit dem Ergebnis der Untersuchung der normalen Platzierung aus Unterabschnitt 4.5.2 überein.

Bemerkenswert bei dieser Performanzuntersuchung ist die Tatsache, dass die Zeiten für die Vorbereitung der Platzierung von Integration und Interaktionssteuerungen nicht signifikant niedriger sind, als die für den HTML-Browser, obwohl die Dateigrößen sich zumindest im Falle der Integration-Komponente signifikant unterscheiden. Die unterschiedliche Dateigröße spiegelt sich wie erwartet in der Zeit für die eigentliche Platzierung wieder.

Aktion	Browser		Integration		Interaktionsst.		Browser	Summe
	pre	post	pre	post	pre	post	bereit	
Mittelwert	961	9506	701	341	773	1215	268	13765

Tabelle 5.7.: Performanz der Platzierung des HTML-Browsers und aller benötigten Komponenten (in ms)

5.5.2. Migration

Wie in Unterabschnitt 4.5.3 erläutert wurde, besteht die Migration einer Komponente aus folgenden drei Phasen: Anhalten und Extrahieren der Sitzungsinformation auf der Quellplattform, Transfer und Starten inkl. Injizieren der Sitzungsinformation auf der Zielplattform. In der Implementierung wurde das Migrationskonzept so umgesetzt, dass die Binärdaten des Bundles nicht zwischen den Plattformen direkt ausgetauscht werden, sondern vom Dienstverzeichnis bereitgestellt werden und die Migration letztlich über das Dienstverzeichnis erfolgt. Sonst hätte eine gesonderte Speicherung der Binärdaten auf den Plattformen erfolgen müssen, da diese von einem bereits installierten Bundle nicht ermittelt werden können. Der genaue Ablauf ist in Abbildung 5.6 dargestellt, wobei im Sequenzdiagramm angenommen wird, dass eine lokale Komponente migriert werden soll. Falls die Migration einer entfernten Komponente durchgeführt werden soll, wird die Migrationsanfrage der Föderationsverwaltungsanwendung erst zu dem Basisdienst auf dem jeweiligen Gerät weitergeleitet.
In Abbildung 5.6 sind die verschiedenen Zeitpunkte durch Zahlen dargestellt, die für die Performanzmessung ermittelt werden. Bei der migrierenden Komponente handelt es sich um den Voice-Browser m. I., der bereits einen Dialogschritt abgerufen hat und somit über Sitzungsinformationen verfügt, die bei Migration berücksichtigt werden müssen. Die Komponente ist 24.870 Byte groß und verfügt im Testfall über 1020 Byte an Sitzungsdaten. Es wurden 10 Wiederholungen durchgeführt, wobei jeweils die JVMs und somit die Knopflerfish-OSGi-Container neu gestartet wurden, um Verfälschungen durch Optimierungen der JVM zu vermeiden. Da die Zeitmessungen auf einem Gerät einfacher sind, wurde diese Testreihe auf einem Gerät (einem Laptop) mit zwei OSGi-Containern durchgeführt.
Das Ergebnis der Performanzmessung der Voice-Browser-Migration ist in Tabelle 5.8 dargestellt. Der signifikante Anteil von über 80% der gesamten Migrationsdauer von 869ms ergibt sich dabei aus Installation, Starten und Initialisieren des Voice-Browsers (Abschnitt 5-6). Im Vergleich zur normalen Installation des Voice-Browsers, die in der gleichen Testreihe erfolgte und im Mittel 219ms benötigte, dauert die Installation mit Initialisierung ca. 3 mal länger. Der Grund hierfür wird in der XML-Verarbeitung der Sitzungsinformationen vermutet, wobei u. a. die Dialogbeschreibung erneut verarbeitet und für die Ausgabe aufbereitet wird. Bemerkenswert ist die kurze Dauer (0ms) des Abschnitts 4-5, bei dem die Binär-

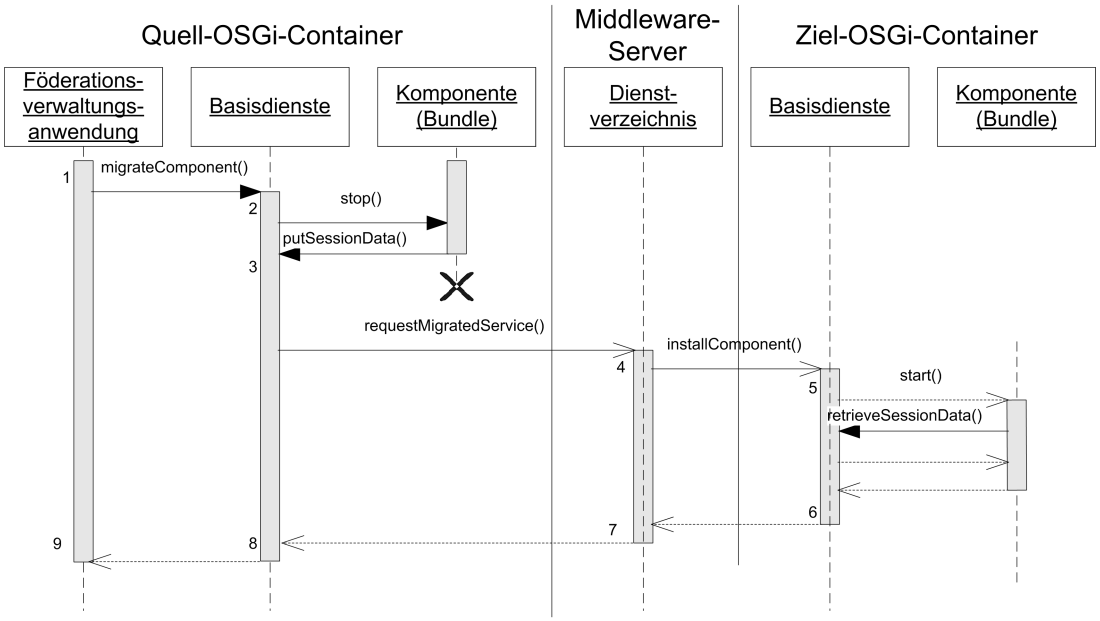

Abbildung 5.6.: Sequenzdiagramm der Migration ausgelöst durch die Föderationsverwaltungsanwendung

Abschnitt	1-2	2-3	3-4	4-5	5-6	6-7	7-8	8-9	Summe (1-9)
Mittelwert	1	93	51	0*	709	0*	15	0*	869

Tabelle 5.8.: Performanz der Migration des Voice-Browsers (in ms) (*Java hat eine minimale zeitliche Auflösung von 10ms, sodass darunterliegende Werte gerundet werden.)

und Sitzungsdaten zur Zielplattform übertragen werden. Der Grund hierfür liegt wahrscheinlich darin, dass es sich beim Test um dieselbe OSGi-Plattform handelte.

Die Migration des Voice-Browsers wurde zusätzlich zwischen dem PDA und dem Laptop getestet. Dabei wird zwischen der Migration vom Laptop zum PDA und vom PDA zum Laptop unterschieden. Die zum Ermitteln der Zeitdauern verwendeten Zeitpunkte sind die gleichen wie in Abbildung 5.6, jedoch werden diese auf PDA und Laptop verteilt ermittelt, sodass nicht mehr jeder nachfolgende Zeitpunkt mit dem vorherigen verrechnet werden kann. Daher werden für die Messungen der Inter-Gerät-Migration nur die Zeitdauern bestimmt, deren Anfangs- und Endzeitpunkt auf einem Gerät ermittelt wurden. Das sind Abschnitt 5-6 (Installieren und Starten des Bundles im Zielcontainer), Abschnitt 4-7 (wie Abschnitt 5-6 zuzüglich Übertragung der Binär- und Sitzungsdaten), Abschnitt 2-3 (Anhalten des Bundles und Extrahieren der Sitzungsdaten) und Abschnitt 1-9 (Gesamte Dauer der Migration). Bemerkenswert ist die ermittelte Verarbeitungs-

Abschnitt	2-3	4-7	5-6	1-9
Vom Laptop zum PDA	76	*4553*	3664	4661
Vom PDA zum Laptop	*629*	755	701	1807

Tabelle 5.9.: Performanz der Migration des Voice-Browsers zwischen Laptop und PDA (in ms)

zeit 0ms zwischen den Schritten 4-5, 6-7 und 8-9. Hierbei wurde von Java der gleiche Zeitstempel zurückgeliefert, trotzdem zwischenzeitlich eine Verarbeitung erfolgte. Die Java-Methode `System.currentTimeMillis()` hat jedoch in der verwendeten Version eine experimentell ermittelte minimale Auflösung von 10ms, sodass die tatsächlichen Werte von einer Rundung betroffen sind. Daraus ergeben sich Zeitdifferenzen von 0ms, auch wenn die eigentliche Differenz beispielsweise 3ms betrug.
Abhängig vom Ziel der Migration bewegen sich die ermittelten Zeitdauern in verschiedenen Größenordnungen, weshalb diese gesondert dargestellt werden. Es wurden 10 Messungen durchgeführt, je fünf pro Migrationsrichtung. Die übrigen Eckdaten wie Bundlegröße und Sitzungsdatengröße sind unverändert. In Tabelle 5.9 sind die ermittelten Zeitdauern dargestellt.
Aus den Differenzen zwischen 4-7 und 5-6 lassen sich jeweils die benötigten Übertragungszeiten zwischen Dienstverzeichnis und Zielplattform errechnen, welche bei der Übertragung vom Laptop zum PDA wie erwartet größer sind als innerhalb des Laptops. Die kursiv geschriebenen Zeiten sind auf dem PDA ermittelt worden und ebenfalls signifikant größer als die zugehörige Zeit für die äquivalente Aktion auf dem Laptop. Hierbei wird die niedrigere Leistungsfähigkeit des PDAs deutlich.

5.5.3. Synchronisierung

In diesem Abschnitt wird das Synchronisierungskonzept am Prototypen analysiert und untersucht, inwiefern es zur zeitlichen Abstimmung der Aktualisierungen geeignet ist. Dazu wurden ein HTML-Browser und ein Voice-Browser auf einem Laptop in einer OSGi-Plattform installiert. Deren gemeinsame Dialogbeschreibung wurde 10mal in Folge aktualisiert und jeweils der Zeitpunkt bestimmt, an dem die Ausgabe des Dialogs erfolgte. Der Voice-Browser wurde mit einer künstlichen Verzögerung aller ein- und ausgehenden Kommunikationen von 1500ms versehen, um die längere Dauer von Übertragungen über drahtlose, evtl. schmalbandige Netzwerke und geringere Verarbeitungskapazität des Endgeräts zu simulieren, da die Voice-Interaktion im Anwendungsfall von einem PDA bereitgestellt wird.
Der Prototyp ermöglicht es dem Nutzer, zwischen drei Strategien zur Synchronisierung auszuwählen: ohne besondere Behandlung, durch verzögerte Ausliefe-

Synchronisierungsstrategie	ohne	verzögerte Auslieferung	verzögerte Darstellung
Zeitdifferenz der Darstellung zwischen HTML- und Voice-Browser	1496	-39	15

Tabelle 5.10.: Gegenüberstellung der Performanzuntersuchung Varianten des Synchronisierungsalgorithmus (in ms)

rung und durch verzögerte Darstellung. Wie in Abschnitt 4.6 beschrieben wurde, berücksichtigt der Ansatz durch verzögerte Darstellung den aktuellen Systemzustand und sollte daher eine genauere Synchronisierung ermöglichen als die verzögerte Auslieferung.
In Tabelle 5.10 ist das Ergebnis der Performanzuntersuchung des Synchronisierungsalgorithmus dargestellt. Wie erwartet ergab sich ohne besondere Behandlung der Synchronisierung eine Zeitdifferenz von 1496ms, die der künstlich eingefügten Verzögerung nahezu entspricht. Bei verzögerter Auslieferung ergab sich eine Differenz von -39ms, d. h. der Voice-Browser begann im Mittel 39ms eher mit der Ausgabe als der HTML-Browser. Durch die Variante der verzögerten Darstellung konnte eine weitere Reduktion der Zeitdifferenz auf 15ms erreicht werden.

5.6. Vergleich der Konzeptvarianten

Wie in Abschnitt 4.7 erwähnt wurde, besitzen einige der Konzepte unterschiedliche Varianten der Realisierung, die über unterschiedliche Vor- und Nachteile verfügen. In diesem Kapitel werden die Konzepte untersucht und die Varianten verglichen, bei denen die Frage offen blieb, welche Variante besser geeignet ist.

5.6.1. Platzierungszeitpunkt

In Unterabschnitt 5.5.1 wurden zwei Messungen für die Komponentenplatzierung an sich (ca. 10,6s) und die Platzierung einschließlich benötigter Komponenten bei Anwendungsinstallation (ca. 13,7s) am Beispiel des HTML-Browsers durchgeführt. Da beim Installieren der Integration und Interaktionssteuerung im Großen und Ganzen die gleichen Schritte notwendig sind, wird die Nachinstallationszeit etwa 3135ms (Differenz aus 13765 und 10630) betragen. Eine geringfügig längere Zeit ist zu erwarten, da zusätzliche Aufrufe zur Ermittlung der Verfügbarkeit und ggf. Neuinstallation erforderlich sind. Hierbei handelt es sich aber um Aufrufe mit kleinem Umfang, sodass nicht zu erwarten ist, dass diese einen großen Einfluss auf die Nachinstallationszeit haben. Für den HTML-Browser bedeutet das, dass

die Vorabinstallation ca. 13,7s benötigt, während die Installation bei Bedarf ca. 10,6s plus ca. 3s bei erster Interaktion benötigt.
Der Unterschied zwischen der alleinigen Platzierung des HTML-Browsers und der gemeinsamen Installation aller benötigten Komponenten beträgt demnach ca. 3 Sekunden. Das entspricht einer ungefähr 30% längeren Dauer beim Vorabinstallieren, sodass der Einfluss auf die Gesamtdauer bis zur Verfügbarkeit des HTML-Browsers nur moderat beeinträchtigt wird. Wenn allerdings statt des HTML-Browsers der simulierte Voice-Browser (24.869 Bytes) oder eine andere, leichtgewichtige Komponente zur Interaktion ausgewählt wird, dann wird sich deren Installationszeit in ähnlichen Bereichen befinden wie die Integration und Interaktionssteuerung, sodass der Einfluss auf die gesamte Installationszeit eine ca. verdreifachte Zeit bewirkt.
Der zeitliche Einfluss auf die gesamte Installationsdauer hängt stark von der Installationsdauer der eigentlich gewünschten Komponente ab. Eine Verlängerung um 30% ist für den Nutzer weniger hinderlich als 200%, wobei das subjektive Empfinden auch von der absoluten Zeit abhängt. Darüber hinaus hängen die Zeiten auch von der Kapazität der jeweiligen Geräte in Bezug auf Netzwerkübertragungsrate, Prozessorleistung und Speichergeschwindigkeit ab, sodass ohne Untersuchung der Einflussfaktoren keine Aussage für andere Geräte abgeleitet werden kann. Allgemein lässt sich aber daraus schlussfolgern, dass bei geringem prozentualen Einfluss auf die gesamte Installationszeit die Vorabinstallation besser als die Bedarfsinstallation geeignet ist.

5.6.2. Verteilungsstrategie

Aus den Performanzuntersuchungen ergibt sich, dass Kommunikation über WebServices innerhalb einer JVM bzw. innerhalb eines OSGi-Containers schneller abläuft, als zwischen verschiedenen OSGi-Containern, auch wenn sie auf dem selben Gerät ausgeführt werden. Daher sind Kommunikationen zwischen verschiedenen OSGi-Containern als aufwändiger zu bewerten, als innerhalb eines OSGi-Containers, sodass Komponenten mit häufigem Nachrichtenaustausch im gleichen Container ausgeführt werden sollten. Das spricht für eine möglichst zentrale Platzierung, wobei der zu erwartende Kommunikationsaufwand geeignet modelliert und in die Platzierungsentscheidung einfließen sollte.
Gegen eine zentrale Platzierung sprechen der dadurch entstehende Single-Point-of-Failure und mögliche Kapazitätsengpässe des zentralen Geräts. Da jedoch im Rahmen der Validierung keine Lasttests durchgeführt wurden und auch die Gefahr der plötzlichen Nichtverfügbarkeit von Geräten nicht betrachtet wurde, kann diese Aussage nicht mit Erfahrungen der prototypischen Implementierung belegt werden.
Zusammenfassend lässt sich für die Verteilungsstrategie feststellen, dass keine der beiden Varianten pauschal die bessere ist. Es kommt in jedem Fall auf verschiedene Kriterien an, die sich z. T. erst zur Laufzeit ergeben (Kapazität der Geräte)

oder aber schon zum Entwicklungszeitpunkt festliegen (Interaktionsmenge zweier Komponenten). Hierfür sei erneut auf verschiedene Ansätze zur Verteilungsplanung verwiesen [AK06, MMRM05, SSD+04, WW05, Ank07].

5.6.3. Synchronisierungsstrategie

In Unterabschnitt 5.5.3 wurde eine Untersuchung der beiden Varianten zur Synchronisierungsausführung durchgeführt. Als Resultat ergibt sich, dass die Verzögerung durch den Client im Testfall besser als die Verzögerung durch den Server geeignet war, weil die Differenz zwischen den Zeitpunkten der Aktualisierung bei clientseitiger Ausführung niedriger (15ms) war als bei serverseitiger Synchronisierungsausführung (39ms).
In [Kne07] wurden u. a. die Synchronisierungsstrategien der kontrollierten und vorläufigen Aktualisierungen einer verteilten Nutzerschnittstelle untersucht. Dazu wurde eine Nutzerbefragung unter 13 Personen durchgeführt. Die Befragten sollten die Toleranzgrenzen für flüssiges und gerade noch akzeptables Feedback bei kontrollierter und vorläufiger *event*-Ebenen-Synchronisierung bestimmen. Daraus ergab sich, dass die Toleranzen für vorläufige Synchronisierung ca. 3mal so groß sind wie für die kontrollierte Synchronisierung. Der Grund hierfür ist vermutlich die Erwartung des Nutzers, von dem primär genutzten Gerät ein Feedback über seine Eingaben zu bekommen. Die übrigen Geräte sind von geringerer Priorität. Da die Toleranz in diesem Fall mit der vom Nutzer zum Wechsel der Geräte benötigten Zeit zusammenhängt, lässt sich schlussfolgern, dass die Synchronisierung auf *event*-Ebene am Besten vorläufig erfolgt.
In einer weiteren Nutzerbefragung in [Kne07] wurden 7 Personen aufgefordert, das Erscheinungsbild des Dialogaufbaus einer verteilten Nutzerschnittstelle zu beurteilen. Damit soll die Akzeptanz der Synchronisierung auf *page*-Ebene eingeschätzt werden. Von den 7 Personen meinten 6, dass der kontrollierte, also synchrone Dialogaufbau die Anwendung konsistenter erscheinen lässt. Daher ist auf *page*-Ebene eine kontrollierte Synchronisierung sinnvoll. Verschiedene Möglichkeiten zum weiteren Verfeinern der Synchronisierungsverfahren wurden in Unterabschnitt 4.6.6 beschrieben. In Anhang C sind die genauen Fragen und Antworten der Nutzerbefragungen dargestellt.
Weitere Schlussfolgerungen lassen sich aus den Nutzerbefragungen nicht ableiten. Mit weitergehenden Nutzerbefragungen kann z. B. die Steuerbarkeit der Anwendung analysiert werden oder inwiefern die Effektivität bei der Interaktion beeinflusst wird.

5.6.4. Sonstige Varianten

Die übrigen Konzeptvarianten der Platzierungsstrategie (gezielt, wahlfrei), Austausch (einfach, komplex), Synchronisierungsgranularität (*event*, *page*) und Migration (ohne Verwendung, mit Verwendung, Datenstromempfänger) können

nicht gegeneinander verglichen werden, weil sie keine Realisierungsalternativen für ein bestimmtes Ziel darstellen und teilweise unterschiedliche Voraussetzungen haben. Über die Varianten der Aufrufbehandlung während der Nichtverfügbarkeit einer Komponente aufgrund von Migration kann keine Aussage getroffen werden, da das Konzept noch nicht implementiert wurde.

5.7. Gegenüberstellung der Lösung mit den Anforderungen

In diesem Abschnitt werden die Anforderungen aus den Abschnitten 2.3, 2.4 und 2.5 den Konzepten und deren Umsetzung gegenübergestellt. Es wird eingeschätzt, inwiefern die entwickelten Lösungen zur Erfüllung der Anforderungen geeignet sind.

Die erste Anforderung an die Integrationsschicht (IA-Anforderung 1) beschreibt einen modularen Aufbau der Interaktionsanwendung, dessen Zweck u. a. die verteilte Ausführung auf verschiedenen Geräten ist. Die entwickelte Interaktionsanwendung besteht aus einzelnen Komponenten (u. a. TF02), die auf beliebigen Geräten der Integrationsschicht ausgeführt werden und miteinander kommunizieren, um die Funktionalität der Interaktionsanwendung bereitzustellen. *IA-Anforderung 1 ist erfüllt.*

IA-Anforderung 2 macht eine Verwaltung der Föderation erforderlich. Die Konzeption der Integrationsschicht sieht eine solche Verwaltungsanwendung vor, in dem sie für die Verwaltung notwendige Schnittstellen der Verzeichnisdienste bereithält. Im Prototyp wurde eine Föderationsverwaltungsanwendung entwickelt, die dem Nutzer eine detaillierte Verwaltung der Föderation, ihrer Komponenten und deren Zuordnung zu Geräten ermöglicht. So kann der Nutzer beliebige Komponenten platzieren, migrieren und austauschen lassen, was ohne entsprechende Verwaltungsanwendung nicht möglich wäre. *IA-Anforderung 2 ist erfüllt.*

IA-Anforderung 3 beschreibt die Notwendigkeit der Robustheit der Interaktionsanwendung gegenüber Ausfällen und daraus resultierender Nichtverfügbarkeit von einzelnen Komponenten. Da es auf diesem Gebiet vielfältige Arbeiten gibt, wurde die Konzeption der Robustheit auf die Migration von Komponenten beschränkt (TF11, SF01, SF02). Diese ermöglicht Komponenten in verschiedenen Situationen die Migration zwischen Plattformen, um so die Verfügbarkeit der Komponenten sicherzustellen. Zwei der drei konzipierte Migrationssituationen werden vom derzeitigen Prototypen unterstützt (Leerlauf und Datenstromempfänger). Die Migration während normaler Verwendung tritt selten auf und wurde daher bei der Implementierung nicht berücksichtigt. *Die IA-Anforderung 3 gilt daher als teilweise erfüllt.*

IA-Anforderung 4 adressiert die Sicherheitsaspekte der Interaktionsanwendung in Bezug auf Abhören und Modifikation der Kommunikationsinhalte zwischen den

Komponenten. Bei der Konzeption der Integrationsschicht wurde festgestellt, dass hierfür keine neuen Ansätze notwendig sind, sondern bestehende Lösungen wie beispielsweise HTTPS eingesetzt werden können. Die SOAP-Engine kSOAP2 bietet allerdings keine Unterstützung für HTTPS, sodass die Kommunikation im Prototypen ungesichert erfolgt. *IA-Anforderung 4 gilt daher als nicht erfüllt.* Da die Verfügbarkeit von sicherer Kommunikation keinen Einfluss auf die übrigen Konzepte hat, kann durch Verwendung einer anderen SOAP-Engine der Prototyp um diesen Aspekt ergänzt werden.

IA-Anforderung 5 beinhaltet die konsistente Repräsentation von Nutzereingaben in den verteilten Nutzerschnittstellen auf *event*-Ebene. Das bedeutet, dass jedes erkannte Ereignis an die anderen Nutzerschnittstellen übertragen werden und dort entsprechend reflektiert werden muss (TF13, TF15, TF18). Der Prototyp unterstützt diese Art der Ereignissynchronisierung, obwohl bei Verwendung des Voice-Browsers die Eingaben stets nur für ganze Felder erkannt werden. Das ist damit zu erklären, dass bei Sprachinterpretation gegen eine Grammatik das Ereignis mit der feinsten Granularität ein Wort der Grammatik ist, das gleichzeitig den Inhalt eines Feldes repräsentiert. Daher ist beim Voice-Browser die *event*-Ebene gleich der *form*-Ebene. *IA-Anforderung 5 ist erfüllt.*

IA-Anforderung 6 sieht einen Mechanismus zur synchronisierten Aktualisierung der verteilten Nutzerschnittstellen vor. Damit sollen Heterogenitäten von Netzwerktechnologien und Rechenkapazitäten der verwendeten Geräte ausgeglichen werden, sodass die Anwendung ein möglichst konsistentes Bild vermittelt. Das Synchronisierungskonzept in Abschnitt 4.6 realisiert dies und wurde auch erfolgreich implementiert (TF12, SF05, SF06). *IA-Anforderung 6 ist erfüllt.*

IA-Anforderung 7 sieht eine Zusammenarbeit von Zielanwendung mit Interaktionsanwendung vor, die ohne Modifikation der Zielanwendung erfolgen soll. Das bedeutet, dass die Interaktionssteuerung bzw. die Generation-Komponente aus jeder beliebigen UI-Beschreibung die entsprechenden Dokumente für die verteilten Nutzerschnittstellen unterschiedlicher Modalitäten erzeugen muss. Das kommt einer Adaption des Inhaltes gleich, wofür es bereits umfangreiche Arbeiten im Bereich multimodaler Anwendungen gibt. Die Schnittstelle zwischen Interaktions- und Zielanwendung ist jedoch lediglich HTTP, sodass für die Zielanwendung transparent ist, dass kein herkömmlicher Web-Browser die Inhalte abruft. Der Prototyp unterstützt zwar die Darstellung von beliebigen Web-Inhalten, allerdings nur durch den HTML-Browser. Eine Adaption beliebiger Inhalte für den Voice-Browser wurde nicht implementiert, könnte aber durch die Modularität für die anderen Komponenten transparent nachgerüstet werden. *IA-Anforderung 7 ist erfüllt.*

IA-Anforderung 8 beinhaltet die dynamische Änderung der vom Nutzer verwendeten Modalitäten. Das bedeutet, dass der Nutzer zur Laufzeit die Menge und Art der Modalitäten ändern, also weitere hinzunehmen oder existierende entfernen kann (TF07, TF16, TF19). Durch die Modularität und die Verwaltungsan-

wendung kann der Nutzer zu beliebigen Zeitpunkten beliebige UI-Komponenten platzieren und verwenden oder deinstallieren. *IA-Anforderung 8 ist erfüllt.*

KP-Anforderung 1 beinhaltet die Verteilung und verteilte Ausführung von Komponenten zur Laufzeit. Das Komponentenverzeichnis wird zur Ermittlung von Komponenteninformationen wie Funktionsbeschreibung und Binärdaten verwendet (TF03, TF05, TF07, TF17). In Zusammenarbeit mit dem Dienstverzeichnis und den Basisdiensten zur Komponentenverwaltung werden Komponenten verteilt und auch verteilt ausgeführt. Die Funktionalität ist im Prototyp umgesetzt. *KP-Anforderung 1 ist erfüllt.*

KP-Anforderung 2 adressiert die Migration von Komponenten zwischen Plattformen zur Aufrechterhaltung ihrer Verfügbarkeit. Der Nutzer kann im Prototyp die Migration über die Föderationsverwaltungsanwendung initiieren, wonach die Basisdienste zusammen mit dem Dienstverzeichnis die Migration durchführen (TF11, SF01, SF02). *KP-Anforderung 2 ist teilweise erfüllt.* Von drei vorgesehenen Migrationssituationen erfüllt der gegenwärtige Prototyp zwei.

KP-Anforderung 3 bezeichnet den Austausch von Komponenten zur Anpassung an geänderte Systembedingungen. Dabei können beliebig viele bestehende Komponenten durch beliebig viele neue ausgetauscht werden. Auslöser kann sowohl der Nutzer als auch die Integrationsschicht selbst sein. Im Prototyp kann der Nutzer mit der Föderationsverwaltungsanwendung manuell Austauschvorgänge durchführen, während Änderungen des Nutzerprofils automatisch von der Integrationsschicht als Anlass für Austauschvorgänge verwendet werden (TF08). *KP-Anforderung 3 ist erfüllt.*

Nach KP-Anforderung 4 muss die Integrationsschicht auf heterogenen Plattformen verfügbar sein. Durch die Verwendung von Bibliotheken, die entweder auf der IBM J9 verfügbar oder im Knopflerfish OSGi-Framework beinhaltet sind, ergibt sich keine Abhängigkeit an bestimmte Systembedingungen. Die minimale Systemkonfiguration ist kompatibel mit IBM J9 und ist somit auf den genannten Plattformen verfügbar. *KP-Anforderung 4 ist erfüllt.*

ZA-Anforderung 1 schreibt die Trennung von Nutzerschnittstelle und Geschäftslogik vor. Da die Interaktionsanwendung (Nutzerschnittstelle) von der Zielanwendung (Geschäftslogik) unabhängig ist und auch keine Auswirkungen auf diese hat, existiert eine klare Trennung zwischen den beiden Elementen. *ZA-Anforderung 1 ist erfüllt.*

Zusammenfassend kann festgestellt werden, dass ein Großteil der Anforderungen aus Kapitel 2 von der Konzeption und der prototypischen Implementierung erfüllt werden. Die Sicherheitsaspekte wurden bei der Umsetzung nicht berücksichtigt und die Migrationskonzepte wurden nur teilweise implementiert.

5.8. Gegenüberstellung der Lösung mit den Kernfragen

In diesem Abschnitt wird untersucht, ob und wenn ja, womit die Kernfragen der Arbeit aus Abschnitt 1.3 durch die entwickelte Lösung beantwortet werden können. Da die in Kapitel 2 aufgestellten Anforderungen teilweise auch aus den Kernfragen hervorgehen und die Anforderungen zum Großteil erfüllt wurden, bilden diese die Grundlage für die Beantwortung der Kernfragen.

In **Kernfrage 1** wird nach Konzepten im Protokollbereich gefragt, welche dem Nutzer eine gleichzeitige Verwendung von verschiedenen Endgeräten zum nutzerfreundlichen Zugriff auf dieselbe Anwendung ermöglichen. Dafür ist es notwendig, dass eine gemeinsame Infrastruktur bzw. Integrationsschicht benutzt wird, auf deren Basis eine modulare Interaktionsanwendung aufsetzt. Die Komponenten unterstehen an der Nutzerschnittstelle der Kontrolle einer zentralen Einheit, die diese verteilten Nutzerschnittstellen koordiniert. Im vorliegenden Prototyp ist dies die Interaktionssteuerung bzw. der Interaction Manager im MMI-F. Die Kommunikation zwischen den Komponenten der Interaktionsanwendung muss sicher erfolgen. Außerdem muss die Interaktionsanwendung dynamisch erweiterbar sein und transparent gegenüber der Zielanwendung auftreten.

Um die Nutzerfreundlichkeit sicherzustellen, sind einerseits Mechanismen zur zeitlichen und inhaltlichen Synchronisierung erforderlich (Abschnitt 4.6). Andererseits trägt die manuelle Föderationsverwaltung zur Nutzerfreundlichkeit bei, da der Nutzer somit die volle Kontrolle über das Verhalten der Föderation hat (Unterabschnitt 5.3.3). Für die Integrationsschicht wurden Konzepte vorgestellt, um die Elemente der Interaktionsanwendung mit hoher Verfügbarkeit bereitzuhalten, da ein Ausfall der Nutzerschnittstelle vom Nutzer direkter und schneller wahrgenommen wird, als ein Ausfall der Geschäftslogik.

In **Kernfrage 2** werden besondere Anforderungen an eine Komponentenplattform hinterfragt, die die angestrebte multimodale und modulare Interaktionsanwendung ausführen soll. Eine solche Komponentenplattform muss auf verschiedenen Plattformen verfügbar sein. Die Komponenten müssen dementsprechend so gestaltet sein, dass diese auf den heterogenen Plattformen ausgeführt werden können. Das wurde bei der Konzeption und Implementierung durch Verwendung des kleinsten gemeinsamen Nenners berücksichtigt (AWT, kSOAP2, kXML). Die dynamische Platzierung von Komponenten (Unterabschnitt 4.5.2) ist notwendig, damit verschiedene Geräte nach Wünschen des Nutzers oder zur Unterstützung in die Föderation einbezogen werden können. Es ist zwar auch möglich, die Komponenten statisch vorab zu verteilen, jedoch ist dann die Flexibilität des Gerätezusammenschlusses eingeschränkt, da neue Komponenten nicht verwendet und Aktualisierungen bestehender Komponenten nicht durchgeführt werden können. Die Plattformen müssen außerdem zur Verfügbarkeitserhöhung beitragen, indem Komponenten beispielsweise migrieren können (Unterabschnitt 4.5.3). Weiterhin

muss die Komponentenplattform den Austausch von Komponenten ermöglichen (Unterabschnitt 4.5.4), um sich an die Bedürfnisse des Nutzers in Bezug auf QoS-Eigenschaften der Komponenten anzupassen. Diese Konzepte werden von der prototypischen Implementierung umgesetzt.
Durch **Kernfrage 3** soll ermittelt werden, welche Voraussetzungen an eine Anwendung gestellt werden, damit diese von der Interaktionsanwendung und damit der Integrationsschicht zum multimodalen Zugriff genutzt werden kann. Eine solche Anwendung muss ein Mindestmaß an Modularität aufweisen. Die Geschäftslogik und die Nutzerschnittstelle wurden voneinander getrennt, damit die Geschäftslogik eine explizite Beschreibung der Nutzerschnittstelle erzeugen kann. Das beinhaltet eine gewisse Transparenz der Nutzerschnittstelle gegenüber der Geschäftslogik, damit die Geschäftslogik unabhängig von der Verarbeitung der Nutzerschnittstelle ausgeführt wird.

5.9. Zusammenfassung der Validierung

In der Validierung wurde das am Anfang der Arbeit in Kapitel 2 eingeführte Anwendungsszenario erneut erläutert, jedoch mit einer detaillierten Berücksichtigung der in Kapitel 4 entwickelten Konzepte. Die Berücksichtigung erfolgte durch Definition von 20 Testfällen, die im Anwendungsszenario vorhanden sind und gegen die prototypische Implementierung abgeprüft wurden. Die Testfälle betreffen dabei teilweise Basisfunktionalität des Prototypen aber beinhalten insbesondere auch die Unterstützung der entwickelten Konzepte. Da nicht alle Konzepte im Prototypen Anwendung finden, wurden für die übrigen Konzepte 6 Sonderfälle definiert. Zusammenfassend ist festzustellen, dass die prototypische Implementierung einen Großteil der Konzepte unterstützt und für die Umsetzung des Anwendungsszenario eingesetzt werden kann.
Daraus folgt, dass die entwickelten Konzepte geeignet sind, unabhängige und heterogene Endgeräte zur Realisierung einer verteilten Nutzerschnittstelle zu verwenden. Die durch die Heterogenität der Endgeräte implizit vorhandenen unterschiedlichen Modalitäten der einzelnen Nutzerschnittstellenteile können zum Vorteil des Nutzers eingesetzt werden, um multimodalen Zugriff auf die gewünschte Anwendung zu gewähren. Das spiegelt sich auch in der Gegenüberstellung der Lösung mit den Anforderungen und Kernfragen der Arbeit wider, wobei ein Großteil der Anforderungen erfüllt wurden. Dennoch werden die Kernfragen nach notwendigen Erweiterungen im Protokollbereich, Anforderungen an Komponentenplattformen und Voraussetzungen der Zielanwendung durch die entwickelten Konzepte beantwortet.

Alle Vermehrung unseres Wissens endet nicht mit einem Schlusspunkt, sondern mit Fragezeichen.

Hermann Hesse

6 Zusammenfassung und Ausblick

6.1. Zusammenfassung der Arbeit

Benutzer von Computersystemen sind vielfältigen Aufgaben ausgesetzt, für die die Interaktionsmechanismen verschiedener Geräte je nach Aufgabe besser oder schlechter geeignet sind. Trotz der schnellen Entwicklung von technischen Geräten und Kommunikationselektronik in den letzten Jahren existieren viele Geräte wie Laptops, PCs, PDAs und Mobiltelefone meist isoliert voneinander. Der Grund hierfür sind heterogene Kommunikationstechnologien, Programmierplattformen und unterschiedliche Herstellerinteressen. Durch die fehlenden Möglichkeiten zur Integration der Geräte wird die gemeinsame Nutzung der Fähigkeiten und Eigenschaften verhindert. Dabei ist es aber oft hilfreich, wenn der Nutzer für die Erledigung bestimmter Aufgaben flexibel zwischen Geräten und Modalitäten entsprechend seiner Präferenzen wechseln kann. Ein solcher Zusammenschluss von Geräten zum gemeinsamen Erreichen eines Ziels wird Geräteföderation genannt. Daher wurde in der Arbeit eine Integrationsschicht mit dem Ziel entwickelt, die durch die Heterogenitäten verfügbaren unterschiedlichen Modalitäten für den Nutzer verwendbar zu machen, sodass dieser von unterschiedlichen Geräten durch multimodalen Anwendungszugriff profitiert. Die Integrationsschicht sollte komponentenbasiert arbeiten und verteilt auf den verschiedenen heterogenen Geräten verfügbar sein. Daraus ergaben sich die folgenden Kernfragen, die durch die Arbeit beantwortet werden sollten:

Kernfrage 1 Welche Konzepte sind auf Protokollebene notwendig, damit dem Nutzer eine gleichzeitige Verwendung unterschiedlicher, mobiler und sta-

tionärer Endgeräte zum Zugriff auf dieselbe Anwendung auf eine nutzerfreundliche Art und Weise ermöglicht wird?

Kernfrage 2 Welche Konzepte muss eine Komponentenplattform unterstützen, wenn die Funktionalität der darauf aufsetzenden Anwendung zur Interaktion in einzelne Komponenten unterteilt wird? Insbesondere ist dabei die Bereitstellung von multimodalen Nutzerschnittstellen durch einzelne, z. T. mobile Endgeräte zu berücksichtigen.

Kernfrage 3 Welche Anforderungen muss eine Anwendung erfüllen, damit die Benutzung föderierter Endgeräte nicht explizit unterstützt werden muss? Wie muss die Integrationsschicht bzw. die Anwendung beschaffen sein, damit für die Anwendung die tatsächliche Art der Nutzerschnittstelle transparent ist, sodass föderierte Endgeräte ohne Einfluss auf die Anwendung oder deren Entwicklungsprozess unterstützt werden können?

Auf Basis dieser Kernfragen und den zur Veranschaulichung aufgestellten Anwendungsfällen wurden Anforderungen definiert, die von der zu entwickelnden Integrationsschicht erfüllt werden müssen. Aus Kernfrage 1 lässt sich ableiten, dass die Interaktionsanwendung streng modular aufgebaut sein muss, eine Synchronisierung zwischen den Teilen der Nutzerschnittstelle notwendig ist und die Komponentenplattform die Mobilität von Code zur Verteilung und Migration von Komponenten ermöglichen muss. Durch eine manuelle Verwaltung der Föderation hat der Nutzer maximalen Einfluss auf deren Konfiguration. Die Analyse von verwandten Arbeiten ergab, dass die hier definierten Anforderungen nur teilweise bis gar nicht berücksichtigt werden.
Die aufgestellten Anforderungen dienen daher als Grundlage für die Konzeption der Integrationsschicht. Für die Umsetzung der Modularität eignet sich ein komponentenorientierter Ansatz. Daher erfolge die Spezifikation der Integrationsschicht unter Berücksichtigung von Teilaspekten wie dem Komponentenmodell, der Komponentenplattform, der Verzeichnisdienste und der notwendigen Konzepte zur Unterstützung der dynamischen Eigenschaften. Die Verzeichnisdienste wurden für die Registrierung von Nutzern, Geräten, Komponenten und deren angebotenen Diensten spezifiziert. Sie sind für die Koordinierung der Kommunikation und Kooperation der Komponenten und Komponentenplattformen notwendig. Das Nutzerverzeichnis ist neben der Registrierung von Nutzern am System für die Verwaltung der Nutzerprofile zuständig, die für verschiedene Teilaspekte der Integrationsschicht abgerufen und berücksichtigt werden. Das Geräteverzeichnis beinhaltet alle Geräte und deren Profile, die für eine Föderation verfügbar sind. Im Komponentenverzeichnis werden die Komponenten in Form von Binärdaten bereitgehalten, die ebenso wie Nutzer und Geräte über ein Profil verfügen, das die Komponente und ihre Funktion charakterisiert. Wenn eine Komponente instanziiert wird, erfolgt ein Eintrag im Dienstverzeichnis, sodass diese Komponente von anderen gefunden und genutzt werden kann.

Die Komponentenplattform muss Funktionen wie *Kopplung* und *Platzierung* von Komponenten bereitstellen. Die Kopplung von Komponenten wurde auf Basis der Choreographie spezifiziert, wobei die Dienste über ihre funktionale Beschreibung gesucht werden. Durch die Platzierung werden die Komponenten auf Plattformen verteilt und dort ausgeführt, wobei der Vorgang der Platzierung durch die Platzierungsstrategie (gezielt vs. wahlfrei), Platzierungszeitpunkt (bei Installation vs. Anfrage) und die Verteilungsstrategie (zentral vs. dezentral) charakterisiert ist. Weiterhin wurden noch Konzepte für die *Migration* und den *Austausch* von Komponenten entwickelt. Die Migration von datenstromempfangenden Komponenten wurde ebenso berücksichtigt, wie der Austausch von beliebig vielen durch beliebig viele Komponenten. Durch diese vier Konzepte wurde die Dynamik des Systems in Bezug auf Verteilung und Kooperation von Komponenten unter Berücksichtigung von Ausfallsicherheit und Adaption an dynamische Umgebungseigenschaften erreicht. Sie stellen die von einer Komponentenplattform zu unterstützenden Konzepte dar, welche durch Kernfrage 2 ermittelt werden sollten.
Die Konzeption der Integrationsschicht ist auf eine Zielanwendung ausgerichtet, die ihre Nutzerschnittstellenbeschreibung in einer expliziten Form beschreibt. Somit ergibt sich eine klare Trennung zwischen Geschäftslogik und Nutzerschnittstelle, wie es in Unterabschnitt 2.5.1 beschrieben wurde. Die explizite Nutzerschnittstellenbeschreibung muss eine Zielanwendung erfüllen, damit sie durch die Integrationsschicht verwendet werden kann (Kernfrage 3). Die Integrationsschicht erzeugt aus dieser Nutzerschnittstellenbeschreibung die entsprechenden Dokumente für die verteilten Nutzerschnittstellen, ohne auf die Zielanwendung zurückgreifen zu müssen.
Zusätzlich zu den für den Nutzer verborgenen Konzepten wurde eine Synchronisierung der verteilten Nutzerschnittstellen auf sowohl *event*- als auch *page*-Ebene konzipiert. Beide Arten können sowohl kontrolliert (d. h. möglichst gleichzeitig) oder so schnell wie möglich (d. h. vorläufig auf *event*-Ebene) erfolgen.
Zur Implementierung der Konzepte wurde ein Prototyp mit OSGi als Komponentenplattform und WebServices als Kommunikationstechnologie implementiert, wobei ein Großteil der entwickelten Konzepte berücksichtigt wurde. Mit Hilfe der Integrationsschicht konnte das Szenario des eingangs geschilderten Wartungsarbeiters erfolgreich umgesetzt werden, was die Notwendigkeit der entwickelten Konzepte belegt. Für einige Mechanismen der Integrationsschicht wurden Performanzuntersuchungen durchgeführt, welche verschiedene Möglichkeiten zur Verbesserung des Gesamtsystems aufzeigen. Beispielsweise die Dauer der Übertragung der Komponentenbinärdaten sowie deren Verarbeitung auf den Endgeräten ist nicht vernachlässigbar, was durch geeignete Caching-Mechanismen reduziert werden kann.
Eine Nutzerbefragung zu den Ansätzen der Synchronisierung ergab, dass die tolerablen Zeiten für kontrollierte Synchronisierung auf *event*-Ebene nur ein Drittel der Zeit für vorläufige Synchronisierung beträgt, weshalb die vorläufige Variante bevorzugt wird. Auf *page*-Ebene ergab eine zweite Nutzerbefragung, dass die

synchronisierte Aktualisierung der Nutzerschnittstellenteile die Darstellung der Anwendung konsistenter erscheinen lässt.
Zusammenfassend lässt sich feststellen, dass die entwickelte Integrationsschicht inkl. der Konzepte, Komponenten und Schnittstellen für die Bereitstellung von multimodalem Anwendungszugriff durch föderierte Endgeräte für den Nutzer ermöglicht. Die entwickelten Konzepte unterstützen dabei die dynamischen Aspekte der Integrationsschicht, die in einem solchen Umfeld eine Rolle spielen. Die Arbeit beantwortet die Forschungsfragen nach benötigten Konzepten, die im Protokollbereich notwendig sind und von einer Komponentenplattform bereitgestellt werden müssen. Außerdem wurde festgestellt, dass die dafür geeigneten Anwendungen eine explizite Nutzerschnittstelle bereitstellen müssen. Der Prototyp dient als Beispielimplementierung einer Integrationsschicht für die Geräteföderation zwecks multimodaler Interaktion und kann als Grundlage für weitere Entwicklungen in diesem Forschungsgebiet dienen.

6.2. Ausblick auf zukünftige Arbeiten

Aus den erreichten Ergebnissen der Arbeit lassen sich einige weitere Fragestellungen und Forschungsthemen ableiten.
Die W3C Multimodal Interaction Activity hat u. a. verschiedene Möglichkeiten der Kombination unterschiedlicher Modalitäten vorgesehen. Sequentielle multimodale Eingaben erfolgen nacheinander, während simultane Eingaben gleichzeitig getätigt werden und zusammengesetzte Eingaben zusätzlich noch integriert werden. Bei Verwendung von verschiedenen Geräten entstehen grundsätzlich größere Verzögerungszeiten als bei Verarbeitung der Informationen auf einem Gerät, sodass sich die Frage stellt, ob diese Kombinationsmöglichkeiten von Modalitäten überhaupt nutzerfreundlich und auf breiter Basis unterstützt werden können. Insbesondere die vorläufige Synchronisierung auf *event*-Ebene hat das Problem, dass erfolgte Integrationen der Eingaben die Nutzerschnittstelle nachträglich ändern können, was zur Verwirrung des Nutzers führen kann.
Obwohl das entwickelte System nicht auf Ein-Nutzer-Betrieb beschränkt ist, wurden keine Untersuchungen mit mehreren gleichzeitigen Nutzern durchgeführt. Eine Umsetzung des Systems für viele Nutzer wirft die Frage auf, ob es notwendig, erforderlich oder gewünscht ist, dass ein Nutzer durch die Verwendung von Modalitäten eines Geräts das ganze Gerät blockiert, oder ob die verschiedenen Modalitäten eines Gerätes gleichzeitig von unterschiedlichen Nutzern verwendet werden können. Da dies möglicherweise nicht nur von der Modalität selbst abhängt, sondern auch vom Inhalt der Anwendung oder Anforderungen des Nutzers in Bezug auf die Anwendung oder Modalitäten, ist zu vermuten, dass eine Vielzahl von Kriterien bei der Entscheidung über geteilt verwendete Geräte beachtet werden müssen. Des Weiteren wurde für die Arbeit angenommen, dass ein Nutzer immer mit genau einer Anwendung über eine Föderation interagiert. Hierbei stellt sich

die Frage, inwiefern die Möglichkeit mehrerer Föderationen den Nutzer bei seiner Arbeit noch mehr unterstützen kann. Dabei ist zu untersuchen, ob und ggf. wie Föderationen aufgeteilt und fusioniert werden können und welche Voraussetzungen zu einer sinnvollen Entscheidung darüber die Integrationsschicht benötigt, damit keine beliebigen Teilungen und Fusionen durchgeführt werden.
Bei der Konzeption wurde vereinfachend angenommen, dass jede Komponentenplattform mit jeder anderen kommunizieren kann. Gerade bei heterogenen Geräten können auch heterogene Netzwerktechnologien vorliegen, die eine direkte Kommunikation zweier Komponentenplattformen verhindern. Insbesondere wenn dies jedoch erforderlich oder gewünscht ist (z. B. zwecks Streaming und Interpretation von Audiosignalen), erzeugt eine indirekte Kommunikation z. B. über einen Proxy eine unnötige Belastung des Systems. Das kann durch eine vorausschauende Planung der Komponentenverteilung vermieden werden. Dafür muss das System um die Modellierung solcher Abhängigkeiten erweitert werden.
Im mobilen Umfeld kann es vorkommen, dass die Zahl an verfügbaren (d. h. erreichbaren) Komponentenplattformen großen Schwankungen unterworfen ist. Durch Ansätze zur Beobachtung und Analyse der erreichbaren Komponentenplattformen könnte verhindert werden, dass die Verfügbarkeit einzelner, verteilt platzierter Komponenten trotz der Dynamik erhalten bleibt, wobei u. U. die Migration der Komponenten verwendet werden kann. Wenn es sich jedoch um ein sehr starkes dynamisches Verhalten handelt, kann der Fall eintreten, dass das System mit Migrationen überlastet wird und ein normaler, wenn auch eingeschränkter Nutzerbetrieb nicht mehr möglich ist. Im Zuge einer aktiven Beobachtung der Geräteumgebung könnte auch eine Unterstützung von komplexen Aktionen wie beispielsweise der Verteilungsadaption der vom Nutzer verwendeten Komponenten nach bestimmten Mustern erfolgen, sodass einfach zwischen einer Föderation nach peer-to-peer-Schema und server-unterstütztem Schema gewechselt werden kann.
Aus Sicht der Mensch-Maschine-Interaktion können verschiedene, naturgemäß weniger technische Aspekte weitergehend beleuchtet werden, die in der Arbeit bewusst ausgeklammert wurden. Beispielsweise ist interessant, welchen Einfluss eine verteilte Nutzerschnittstelle auf die Wahrnehmung durch den Nutzer hat. Wieviele Nutzerschnittstellenteile sind von einem Nutzer bedienbar? Was gibt es bei der Entwicklung der einzelnen Teile zu beachten, damit sie möglichst ohne große Probleme verwendet werden können? Hierbei spielen auch psychologische Aspekte des Entwurfs von Nutzerschnittstellen eine wichtige Rolle.
Das in der Arbeit entwickelte System hat die Auswirkungen von Eingabefehlern nicht betrachtet. Hierbei stellt sich die Frage, ob und ggf. wie dem Nutzer ein Fehler bei Interaktion mit einer Modalität in den anderen reflektiert wird. Hieraus ergibt sich ein Querbezug zu Systemen, die statt der strikten Trennung von Nutzerschnittelle und Geschäftslogik auch weniger strikte Verteilungen erlauben, wie es z. B. mit AJAX möglich ist. Teile der Geschäftslogik sind hier rein lokal auf dem Endgerät, haben jedoch u. U. Auswirkungen auf die anderen

Teile der Nutzerschnittstelle. Zur Unterstützung solcher Szenarien sind weitere Integrationskonzepte notwendig, die nicht nur das Datenmodell als Synchronisierungsbasis als Grundlage haben.

Literaturverzeichnis

[ACD+03] Andrews, Tony, Francisco Curbera, Hitesh Dholakia, Yaron Goland, Johannes Klein, Frank Leymann, Kevin Liu, Dieter Roller, Doug Smith, Satish Thatte, Ivana Trickovic und Sanjiva Weerawarana: *BPEL: Business Process Execution Language for Web Services Specification (BPEL4WS) - Version 1.1*, 2003.

[ADJ+04] Anegg, Hermann, Thomas Dangl, Michael Jank, Georg Niklfeld, Michael Pucher, Raimund Schatz, Rainer Simon und Florian Wegscheider: *Multimodal Interfaces in Mobile Devices - The MONA Project.* In: *MobEA II, Emerging Applications for Wireless and Mobile Access Workshop*, 2004.

[Ait06] Aitenbichler, Erwin: *Development Tools for Mundo Smart Environments.* In: Kortuem, Gerd (Herausgeber): *Workshop on Software Engineering Challenges for Ubiquitous Computing*, Seiten 89–90, Lancaster University, Juni 2006.

[AK06] Anke, Jürgen und Klaus Kabitzsch: *Cost-based Deployment Planning for Components in Smart Item Environments.* In: *11th IEEE International Conference on Emerging Technologies and Factory Automation*, Prague, Czech Republic, 20-22 September 2006.

[AKM05] Aitenbichler, Erwin, Jussi Kangasharju und Max Mühlhäuser: *Experiences with MundoCore.* In: *Third IEEE Conference on Pervasive Computing and Communications (PerCom'05) Workshops*, Seiten 168–172, Kauai Island, Hawaii, März 2005. IEEE Computer Society.

[AKM07] Aitenbichler, Erwin, Jussi Kangasharju und Max Mühlhäuser: *MundoCore: A Light-weight Infrastructure for Pervasive Computing.* Pervasive and Mobile Computing, 2007. doi:10.1016/j.pmcj.2007.04.002 (332–361).

[AM07] Aitenbichler, Erwin und Max Mühlhäuser: *Softwareentwicklung für Mundo Smart Environments.* In: *KiVS'07 Workshop*

'Service-Oriented Architectures und Service-Oriented Computing' (SOA/SOC), 2007. (383–394).

[Ank07] ANKE, JÜRGEN: *Verteilungsplanung für Komponenten in Smart Item Umgebungen.* Doktorarbeit, Technische Universität Dresden, eingereicht 2007.

[Aut87] AUTRUM, HANSJOCHEM (Herausgeber): *Von der Naturforschung zur Naturwissenschaft.* Springer, Berlin, 1987.

[Avi85] AVIZIENIS, A.: *The N-Version Approach to Fault-Tolerant Software.* In: *IEEE Transactions on Software Engineering*, Band SE-11, Seiten 1491–1501, 1985.

[BA03] BRAUN, ELMAR und ERWIN AITENBICHLER: *Interaktion als Dienst in Mundo.* In: *INFORMATIK 2003 Innovative Informatikanwendungen Band 1*, Seiten 222–227, 2003.

[BAKM04] BRAUN, ELMAR, GERHARD AUSTALLER, JUSSI KANGASHARJU und MAX MÜHLHÄUSER: *Accessing Web Applications with Multiple Context-Aware Devices.* In: MATERA, MARISTELLA und SARA COMAI (Herausgeber): *Engineering Advanced Web Applications*, Seiten 353–366. Rinton Press, 2004.

[BBD+02] BABCOCK, BRIAN, SHIVNATH BABU, MAYUR DATAR, RAJEEV MOTWANI und JENNIFER WIDOM: *Models and issues in data stream systems.* In: *PODS '02: Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, Seiten 1–16, New York, NY, USA, 2002. ACM Press.

[BCL07] BARRALON, NICOLAS, JOËLLE COUTAZ und CHRISTOPHE LACHENAL: *Coupling Interaction Resources and Technical Support.* In: *In Proc. HCI International 2007, Beijing, July 2007*, 2007.

[BD03] BERNSEN, NIELS OLE und LAILA DYBKJAER: *Best practice in natural and multimodal interactivity engineering - CLASS Deliverable 1.5+6.* Technischer Bericht, Odense, 2003.

[Ber02] BERNSEN, NILS OLE: *Multimodality in Language and Speech Systems — From Theory to Design Support Tool.* In: GRANSTRÓM, BJÓRN (Herausgeber): *Multimodality in Language and Speech Systems*, Band 19 der Reihe *Text, Speech and Language Technology*, Seiten 93–149. Kluwer Academic Publishers, 2002.

[BG04] BOLSHAKOV, IGOR A. und ALEXANDER GELBUKH: *COMPUTATIONAL LINGUISTICS: Models, Resources, Applications.* 2004.

[BM04] Braun, Elmar und Max Mühlhäuser: *Extending XML UIDLs for Multi-Device Scenarios.* In: Luyten, Kris, Marc Abrams, Jean Vanderdonckt und Quentin Limbourg (Herausgeber): *Proceedings of Developing User Interfaces with XML: Advances on User Interface Description Languages, Satellite Workshop of Advanced Visual Interfaces 2004*, Seiten 143–149, Mai 2004.

[BM05] Braun, Elmar und Max Mühlhäuser: *Interacting with Federated Devices.* In: Ferscha, Alois, Rene Mayrhofer, Thomas Strang, Claudia Linnhoff-Popien, Anind Dey, Andreas Butz und Albrecht Schmidt (Herausgeber): *Advances in Pervasive Computing, Adjunct Proceedings of the Third International Conference on Pervasive Computing*, Seiten 153–160. Austrian Computer Society, Mai 2005.

[Bol80] Bolt, Richard A.: *Voice and gesture at the graphics interface.* In: *SIGGRAPH '80: Proceedings of the 7th annual conference on Computer graphics and interactive techniques*, Seiten 262–270, New York, NY, USA, 1980. ACM Press.

[BP05] Berti, Silvia und Fabio Paternò: *Migratory MultiModal interfaces in MultiDevice environments.* In: *ICMI '05: Proceedings of the 7th international conference on Multimodal interfaces*, Seiten 92–99, New York, NY, USA, 2005. ACM Press.

[BPBH06] Burmeister, Rodger, Christoph Pohl, Siegfried Bublitz und Pascale Hugues: *SNOW - A Multimodal Approach for Mobile Maintenance Applications.* In: *Proceedings of the 15th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises*, Seiten 131–136, 2006.

[Bra87] Bratman, M. E.: *Intention, Plans, and Practical Reason.* Harvard University Press, Cambridge, MA, 1987.

[BSLG98] Beigl, Michael, Albrecht Schmidt, Markus Lauff und Hans-Werner Gellersen: *The UbicompBrowser.* In: *roceedings of the 4th ERCIM Workshop on User Interfaces for All (UI4ALL)*, Stockholm, Sweden, Oktober 1998.

[Bun98] Bunt, Harry: *Issues in Multimodal Human-Computer Communication.* In: *Multimodal Human-Computer Communication, Systems, Techniques, and Experiments*, Seiten 1–12, London, UK, 1998. Springer-Verlag.

[CC91] Coutaz, J. und J. Caelen: *A taxonomy for multimedia and multimodal user interface.* In: *1st ERCIM Workshop on Multimodal Human-Computer Interaction*, Lisbon, 1991.

[CD00] Cheesman, John und John Daniels: *UML components: a simple process for specifying component-based software.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2000.

[CDML03] Coles, Alistair, Eric Deliot, Tom Melamed und Kevin Lansard: *A framework for coordinated multi-modal browsing with multiple clients.* In: *WWW '03: Proceedings of the 12th international conference on World Wide Web*, Seiten 718–726, New York, NY, USA, 2003. ACM Press.

[CHML06] Carter, Scott, Amy Hurst, Jennifer Mankoff und Jack Li: *Dynamically adapting GUIs to diverse input devices.* In: *Assets '06: Proceedings of the 8th international ACM SIGACCESS conference on Computers and accessibility*, Seiten 63–70, New York, NY, USA, 2006. ACM Press.

[CJM+97a] Cohen, Philip R., Michael Johnston, David McGee, Sharon Oviatt, Jay Pittman, Ira Smith, Liang Chen und Josh Clow: *QuickSet: Multimodal Interaction for Distributed Applications.* Technischer Bericht, Portland, 1997.

[CJM+97b] Cohen, Philip R., Michael Johnston, David McGee, Sharon Oviatt, Jay Pittman, Ira Smith, Liang Chen und Josh Clow: *QuickSet: multimodal interaction for simulation set-up and control.* In: *Proceedings of the fifth conference on Applied natural language processing*, Seiten 20–24, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc.

[DFAB04] Dix, Alan, Janet Finlay, Gregory D. Abowd und Russel Beale: *Human Computer Interaction.* Pearson Education Limited, Essex, 3rd Auflage, 2004.

[DRONP01] Dan R. Olsen, Jr., S. Travis Nielsen und David Parslow: *Join and capture: a model for nomadic interaction.* In: *UIST '01: Proceedings of the 14th annual ACM symposium on User interface software and technology*, Seiten 131–140, New York, NY, USA, 2001. ACM Press.

[EAWJ02] Elnozahy, E. N. (Mootaz), Lorenzo Alvisi, Yi-Min Wang und David B. Johnson: *A survey of rollback-recovery protocols in message-passing systems.* ACM Computing Surveys (CSUR), 34(3):375–408, 2002.

[Erl05] Erl, Thomas: *Service-Oriented Architecture: Concepts, Technology, and Design.* Prentice Hall PTR, Upper Saddle River, NJ, USA, 2005.

[ERS03] Elting, Christian, Stefan Rapp und Gregor Mand Michael Strube: *Architecture and implementation of multimodal plug and play.* In: *ICMI '03: Proceedings of the 5th international conference on Multimodal interfaces*, Seiten 93–100, New York, NY, USA, 2003. ACM Press.

[FCFG99] Ferreira, Alexandre G., Renato Cerqueira, Waldemar Celes Filho und Marcelo Gattass: *Multiple Display Viewing Architecture for Virtual Environments over Heterogeneous Networks.* In: *SIBGRAPI '99: Proceedings of the XII Brazilian Symposium on Computer Graphics and Image Processing*, Seiten 83–92, Washington, DC, USA, 1999. IEEE Computer Society.

[Fou] Foundation of Intelligent Physical Agents: *Standard Status Specifications.* Online verfügbar: `http://www.fipa.org/repository/standardspecs.html`, Letzter Zugriff am 04.07.2007.

[Fou04] Foundation of Intelligent Physical Agents: *FIPA Agent Management Specification*, März 2004.

[FPV98] Fuggetta, Alfonso, Gian Pietro Picco und Giovanni Vigna: *Understanding Code Mobility.* IEEE Transactions on Software Engineering, 24(5), Mai 1998.

[FSF+04] Fu, R. Y., H. Su, J. C. Fletcher, W. Li, X. X. Liu, S. W. Zhao und C. Y. Chi: *A framework for device capability on demand and virtual device user experience.* IBM J. Res. Dev., 48(5/6):635–648, 2004.

[GHKP06] Göbel, Steffen, Falk Hartmann, Kay Kadner und Christoph Pohl: *A Device-Independent Multimodal Mark-up Language.* In: *INFORMATIK 2006: Informatik für Menschen, Band 2*, Seiten 170–177, 2006.

[GLM+05] Ghigny, C., J-Y. Lawson, B. Macq, L. Nigay und D.G. Trevisan: *Architecture of the OpenInterface platform.* Technischer Bericht, SIMILAR Projektkonsortium, 2005. Deliverable D58.

[Gro06a] Group, W3C XForms Working: *XForms 1.0 (Second Edition)*, 2006. Online verfügbar: `http://www.w3.org/TR/2006/REC-xforms-20060314/`, Letzter Zugriff am 31.08.2007.

[Gro06b] GROUP, WIDEX WORKING: *Widget Description Exchange Service Charter*, 2006.

[Gro07] GROUP, WIDEX WORKING: *Widget Description Exchange Service (WIDEX) Requirements*, 2007.

[GY03] GRUNDY, JOHN und BIAO YANG: *An environment for developing adaptive, multi-device user interfaces.* In: *AUIC '03: Proceedings of the Fourth Australasian user interface conference on User interfaces 2003*, Seiten 47–56, Darlinghurst, Australia, Australia, 2003. Australian Computer Society, Inc.

[GZ02] GRUNDY, JOHN und WENJING ZOU: *An Architecture for Building Multi-device Thin-Client Web User Interfaces.* In: *CAiSE '02: Proceedings of the 14th International Conference on Advanced Information Systems Engineering*, Seiten 728–732, London, UK, 2002. Springer-Verlag.

[HAH+02] HARTL, ANDREAS, ERWIN AITENBICHLER, GERHARD AUSTALLER HEINEMANN, ANDREAS, TOBIAS LIMBERGER BRAUN, ELMAR und MAX MÜHLHÄUSER: *Engineering Multimedia-Aware Personalized Ubiquitous Services.* In: *IEEE Fourth International Symposium on Multimedia Software Engineering (MSE'02)*, Seiten 344–351, 2002.

[Her87] HERMANN VON HELMHOLTZ: *Über die Entwicklungsgeschichte der neueren Naturwissenschaften.* In: HANSJOCHEM AUTRUM [Aut87], Seiten 33–62.

[HL06] HUA, ZHIGANG und HANQING LU: *Web Browsing on Small-Screen Devices: A Multiclient Collaborative Approach.* IEEE Pervasive Computing, 5(2):78–84, 2006.

[HPN00] HAN, RICHARD, VERONIQUE PERRET und MAHMOUD NAGHSHINEH: *WebSplitter: a unified XML framework for multi-device collaborative Web browsing.* In: *CSCW '00: Proceedings of the 2000 ACM conference on Computer supported cooperative work*, Seiten 221–230, New York, NY, USA, 2000. ACM Press.

[HR83] HAERDER, THEO und ANDREAS REUTER: *Principles of transaction-oriented database recovery.* ACM Computing Surveys (CSUR), 15(4):287–317, 1983.

[HR95] HAYES-ROTH, BARBARA: *An architecture for adaptive intelligent systems.* Artificial Intelligence, 72(1-2):329–365, 1995.

[HXLM05] Hua, Zhigang, Xing Xie, Hanqing Lu und Wei-Ying Ma: *ASAP: A Synchronous Approach for Photo Sharing across Multiple Devices.* In: *Proceedings of the 11th International Multimedia Modelling Conference, 2005.*, Seiten 206–213, 2005.

[HZ05] Haitao Zhang, Fei-Yue Wang, Yunfeng Ai: *An OSGi and agent based control system architecture for smart home.* In: *The 2005 IEEE International Conference on Networking, Sensing and Control*, Seiten 13–18, 2005.

[IEE93] IEEE Standards Board: *IEEE Standard for Information Technology - Protocols for Distributed Interactive Simulation Applications*, 1993. IEEE-Std-1278-1993.

[Jam86] James, Geoffrey: *The Tao of Programming.* Info Books, 1986. Auszüge Online verfügbar: `http://www.canonical.org/~kragen/tao-of-programming.html`, Letzter Zugriff am 12.11.2007.

[JPSF01] Johanson, Brad, Shankar Ponnekanti, Caesar Sengupta und Armando Fox: *Multibrowsing: Moving Web Content across Multiple Displays.* In: *UbiComp '01: Proceedings of the 3rd international conference on Ubiquitous Computing*, Seiten 346–353, London, UK, 2001. Springer-Verlag.

[Kad06] Kadner, Kay: *A flexible architecture for multimodal applications using federated devices.* In: *Proceedings of Visual Languages and Human-Centric Computing (VL/HCC'06)*, Seiten 236–237, Brighton, UK, September 2006. IEEE Computer Society.

[KM07] Kadner, Kay und Stephan Müller: *Integration of distributed user input to extend interaction possibilities with local applications.* In: *Proceedings of Engineering Interactive Systems*, 2007. accepted for publication.

[Kne07] Knechtel, Martin: *Konzeption und Implementierung eines Synchronisierungsverfahrens für verteilte Nutzerschnittstellen in heterogenen Netzwerken.* Diplomarbeit, Technische Universität Dresden, Juli 2007.

[KR07] Kadner, Kay und David Roussel: *Documentation for Aircraft Maintenance based on Topic Maps.* In: *Leveraging the Semantics of Topic Maps: Second International Conference on Topic Maps Research and Applications, TMRA 2006*, Seiten 56–61, Leipzig, Germany, 2007.

[KS04] Kadous, Mohammed Waleed und Claude Sammut: *MICA: Pervasive Middleware for Learning, Sharing and Talking.* In: *PERCOMW '04: Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops*, Seite 176, Washington, DC, USA, 2004. IEEE Computer Society.

[KSKB03] Kleindienst, Jan, Ladislav Serédi, Pekka Kapanen und Janne Bergman: *Loosely-coupled approach towards multi-modal browsing.* Universal Access in the Information Society, 2(2):173–188, 2003.

[KT87] Koo, Richard und Sam Toueg: *Checkpointing and rollback-recovery for distributed systems.* IEEE Trans. Softw. Eng., 13(1):23–31, 1987.

[KWK04] Kray, Christian, R. Wasinger und Gerd Kortuem: *Concepts and issues in interfaces for multiple users and multiple devices.* In: *Proceedings of the workshop on multi-user and ubiquitous user interfaces (MU3I) at IUI 2004*, Seiten 7–11, Funchal, Madeira, Portugal, 2004.

[Lam78] Lamport, Leslie: *Time, Clocks and the Ordering of Events in a Distributed System.* Communications of the ACM, 1978.

[LEW02] Loy, Marc, Robert Eckstein und Dave Wood: *Java Swing.* O'Reilly Media, 2002.

[LR93] Lewis, Clayton und John Rieman: *Task-Centered User Interface Design.* 1993. Online verfügbar: `http://www.hcibib.org/tcuid/index.html`, Letzter Zugriff am 19.11.2007.

[Mae95] Maes, Pattie: *Artificial life meets entertainment: lifelike autonomous agents.* Communications of the ACM, 38(11):108–114, 1995.

[Mat88] Mattern, F.: *Virtual Time and Global States in Distributed Systems.* In: *Workshop on Parallel and Distributed Algorithms*, 1988.

[May03] Maybury, Mark T.: *Universal multimedia information access.* Universal Access in the Information Society, 2(2):96–104, 2003.

[Mil95] Mills, David L.: *Improved algorithms for synchronizing computer network clocks.* IEEE/ACM Trans. Netw., 3(3):245–254, 1995.

[Mil96] Mills, D.: *Simple Network Time Protocol (SNTP) Version 4 for IPv4, IPv6 and OSI*, Oktober 1996. RFC 2030.

[Mül06] Müller, Stephan: *Konzeption einer Architektur zur Integration multimodaler Eingaben in bestehende Anwendungen.* Diplomarbeit, Technische Universität Dresden, 2006.

[MMB+02] Myers, Brad, Robert Malkin, Michael Bett, Alex Waibel, Ben Bostwick, Robert C. Miller, Jie Yang, Matthias Denecke, Edgar Seemann, Jie Zhu, Choon Hong Peck, Dave Kong, Jeffrey Nichols und Bill Scherlis: *Flexi-Modal and Multi-Machine User Interfaces.* icmi, 00:343, 2002.

[MMF00] Marsic, Ivan, Attila Medl und James Flanagan: *Natural communication with information systems.* In: *Proceedings of the IEEE*, Band 88, Seiten 1354–1366, August 2000.

[MMRM05] Malek, Sam, Marija Mikic Rakic und Nenad Medvidovic: *A Decentralized Redeployment Algorithm for Improving the Availability of Distributed Systems.* In: *Proceedings of the 3rd International Working Conference on Component Deployment (CD 2005)*, Seiten 99–114, Grenoble, France, November 2005.

[MPS04] Mori, Guilio, Fabio Paternó und Carmen Santoro: *Design and Development of Multidevice User Interfaces through Multiple Logical Descriptions.* IEEE Transactions on Software Engineering, 30(8):507–520, 2004.

[MR03] Mrohs, Bernd und Christian Räck: *Multi-Device, Ambient-Aware Information Delivery.* In: *International Conference on Ubiquitous Computing (UbiComp) 2003*, 2003.

[MtB07] Beek, Antonio Bucchiarone, Stefania Gnesi Maurice ter: *Web Service Composition Approaches: From Industrial Standards to Formal Methods.* In: *Second International Conference on Internet and Web Applications and Services (ICIW)*, Seiten 15–20, 2007.

[Mye01] Myers, Brad A.: *Using handhelds and PCs together.* Communications of the ACM, 44(11):34–41, 2001.

[NC93] Nigay, Laurence und Joelle Coutaz: *A design space for multimodal systems: concurrent processing and data fusion.* In: *CHI '93: Proceedings of the SIGCHI conference on Human factors in computing systems*, Seiten 172–178, New York, NY, USA, 1993. ACM Press.

[NM06] Nichols, Jeffrey und Brad A. Myers: *Controlling Home and Office Appliances with Smart Phones.* IEEE Pervasive Computing, 5(3):60–67, 2006.

[NMH+02] Nichols, Jeffrey, Brad A. Myers, Michael Higgins, Joseph Hughes, Thomas K. Harris, Roni Rosenfeld und Mathilde Pignol: *Generating remote control interfaces for complex appliances.* In: *UIST '02: Proceedings of the 15th annual ACM symposium on User interface software and technology*, Seiten 161–170, New York, NY, USA, 2002. ACM Press.

[OAS] OASIS User Interface Markup Language (UIML) Technical Committee: *User Interface Markup Language (UIML).* `http://www.oasis-open.org/committees/uiml/`.

[Ovi02] Oviatt, Sharon: *Multimodal Interfaces.* In: Jacko, J. und A. Sears (Herausgeber): *The Human-Computer Interaction Handbook: Fundamentals, Evolving Technologies and Emerging Applications*, Seiten 286–304. Lawrence Erlbaum Associates Inc., Mahwah, New Jersey, USA, 2002.

[Pen07] Pengjie, Li: *Design and implementation of a Web browser for the use with federated devices based on OSGi.* Großer Beleg, Technische Universität Dresden, Betreuer: Dipl.-Inf. Kay Kadner, 2007.

[PFL+02] Petrie, Helen, Wendy Fisher, Ine Langer, Gerhard Weber, Keith Gladstone, Cathy Rundle und Liesbeth Pyfers: *Universal Interfaces to Multimedia Documents.* International Conference on Multimodal User Interfaces, 00:319, 2002.

[PHK07a] Pereira, Alessandro Costa, Falk Hartmann und Kay Kadner: *A Distributed Staged Architecture for Multimodal Applications.* In: Oquendo, F. (Herausgeber): *European Conference on Software Architecture (ECSA) 2007*, Nummer 4758 in *Lecture Notes in Computer Science*, Seiten 195–206. Springer Verlag Berlin Heidelberg, 2007.

[PHK07b] Pereira, Alessandro Costa, Falk Hartmann und Kay Kadner: *A Distributed Staged Architecture for Multimodal Applications (Extended Abstract).* In: *Software Engineering 2007*, Seiten 255–256, 2007.

[Pra96] Pradhan, Dhiraj K. (Herausgeber): *Fault-tolerant computer system design.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1996.

[PSG00] PHAM, THAI-LAI, GEORG SCHNEIDER und STUART GOOSE: *A situated computing framework for mobile and ubiquitous multimedia access using small screen and composite devices.* In: *MULTIMEDIA '00: Proceedings of the eighth ACM international conference on Multimedia*, Seiten 323–331, New York, NY, USA, 2000. ACM Press.

[Ric05] RICHTER, KAI: *A transformational approach to multi-device interfaces.* In: *CHI '05: CHI '05 extended abstracts on Human factors in computing systems*, Seiten 1126–1127, New York, NY, USA, 2005. ACM Press.

[Rig05] *A Component-Based Infrastructure for Pervasive User Interaction.* In: *International Workshop on Software Techniques for Embedded and Pervasive Systems (STEPS)*, Munich, Germany, May 2005.

[Röm01] RÖMER, KAY: *Time synchronization in ad hoc networks.* In: *MobiHoc '01: Proceedings of the 2nd ACM international symposium on Mobile ad hoc networking & computing*, Seiten 173–182, New York, NY, USA, 2001. ACM Press.

[Roe95] ROEHL, BERNIE: *Some Thoughts on Behavior in VR Systems*, 1995. Online verfügbar: `http://ece.uwaterloo.ca/~broehl/behav.html`, Letzter Zugriff am 11.07.2007.

[Sch03a] SCHILLER, JOCHEN: *Mobilkommunikation.* Pearson Studium, München [u.a.], 2. Auflage, 2003.

[Sch03b] SCHLICHTING, H. JOACHIM: *Die Welt jenseits der geschliffenen Gläser - Zur Bedeutung des Sehens in der klassischen Physik.* PhyDid, 2003. Online verfügbar: `http://www.phydid.de/showpdf.php?artikel_id=14`, Letzter Zugriff am 04.09.2007.

[SDPG01] SCHNEIDER, G., S. DJENNANE, T.L. PHAM und S. GOOSE: *Multimodal Multi-device UIs for Ubiquitous Access to Multimedia Gateways.* In: *Seventeenth International Joint Conference on Artificial Intelligence: Workshop on Artificial Intelligence in Mobile Systems (AIMS)*, Seattle, USA, 2001.

[SHR07] SALMINEN, TIMO, SIMO HOSIO und JUKKA RIEKKI: *Middleware Based User Interface Migration: Implementation and Evaluation.* In: *Proceedings of the 4th International Conference on Mobile Technology, Applications and Systems (Mobility 2007)*, Seiten 366–371, 2007.

[Sin03] Sinha, Anoop Kumar: *Informally Prototyping Multimodal, Multidevice User Interfaces.* Doktorarbeit, University of California, Berkeley, 2003.

[SKSY06] Springer, Thomas, Kay Kadner, Frank Steuer und Ming Yin: *Middleware Support for Context-Awareness in 4G Environments.* In: *IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks*, Seiten 203–211, Niagara-Falls, Buffalo-NY, US, 2006.

[SM03] Schaefer, Robbie und Wolfgang Mueller: *Multimodal Interactive User Interfaces for Mobile Multi-Device Environments.* In: *Workshop Multi-Device Interfaces for Ubiquitous Peripheral Interaction at Ubicomp 2003*, Seattle, USA, 2003.

[SMB05] Schaefer, Robbie, Wolfgang Mueller und Steffen Bleul: *A Dialog Model for Multi Device Interfaces with Different Modalities.* In: *HCI International 2005*, Las Vegas, USA, 2005.

[SO00] Scott Oaks, Henry Wong: *Jini in a Nutshell.* O'Reilly Media, Inc., 2000.

[SS07] Schill, Alexander und Thomas Springer: *Verteilte Systeme: Grundlagen und Basistechnologien (eXamen.press).* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.

[SSD+04] Stewart, Christopher, Kai Shen, Sandhya Dwarkadas, Michael L. Scott und Jian Yin: *Profile-Driven Component Placement for Cluster-Based Online Services.* IEEE Distributed Systems Online, 5(10), 2004.

[SSW96] Scholze-Stubenrecht, Dr. Werner und Dr. Matthias Wermke (Herausgeber): *DUDEN - Die deutsche Rechtschreibung.* Dudenredaktion, 1996. 21. Fassung.

[Thea] The BIB3R Consortium: *Berlin's Beyond-3G Testbed and Serviceware Framework for Advanced Mobile Solutions.* Online verfügbar: `http://www.bib3r.de/`, Letzter Zugriff am 07.07.2007.

[Theb] The Mozilla Foundation: *XML User Interface Language (XUL).* `http://www.mozilla.org/projects/xul/`.

[The05a] The OSGi Alliance: *OSGi Service Platform, Core Specification*, August 2005.

[The05b] The SNOW Consortium: *SNOW Project Homepage*, 2005. `http://www.snow-project.org`.

[UMB] *UMBC AgentWeb, KQML - Knowledge Query and Manipulation Language*. Online verfügbar: `http://www.cs.umbc.edu/kqml/`, Letzter Zugriff am 04.07.2007.

[UvBTHB01] Urting, David, Stefan van Baelen Tom Holvoet und Yolande Berbers: *Embedded Software Development: Components and Contracts*. In: *Proceedings of the IASTED International Conference on Parallel and Distributed Computing and Systems*, Seiten 685–690, Anaheim, USA, 2001. ACTA Press.

[W3C06] W3C Voice Browser Working Group: *State Chart XML (SCXML): State Machine Notation for Control Abstraction*, 2006. Online verfügbar: `http://www.w3.org/TR/2006/WD-scxml-20060124/`, Letzter Zugriff am 27.06.2007.

[W3C07] W3C Voice Browser Working Group: *Voice Browser Call Control: CCXML Version 1.0*, 2007. Online verfügbar: `http://www.w3.org/TR/2007/WD-ccxml-20070119/`, Letzter Zugriff am 27.06.2007.

[Wai07] Wainhouse Research: *The Wainhouse Research Bulletin - News and Views on Real-Time Unified Communication*. Monatlicher Newsletter, Band 8, Ausgabe 29, 29. August 2007. Online verfügbar: `http://www.wainhouse.com/files/wrb-08/wrb-0829.pdf`, Letzter Zugriff am 04.09.2007.

[WAQ04] West, David, Trent Apted und Aaron Quigley: *A context inference and multi-modal approach to mobile information access*. In: *Artificial Intelligence in Mobile Systems*, Seiten 28–35, Nottingham, England, 2004.

[WDJS04] Wegscheider, Florian, Thomas Dangl, Michael Jank und Rainer Simon: *A Multimodal Interaction Manager for Device Independent Mobile Applications*. In: *13th International World Wide Web Conference*, Seiten 272–273, 2004.

[Wil00] Wilfredo, Torres: *Software Fault Tolerance: A Tutorial*. Technischer Bericht, 2000.

[WJ95] Wooldridge, Michael und Nick Jennings: *Intelligent Agents: Theory and Practice*. Knowledge Engineering Review, 10(2):115–152, 1995.

[WM03] Wang, Jingtao und Jennifer Mankoff: *Theoretical and architectural support for input device adaptation*. In: *CUU '03: Proceedings of the 2003 conference on Universal usability*, Seiten 85–92, New York, NY, USA, 2003. ACM Press.

[Wor02a] World Wide Web Consortium (W3C): *Multimodal Interaction Activity*, 2002. Online verfügbar: `http://www.w3.org/2002/mmi/`, Letzter Zugriff am 11.07.2007.

[Wor02b] World Wide Web Consortium (W3C): *Multimodal Interaction Framework*, 2002. Online verfügbar: `http://www.w3.org/TR/2003/NOTE-mmi-framework-20030506/`, Letzter Zugriff am 11.07.2007.

[Wor03] World Wide Web Consortium (W3C): *Multimodal Interaction Requirements*, 2003. Online verfügbar: `http://www.w3.org/TR/2003/NOTE-mmi-reqs-20030108/`, Letzter Zugriff am 11.07.2007.

[Wor05a] World Wide Web Consortium (W3C): *Extensible MultiModal Annotation markup language*, 2005. Online verfügbar: `http://www.w3.org/TR/2007/WD-emma-20070409/`, Letzter Zugriff am 11.07.2007.

[Wor05b] World Wide Web Consortium (W3C): *Synchronized Multimedia Integration Language (SMIL 2.1)*, 2005. Online verfügbar: `http://www.w3.org/TR/2005/REC-SMIL2-20051213`, Letzter Zugriff am 28.07.2007.

[Wor05c] World Wide Web Consortium (W3C): *Web Services Choreography Description Language Version 1.0*, 2005. Online verfügbar: `http://www.w3.org/TR/2005/CR-ws-cdl-10-20051109/`, Letzter Zugriff am 15.08.2007.

[Wor06] World Wide Web Consortium (W3C): *Multimodal Architecture and Interfaces*, 2006. Online verfügbar: `http://www.w3.org/TR/2006/WD-mmi-arch-20061211/`, Letzter Zugriff am 27.06.2007.

[WW05] Wu, Qing und Zhaohui Wu: *Adaptive Component Allocation in ScudWare Middleware for Ubiquitous Computing.* In: *EUC*, Seiten 1155–1164, 2005.

[XSL99] XSL Working Group and XML Linking Working Group: *XML Path Language (XPath), Version 1.0*, 1999. Online verfügbar: `http://www.w3.org/TR/1999/REC-xpath-19991116`, Letzter Zugriff am 22.10.2007.

[Yan03] Yang, Feng-Chao: *Design and implement of the home networking service agent federation using open service gateway.* In: *Proceedings of the International Conference on Integration of Knowledge Intensive Multi-Agent Systems*, Seiten 628–633, 2003.

[ZLWS06] Zhang, Zhe, Daxin Liu, Zhengxian Wei und Changsong Sun: *Research on Triple Modular Redundancy Dynamic Fault-Tolerant System Model.* imsccs, 01:572–576, 2006.

A
Testfälle für die Validierung

TF01 - Anmelden des Nutzers

Beschreibung Der Nutzer möchte sich am System anmelden, um dieses zu verwenden. Das Anmelden ist wesentlich für die Funktion der Integrationsschicht, da alle Komponenten und Geräte zur Laufzeit einem Nutzer zugeordnet werden.

Vorbedingung Der Nutzer ist noch nicht angemeldet. Das System ist funktionsbereit. Der Nutzer meldet sich mit einem Namen am System an.

Nachbedingung Der Nutzer ist angemeldet, d. h. er hat einen entsprechenden Eintrag (eine Nutzer-ID) im Nutzerverzeichnis bekommen. Wenn bereits ein Nutzer mit demselben Namen existiert, wird das Anmelden nicht durchgeführt und der Nutzer darauf hingewiesen.

Erfüllungsgrad Das System erfüllt diesen Testfall.

TF02 - Anzeigen der in der Integrationsschicht verfügbaren Komponenten (UI und sonstige)

Beschreibung Der Nutzer lässt sich mit Hilfe der Föderationsverwaltungsanwendung die verfügbaren Komponenten anzeigen, welche zur Installation in der Integrationsschicht ausgewählt werden können. Dabei werden sowohl die UI-Komponenten als auch die unterstützenden Komponenten angezeigt. Aus dieser Liste wählt der Nutzer den HTML-Browser zur Platzierung aus.

Vorbedingung Der Nutzer ist angemeldet. Ein angemeldeter Nutzer ist gleichzeitig Vorbedingung für alle weiteren Testfälle.

Nachbedingung Der Nutzer wählt aus der Liste verfügbarer Komponenten den HTML-Browser zur Platzierung aus.

Erfüllungsgrad Das System erfüllt diesen Testfall.

TF03 - Platzieren des HTML-Browsers auf dem Laptop

Beschreibung Der HTML-Browser wird vom Nutzer zur Platzierung angefordert. Dabei wird explizit der Laptop als Gerät zur Platzierung angegeben.

Vorbedingung Der HTML-Browser ist noch nicht auf dem Laptop installiert.

Nachbedingung Der HTML-Browser ist auf dem Laptop installiert und wird ausgeführt.

Erfüllungsgrad Das System erfüllt diesen Testfall.

TF04 - Komposition von HTML-Browser und Integration-Komponente durch Ermitteln der Referenz

Beschreibung Der Nutzer konfiguriert den HTML-Browser für den föderierten Modus, sodass dieser die erkannten Eingaben zur weiteren Verarbeitung (d. h. evtl. Fusion) an die Integration-Komponente übertragen muss. Daher stellt der HTML-Browser eine entsprechende Anfrage an das Dienstverzeichnis, welches eine Referenz auf eine Integration-Komponente zurückgeliefert.

Vorbedingung Der Nutzer macht Eingaben im HTML-Browser, welcher sich im Föderationsmodus befindet. Eine Referenz auf eine Integration-Komponente ist noch nicht vorhanden.

Nachbedingung Der HTML-Browser überträgt die erkannten Eingaben an die Integration-Komponente.

Erfüllungsgrad Das System erfüllt diesen Testfall.

TF05 - Automatisches Platzieren der Integration-Komponente

Beschreibung In Testfall TF04 wird eine Referenz auf eine Integration-Komponente angefordert, welche es zu diesem Zeitpunkt noch nicht gibt, sodass eine neue platziert werden muss. Da diese Platzierung nicht direkt vom Nutzer angefordert wurde, ist auch kein Gerät definiert, auf dem die Komponente platziert werden soll. Da der Nutzer eine dezentrale Verteilungsstrategie im Nutzerprofil eingestellt hat, wird der Integrationsschicht-Server zur Platzierung ausgewählt. Anschließend wird die Platzierung durchgeführt.

Vorbedingung Die Integration-Komponente ist noch nicht installiert, eine Referenz fehlt demnach und wird angefordert. Im Nutzerprofil ist eine dezentrale Verteilungsstrategie eingestellt.

Nachbedingung Die Integration-Komponente ist auf dem Integrationsschicht-Server installiert und wird ausgeführt.

Erfüllungsgrad Das System erfüllt diesen Testfall.

TF06 - Inhalt der Anwendung anfordern und auf dem HTML-Browser darstellen

Beschreibung Der Nutzer gibt in das Adressfeld des HTML-Browsers „d3ml“ ein, wodurch eine in D3ML-kodierte Testanwendung aufgerufen wird. Daraufhin wird das zugehörige Dokument geladen, für den HTML-Browser aufbereitet und an diesen zur Darstellung übertragen.

Vorbedingung Der HTML-Browser wurde gestartet. Der Nutzer gibt eine URL (hier „d3ml“) ein und bestätigt diese. Die Komponenten für Integration und Interaktionssteuerungs sind installiert und werden ausgeführt.

Nachbedingung Der Inhalt des D3ML-Dokuments wird im HTML-Browser angezeigt.

Erfüllungsgrad Das System erfüllt diesen Testfall.

TF07 - Platzieren des Voice-Browsers auf dem PDA

Beschreibung Der Nutzer meldet sich auf dem PDA an und lässt den Voice-Browser lokal platzieren, welchen er anschließend zur Interaktion verwendet. Das entspricht im Prinzip den Testfällen TF01 bis TF04.

Vorbedingung Der Nutzer ist auf dem PDA nicht angemeldet. Der Voice-Browser ist auf dem PDA nicht installiert.

Nachbedingung Der Nutzer ist auf dem PDA angemeldet und der Voice-Browser ist auf dem PDA installiert.

Erfüllungsgrad Das System erfüllt diesen Testfall.

TF08 - Austausch des Voice-Browsers mit integrierter Spracherkennung durch einen ohne integrierte Spracherkennung

Beschreibung Der Nutzer erstellt mit Hilfe der Föderationsverwaltungsanwendung einen Austauschkonfigurationsdokument, worin er die Service-ID des Voice-Browsers m. I. als auszutauschenden Dienst und die Komponenten-ID des Voice-Browsers o. I. als Austauschkomponente einträgt. Nach Bestätigung der Austauschkonfiguration wird der Austausch vom ServiceDirectory durchgeführt und der neue Voice-Browser arbeitet an der Stelle weiter, wo der alte aufgehört hat.

Vorbedingung Der Voice-Browser m. I. ist auf dem PDA installiert.

Nachbedingung Der Voice-Browser o. I. ist auf dem PDA installiert, wird ausgeführt und besitzt den selben Dialogzustand wie der Voice-Browser m. I. vor dem Austausch.

Erfüllungsgrad Das System erfüllt diesen Testfall.

TF09 - Installieren von benötigten Komponenten des neuen Voice-Browsers

Beschreibung Der Austausch des Voice-Browsers aus TF08 beinhaltet die übliche Installation einer Komponente, namentlich des Voice-Browsers o. I. auf dem PDA. Dabei wird abgeprüft, ob und ggf. wie benötigte Komponenten ebenfalls installiert werden sollen. Der Nutzer hat vor dem Voice-Browser-Austausch das Nutzerprofil geändert, sodass benötigte Komponenten nach zentraler Verteilungsstrategie bei Installation der benötigenden Komponente ebenfalls installiert werden. Beim Voice-Browser o. I. bedeutet das die automatische Installation der Voice-Interpretation auf dem PDA.

Vorbedingung Der Nutzer hat im Nutzerprofil vor Auslösen des Austauschs die zentrale Verteilungsstrategie und Installation von benötigten Komponenten zum Installationszeitpunkt ausgewählt. Die Voice-Interpretation ist nicht installiert. Der Voice-Browser o. I. wird auf dem PDA installiert.

Nachbedingung Die Voice-Intepretation ist auf dem gleichen Gerät (dem PDA) wie der Browser installiert.

Erfüllungsgrad Das System erfüllt diesen Testfall.

TF10 - Erzeugen von Text aus Sprache durch die Voice-Interpretation

Beschreibung Der Nutzer macht Eingaben in den Voice-Browser, die zur Voice-Interpretation weitergeleitet werden. Diese empfängt die Informationen und prüft mit der definierten Grammatik, ob es sich bei den Nutzereingaben um gültige Werte handelt. Ist dies der Fall, wird der erkannte Text an den Voice-Browser zurückgeliefert. Für diesen Anwendungsfall ist unerheblich, ob die Interpretation innerhalb oder außerhalb des Voice-Browsers erfolgt. Der einzige Unterschied ist die explizite Kommunikation zwischen beiden Komponenten.

Vorbedingung Die Voice-Interpretation und der Voice-Browser sind verfügbar. Der Nutzer macht Eingaben in den Voice-Browser. Die Voice-Interpretation ist mit einer Grammatik konfiguriert.

Nachbedingung Die Eingaben des Nutzers wurden erkannt und als Text zurückgeliefert.

Erfüllungsgrad Das System erfüllt diesen Testfall.

TF11 - Migration der Voice-Interpretation vom PDA auf den Integrationsschicht-Server

Beschreibung Da die Voice-Interpretation auf dem PDA auf Grund der geringeren Hardware-Ressourcen nur mit einer eingeschränkten Grammatik betrieben werden kann, verschiebt der Nutzer diese Komponente auf den Integrationsschicht-Server. Dazu wählt der Nutzer mittels Föderationsverwaltungsanwendung die Komponente und das Zielgerät aus, woraufhin die Migration durchgeführt wird. Durch die Migration gehen keine bereits teilweise verarbeiteten Informationen verloren.

Vorbedingung Die Voice-Interpretation ist auf dem PDA installiert und wird ausgeführt.

Nachbedingung Die Voice-Interpretation ist auf dem Integrationsschicht-Server installiert und wird ausgeführt. Die Interaktion des Nutzers kann ohne Verlust fortgesetzt werden.

Erfüllungsgrad Das System erfüllt diesen Testfall.

TF12 - Navigation duch die Anwendung mit Synchronisierung auf *page*-Ebene

Beschreibung Der Nutzer verwendet die Anwendung und navigiert durch verschiedene Dialoge. Die Dialoge entsprechen jeweils neuen Seiten bzw. Dokumenten für den HTML- und Voice-Browser, die eine Aktualisierung der Seite zur Folge haben. Diese Aktualisierung erfolgt synchronisiert, d. h. der Synchronsierungsalgorithmus kommt zum Einsatz, um die Durchführung der UI-Aktualisierung langsamerer Geräte zu verzögern. Dabei kann die Aktualisierung beliebig durch beide Browser ausgelöst werden.

Vorbedingung HTML-Browser, Voice-Browser, Integration-Komponente und Interaktionssteuerung sind installiert und die Zielanwendung wurde aufgerufen.

Nachbedingung Der Nutzer hat die Informationen der verschiedenen Dialoge erhalten, wobei bei der Seiten-Aktualisierung der Synchronisierungsalgorithmus zum Einsatz kommt.

Erfüllungsgrad Das System erfüllt diesen Testfall.

TF13 - Ein- und Ausgabe von Formularinformationen mit Synchronisierung auf *event*-Ebene

Beschreibung Einer der dem Nutzer angezeigten Dialoge enthält Formularfelder, die der Nutzer ausfüllt. Dazu kann er ein beliebiges Endgerät verwenden. Dabei werden die Eingaben sofort auf dem Endgerät angezeigt und parallel zur Integration übertragen.

Vorbedingung Es wird ein Dialog mit Formularfeldern auf mindestens zwei Endgeräten angezeigt.

Nachbedingung Die vom Nutzer getätigten Eingaben auf einem Endgerät werden auf dem anderen reflektiert, wobei jedes Ereignis repliziert wird.

Erfüllungsgrad Das System erfüllt diesen Testfall teilweise. Da der Voice-Browser nur Eingaben auf *field*-Ebene unterstützt, können diese Eingaben nur auf *field*-Ebene synchronisiert werden. Beim HTML-Browser ist eine Synchronisierung auf *event*-Ebene möglich.

TF14 - Austausch der Integrationskomponente

Beschreibung Damit die Eingaben von verschiedenen UI-Komponenten integriert werden können, ist eine Integration-Komponente notwendig, die diese Integration unterstützt. Da bisher nur eine Integration-Komponente installiert wurde, die lediglich die Eingaben kanalisiert, muss diese gegen die Integration-Komponente mit einfacher Fusion ausgetauscht werden. Wie beim Austausch des Voice-Browsers erstellt der Nutzer ein Austauschkonfigurationsdokument mit der Service-ID des alten Dienstes und der Komponenten-ID der neuen Komponente und schickt dieses ab, woraufhin der Austausch durchgeführt wird.

Vorbedingung Die Integration-Komponente ohne Fusion wird verwendet. Das Austauschkonfigurationsdokument wird angelegt und abgeschickt.

Nachbedingung Die Integration-Komponente mit Fusion hat die ohne Fusion abgelöst, also ist installiert und wird ausgeführt.

Erfüllungsgrad Das System erfüllt diesen Testfall.

TF15 - Einfache Integration von Eingaben und kontrollierte Repräsentation auf dem Laptop

Beschreibung Nachdem die Integration-Komponente eine einfache Fusion unterstützt, soll diese Funktionalität auch genutzt werden. Dazu hält der Arbeiter auf dem Laptop die Umschalt-Taste gedrückt und führt gleichzeitig eine Voice-Eingabe durch. Diese beiden Informationen werden in der Integration-Komponente zusammengefügt, sodass das vormals kleingeschriebene Ergebnis der Spracherkennung in Großbuchstaben in den Anwendungen erscheint.

Vorbedingung Die Integration-Komponente mit einfacher Fusion wird verwendet.

Nachbedingung Die gedrückte Umschalt-Taste am Laptop und die Spracheingabe werden zu einem großbuchstabigen Ergebnis fusioniert.

Erfüllungsgrad Das System erfüllt diesen Testfall.

TF16 - Deinstallieren des Voice-Browser nach Gebrauch

Beschreibung Der Voice-Browser wird nicht mehr benötigt und wird daher vom Nutzer deinstalliert. Dafür wählt er mit der Föderationsverwaltungsanwendung den PDA aus und selektiert den Voice-Browser aus der angezeigten Liste installierter Komponenten, welcher nach Bestätigung der Auswahl gestoppt und entfernt wird. Dabei ist unerheblich, ob der Nutzer die Föderationsverwaltung auf dem Laptop oder die auf dem PDA dafür verwendet.

Vorbedingung Der Voice-Browser ist auf dem PDA installiert und wird ausgeführt.

Nachbedingung Der Voice-Browser ist auf dem PDA nicht installiert.

Erfüllungsgrad Das System erfüllt diesen Testfall.

TF17 - Gezieltes Platzieren des HTML-Browsers auf dem Beamer-PC

Beschreibung Der Nutzer wählt mit der Föderationsverwaltungsanwendung aus der Liste verfügbarer Komponenten den HTML-Browser aus und lässt ihn auf dem Beamer-PC installieren.

Vorbedingung Der HTML-Browser ist auf dem Beamer-PC nicht installiert.

Nachbedingung Der HTML-Browser ist auf dem Beamer-PC installiert.

Erfüllungsgrad Das System erfüllt diesen Testfall.

TF18 - Umschalten von kontrollierter auf vorläufige *event*-Level Aktualisierung

Beschreibung Damit der Nutzer flüssiger mit der Anwendung agieren kann, wechselt er von kontrollierter auf vorläufige Aktualisierung von Eingaben auf *event*-Ebene. Dazu modifiziert er den entsprechenden Teil des Nutzerprofils mit der Föderationsverwaltungsanwendung. Anschließend werden die Eingaben sofort lokal und anschließend so schnell wie möglich auf die übrigen UI-Komponenten verteilt.

Vorbedingung Die *event*-Level Aktualisierung erfolgt kontrolliert.

Nachbedingung Die *event*-Level Aktualisierung erfolgt vorläufig.

Erfüllungsgrad Das System erfüllt diesen Testfall.

TF19 - Deinstallieren der HTML-Browser von Beamer-PC und Laptop

Beschreibung Analog zu TF16 werden die HTML-Browser von Beamer-PC und Laptop durch den Nutzer explizit entfernt, indem er die entsprechenden Komponenten der entsprechenden Geräte in der Föderationsverwaltungsanwendung auswählt und deinstallieren lässt. Anschließend befinden sich keine UI-Komponenten mehr auf den Endgeräten.

Vorbedingung Der HTML-Browser ist auf Beamer-PC und Laptop installiert und wird ausgeführt.

Nachbedingung Der HTML-Browser wurde von Beamer-PC und Laptop entfernt.

Erfüllungsgrad Das System erfüllt diesen Testfall.

TF20 - Abmelden des Nutzers

Beschreibung Der Nutzer hat die Verwendung der Integrationsschicht beendet und meldet sich ab. Daraufhin werden alle übrigen, dem Nutzer gehörenden Komponenten deinstalliert.

Vorbedingung Der Nutzer ist angemeldet.

Nachbedingung Der Nutzer ist abgemeldet. Alle seine Komponenten wurden deinstalliert.

Erfüllungsgrad Das System erfüllt diesen Testfall.

SF01 - Migration einer nichtverwendeten Komponente zwischen zwei Plattformen

Beschreibung Dieser Testfall bezieht sich auf die Migration einer Komponente, die zum Zeitpunkt der Migration nicht verwendet wird, d. h. es existiert kein aktiver Aufruf. Im Falle einer UI-Komponente wird diese gerade nicht vom Nutzer verwendet. Der aktuelle Zustand der Komponente zum Zeitpunkt des Beginns der Migration ist nach der Migration immer noch vorhanden. Der Nutzer wählt Komponente und Zielgerät mit der Föderationsverwaltungsanwendung aus.

Vorbedingung Die zu migrierende Komponente besitzt einen bestimmten Zustand und befindet sich auf Plattform A.

Nachbedingung Die zu migrierende Komponente befindet sich auf Plattform B und besitzt denselben Zustand.

Erfüllungsgrad Das System erfüllt den Sonderfall.

SF02 - Migration einer verwendeten Komponente zwischen zwei Plattformen

Beschreibung Im Gegensatz zur Migration einer nichtverwendeten Komponente trifft in diesem Spezialfall die Migrationsanforderung an die Komponente in einem Augenblick ein, indem diese gerade von einer anderen benutzt wird. Dabei wird entsprechend des *Late Response*-Mechanismus (siehe Unterabschnitt 4.5.3) der gegenwärtige Aufruf kontrolliert abgebrochen, nach Abschluss der Migration weiterverarbeitet und die entsprechende Antwort zurückgeliefert.

Vorbedingung Eine Komponente bearbeitet gerade einen Aufruf durch eine andere Komponente, während die Anforderung zur Migration eintrifft.

Nachbedingung Die Migration ist erfolgt und der Aufruf wurde entsprechend des *Late Response*-Mechanismus beendet.

Erfüllungsgrad Das System erfüllt den Sonderfall nicht.

SF03 - Implizite Aufrufumleitung

Beschreibung Dieser Sonderfall beschreibt die Behandlung eines Aufrufs, der an eine gerade migrierende Komponente gerichtet ist und somit von dieser nicht akzeptiert werden kann.

Vorbedingung Eine Komponente ist gerade dabei, migriert zu werden, während ein Aufruf an diese gerichtet wird.

Nachbedingung Der Aufruf wurde erfolgreich beendet, nachdem er für den Aufrufer transparent von der alten Plattform an die neue weitergeleitet wurde.

Erfüllungsgrad Das System erfüllt den Sonderfall nicht.

SF04 - Explizite Aufrufumleitung

Beschreibung Dieser Sonderfall beschreibt die Behandlung eines Aufrufs, der an eine gerade migrierende Komponente gerichtet ist und somit von dieser nicht akzeptiert werden kann.

Vorbedingung Eine Komponente ist gerade dabei, migriert zu werden, während ein Aufruf an diese gerichtet wird.

Nachbedingung Der Aufruf wurde erfolgreich beendet, nachdem der Aufruf von der alten Plattform mit Referenz auf die neue Plattform abgewiesen wurde.

Erfüllungsgrad Das System erfüllt den Sonderfall nicht.

SF05 - Server-basierte Synchronisierungsausführung

Beschreibung Die Synchronisierung von Aktualisierungen der Nutzerschnittstelle auf *page*-Ebene erfolgt durch Verzögerung der Auslieferung auf Serverseite. Dazu hat der Nutzer in seinem Nutzerprofil als Synchronisierungsart server-basiert ausgewählt und die Synchronisierung auf *page*-Ebene aktiviert.

Vorbedingung Der Nutzer verwendet zwei UI-Komponenten, aktiviert die Synchronisierung auf *page*-Ebene mit server-basierter Synchronisierungsart.

Nachbedingung Die Nutzerschnittstellen werden synchronisiert aktualisiert, indem der Server die Auslieferung verzögert.

Erfüllungsgrad Das System erfüllt den Sonderfall.

SF06 - Client-basierte Synchronisierungsausführung

Beschreibung Die Situtation ist die gleiche wie in SF06, nur dass der Nutzer eine Synchronisierung durch den Client im Nutzerprofil eingstellt hat.

Vorbedingung Der Nutzer verwendet zwei UI-Komponenten, aktiviert die Synchronisierung auf *page*-Ebene mit client-basierter Synchronisierungsart.

Nachbedingung Die Nutzerschnittstellen werden synchronisiert aktualisiert, indem die UI-Komponente die Ausgabe verzögert.

Erfüllungsgrad Das System erfüllt diesen Sonderfall.

B
Komponentenprofile

B.1. HTML-Browser

```
<componentprofile cID="2">
  <functions>
    <function id="1" type="Rendering" modality="visual" />
    <function id="2" type="Recognition" modality="keyboard" />
    <function id="3" type="Recognition" modality="mouse" />
    <function id="4" type="Interpretation" modality="keyboard" />
    <function id="5" type="Interpretation" modality="mouse" />
    <function id="6" type="Integration" domain="local" />
    <function id="7" type="Styling" modality="visual" />
  </functions>
  <couplingprofile>
    <function id="1">
      <providedports>
        <providedport portid="P1">
          <methodname>render</methodname>
          <methodsignature>
            <param name="url" type="String" />
            <param name="xhtmlDescription" type="String" />
            <param name="options" type="String" />
            <returnvalue />
          </methodsignature>
        <providedport>
        <providedport portid="P2">
          <methodname>reactEvent</methodname>
```

```
25          <methodsignature>
26            <param name="clientID" type="String" />
27            <param name="event" type="String" />
28            <returnvalue />
29          </methodsignature>
30        <providedport>
31      </providedports>
32    </function>
33    <function id"6">
34      <requiredports>
35        <requiredport cID="2" functionid="1" portid="P1" />
36        <requiredport cID="2" functionid="1" portid="P2" />
37        <requiredport cID="2" functionid="1" portid="P3" />
38        <requiredport>
39          <description />
40        </requiredport>
41      </requiredports>
42    </function>
43  </couplingprofile>
44  <requirements>
45    <hardware>
46      <inputdevice type="keyboard" />
47      <inputdevice type="mouse" />
48      <outputdevice type="display" />
49    </hardware>
50    <software />
51  </requirements>
52 </componentprofile>
```

Listing B.1: Komponentenprofil des HTML-Browsers

B.2. Integration ohne Fusion

```
1 <componentprofile cID="3">
2   <functions>
3     <function id="1" type="Integration" domain="global" />
4   </functions>
5   <couplingprofile>
6     <function id="1">
7       <providedports>
8         <providedport portid="P1">
9           <methodname>reportEvent</methodname>
10          <methodsignature>
11            <param name="clientID" type="String" />
```

```
12            <param name="event" type="String" />
13            <returnvalue />
14          </methodsignature>
15          <description>
16            <qosattrib type="fusiontype" value="none" />
17          </description>
18        <providedport>
19        <providedport portid="P2">
20          <methodname>doGet</methodname>
21          <methodsignature>
22            <param name="clientID" type="String" />
23            <param name="getEvent" type="String" />
24            <returnvalue />
25          </methodsignature>
26          <description>
27            <qosattrib type="fusiontype" value="none" />
28          </description>
29        </providedport>
30        <providedport portid="P3">
31          <methodname>doPut</methodname>
32          <methodsignature>
33            <param name="clientID" type="String" />
34            <param name="putEvent" type="String" />
35            <returnvalue />
36          </methodsignature>
37          <description>
38            <qosattrib type="fusiontype" value="none" />
39          </description>
40        </providedport>
41      </providedports>
42      <requiredports>
43        <requiredport cID="4" functionid="1" portid="P1" />
44        <requiredport cID="4" functionid="1" portid="P2" />
45        <requiredport cID="4" functionid="1" portid="P3" />
46      </requiredports>
47    </function>
48  </couplingprofile>
49  <requirements>
50    <hardware />
51    <software />
52  </requirements>
53 </componentprofile>
```

Listing B.2: Komponentenprofil der Integration-Komponente

B.3. Integration mit Fusion

```
<componentprofile cID="8">
  <functions>
    <function id="1" type="Integration" domain="global" />
  </functions>
  <couplingprofile>
    <function id="1">
      <providedports>
        <providedport portid="P1">
          <methodname>reportEvent</methodname>
          <methodsignature>
            <param name="clientID" type="String" />
            <param name="event" type="String" />
            <returnvalue />
          </methodsignature>
          <description>
            <qosattrib type="fusiontype" value="simple" />
          </description>
        <providedport>
        <providedport portid="P2">
          <methodname>doGet</methodname>
          <methodsignature>
            <param name="clientID" type="String" />
            <param name="getEvent" type="String" />
            <returnvalue />
          </methodsignature>
          <description>
            <qosattrib type="fusiontype" value="simple" />
          </description>
        </providedport>
        <providedport portid="P3">
          <methodname>doPut</methodname>
          <methodsignature>
            <param name="clientID" type="String" />
            <param name="putEvent" type="String" />
            <returnvalue />
          </methodsignature>
          <description>
            <qosattrib type="fusiontype" value="simple" />
          </description>
        </providedport>
      </providedports>
      <requiredports>
        <requiredport cID="4" functionid="1" portid="P1" />
```

```
44        <requiredport cID="4" functionid="1" portid="P2" />
45        <requiredport cID="4" functionid="1" portid="P3" />
46      </requiredports>
47    </function>
48  </couplingprofile>
49  <requirements>
50    <hardware />
51    <software />
52  </requirements>
53 </componentprofile>
```

Listing B.3: Komponentenprofil der Integration-Komponente

B.4. Interaktionssteuerung

```
1  <componentprofile cID="4">
2    <functions>
3      <function id="1" type="Interaction Manager"/>
4      <function id="2" type="System+Environment"/>
5      <function id="3" type="SessionManagement" />
6      <function id="4" type="Generation" />
7    </functions>
8    <couplingprofile>
9      <function id="1">
10       <providedports>
11         <providedport portid="P1">
12           <methodname>reportEvent</methodname>
13           <methodsignature>
14             <param name="clientID" type="String" />
15             <param name="event" type="String" />
16             <returnvalue />
17           </methodsignature>
18         </providedport>
19         <providedport portid="P2">
20           <methodname>doGet</methodname>
21           <methodsignature>
22             <param name="clientID" type="String" />
23             <param name="getEvent" type="String" />
24             <returnvalue />
25           </methodsignature>
26         </providedport>
27         <providedport portid="P3">
28           <methodname>doPut</methodname>
29           <methodsignature>
```

```
30            <param name="clientID" type="String" />
31            <param name="putEvent" type="String" />
32            <returnvalue />
33          </methodsignature>
34        </providedport>
35      </providedports>
36    </function>
37  </couplingprofile>
38  <requirements>
39    <hardware />
40    <software />
41  </requirements>
42 </componentprofile>
```

Listing B.4: Komponentenprofil der Interaktionssteuerungs-Komponente

B.5. Voice-Browser mit Interpretation

```
1  <componentprofile cID="5">
2    <functions>
3      <function id="1" type="Rendering" modality="voice" />
4      <function id="2" type="Recognition" modality="voice" />
5      <function id="3" type="Interpretation" modality="voice" />
6      <function id="4" type="Styling" modality="voice" />
7    </functions>
8    <couplingprofile>
9      <function id="1">
10       <providedports>
11         <providedport portid="P1">
12           <methodname>render</methodname>
13           <methodsignature>
14             <param name="url" type="String" />
15             <param name="xhtmlDescription" type="String" />
16             <param name="options" type="String" />
17             <returnvalue />
18           </methodsignature>
19           <description>
20             <qosattrib type="format" value="vxml" />
21           </description>
22         <providedport>
23         <providedport portid="P2">
24           <methodname>reactEvent</methodname>
25           <methodsignature>
26             <param name="clientID" type="String" />
```

```
27            <param name="event" type="String" />
28            <returnvalue />
29          </methodsignature>
30        <providedport>
31      <requiredports>
32        <requiredport cID="3" functionid="1" portid="P1" />
33      </requiredports>
34    </function>
35  </couplingprofile>
36  <requirements>
37    <hardware>
38      <inputdevice type="voice" />
39      <outputdevice type="voice" />
40    </hardware>
41    <software />
42  </requirements>
43 </componentprofile>
```

Listing B.5: Komponentenprofil des Voice-Browsers mit Interpretation

B.6. Voice-Browser ohne Interpretation

```
1  <componentprofile cID="6">
2    <functions>
3      <function id="1" type="Rendering" modality="voice" />
4      <function id="2" type="Recognition" modality="voice" />
5      <function id="3" type="Interpretation" modality="voice" />
6      <function id="4" type="Styling" modality="voice" />
7    </functions>
8    <couplingprofile>
9      <function id="1">
10       <providedports>
11         <providedport portid="P1">
12           <methodname>render</methodname>
13           <methodsignature>
14             <param name="url" type="String" />
15             <param name="xhtmlDescription" type="String" />
16             <param name="options" type="String" />
17             <returnvalue />
18           </methodsignature>
19           <description>
20             <qosattrib type="format" value="vxml" />
21           </description>
22         <providedport>
```

```
23        <providedport portid="P2">
24          <methodname>reactEvent</methodname>
25          <methodsignature>
26            <param name="clientID" type="String" />
27            <param name="event" type="String" />
28            <returnvalue />
29          </methodsignature>
30        <providedport>
31      <requiredports>
32        <requiredport cID="3" functionid="1" portid="P1" />
33      </requiredports>
34    </function>
35    <function id="3">
36      <providedports>
37        <providedport portid="P3">
38          <methodname>resultReceived</methodname>
39          <methodsignature>
40            <param name="result" type="String" />
41          </methodsignature>
42          <returnvalue />
43        </providedport>
44      </providedports>
45      <requiredports>
46        <requiredport>
47          <requiredport cID="7" functionid="1" portid="P1" />
48        </requiredport>
49      </requiredports>
50    </function>
51  </couplingprofile>
52  <requirements>
53    <hardware>
54      <inputdevice type="voice" />
55      <outputdevice type="voice" />
56    </hardware>
57    <software />
58  </requirements>
59 </componentprofile>
```

Listing B.6: Komponentenprofil des Voice-Browsers ohne Interpretation

B.7. Voice-Interpretation

```
1 <componentprofile cID="7">
2   <functions>
```

```
3     <function id="1" type="Interpretation" modality="voice" />
4   </functions>
5   <couplingprofile>
6     <function id="1">
7       <providedports>
8         <providedport portid="P1">
9           <methodname>reportEvent</methodname>
10          <methodsignature>
11            <param name="clienturl" type="String" />
12            <param name="event" type="String" />
13            <returnvalue />
14          </methodsignature>
15        <providedport>
16        <providedport portid="P2">
17          <methodname>setGrammar</methodname>
18          <methodsignature>
19            <param name="grammar" type="String" />
20            <returnvalue />
21          </methodsignature>
22        <providedport>
23      <requiredports>
24        <requiredport cID="6" functionid="3" portid="P3" />
25      </requiredports>
26    </function>
27  </couplingprofile>
28  <requirements>
29    <hardware />
30    <software />
31  </requirements>
32 </componentprofile>
```

Listing B.7: Komponentenprofil der Voice-Interpretation-Komponente

C

Nutzerbefragungen zur Synchronisierung

Die Nutzerbefragungen wurde von Martin Knechtel im Rahmen seiner Diplomarbeit [Kne07] durchgeführt. Die erste Befragung befasste sich mit den tolerablen Verzögerungszeiten bei Synchronisierung auf *event*-Ebene. Der Nutzer bekam zwei Fenster (HTML-Browser in Java implementiert) mit dem gleichen Inhalt auf einem Bildschirm angezeigt (Abbildung C.1 oben). Über den Controller (Abbildung C.1 unten) kann die Verzögerung eingestellt werden, die auf die Ereignisverarbeitung angewendet wird. Eine gewählte Verzögerung von beispielsweise 1 Sekunde bedeutet, dass zwischen dem Tastendruck und dem Erscheinen des Buchstabens mindestens eine Sekunde vergeht (zusätzlich zu der Zeit für die ohnehin notwendigen Verarbeitungsschritte). Bei kontrollierter Aktualisierung erfolgt die Ausgabe gleichzeitig in beiden Fenstern, während bei vorläufiger Aktualisierung das primäre Fenster sofort eine Ausgabe erzeugt und das andere Fenster eine Sekunde später reagiert.
Die Nutzer wurden gebeten, Eingaben in die Textfelder der Browser zu tätigen. Dabei konnten die zwei Browser wahlweise mit kontrollierter oder vorläufiger Aktualisierung betreiben (Einstellungen „distribution“ bzw. „confirmation“ im Browserfenster oben). Den Nutzern wurden folgende Fragen gestellt:

1. Wie hoch ist die maximale Verzögerung, damit die Nutzerschnittstelle gerade noch bedienbar ist?

2. Wie hoch ist die maximale Verzögerung, damit die Nutzerschnittstelle flüssiges Feedback liefert?

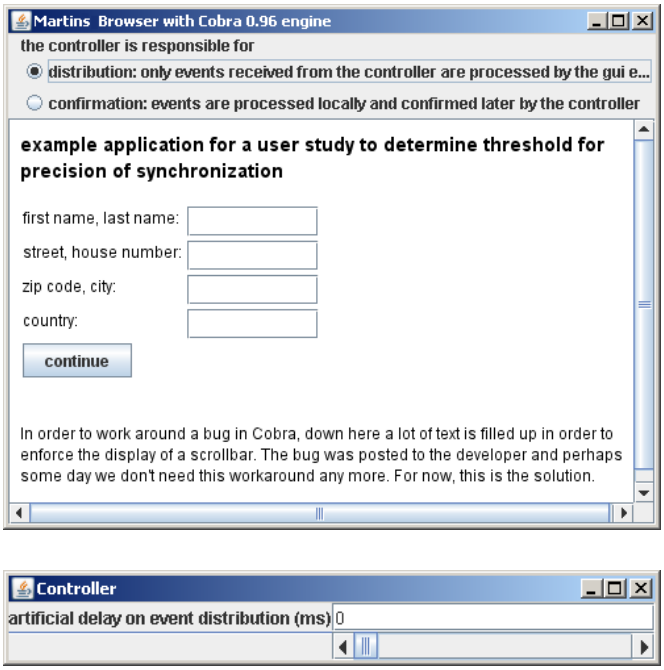

Abbildung C.1.: Prototyp 1 zur Ermittlung tolerabler Verzögerungen bei Synchronisierung auf *event*-Ebene (oben: Nutzerschnittstelle, unten: Controller zur Einstellung der Verzögerung, aus [Kne07]

Das Ergebnis der Nutzerbefragung ist in Tabelle C.1 dargestellt. Die tolerable Verzögerung für flüssiges Feedback bei kontrollierter Aktualisierung beträgt ca. 300ms, bei ca. 800ms Verzögerung ist die Nutzerschnittstelle gerade noch so bedienbar. Bei kontrollierter Aktualisierung ist flüssiges Feedback bei ca. 950ms gegeben und die Benutzung bei ca. 2700ms gerade noch so möglich. Zwischen kontrollierter und vorläufiger Aktualisierung und zwischen flüssigem und gerade noch akzeptablem Feedback besteht ebenfalls ungefähr der Faktor 3.
In der zweiten Nutzerbefragung sollte ermittelt werden, wie sich eine Synchronisierung auf *page*-Ebene auf die Erscheinung der Anwendung gegenüber dem Nutzer auswirkt. Dazu wurden zwei HTML-Seiten erstellt, die sich jeweils verlinken, sodass die Navigation durch eine Web-Anwendung simuliert wird. Es kommt wiederum eine verteilte Nutzerschnittstelle mit zwei Browsern zum Einsatz, wobei sich der eine auf einem PC und der andere auf einem PDA befindet. Wahlweise kann bei Aktualisierung der Seiten die Synchronisierung durchgeführt werden. Die Nutzer wurden nach einigen Dialogaktualisierungen gefragt, in welchem Modus (mit oder ohne Synchronisierung) die Anwendung konsistenter erscheint. Das Ergebnis ist in Tabelle C.2 dargestellt. Demzufolge fanden 6 von 7 Befragten die synchronisierte Darstellung konsistenter.

Teilnehmer	kontrollierte Aktualisierung		vorläufige Aktualisierung	
	I	II	I	II
1	1000	500	10000	500
2	1000	500	4000	2000
3	1020	340	1020	340
4	2000	800	1200	800
5	1010	250	750	200
6	730	510	670	340
7	510	110	1350	730
8	400	110	350	110
9	1080	310	3000	2250
10	750	500	2500	1500
11	1000	90	8000	2000
12	200	100	500	150
13	200	70	2000	1500
Mittelwert	838,46	322,31	2718,46	955,38

Tabelle C.1.: Tolerable Verzögerungen (in ms, maximale Verzögerung für Bedienbarkeit (I) und flüssiges Feedback (II), nach [Kne07])

Anzahl der Probanden:	7
konsistenterer Eindruck ohne Ausgabesynchronisierung:	0
konsistenterer Eindruck mit Ausgabesynchronisierung:	6
kein wahrnehmbarer Unterschied:	1

Tabelle C.2.: Befragungsergebnisse zur Verwendung der Ausgabesynchronisation auf *page*-Ebene (aus [Kne07])