Universität Rostock Traditio et Innovatio

Service-basierte Infrastruktur für pervasive Lehr- und Lernarrangements

Der Fakultät für Informatik und Elektrotechnik
der Universität Rostock zur Erlangung des
akademischen Grades eines

Dr.-Ing.

vorgelegte Dissertation von

Herrn Dipl.-Inf. Raphael Zender

aus Wismar

Datum der Einreichung: 08. Oktober 2010
Datum der Verteidigung: 16. März 2011

1. Gutachter: Prof. Dr. habil Djamshid Tavangarian (Universität Rostock)
2. Gutachter: Prof. Dr. habil Peter Forbrig (Universität Rostock)
3. Gutachter: Prof. Dr. Christoph Meinel (Hasso-Plattner-Institut)

urn:nbn:de:gbv:28-diss2011-0071-5

Raphael Zender
Großbeererstr. 112
14482 Potsdam

Hiermit versichere ich, dass ich die von mir vorgelegte Arbeit selbstständig verfasst habe, dass ich die verwendeten Quellen, Internet-Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Rostock, den 08. Oktober 2010

_____

Raphael Zender

Zusammenfassung

Mit der wachsenden Bedeutung von Mobilität, allgegenwärtigem IT-Zugang und dynamischer Allokation anpassbarer Inhalte ist eine zunehmende Integration von Technologien des Pervasive Computing in Lehr- und Lernprozesse erkennbar. Diese Dissertationsschrift analysiert die infrastrukturellen Herausforderungen für pervasive Umgebungen im Allgemeinen und pervasive Hochschulen im Speziellen. Sie bewertet basierend auf dieser Analyse die Eignung heutiger Kommunikationsmodelle als infrastrukturelle Basis und motiviert als Ergebnis dieses Vergleichs den Einsatz Service-orientierter Architekturen (SOA).

Da sich in den jeweiligen Anwendungsbereichen der einzelnen Komponenten pervasiver Umgebungen unterschiedliche SOA-Implementierungen etabliert haben, müssen Interoperabilitätsmechanismen bereitgehalten werden. Dieser Arbeit stellt bekannte Interoperabilitätsansätze vor, diskutiert und bewertet diese hinsichtlich ihrer Eignung zur Interoperabilität in pervasiven Umgebungen. Es wird anschließend ein systematischer Interoperabilitätsansatz vorgestellt. Für die zuvor klassifizierten SOA-Implementierungen werden weiterhin Strategien zu deren Integration in diesen Ansatz erarbeitet und diskutiert.

Die erzielten Ergebnisse werden im Rahmen universitärer Fallstudien evaluiert. Es wird gezeigt, dass durch die resultierende Systematik, Transparenz und Flexibilität traditionelle Lehr- und Lernarragements bereichert und neue Innovationen im universitären Umfeld entstehen.

Abstract

The increasing importance of mobility, ubiquitous IT access, and dynamic allocation of adaptable content causes more and more integration of pervasive computing technology into learning processes. This thesis analyzes the infrastructural challenges for pervasive environments in general and especially pervasive universities. Based on this analysis, the applicability of todays communication models as infrastructural basis will be assessed. Accordingly, the use of Service-oriented Architectures (SOA) will be motivated.

Pervasive environments consist of components from different areas of application. As different SOA instances have been established for these areas, pervasive environments require appropriate interoperability mechanisms. This thesis introduces existing interoperability approaches. They will be discussed and assessed due to their applicability for those environments. Furthermore, a more systematic interoperability approach will be introduced and the integration of existing SOA instances into this approach wil be described as well as discussed.

Results will be evaluated in University-related case studies. It will be shown, that the emerged systematic, transparency, and flexibility enriches traditional learning arrangements as well as creates new innovations for universities.

Danksagung

Diese Dissertation entstand während meiner Zeit als Stipendiat des Graduiertenkollegs *Multimodal Smart Appliance Ensembles for Mobile Applications* (MuSAMA) am Lehrstuhl für Rechnerarchitektur der Universität Rostock. Ohne die organisatorische und fachliche Grundlage des durch die Deutsche Forschungsgemeinschaft (DFG) geförderten Graduiertenkollegs wäre diese Arbeit nicht möglich gewesen.

An erster Stelle möchte ich meinem Chef und Mentor Prof. Djamshid Tavangarian für seine wertvolle fachliche und menschliche Unterstützung danken. In den lehrreichen und stets angenehmen Jahren am Lehrstuhl hatte er stets ein offenes Ohr und nicht selten unkonventionelle Lösungen für die alltäglichen Herausforderungen im Alltag eines Doktoranden. Weiterhin gilt mein Dank Prof. Peter Forbrig und Prof. Christoph Meinel für ihr Interesse an meiner Arbeit sowie die Diskussionen und Anregungen im Vorfeld.

Einen nicht unwesentlichen Anteil am Gelingen dieser Arbeit hatte Prof. Ulrike Lucke die es verstand, mich immer wieder aufs Neue zu motivieren und inspirieren und die mich mit den Gelegenheiten und Tücken der wissenschaftliche Forschung vertraut machte. Auch den weiteren Kollegen am Lehrstuhl für Rechnerarchitektur, allen voran Dr. Daniel Versick, Dr. Heiko Kopp sowie das Team um Peter Eschholz, gilt mein aufrichtiger Dank für die vielfältige Unterstützung und familiäre Atmosphäre in den vergangenen Jahren. Es ist großartig ein Teil dieses herausragenden Teams zu sein.

Zudem danke ich Dr. Frank Burghardt für das Korrekturlesen meiner Arbeit sowie die praktischen Hinweise durch die diese Schrift weiter an Qualität gewonnen hat.

Auch meinen Mitstreitern und den Leitern des Graduiertenkollegs MuSAMA möchte ich meinen aufrichtigen Dank sowie großen Respekt für die außerordentlichen Leistungen aussprechen, an denen ich Teil haben durfte. Dabei ist besonders Dr. Enrico Dressler hervorzuheben, mit dem zusammen ich viele Arbeiten und Veröffentlichungen, die zu dieser Dissertation führten, verwirklichen konnte. Ebenso danke ich den vielen Studierenden und Absolventen, die mich auf meinem Weg zu dieser Dissertationsschrift auf vielfältige Art begleitet haben.

Meinen Eltern bin ich dankbar für die soliden Werte und Charaktereigenschaften, die sie mir stets auf liebevolle und aufopferungvolle Weise vermittelt haben. Ich weiß, dass ich ohne sie nicht dort stehen würde wo ich jetzt stehe. Sie sind die wertvollsten, aufrichtigsten und stärksten Eltern, die ich mir hätte wünschen können und was sie für mich und meine Geschwister leisteten ist mit nichts auf der Welt aufzuwiegen.

Ich danke auch meinem guten Freund Peter Danielis für unsere fachlichen Diskussionen, viele neue Ansätze aber auch die nötige Ablenkung nicht nur während der Durchführung dieser Dissertation sondern in den vergangenen elf Jahren.

Mein größter Dank jedoch gilt meiner Freundin Ines, die mir mit aufrichtiger Liebe, wundervollem Verständnis und so manchem aufmunternden Wort bei dieser Dissertation zur Seite stand. Sie hat mir die Kraft, Inspiration und wertvolle Hilfe gegeben, die diese Arbeit zum Abschluss geführt haben.

Inhaltsverzeichnis

1 Einleitung **7**
- 1.1 Motivation: Pervasive Umgebungen . . . 7
  - 1.1.1 Einführung in Pervasive Computing . . . 7
  - 1.1.2 Herausforderungen pervasiver Umgebungen . . . 9
  - 1.1.3 Multimodal Smart Appliance Ensembles for Mobile Applications (MuSAMA) . . . 10
- 1.2 Anwendungsgebiet: Moderne Bildungseinrichtungen . . . 11
  - 1.2.1 Organisatorische Herausforderungen . . . 11
  - 1.2.2 Technologische Herausforderungen . . . 12
- 1.3 Zielstellung dieser Arbeit . . . 13
- 1.4 Aufbau dieser Arbeit . . . 14

2 Service-orientierte Architekturen – Eine Bestandsaufnahme **15**
- 2.1 Kommunikationsmodelle für pervasive Umgebungen . . . 15
- 2.2 Das SOA-Konzept und seine Implementierungen . . . 20
  - 2.2.1 Web Services – SOA für das Internet . . . 22
  - 2.2.2 UPnP – SOA für Ad-hoc-Gerätekooperation . . . 27
  - 2.2.3 Jini – SOA für verteilte Java-Anwendungen . . . 29
  - 2.2.4 DNS-SD – SOA über vorhandene DNS-Infrastruktur . . . 30
  - 2.2.5 Bluetooth SDP – SOA für Personal Area Networks . . . 32
  - 2.2.6 Weitere SOA-Umsetzungen . . . 33
- 2.3 Technische Klassifizierungen . . . 34
  - 2.3.1 Einsatz von Brokern . . . 34
  - 2.3.2 Aktivität des Service Busses . . . 35
  - 2.3.3 Spezifikation der Service-Nutzung . . . 36
  - 2.3.4 Technologiebindung . . . 36
  - 2.3.5 Übersicht der SOA-Klassifizierungen . . . 37
- 2.4 SOA-Kommunikationsmodell für pervasive Umgebungen . . . 37

3 SOA-Interoperabilität für pervasive Umgebungen **39**
- 3.1 Ansätze zur SOA-Interoperabilität . . . 39
  - 3.1.1 Technologiebrücken . . . 40
  - 3.1.2 Abstrakter Provider . . . 41
  - 3.1.3 Abstrakter Consumer . . . 42
  - 3.1.4 Zentrale Vereinheitlichung . . . 43

3.2 Der General Purpose Access Point (GPAP) . . . 45
3.2.1 Zentrale Vereinheitlichung auf dem GPAP Service Layer . . . 47
3.2.2 Die Service Technology-independent Language (STiL) . . . 49
3.2.3 GPAP-Plugins - Strategien zur Einbindung von SOA-Klassen . . . 56
3.2.4 Topologien einer SOA-Community . . . 59
3.3 Nutzung von Kontextinformationen . . . 63
3.3.1 Kontextarten und -Modelle . . . 64
3.3.2 Kontextinformationen in service-orientierten Architekturen . . . 65

4 Implementierung des GPAP Service Layers 68
4.1 GPAP Service-Layer . . . 68
4.1.1 `ServiceLayer`-Klasse . . . 68
4.1.2 Discovery-Manager und -Plugins . . . 70
4.1.3 Provision-Manager und -Plugins . . . 72
4.1.4 Broker-Manager und -Plugins . . . 74
4.1.5 Service Bus . . . 75
4.1.6 Community-Implementierungen . . . 77
4.2 Evaluierung und Kontext . . . 79

5 Evaluierung: Die Universität der Zukunft 80
5.1 Die Pervasive Universität . . . 80
5.1.1 Der Weg zur Pervasiven Universität . . . 80
5.1.2 Service-orientierte Universität . . . 82
5.1.3 Ausgangssituation der Universität Rostock . . . 86
5.2 Service-orientierte Lehr- und Lernarrangements . . . 87
5.2.1 Basis-Services . . . 87
5.2.2 Fallstudie: Lecture as a Service . . . 89
5.2.3 Fallstudie: Immersive Learning . . . 92
5.3 SOA-übergreifendes Lehr- und Lernarrangement . . . 97
5.3.1 Zentrale Vereinheitlichung ohne STiL . . . 97
5.3.2 Fallstudie: Immersive Messaging . . . 98
5.4 Kontextbasierte Arrangements . . . 100
5.4.1 Fallstudie: Kontext als Service . . . 101
5.4.2 Fallstudie: Kontextbasierte Stud.IP-Anpassung . . . 102
5.4.3 Fallstudie: Kontextbasierte Jukebox . . . 104

6 Zusammenfassung und Ausblick 107
6.1 Zusammenfassung und erzielte Ergebnisse . . . 107
6.2 Ausblick und weiterführende Fragestellungen . . . 109

Thesen 112

A XML-Schema der Service Technology-independent Language (STiL) 114

B Service-Typen der STiL 119

Abkürzungsverzeichnis 121

Literaturverzeichnis **124**

Kapitel 1

Einleitung

1.1 Motivation: Pervasive Umgebungen

1.1.1 Einführung in Pervasive Computing

Den modernen Mensch des 21. Jahrhunderts umgibt eine Wolke aus Informationstechnologien (IT). Diese besteht aus einer Vielzahl an Geräten und Anwendungen, die Informationen konsumieren, verarbeiten, liefern oder umwandeln. Das nach wie vor gültige *Mooresche Gesetz* der Verdopplung der Integrationsdichte von Transistoren auf einem Chip alle achtzehn Monate, führt zu immer kleineren Gerätedimensionen bei steigender Leistung. So reicht das breite Gerätespektrum heute von Personal Computern und Laptops über das allgegenwärtige Mobiltelefon oder sogar Smartphone bis hin zu mehr oder weniger unsichtbarer IT in unseren Wohnungen und Fahrzeugen. In seiner Vision des *Ubiquitious Computing* hat Mark Weiser bereits 1991 die drastisch zunehmende IT-Durchdringung des Alltags, aber parallel dazu auch das Verschwinden dieser Technologievielfalt aus unserem unmittelbaren Sichtfeld prognostiziert:

> Die Technologien, die aus dem direkten Blickfeld verschwinden, sind am einflussreichsten. Sie verschmelzen mit dem alltäglichen Leben, bis sie daraus nicht mehr wegzudenken sind. [...] Erst wenn die Dinge verschwinden [...] sind wir frei diese zu nutzen ohne uns auf sie zu konzentrieren und können uns darüber hinaus neuen Zielen öffnen.
>
> (frei übersetzt aus [Weiser 91])

Die Pervasive Computing-Forschung hat dabei nicht nur das Ziel, aus Nutzersicht die Konzentration auf Geschäftsprozesse und persönliche Aufgaben zu ermöglichen, sondern sogar strebt eine proaktive Unterstützung beim Erreichen dieser Ziele an [Encarnação 05]. Das Potential der resultierenden pervasiven Systeme ist beträchtlich. An dieser Stelle sei nur eine Auswahl der beliebtesten Beispiele angeführt:

- Displays, die abhängig von der aktuellen Nutzersituation selbstständig nützliche Informationen anzeigen (z. B. digitale Zeitung zum Frühstück)

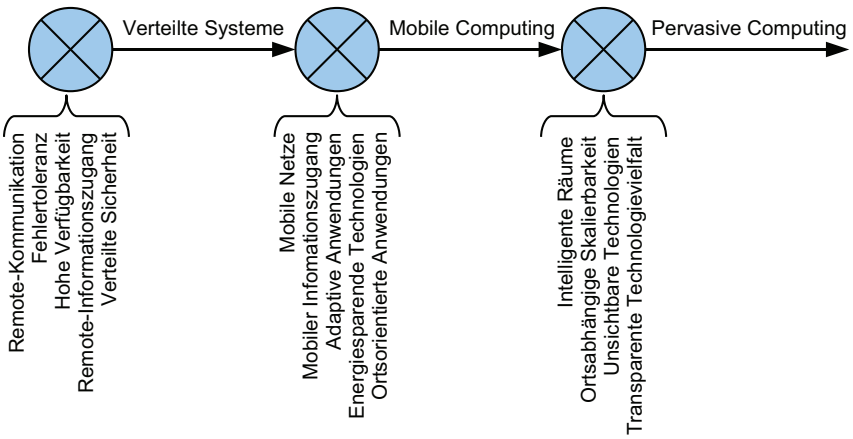


Abbildung 1.1: Die Evolution moderner Computing-Paradigmen (nach [Satyanarayanan 01]) führt von Verteilten Systemen über das Paradigma des Mobile Computing zum Pervasive Computing.

- Räume, die sich vollautomatisch an Nutzerbedürfnisse anpassen und beispielsweise Temperatur, Helligkeit und selbst die Ausrichtung von Möbeln und anderen Gegenständen kontrollieren
- Fahrzeuge, die selbstständig miteinander kommunizieren, um sich beispielsweise über Staus bzw. Unfälle zu informieren und ihrem Fahrer alternative Routen zu empfehlen
- Persönliche Nutzerendgeräte (z. B. Laptops, Mobiltelefone) von Studenten, die automatisch und spontan untereinander Lernmaterialen bzw. andere relevante Informationen autauschen

Pervasive Computing bezeichnet kein klar abgrenzbares Konzept bzw. eine definierbare Methodik, sondern ein Forschungsfeld und ein damit verbundenes Paradigma, welches zum Ziel hat die exisitierende allgegenwärtige Informationsverarbeitung möglichst transparent nutzbar zu machen, um den Nutzer zu unterstützen. Selbst die Bezeichnung des Forschungsfeldes/Paradigmas variiert. Während Mark Weiser noch von *Ubiquitious Computing* sprach, haben sich inzwischen auch der industriell geprägte Begriff des *Pervasive Computing* sowie vor allem in Europa *Ambient Intelligence* durchgesetzt. Der Begriff *Ubiquitious Computing* wird heute vorwiegend für langfristige und teils visionäre Konzepte und Ideen verwendet, während *Ambient Intelligence* vor allem den Faktor der Intelligenz der Nutzerumgebung thematisiert. In dieser Arbeit wird der Begriff *Pervasive Computing* genutzt, da dieser verstärkt die Verwendung der bereits exisitierenden und den Alltag durchdringenden Technologien fokussiert, um allgegenwärtige Informationsverarbeitung bereits kurzfristig verfügbar zu machen [Langheinrich 03].

Weisers Vision hat sich evolutionär aus vorhergehenden Computing-Paradigmen entwickelt, wie Abbildung 1.1 illustriert [Satyanarayanan 01]. Ihre Wurzeln liegen in den 1970er bis 1990er Jahren, in den Forschungen zu verteilten Rechensystemen. Die damals entwickelten Werkzeuge, Protokolle und Modelle zur Langstreckenkommunikation, Steigerung der Fehlertoleranz und Verfügbarkeit sowie zu verteilten Informationen und Sicherheitsmechanis-

men finden sich noch heute in den Forschungen zu Pervasive Computing wieder. Mit der Einführung erschwinglicher mobiler Clients wurden Verteilte Systeme in den frühen 1990er Jahren, um Technologien und Mechanismen für mobile Netze und somit mobilen Zugang zu Informationen sowie adaptive und ortsorientierte Anwendungen bereichert. Auch der Faktor der Energieeffizienz hatte für mobile Geräte einen höheren Stellenwert als für stationäre Systeme und führte damals wie heute zu energiesparenden Technologien. Nach [Satyanarayanan 01] werden vor allem vier Forschungsbereiche als weitere Motoren aber auch Herausforderungen für pervasive Systeme betrachtet:

- **Intelligente Räume** mit eingebetten Systemen, die den Raum weitestgehend an die Bedürfnisse der Nutzer anpassen
- **Ortsabhängige Skalierbarkeit** zur effizienten Kommunikation zwischen Nutzern und intelligenten Räumen, in Abhängigkeit von ihrer räumlichen Beziehung zueinander
- **Unsichtbare Technologien**, die aufgrund ihrer Integration oder Größe kaum mehr bewusst vom Nutzer wahrgenommen werden
- **Transparente Technologievielfalt**, die Nutzern unabhängig von der verfügbaren Hard- oder Software eine möglichst einheitliche bzw. gewohnte Umgebung präsentiert

Zusätzlich können weitere Pervasive Computing-Motoren wie Fortschritte in der Nanotechnik und Mikroelektronik (z. B. elektronisches Papier, RFID, mikroskopische Sensoren) [Langheinrich 03], Künstliche Intelligenz, Web 2.0 und Cloud Computing identifiziert werden. Im folgenden Abschnitt wird auf die spezifischen Entwicklungen und Herausforderungen pervasiver Umgebungen als konkretes Pervasive Computing-Anwendungsfeld eingegangen.

1.1.2 Herausforderungen pervasiver Umgebungen

Eine konkrete Ausprägung der Pervasive Computing-Forschung sind *pervasive Umgebungen*. Diese weitgehend transparent von IT durchdrungenen Räume oder sogar raumübergreifenden Bereiche sind in der Lage Nutzeraktivitäten zu erfassen, auszuwerten und darauf derart zu reagieren, dass sie den Nutzer proaktiv im Erreichen seiner Ziele unterstützen. Neben den bereits erläuterten Herausforderungen aus dem Bereich des Pervasive Computing, werden Entwickler pervasiver Umgebungen mit einer Reihe zusätzlicher und konkretisierter Herausforderungen konfrontiert, von denen die wichtigsten in diesem Abschnitt näher betrachtet werden.

Der analog zu pervasiven Umgebungen verwendete Begriff *Smart Environment* verdeutlicht, dass ein scheinbar intelligentes Verhalten des Raumes bzw. der Umgebung des Nutzers angestrebt wird. Grundlage für dieses Verhalten ist eine zuverlässige und effiziente **Intentionserkennung**. Diese erfordert wiederum eine ausreichende Sensorik (z. B. zur Lokalisierung oder Authentifizierung von Nutzern) innerhalb der pervasiven Umgebung sowie **Modelle und Deskriptoren** für physische Umgebungen. Ferner sind **Methoden zur Strategieentwicklung** (z. B. Plannungsalgorithmen) erforderlich, die auf Grundlage der Nutzerintentionen und der Umgebungsmodelle sowie mit Hilfe geeigneter Aktoren ein möglichst proaktives Systemverhalten realisieren.

Bei vielen so genannten “intelligenten Umgebungen” wie dem *Reactive Video-Conferencing Room* [Cooperstock 97] oder Microsofts *EasyLiving*-Projekt [Brumitt 00] muss bereits der Entwickler beim Design der Umgebung wissen, welche Zustände diese haben kann und wie die dazu passende Systemreaktion aussieht. Das Resultat dieser Arbeiten ist, dass pervasive Umgebungen heute statische und teure Einzellösungen sind, die fast ausschließlich im Bereich der akademischen oder industriellen Forschung eine Rolle spielen. Damit diese auch für Privatanwender erschwinglich werden ist ein Umdenken in Richtung dynamischer Geräteansammlungen (Ensembles [Heider 06]) aus bereits heute weit verbreiteten Geräten (z. B. Laptops, Smartphones) anstelle statischer vorkonfigurierter Systeme erforderlich. Die neue Dynamik erfordert eine **Infrastruktur, die ohne Vorwissen in der Lage ist unterschiedliche Geräte und Anwendungen zu integrieren**. Sie muss weiterhin transparent die **Heterogenität überbrücken**, die durch die Verwendung von bereits auf dem Markt befindlichen Geräten entsteht und trotzdem deren vollen Funktionsumfang zur Verfügung stellen.

Zukünftige pervasive Umgebungen werden sich demnach durch ein hohes Maß an Dynamik und Heterogenität auszeichnen und daher vor völlig neue Herausforderungen gestellt. Die vorliegende Arbeit entstand im Rahmen des Graduiertenkollegs Multimodal Smart Appliance Ensembles for Mobile Applications (MuSAMA) der Deutschen Forschungsgemeinschaft (DFG), welches diese Herausforderung fokussiert und im folgenden Abschnitt näher betrachtet wird.

1.1.3 Multimodal Smart Appliance Ensembles for Mobile Applications (MuSAMA)

Dem Graduiertenkolleg (GRK) *Multimodal Smart Appliance Ensembles for Mobile Applications* (MuSAMA) liegt die These zugrunde, dass zukünftige pervasive Umgebungen nicht fest installiert und vorkonfiguriert sein werden, sondern sich in ihrer Zusammensetzung unvorhersehbar ändern können. Die einzelnen Komponenten (Geräte und Anwendungen) einer bestimmten Zusammensetzung (Ensemble) müssen in der Lage sein miteinander zu kooperieren und den Nutzer somit zielgerichtet, spontan und ohne menschliche Anleitung zu unterstützen [Kirste 08]. Dabei werden vier Forschungsschwerpunkte fokussiert:

1. **Kontexterkennung und -analyse** für Fragestellungen der Erfassung, Modellierung und Bewertung von Umgebungsparametern und den daraus abgeleiteten Situationen (Kontext).
2. **Multimodale Interaktion und Visualisierung** für Fragestellungen der verteilten Darstellung von Informationen und der Interaktion zwischen Nutzern und der pervasiven Geräteinfrastruktur
3. **Intentionserkennung und Strategieentwicklung** für Fragestellungen zur Ableitung von Nutzerzielen sowie der Entwicklung von Strategien, um diese umzusetzen
4. **Datenhaltung, Ressourcen- und Infrastrukturmanagement** für Fragestellungen der Kommunikation innerhalb der unterliegenden Infrastruktur und das Management der verfügbaren Ressourcen – In diesem Forschungsschwerpunkt ist auch die vorliegende Arbeit angesiedelt.

Die in den Forschungsschwerpunkten erzielten Ergebnisse werden im Rahmen des Szenarios einer “Pervasiven Universität” evaluiert, einer weitläufigen und raumübergreifenden pervasiven Umgebung aus dem Anwendungsbereich der universitären Aus- und Weiterbildung. Insbesondere verschiedene Varianten eines Meeting-Szenarios in einem *Smart Room* (als Teil der pervasiven Universität) vereinen den Großteil der MuSAMA-Arbeiten. Eine dieser Varianten wird im folgenden näher betrachtet, um einen Überblick über die Ziele des GRK zu erhalten.

Im Meeting-Szenario wird von einem mit der notwendigen Infrastruktur ausgestatteten Smart Room ausgegangen, den die Teilnehmer eines Meetings vor dessen Beginn betreten. Der Raum erkennt automatisch, welche Geräte vorhanden sind und welche Anzeige- bzw. Kooperationsmöglicheiten dadurch bestehen. Anhand einer Agenda und Identifikation der einzelnen Personen (z. B. Sensor-Tag) bzw. deren Position im Raum kann der aktuell Vortragende ermittelt werden. Der Raum kann dessen Laptop-Bildschirminhalt automatisch auf eine Leinwand oder freie Fläche projizieren, den Raum abdunkeln und ggf. weitere Schritte ausführen bevor der Vortragende beginnt. Nach dem Vortrag könnte ein Wechsel der Vortragendenrolle (z. B. durch eine ausreichend aussagekräftige Veränderung der Teilnehmer-Positionen erkannt) erfolgen oder der Raum könnte eine Diskussionsrunde unterstützen (z. B. mit Brainstorming-Software oder dem automatischen Anzeigen für die Diskussion relevanter Webseiten). Nach dem Meeting wäre ein Verteilen der automatisch erfassten Meeting-Ergebnisse an alle Teilnehmer sinnvoll. Dabei wäre eine Anpassung der Inhalte beispielsweise für die jeweiligen Geräte oder Rollen der Teilnehmer denkbar.

Dieses MuSAMA-Szenario stellt nur ein Teil der Möglichkeiten und Herausforderungen dar, die durch den Einzug pervasiver Systeme in den universitären Alltag entstehen. Diese dienen als Referenzumgebung für die in der vorliegenden Dissertationsschrift entwickelten Infrastruktur und der Evaluierung der erzielten Ergebnisse. Daher erfolgt im folgenden Abschnitt eine nähere Betrachtung der Herausforderungen moderner Bildungseinrichtungen auf ihrem Weg zu pervasiven Institutionen.

1.2 Anwendungsgebiet: Moderne Bildungseinrichtungen

1.2.1 Organisatorische Herausforderungen

Vor allem durch die zunehmende Globalisierung stehen deutsche Hochschulen heute vor einer Reihe organisatorischer Herausforderungen. Im Rahmen der Entwicklung eines einheitlichen europäischen Hochschulwesens durch den Bologna-Prozess wurden in den vergangenen Jahren bereits europaweit vergleichbare Abschlüsse und Anerkennungen von Studienleistungen eingeführt. Unter anderem wird die **Mobilität europäischer Studenten** durch diese Maßnahmen gesteigert.

Insbesondere deutsche Hochschulen sehen sich außerdem mit einer steigenden Anzahl ausländischer Studenten konfrontiert, die sich von dem Studium in Deutschland eine Verbesserung der eigenen Berufschancen erhoffen. Fast die Hälfte der Bildungsausländer stammt dabei aus Ländern mit einem geringen Pro-Kopf-Einkommen [W. Isserstedt and J. Link 08], wodurch die Reise nach Deutschland und das Studium für diese Zielgruppe erschwert wird.

Durch **hochschul- und länderübergreifende Dienste** (z. B. Austausch von Lerninhalten, Zusammenarbeit in Lehre, Forschung und Verwaltung) können diese Kosten gesenkt und sogar ein deutsches Studium im Heimatland ermöglicht werden (z. B. im Rahmen des Programms "Studienangebote deutscher Hochschulen im Ausland" des Deutscher Akademischer Austauschdienst, DAAD).

In Deutschland wiederum weckt die Einführung von Studiengebühren für die Verbesserung von Studium und Lehre an vielen Hochschulen in Studenten und Dozenten das berechtigte **Verlangen nach hochwertigen Inhalten und Lernmedien**. Auch diese könnten über Kooperationskonzepte aus anderen Hochschulen bezogen und in den lokalen Lehrplan eingebunden werden, falls entsprechende Kompetenzen oder Ressourcen an der eigenen Hochschule fehlen.

Organisatorische Herausforderungen wie die wachsende Studentenmobilität und der Bedarf nach hochschulübergreifenden Diensten wie dem Austausch von hochwertigen Lehrinhalten erfordern eine flexible und systematische Infrastruktur, die klare Schnittstellen für externe Kooperationspartner und Gäste bietet. Zusätzlich exisitieren eine Reihe technologischer Herausforderungen für moderne Bildungseinrichtungen, die weitere Anforderungen an die Infrastruktur stellen und daher im Folgenden näher beleuchtet werden.

1.2.2 Technologische Herausforderungen

Im Jahr 2010 präsentierte der renomierte und jährlich erscheinende Horizon Report sechs Technologien, die sich innerhalb der nächsten fünf Jahre in Lehre, Lernen oder kreativer Forschung wahrscheinlich durchsetzen werden [Johnson 10]:

- **Mobile Computing** (innerhalb der nächsten 12 Monate): Die Nutzung mobiler, netzwerkfähiger Endgeräte (z. B. Smartphones und Netbooks) war bereits im vergangenen Horizon Report [Johnson 09] eine der Schlüsseltechnologien für moderne Hochschulen. Ihre Nutzung hat drastisch zugenommen und somit ein hohes Potential für tragbare Arbeitswerkzeuge für Produktivität, Lernen und Kommunikation eröffnet.
- **Open Content** (innerhalb der nächsten 12 Monate): Die freie Verfügbarkeit elektronisch aufbereiteter Kursinhalte ist als Ausprägung der *Web 2.0*-Kultur und Reaktion auf die steigenden Ausbildungskosten sowie den Wunsch nach Bildungszugang auch in strukturschwachen Regionen zu verstehen. Analog zum Prinzip des Pervasive Computings, können Lernende durch Open Contant selbst bestimmen wann und wie sie Inhalte konsumieren wollen.
- **Elektronische Bücher** (innerhalb der nächsten 2-3 Jahre): Als weiterer Content wurde in den letzten zwölf Monaten ein deutlicher Anstieg in der Akzeptanz und Nutzung sogenannter *eBooks* verzeichnet. Diesen Trend bedient auch die Geräteindustrie durch die Einführung neuer eBook-Reader wie Apples *iPad*.
- **Simple Augmented Reality** (innerhalb der nächsten 2-3 Jahre): Die Anreicherung realer Objekte mit digitalen Informationen hat sich dank moderner Smartphones als "Fenster in den digitalen Raum"bereits dem Privatanwender erschlossen und besitzt auch als Werkzeug für Lehr-/Lernszenarien ein hohes Potential.

- **Gestenbasiertes Computing** (innerhalb der nächsten 4-5 Jahre): Durch berührungsempfindliche Displays längst auf dem Privatanwendermarkt etabliert, gibt es inzwischen auch in der Forschung und Lehre Prototypen für die gestenbasierte Interaktion mit Computern. Insbesondere dise Technologie profitiert direkt von Ergebnissen im Bereich Intentionserkennung der Pervasive Computing-Forschung.
- **Visuelle Datenanalyse** (innerhalb der nächsten 4-5 Jahre): Visuelle Methoden zum Erkennen und Verstehen von Mustern in großen Datensätzen erhalten als Kombination aus Statistik, Datamining und Visualisierung Einzug in die Forschungslandschaft.

Insbesondere die Einführung von Mobile Computing, Open Content und elektronischen Büchern stellt die universitäre Infrastruktur vor die Herausforderung neue Lernwerkzeuge und Inhalte integrieren zu müssen. Dies beinhaltet deren automatisierte Veröffentlichung, Suche und Nutzung durch verschiedene Geräte und Anwendungen. Weiterhin sind Mechanismen erforderlich, um Inhalte an verschiedene Geräteklassen (z. B. Laptop, Smartphone, MP3-Player) anzupassen.

Mit der wachsenden Bedeutung von Mobilität, allgegenwärtigem IT-Zugang und dynamischer Allokation anpassbarer Inhalte ist eine zunehmende Integration von Technologien des Pervasive Computing in Lehr- und Lernprozesse erkennbar. Moderne Bildungseinrichtungen sind mehr und mehr auf dem Weg pervasive Institutionen zu werden, eine Ausprägung pervasiver Umgebungen mit auf Hochschulen zugeschnittenen Komponenten und Interaktionsmustern. Die mit dem Begriff *Pervasive University* [Tavangarian 09] bezeichnete Hochschule der Zukunft muss sich dabei ebenfalls den in Abschnitt 1.1.2 betrachteten Herausforderungen stellen.

1.3 Zielstellung dieser Arbeit

Das Ziel dieser Arbeit ist die Konzeption und prototypische Implementierung einer Infrastruktur für pervasive Umgebungen unter besonderer Berücksichtigung universitärer Instanzen dieser Umgebungen. Dieses komplexe Ziel lässt sich in zwei Teilziele aufspalten.

Pervasive Umgebungen und moderne Bildungseinrichtungen haben ähnliche infrastrukturelle Herausforderungen zu meistern. Das erste Ziel dieser Arbeit ist es, diese Herausforderungen zu erarbeiten, ein geeignetes infrastrukturelles Modell zur Bewältigung dieser Herausforderungen zu finden und seinen Einsatz in pervasiven Umgebungen und Universitäten exemplarisch anhand passender Fallstudien zu demonstrieren.

Die Service-orientierte Architektur ist am Besten als infrastrukturelles Modell für dieses Umfeld geeignet, wie die vorliegende Dissertationsschrift belegt. Allerdings hat sich durch den SOA-Hype für verschiedene Anwendungsfelder ein breites Spektrum an SOA-Implementierungen etabliert, die größtenteils nicht kompatibel zueinander sind. Da sich pervasive Umgebungen heute aus einer Vielzahl von Komponenten aus unterschiedlichen Anwendungsbereichen zusammensetzen, muss der resultierenden SOA-Heterogenität durch Interoperabilitätsansätze begegnet werden. Eine Infrastruktur für diese Umgebungen wäre unvollständig ohne die Berücksichtigung ihrer Probleme. Das zweite Ziel dieser Arbeit ist daher

der Vergleich existierender Ansätze zur SOA-Interoperabilität, deren Bewertung hinsichtlich ihrer Eignung für die hochdynamische und transparente pervasive Umgebungen sowie die Erarbeitung eines systematischen Ansatzes, der sich insbesondere im pervasiven Umfeld einsetzen lässt. Diese Ergebnisse werden exemplarisch anhand moderner Lehr- und Lernarragements evaluiert.

1.4 Aufbau dieser Arbeit

Der weitere Teil dieser Arbeit ist wie folgt gegliedert. In Kapitel 2 werden verschiedene Kommunikationsmodelle für pervasive Umgebungen verglichen und bewertet. Es wird die Eignung des Broker-Modells als Grundlage Service-orientierter Architekturen motiviert und im Anschluss ein Überblick über existierende SOA-Implementierungen mitsamt deren Klassifizierung gegeben.

Kapitel 3 behandelt das Problem der SOA-Heterogenität und zeigt existierende Lösungsansätze auf. Der auf Abstraktion basierende und vielversprechendste Ansatz der *Zentralen Vereinheitlichung* wird durch die zentrale Komponente *General Purpose Access Point* (GPAP) in die Infrastruktur einer pervasiven Umgebung eingebettet. Ein besonderer Fokus liegt dabei auf der im Rahmen dieser Arbeit konzipierten *Service Technology-independent Language* (STiL) als Abstraktionsformat. Außerdem werden Plugin-basierte Integrationsstrategien für die zuvor klassifizierten SOA-Implementierungen konzipiert. Abschließend wird in einem Exkurs die Rolle der für pervasive Umgebungen sehr wichtigen Kontextinformationen und deren Zusammenspiel mit Service-Orientierung und SOA-Interoperabilität behandelt.

Das nächste Kapitel behandelt detailliert die prototypische Implementierung der in Kapitel 3 entwickelten Konzepte.

Kapitel 5 evaluiert die vorangegengenen Konzepte und Implementierungen im Kontext pervasiver Hochschulen. Nach einer umfangreichen Einführung in das Thema werden die Aspekte SOA, SOA-Interoperabilität und Kontext-Berücksichtigung anhand ausgewählter Fallstudien der Universität Rostock praktisch vertieft und Vorgehensweisen zu deren Integration und Nutzung in modernen Lehr- und Lernarragements aufgezeigt.

Diese Arbeit wird durch eine Zusammenfassung und einen Ausblick auf weitere Arbeiten und offene Fragestellungen abgerundet.

Kapitel 2

Service-orientierte Architekturen – Eine Bestandsaufnahme

Pervasive Umgebungen bestehen für gewöhnlich aus einer Reihe unterschiedlicher Geräte, die in der Lage sein müssen, miteinander zu kommunizieren. So muss eine möglicherweise fest installierte Infrastruktur beispielsweise Informationen mit den persönlichen Geräten der im Raum befindlichen Nutzer austauschen, um diesen ggf. zu identifizieren und seine Intention in der Umgebung zu ermitteln. Derartige Grundlagen intelligenten Verhaltens erfordern eine Kommunikationsinfrastruktur, die die besonderen Bedingungen in pervasiven Umgebungen bereits auf Ebene des zugrundeliegenden Kommunikationsmodells berücksichtigt.

2.1 Kommunikationsmodelle für pervasive Umgebungen

Vor allem durch die zunehmende Nutzer- und Gerätenmobilität sowie die technologische Heterogenität zukünftiger pervasiver Umgebungen entstehen eine Reihe von Anforderungen an die unterliegende Netzwerkinfrastruktur sowie das Kommunikationsmodell für die darin kooperierenden Komponenten (nach [Henricksen 01]):

- Um größtmögliche Interoperabilität und Transparenz zu gewährleisten, muss **von Unterschieden in der Hard- und Software einzelner Komponenten abstrahiert** werden können.
- Für eine hohe Robustheit, Fehlertoleranz und Mobilität muss das Wegfallen einzelner Komponenten innerhalb angemessender Zeiträume bemerkt und (falls verfügbar) eine gleichwertige Komponente als Ersatz **zur Laufzeit eingebunden** oder vorhandene Komponenten **spontan angepasst** werden können.
- Für die Unterstützung mobiler Nutzer und Geräte muss die Infrastruktur Mechanismen zur **Verteilung und Migration** von Komponenten anbieten.
- Für maximale Flexibilität müssen hinzukommende Komponenten **ohne umfassendes Vorwissen gefunden** und in eine bestehende Infrastruktur eingebunden werden können.

- Neu entwickelte und angepasste Anwendungen müssen zur Laufzeit in die Infrastruktur eingebunden werden können, wodurch diese entsprechende **Laufzeitumgebungen** (z. B. eine Java Virtual Machine, JVM) zur Verfügung stellen muss.
- Intelligente und somit situationsabhängige Systemreaktionen erfordern eine Infrastruktur, die die **Einbindung von Sensoren und anderen Kontextquellen** sowie die **Übertragung der gemessenen Kontextinformationen** unterstützt.
- Da die eigentliche Systemkomplexität für den Nutzer transparent sein muss, ist eine **gut skalierbare** aber auch **interaktive** Kommunikationsinfrastruktur erforderlich.
- Vor allem durch die heterogene Gerätelandschaft und neuartige Interaktionsmethoden zwischen Mensch und Maschine sind **generische Nutzerschnittstellen mit einer ausgereiften Bedienbarkeit** erforderlich.

Die Anforderungen an die Nutzerschnittstellen sowie die Anpassbarkeit von Anwendungen werden im Rahmen dieser Arbeit nicht als direkte Aufgabe der Infrastruktur sondern vielmehr der Anwendungsentwickler betrachtet und im Rahmen des Graduiertenkollegs MuSAMA in einem anderen Forschungsschwerpunkt (Multimodale Interaktion und Visualisierung) behandelt. Jedoch wurden im Rahmen der vorliegenden Arbeit zwei weitere Anforderungen identifiziert, die bereits auf Ebene der Infrastruktur Beachtung finden sollten:

- Um pervasive Umgebungen möglichst kurzfristig zu realisieren müssen möglichst viele bereits auf dem Markt **verfügbare Komponenten ohne zusätzliche Modifikationen** integrierbar sein.
- Um Sicherheitsaspekte einzelner Komponenten zu berücksichtigen (z. B. Datenschutz für das Adressbuch eines Mobiltelefons) müssen die integrierten Komponenten autonom bleiben dürfen und ihre **Integrationsintensität selbst definieren** können. Dies beinhaltet insbesondere auch, dass die Komponenten die Kontrolle darüber behalten, wer sie benutzt beziehungsweise Zugriff auf ihre Daten hat.

Diese Anforderungen verlangen auf der einen Seite nach einer flexiblen Netzwerkinfrastruktur, die alle verfügbaren Komponenten unter Berücksichtigung ihrer individuellen Netzwerktechnologien miteinander vernetzt. Die Entwicklung einer derartigen Infrastruktur ist einer der Forschungsaspekte des Graduiertenkollegs MuSAMA, jedoch nicht im Fokus der vorliegenden Arbeit. Stattdessen wird angenommen, dass die meisten Komponenten auf ISO/OSI-Layer 4 (Transport Layer) miteinander vernetzt sind und der generelle Datenaustausch möglich ist.

Auf der anderen Seite ist ein Kommunikationsmodell erforderlich, dass die vernetzten Komponenten dazu entsprechend den genannten Anforderungen befähigt auf Anwendungsebene zu kooperieren. Abbildung 2.1 zeigt die wichtigsten traditionellen und modernen Kommunikationsmodelle, die als Kandidaten für das geforderte Modell in Frage kommen [Lehsten 08]. Dabei sind diese nicht als echte Alternativen zueinander zu sehen, da sie in einzelnen Aspekten auch aufeinander aufbauen.

Das grundlegende Kommunikationsmodell ist das **Client/Server-Modell** (Abbildung 2.1a), bei dem *Clients* Anfragen nach Ressourcen oder Informationen an einen *Server* stellen (*Request*). Der Server antwortet mit einem *Response*, der bestenfalls die gewünschten Daten aber möglicherweise auch eine Fehlermeldung enthält. Dieses weit verbreitete Modell ist die

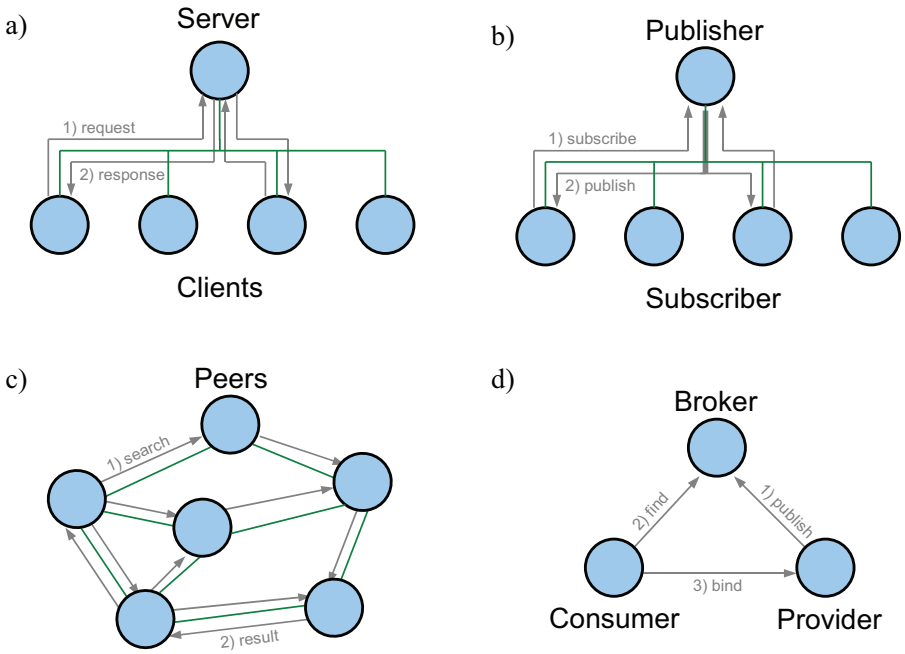


Abbildung 2.1: Verschiedene Kommunikationsmodelle als Infrastruktur-Kandidaten für Pervasive Umgebungen [Lehsten 08]: a) Client/Server-Modell, b) Publisher/Subscriber-Modell, c) Peer-to-Peer-Modell und d) Broker-Modell

Grundlage elementarer Internetmechanismen (z. B. HTTP und E-Mail), jedoch absolut unzureichend für Pervasive Umgebungen. Es erfordert beispielsweise Vorwissen über den Server und dessen angebotene Daten und skaliert sehr schlecht bei einer großen Menge an Clients. Weiterhin ist diese Client-initiierte Kommunikation ungeeignet für Clients, die auf die Aktualität von Daten angewiesen sind, da Updates beim Server erst nach einem weiteren Request/Response bemerkt werden können.

Vor allem diese Aktualität steht im Fokus des **Publisher/Subscriber-Modells** (Abbildung 2.1b). Hier registrieren sich *Subscriber* bei einem *Publisher* für den Erhalt von Daten (*subscribe*). Dieser verwaltet eine Liste der registrierten Subscriber und informiert diese falls neue für sie relevante Daten eintreffen (*publish*). Insbesondere die Registrierung beim Subscriber kann über das Client/Server-Modell erfolgen. Sobald Subscriber registriert sind, bietet das Publisher/Subscriber-Modell aber eine höhere Aktualität der Daten. Allerdings skaliert auch dieses Modell alleinstehend schlecht bei vielen Subscribern und auch hier ist Vorwissen über den Publisher sowie seine lieferbaren Daten nötig.

Bei dem vergleichbaren **Producer/Consumer-Modell** entfällt die Registrierung beim Datenlieferanten. Dieser sendet Daten auf einem vordefinierten Kanal, auf dem Consumer lauschen. Die Notwendigkeit von Vorwissen über den Kanal und zu erwartenden Daten bleibt. Allerdings ist hier eine bessere Skalierbarkeit möglich, da der Producer die Daten nur einmal verschickt und die Consumer selbst den Erhalt sicherstellen müssen.

Im **Peer-to-Peer (P2P)-Modell** (Abbildung 2.1c) gibt es keine feste Rollenverteilung mehr. Jeder Teilnehmer ist ein gleichberechtigter *Peer*. Um Ressourcen und Informationen zu finden, sendet der suchende Peer eine Anfrage an einen oder mehrere ihm bekannte Peers (*search*). Über ein Routing-Verfahren (abhängig vom spezifischen P2P-Protokoll) wird die Anfrage an Peers weitergeleitet, die die angefragen Daten liefern können. Die Daten werden anschließend an den anfragenden Peer geleitet (*result*). Dieses Modell hat seine Stärken vor allem in der Skalierbarkeit. Da P2P-Netzwerke meist aus einer großen Menge an Knoten bestehen, sind die meisten Daten redundant gespeichert, so dass nicht nur ein einzelner Teilnehmer für ein Datum verantwortlich ist. Auch die Last der Anfrage wird über mehrere Knoten verteilt. Weiterhin ist nur geringes Vorwissen notwendig, um eine Information zu finden. Ein Peer muss lediglich einen weiteren Peer des Netzwerkes kennen. Der Speicherort einer spezifischen Information ist für den anfragenden Peer transparent. Einen schlechten Ruf haben P2P-Netzwerke vor allem durch die häufige Nutzung in Form von Tauschbörsen für urheberrechtlich geschützte Daten bekommen. Außerdem besteht ohne zusätzliche Mechanismen keine Kontrolle darüber, wer welche Daten konsumiert. Daher ist dieses Modell für pervasive Umgebungen, in denen Komponenten den Zugriff auf ihre Daten kontrollieren wollen, nur bedingt einsetzbar.

In Abbildung 2.1d ist das **Broker-Modell** dargestellt. *Provider* bieten Informationen oder Ressourcen an, indem sie entsprechende Meta-Informationen bei einem *Broker* registrieren (*publish*). *Consumer*, die Informationen oder Ressourcen suchen, stellen eine Anfrage über alle Meta-Informationen beim Broker (*find*) und erhalten die Adressen der Provider, die zur Anfrage passende Daten registriert haben. Abschließend wählt der Consumer selbstständig einen der resultierenden Provider aus und versucht dessen Angebot zu nutzen (*bind*). Nach diesem Modell funktionieren beispielsweise Suchmaschinen, wobei diese die Registrierung von Providern oft selbst vornehmen. Da der Provider während des bind-Prozesses selbstständig entscheiden kann, ob der Consumer zur Nutzung berechtigt ist, ist dieses Modell zur sicherheitskritischen Kommunikation zwischen autonomen Komponenten geeignet. Weiterhin skaliert dieses Modell hervorragend, da nur bei der Suche mit dem Broker eine zentrale Komponente erforderlich ist. Die Nutzung der Ressourcen und Informationen erfolgt ausschließlich zwischen den beteiligten Komponenten und kann selbst dann fortgesetzt werden, wenn der Broker ausfällt. Weiterhin ist das Broker-Modell für sehr robuste Anwendungen prädestiniert. Bei Problemen oder sogar Ausfällen eines Providers kann der Consumer selbstständig einen alternativen Provider aus dem Suchergebnis des Brokers wählen (sofern vorhanden). Für das Broker-Modell ist jedoch Vorwissen über einen Broker notwendig. Daher erfordert seine Nutzung für pervasive Umgebungen zusätzliche Mechanismen zur Broker-Suche.

Wie Tabelle 2.1 verdeutlicht, erfüllt streng genommen keines der Kommunikationsmodelle in seiner Rohform alle definierten Anforderungen. Viele der Anforderungen lassen sich jedoch durch zusätzliche Mechanismen auf Grundlage der einzelnen Modelle realisieren. Vor allem das Broker-Modell ist eine geeignete Grundlage für derartige Erweiterungen, da es insbesondere dem Großteil der aus Dynamik und Mobilität resultierenden Anforderungen gerecht wird. Zusammen mit weiteren Mechanismen und Datenstrukturen bildet es daher die Grundlage für das Konzept *Service-orientierter Architekturen (SOA)*, die sich für die Realisierung hochdynamischer, vernetzter Systeme etabliert haben. In den folgenden Abschnitten werden die SOA-Besonderheiten erläutert und die wichtigsten Implementierungen dieses Konzeptes vorgestellt.

	Client/Server	Publisher/ Subscriber	Peer-to-Peer	Broker
Hardware- und Software-Abstraktion	nein	nein	nein	nein
Spontane Komponenten-einbindung	nein	nein	ja	ja
Spontane Komponenten-anpassung	nein	nein	nein	nein
Komponenten-verteilung und -migration	nein	nein	ja	ja
Kein Vorwissen erforderlich	nein	nein	nein	nein
Bereitgestellte Laufzeitumgebungen	nein	nein	nein	nein
Kontext-Unterstützung	nein	nein	nein	nein
Gute Skalierbarkeit	nein	nein	ja	ja
Integration exitierender Komponenten	nein	nein	nein	nein
Variable Integrationsintensität	ja	ja	nein	ja

Tabelle 2.1: Übersicht über die Eignung der wichtigsten Kommunikationsmodelle für die Infrastrukturen pervasiver Umgebungen.

2.2 Das SOA-Konzept und seine Implementierungen

Das Broker-Modell ist alleinstehend noch nicht für eine spontane Integration von Komponenten in einer pervasiven Umgebung geeignet, da es keine Spezifikation zur Art und Struktur der ausgetauschten Ressourcen und Informationen bietet. Diese wird jedoch für deren automatische Integration in maschinenlesbarer Form benötigt. Daher zeigt dieses Modell erst durch die Standardisierung der übertragenden Daten sowie der dafür genutzten Protokolle in Form von Services seine Stärken und hat sich unter dem Schlagwort der *Service-orientierten Architektur (SOA)* etabliert.

Der SOA-Begriff wurde erstmals im Jahr 1996 vom Marktforschungsunternehmen Gartner für ein Architekturkonzept gebraucht, das es Unternehmen ermöglicht in ihren Anwendungen gemeinsame Programmlogik sowie Daten zu nutzen [Schulte 96]. Das Konzept hat sich während des SOA-Hypes in den späten 90er Jahren rasant als Möglichkeit für Unternehmen verbreitet, schnell auf sich ändernde Geschäftsprozesse sowie Integrationsherausforderungen zu reagieren [Bieberstein 06]. In den vergangenen Jahren legte sich jedoch die Begeisterung der Business-Welt, nachdem diese einsah, dass man SOA nicht im Vorbeigehen mitnehmen kann, sondern tiefgreifende Umstrukturierungen im Unternehmen erforderlich werden [Scholler 08]. Dennoch ist das Konzept nach wie vor eine vielversprechende Lösung für IT-Integrationsprozesse in Industrie und Forschung, insbesondere für dem Einsatz in dynamischen pervasiven Umgebungen.

Im Mittelpunkt des SOA-Konzeptes steht der *Service*. Durch die deutlichen Unterschiede zwischen den existierenden SOA-Implementierungen gibt es bis heute keine einheitliche und allgemein anerkannte Definition des Service-Begriffes. Folgende Definition (angelehnt an [Melzer 07]) wird für die vorliegende Arbeit herangezogen, da sie über verschiedene SOA-Implementierungen hinaus anwendbar ist und die Service-Kerncharakteristika umfasst.

Definition 1 ***Services*** *sind wiederverwendbare und gekapselte Softwarekomponenten mit klar definierten, plattformunabhängigen und lokal oder über Netzwerke nutzbaren Schnittstellen.*

Daraus abgeleitet sind die wichtigsten Charakteristika eines Services:

- **Wiederverwendbarkeit**: Ist die Nutzung des Services abgeschlossen, wird er danach wieder zur Nutzung freigegeben. Viele Services sind auch zustandslos umgesetzt, so dass die explizite Freigabe nicht erforderlich ist.
- **Kapselung**: Gemäß des Prinzips des *Information Hiding* ist die spezifische Implementierung der Funktionalität eines Services hinter seiner abstrahierenden Schnittstelle verborgen.
- **Nutzung lokal oder über ein Netzwerk**: Services können sowohl lokal als auch über ein Netzwerk genutzt werden. Zur Bekanntmachung der Service-Schnittstelle kommen daher oftmals Serialisierungskonzepte (z. B. XML-Beschreibungen) zum Einsatz.
- **Plattformunabhängigkeit**: Services können unabhängig von ihrer spezifischen Implementierung in unterschiedlichen Laufzeitumgebungen verwendet werden.

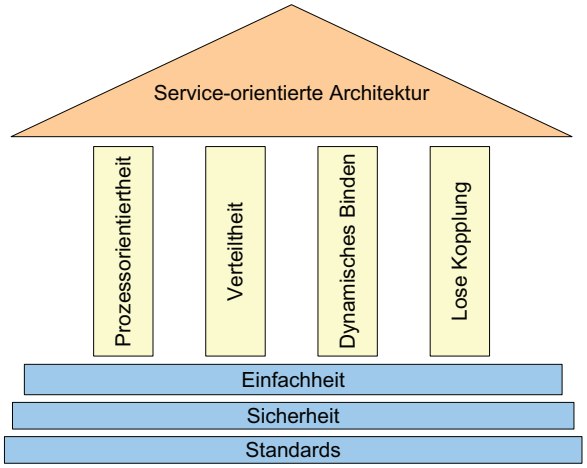


Abbildung 2.2: Unabhängig von konkreten SOA-Implementierungen illustriert der SOA-Tempel die allgemeinen Anforderungen und Merkmale einer SOA (basierend auf [Melzer 07]).

Services bilden zusammen mit der Flexibilität des Broker-Modells die Grundlage jeder SOA. Die zahlreichen Definitionen des SOA-Begriffes sind durch die jeweilige Sichtweise auf das SOA-Konzept und die Vielfalt der auf dem Markt befindlichen Implementierungen geprägt. In dieser Arbeit wird die SOA-Definition von Mark Colan (SOA- und Web Services-Evangelist bei IBM) herangezogen, da sie vor allem die SOA-Charakteristika betont, die für pervasive Umgebungen ausschlaggebend sind und zugleich so generisch ist, dass sie für nahezu alle wichtigen SOA-Implementierungen herangezogen werden kann (frei übersetzt aus [Colan 04]):

Definition 2 *Eine* ***Service-orientierte Architektur*** *ist eine Systemarchitektur, in der Softwarefunktionen als lose gekoppelte und klar definierte Komponenten ('Services') umgesetzt wurden, um die Interoperabilität zwischen ihnen zu erleichtern sowie die Flexibilität und Wiederverwendbarkeit des Systems zu steigern.*

Um eine derartige SOA erfolgreich einsetzen zu können sind Standards für die Beschreibung von Service-Schnittstellen erforderlich. Zusätzlich fördert die Berücksichtigung von Sicherheitsanforderungen sowie eine möglichst einfach gehaltene Architektur die SOA-Akzeptanz und somit die SOA-Einführung. Darauf aufbauend sind die wichtigsten Merkmale Service-orientierter Architekturen [Melzer 07]:

- **Prozessorientierung**: Durch die Kapselung von Service-Implementierungen ist die Konzentration auf die Funktionalität von Services möglich.
- **Verteiltheit**: Mehrere Netzwerkteilnehmer nutzen Services bzw. bieten eigene Services an.
- **Dynamisches Binden**: Durch einen Broker oder äquivalente Mechanismen werden Services von Anwendungen oder anderen Services dynamisch gefunden.
- **Lose Kopplung**: Services werden von Anwendungen oder anderen Services unter Nutzung ihrer Schnittstellenbeschreibung zur Laufzeit eingebunden.

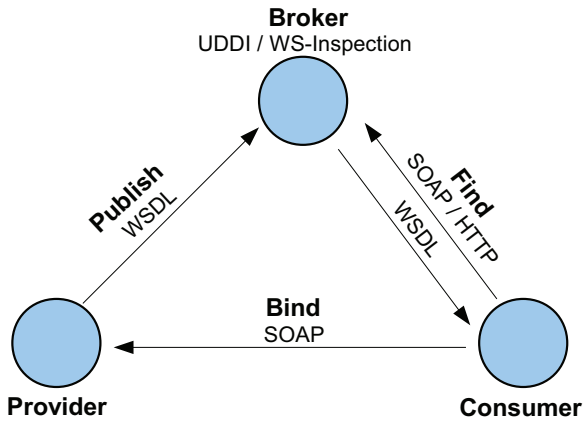


Abbildung 2.3: Web Services nutzen WSDL, SOAP sowie Broker-Implementierungen wie UDDI und WS-Inspection, um die SOA-Operationen *Publish*, *Bind* und *Find* umzusetzen.

Die geschilderten Anforderungen und Merkmale sind im sogenannten SOA-Tempel in Abbildung 2.2 illustriert und unabhängig von der konkreten SOA-Implementierung. Die folgenden Abschnitte erläutern die derzeit wichtigsten Implementierungen mit ihren Besonderheiten und Anwendungsbereichen, angefangen mit Web Services. Da diese heute als Referenzimplementierung für das SOA-Konzept gelten, werden die zugehörigen Standards, Komponenten und Prozesse etwas ausführlicher dargestellt als für die nachfolgenden Implementierungen.

2.2.1 Web Services – SOA für das Internet

Aufgrund der bereits sehr ausgereiften Spezifikation, dem großen Spektrum an Implementierungen und dem weltumspannenden Internet als Anwendungsgebiet sind Web Services heute eine sehr populäre SOA-Implementierung. In vielen Fällen werden Web Services sogar mit SOA gleichgesetzt, was allerdings aufgrund der zahlreichen alternativen Implementierungen nicht gerechtfertigt ist. Der Begriff Web Services umfasst verschiedene, teilweise konkurrierende Konzepte und Technologien, mit denen sich das SOA-Konzept im Internet realisieren lässt. Daher sind selbst Web Services streng genommen keine konkrete und eindeutige SOA-Implementierung. Allerdings haben sich in den letzten Jahren vor allem zwei Standards herauskristallisiert, die heute kennzeichnend für Web Services sind: *Die Web Service Description Language* (WSDL) zur Schnittstellenbeschreibung für Services und *SOAP* (früher für *Simple Object Access Protocol*, heute nicht mehr als Akronym verwendet) als Kommunikationsprotokoll zwischen den SOA-Teilnehmern. Weiterhin haben sich eine Reihe von Broker-Implementierungen einen Namen gemacht, allem voran *Universal Description, Discovery and Integration* (UDDI) und *Web Service Inspection* (WS-Inspection). In diesem Abschnitt wird die Funktionsweise von Web Services vor allem basierend auf diesen Technologien und Standards betrachtet.

Abbildung 2.3 illustriert den Einsatz der genannten Technologien im klassischen SOA-Dreieck mit drei Rollen und Operationen zwischen diesen Rollen. Provider hosten spezifische Web Services und veröffentlichen deren Schnittstellenbeschreibungen in Form von plattformunabhängigen WSDL-Dokumenten.

WSDL basiert auf der menschen- und maschinenlesbaren Extensible Markup Language (XML). Die vom World Wide Web Consortium (W3C) standardisierte Sprache liegt seit 2007 in Version 2.0 [Chinnici 07] vor, allerdings ohne große Verbreitung in realen Systemen [Melzer 07]. Die folgenden Ausführungen sind daher auf das verbreitete WSDL 1.1 [Christensen 01] aus dem Jahr 2001 bezogen. Listing 2.1 zeigt beispielhaft die WSDL-Beschreibung eines Web Services, der ISBN-Nummern in Buchtitel auflöst.

```
<definitions targetNamespace="http://isbn/" name="ISBNSearchService">

 <types/>

 <message name="Get_Booktitle">
  <part name="ISBN_Number" type="xsd:string"/>
 </message>

 <message name="Get_BooktitleResponse">
  <part name="Booktitle" type="xsd:string"/>
 </message>

 <portType name="ISBN_Resolution_Service">
  <operation name="Get_Booktitle">
   <input message="tns:Get_Booktitle"/>
   <output message="tns:Get_BooktitleResponse"/>
  </operation>
 </portType>

 <binding name="ISBN_Resolution_ServicePortBinding"
    type="tns:ISBN_Resolution_Service">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
    style="rpc"/>
  <operation name="Get_Booktitle">
   <soap:operation soapAction=""/>
   <input>
    <soap:body use="literal" namespace="http://isbn/"/>
   </input>
   <output>
    <soap:body use="literal" namespace="http://isbn/"/>
   </output>
  </operation>
 </binding>

 <service name="ISBNSearchService">
  <port name="ISBN_Resolution_ServicePort"
    binding="tns:ISBN_Resolution_ServicePortBinding">
   <soap:address location="http://139.30.7.75:8081/isbn"/>
  </port>
 </service>

</definitions>
```

Listing 2.1: Beispiel: WSDL-Dokument

WSDL-Beschreibungen enthalten eine Menge von Service-Definitionen (`definitions`), die wiederum sechs Hauptelemente beinhalten. Die ersten drei Elemente beschreiben den Service abstrakt auf der Ebene seiner Funktionalität:

- Das Element `types` definiert Datentypen, die für den Nachrichtenaustausch benötigt werden. In Listing 2.1 ist dieses Element leer, da nur Strings verwendet werden, die wiederum ein grundlegender atomarer XML-Datentyp sind und daher nicht weiter definiert werden müssen.
- `message` definiert abstrakt die Nachrichtentypen, die zwischen Consumer und Service ausgetauscht werden. Listing 2.1 enthält zwei Nachrichtentypen, die Anfrage nach einem Buchtitel und die dazugehörige Antwort.
- Unter `portType` werden die Operationen des Services definiert wobei für die Eingabe- und Ausgabenachricht auf die bereits definierten Nachrichten verwiesen wird. Der Beispiel-Service enthält nur eine einzige Operation (`Get_Booktitle`).

Die restlichen drei Elemente beschreiben den Service konkret auf der Ebene seiner technischen Details:

- In `binding` wird beschrieben, wie der Service aufgerufen wird. Der Service in Listing 2.1 besitzt nur das Binding `soap`, welches den Aufruf über Remote Procedure Calls (RPC) vorsieht. Über dieses Binding kann eine einzige Operation aufgerufen werden. Deren erwartete Eingabe- und Ausgabedatentypen sind als `literal` definiert, womit auf Datentypen aus dem SOAP-XML-Schema verwiesen wird, welches auch Strings definiert.
- Das Element `service` enthält eine Menge konkreter Ports als Kommunikationsendpunkte für den Service. Für den Beispiel-Service steht nur ein Endpunkt zur Verfügung.
- Der `port` definiert die konkrete Adresse (im Beispiel `http://139.30.7.75:8081/isbn`) für ein bereits definiertes Binding (im Beispiel `soap`).

Durch die Beschreibung der Service-Schnittstelle in Form des WSDL-Dokuments ist es Consumern möglich den Service zu nutzen. Dazu müssen sie das WSDL-Dokument jedoch zunächst im Internet finden. Um dies zu erleichtern, veröffentlicht der Provider es, bzw. einen Verweis auf das Dokument, zusammen mit Meta-Informationen über den Service bei einem öffentlichen Broker. Für Web Services gibt es unterschiedliche Broker-Konzepte. Zu den wichtigsten gehören UDDI und WS-Inspection, die im Folgenden kurz betrachtet und gegenübergestellt (Abbildung 2.4) werden.

UDDI (seit 2004 in Version 3.0.2 [Clement 04]) ist kein dedizierter Web Service-Broker, sondern hat – getrieben von IBM und Microsoft – grundlegend das Ziel ein Verzeichnis über Unternehmen und ihre als Services angebotenen Dienstleistungen zur Verfügung zu stellen – unabhängig von einer konkreten SOA-Implementierung und sogar über SOA hinaus. Dieses hochgesteckte Ziel führte dazu, dass UDDI ein Managementsystem für Metadaten (mit Fokussierung rechtlicher und wirtschaftlicher Aspekte) wurde, statt eines leichtgewichtigen und effizienten SOA-Brokers, wie er in hochdynamischen pervasiven Umgebungen benötigt wird.

Wie in Abbildung 2.4 dargestellt, kann UDDI als ein klassischer SOA-Broker betrieben werden. Provider registrieren ihre durch WSDL beschriebenen Services bei einem zentralen Ver-

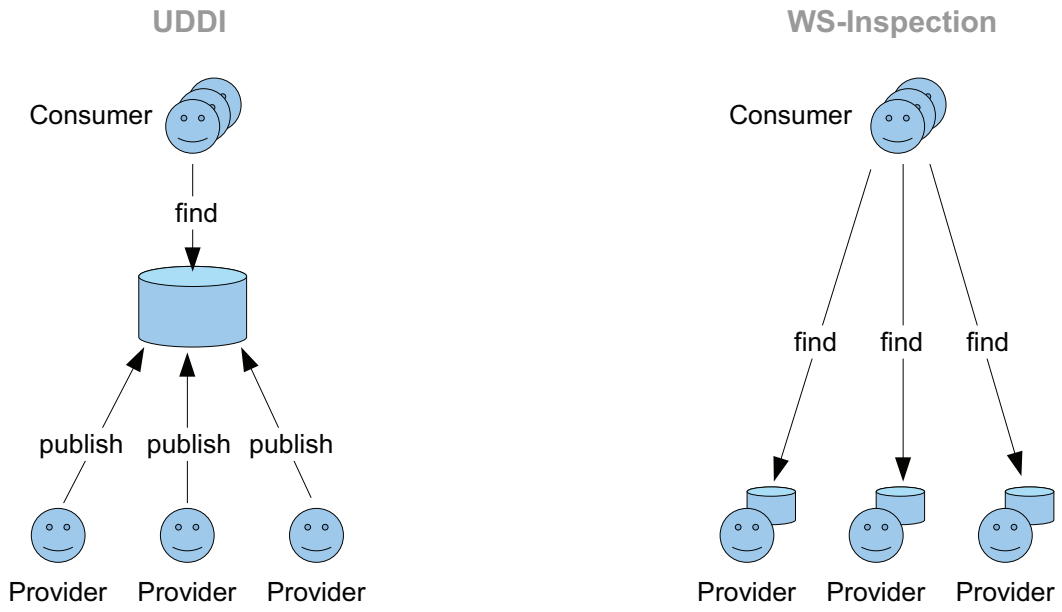


Abbildung 2.4: UDDI und WS-Inspection verfolgen zwei unterschiedliche Ansätze zur Realisierung eines Web Service-Brokers (nach [Melzer 07]).

zeichnis (*publish*). Die UDDI-Spezifikation sieht sogar eine SOAP-Schnittstelle für die Registrierung vor. Consumer können ebenfalls über eine SOAP-Schnittstelle Anfragen auf den UDDI-Datenbestand ausführen (*find*) und somit ihren Anforderungen entsprechende Dienste finden.

Sowohl der publish- als auch der find-Prozess sind allerdings sehr komplex im Vergleich zu anderen Broker-Lösungen: Provider müssen zu einem Service vier verschiedene Datenstrukturen anlegen (weitere sind fakultativ), die teilweise redundant zur ohnehin verfügbaren WSDL-Beschreibung sind und zusätzlich umfangreiche Informationen zum Service-anbietenden Unternehmen enthalten. Selbst mit den verfügbaren UDDI-APIs erfordert dies umfangreiche Kenntnisse über die UDDI-Funktionsweise sowie den umfassenden Kontext jedes Dienstes. Ein daraus resultierendes Problem ist die mangelnde Wartung und Aktualität von UDDI-Verzeichnissen und somit unzuverlässige Suchergebnisse.

Bereits 2005 deaktivierten IBM und Microsoft ihre UDDI-basierten öffentlichen Registries, verwiesen jedoch auf die Integration von UDDI-Mechanismen in eigenständige Produkte. Da UDDI aufgrund seiner Schwergewichtigkeit, Komplexität und Zentralierung auch ungeeignet für dynamische pervasive Umgebungen ist, wird an dieser Stelle auf weitere Details zu dieser Technologie verzichtet.

Im Gegensatz zu UDDI setzt WS-Inspection auf eine dezentrale Lösung. Das WS-Inspection-Konzept sieht vor, dass jeder Provider sein eigenes WSIL-Dokument als XML-Liste der von ihm angebotenen Dienste bereit stellt. Dieses Dokument ist üblicherweise im Root-Verzeichnis des Providers per HTTP zu finden (`http://<Provider-Domainname>/inspection.wsil`). WS-Inspection wurde von IBM und Microsoft 2001 spezifiziert [Ballinger 01] und 2007 ak-

tualisiert. Ursprünglich als Bindeglied zwischen UDDI und WSDL vorgesehen, wird WS-Inspection heute auch ohne UDDI-Bezug verwendet.

Listing 2.2 zeigt beispielhaft ein WS-Inspection-Dokument. Dieses enthält eine Referenz auf einen Service (`service`) mit dessen Namen (`name`), abstrakter Beschreibung (`abstract`) und einer Referenz auf die konkrete Beschreibung in Form seines WSDL-Dokumentes (`description`). Neben Services kann auch auf andere WSIL-Dokumente verwiesen werden (`link`), sodass umfassende WS-Inspection-Netzwerke möglich sind. Es ist denkbar mit WS-Inspection zentrale Zugangspunkte (WSIL-Wurzeldokumente) zu definieren, über die baumartig auf das Netzwerk zugegriffen werden kann. Vor allem durch seine dezentrale und leichtgewichtige Struktur ist WSIL besser für dynamische pervasive Umgebungen geeignet, wobei auch hier wie bei UDDI Vorwissen über den Broker (das WSIL-Netz) erforderlich ist.

```
<?xml version="1.0"?>
<inspection xmlns="http://schemas.xmlsoap.org/ws/2001/10/inspection/">
 <service>
  <name>ISBN Resolution Service</name>
  <abstract>Resolves ISBN codes into booktitles</abstract>
  <description referencedNamespace="http://schemas.xmlsoap.org/wsdl/"
   location="http://139.30.7.75:8081/isbn?wsdl" />
 </service>
</inspection>
```

Listing 2.2: Beispiel: WSIL-Dokument

Spätestens bei der Service-Nutzung durch den Consumer verwendet der Großteil der Web Services das XML-basierte und somit plattform- und programmiersprachenunabhängige Nachrichtenformat SOAP. Durch entsprechende APIs und Bibliotheken (z. B. für Java) werden SOAP-Nachrichten im Normalfall vor Nutzern und Entwicklern verborgen. Daher wird an dieser Stelle auch nur beispielhaft und in gestraffter Form auf die Schlüsselelemente aus der SOAP-Spezifikation [Gudgin 07] eingegangen.

```
<?xml version="1.0" ?>
<S:Envelope  xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
 <S:Body>
  <ns2 :Get_Booktitle xmlns:ns2="http://isbn/">
   <ISBN_Number>978-3503061006</ISBN_Number>
  </ns2:Get_Booktitle>
 </S:Body>
</S:Envelope>
```

Listing 2.3: Beispiel: SOAP-RPC-Request

SOAP-Nachrichten sind nicht an ein bestimmtes Transportprotokoll gebunden, sondern können als Nutzlast über unterschiedliche Protokolle übertragen werden. In der Regel kommt jedoch HTTP zum Einsatz. SOAP-Nachrichten bestehen immer aus einem *Envelope*, der einen optionalen `Header` sowie einen obligatorischen `Body` enhält. Der Header wurde nicht näher spezifiziert, kann aber beispielsweise dazu genutzt werden Nachrichten mit zusätzlichen Meta-Informationen anzureichern. Der Body enhält die eigentliche Nutzlast der SOAP-Nachricht.

```
<?xml version="1.0" ?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
 <S:Body>
  <ns2:Get_BooktitleResponse xmlns:ns2="http://isbn/">
   <Booktitle>Das Nibelungenlied</Booktitle>
  </ns2:Get_ BooktitleResponse>
 </S:Body>
</S:Envelope>
```

Listing 2.4: Beispiel: SOAP-RPC-Response

Listing 2.3 stellt exemplarisch einen Remote Procedure Call dar, mit dem ein Request an den bereits erwähnten ISBN-Web Service durchgeführt wird. Wie in der WSDL-Beschreibung gefordert, wird dazu eine Nachricht vom Typ `Get_Booktitle` mit dem Argument `Booktitle` aufgerufen.

Zu jedem Remote Procedure Call gehört eine Response-Nachricht, die ebenfalls in einer SOAP-Nachricht verpackt wurde. Listing 2.4 zeigt die Antwortnachricht vom Typ `Get_BooktitleResponse` mit dem Argument `Booktitle`, welches ebenfalls bereits in der WSDL-Beschreibung definiert wurde.

SOAP-Nachrichten können durch die intensive XML-Nutzung unverhältnismäßig groß werden. Die in Listing 2.3 aufgeführte Nachricht ist z.B. fast 20mal so groß wie die eigentliche Information, hinzu kommt der Overhead des Transportprotokolls. Dank der hohen Verfügbarkeit hoher Übertragungsbandbreiten hat sich dieser Nachteil kaum auf die Verbreitung von SOAP ausgewirkt, so dass SOAP sogar über die Grenzen von Web Services hinaus eingesetzt wird, beispielsweise in der SOA-Implementierung *Universal Plug and Play*, die im folgenden Abschnitt näher beschrieben wird.

2.2.2 UPnP – SOA für Ad-hoc-Gerätekooperation

Während Web Services die Anwendungskooperation in weitläufigen Netzwerken unterstützt, konzentriert sich *Universal Plug and Play* (UPnP) [Jeronimo 03] auf die spontane Ansteuerung von Geräten in lokalen IP-basierten Netzwerken. Ursprünglich von Microsoft eingeführt, wird UPnP inzwischen vom UPnP-Forum spezifiziert und standardiert. Das Forum zertifiziert außerdem UPnP-konforme Geräte verschiedener Hersteller. Da UPnP-Geräte gemäß dem *Zero Configuration*-Konzept einem Netzwerk ohne eine vorher zugewiesene IP-Adresse beitreten können, hat diese Technologie heute vor allem für die schnelle Konfiguration von Heimnetzwerken eine Bedeutung. Durch die Erweiterung *UPnP Audio and Video* (UPnP-AV) erfährt UPnP außerdem eine weite Verbreitung für die Verteilung von Multimedia-Inhalten in lokalen Netzwerken.

UPnP definiert, wie standardisierte und IP-basierte Kommunikationsprotokolle (z. B. UDP, TCP, HTTP und SOAP) für die folgenden sechs UPnP-Prozesse zu nutzen sind:

- *Addressing*: Bezug einer IP-Adresse (per *Dynamic Host Configuration Protocol* oder *Zero Configuration Networking*)

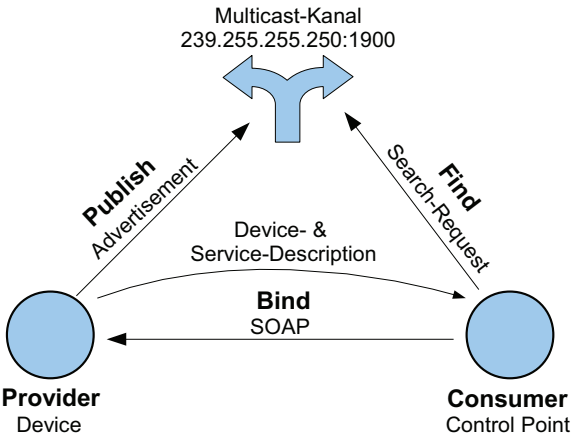


Abbildung 2.5: Für größtmögliche Dynamik verzichtet UPnP auf eine dedizierte Broker-Komponente. Stattdessen nutzen Provider und Consumer einen bekannten Multicast-Kanal zum Anbieten und Finden von Services.

- *Discovery*: Anbieten und Finden von Geräten bzw. dessen Services (per IP-Multicast und dem *Simple Service Discovery Protocol*, SSDP)
- *Description*: Austausch von Geräte- bzw. Service-Beschreibungen (per HTTP mit XML-basierten Nachrichten)
- *Control*: Aufruf von Gerätefunktionen (per SOAP)
- *Event Notification*: Meldung von Ereignissen wie z. B. der Änderung von Statusvariablen (per General Event Notification Architecture, GENA)
- *Presentation*: Benutzeroberfläche für *Control* und *Eventing* (per HTTP und HTML)

UPnP-Services liegen als XML-basierte *Service Descriptions* auf dem anbietenden Gerät vor. Im Rahmen der darauf aufbauenden SOA-Funktionalität sind insbesondere die Abläufe der Prozesse *Discovery*, *Description* und *Control* interessant. Abbildung 2.5 illustriert die in diesen Prozessen spezifizierte Umsetzung des SOA-Dreiecks.

Der Provider wird durch die UPnP-Komponente *Device* und der Consumer durch den *Control Point* repräsentiert. Geräte können auch beide Komponenten umsetzen, indem sie einerseits Services anbieten und andererseits die Services anderer Geräte nutzen. Auffällig ist, dass UPnP keinen Broker vorsieht, da die Architektur völlig dezentral konzipiert ist. Stattdessen verwendet UPnP SSDP zum Anbieten und Finden von Services: Provider veröffentlichen ihre Services, indem sie eine HTTP-basierte Nachricht (*Advertisement*) an einen spezifizierten Multicast-Kanal (dedizierte IP-Adresse) schicken. Diese enthält die Adresse des Anbieters sowie der *Device Description*, die wiederum die *Service Description* enthält, und eine Gültigkeitsdauer des Advertisements. Consumer lauschen wiederum auf dem Multicast-Kanal und fragen bei Interesse die Device Description per Unicast ab. Neben dem Advertisement können Consumer auch selbstständig *Search*-Requests mit den gesuchten Service-Typen an den Multicast-Kanal senden. Die Provider lauschen ebenfalls auf dem Kanal und antworten ggf. per Unicast mit den geforderten Descriptions. Die Service Descriptions spezifizieren unter

anderem die Operationen eines Services. Genau wie Web Services, verwendet UPnP darauf aufbauend SOAP zur Service-Nutzung.

Einer der schwerwiegendsten Nachteile von UPnP ist die mangelnde Unterstützung von Sicherheitsmechanismen. Vor allem bei der Netzwerkkonfiguration ist beispielsweise eine gegenseitige Authentifizierung zwischen Providern und Consumern wünschenswert. Somit ist UPnP für sicherheitskritische Anwendungen ungeeignet, da Provider und Consumer eigene Sicherheitsmechanismen neben UPnP implementieren müssten.

Insbesondere die nutzertransparente Unterstützung von dynamischen und spontan entstehenden Gerätekooperationen hat zum Erfolg von UPnP geführt. Dabei setzt UPnP auf eine dezentrale, da Multicast-basierte Lösung. Auch die im folgenden Abschnitt behandelte Implementierung Jini verwendet IP-Multicasts für spontane Szenarien, kombiniert diese jedoch mit der zentralen Service-Verwaltung durch einen Broker.

2.2.3 Jini – SOA für verteilte Java-Anwendungen

Sun Micosystems offene Netzwerkarchitektur *Jini* [Newmarch 06] fokussiert die Kooperation zwischen Anwendungen eines bestehenden Netzwerkes. Wie die bisherigen SOA-Implementierungen ist auch Jini unabhängig von konkreten Programmiersprachen und Laufzeitumgebungen. Dennoch hat Jini heute nur im Zusammenspiel mit Java und der Java Virtual Machine (JVM) eine Relevanz, da das Referenzframework *Jini Starter Kit* (aktuelle Version 2.1) die verbreitetste Umsetzung der Jini-Spezifikation darstellt. Eine wichtige Bedeutung hat Jini vor allem für *JavaSpaces* [Freeman 02], einer Implementierung des *Object Spaces*- Konzeptes zum Austausch von Objekten in verteilten Anwendungen, wie sie auch in Kooperationsszenarien des Pervasive Computing zum Tragen kommen [Seungyong 06].

Auch der Zugriff auf Jini-Services, welche im Allgemeinen in Java implementiert sind und in der JVM des Providers laufen, erfolgt über eine veröffentlichte Schnittstelle, das *Service-Objekt*. Dieses serialisierte Java-Objekt wird direkt in die Consumer-Anwendung eingebunden und verbirgt die gesamte Kommunikation mit dem Service. Es ist möglich, dass das Service-Objekt der eigentliche Service ist, der dann in der JVM des Consumers läuft und keine weitere Kommunikation mit dem Provider erfordert. Diese “Code-Verschiebung” bedingt jedoch die komplette Serialiserbarkeit des Services und ist beispielweise beim Einbinden von Algorithmen-Implementierungen ohne Nutzung von Provider-Ressourcen sinnvoll. In den meisten Fällen ist das Service-Objekt jedoch ein Proxy, der mit dem eigentlichen Service über *Remote Method Invocation* (RMI), die *Common Object Request Broker Architecture* (CORBA) oder selbst SOAP kommuniziert. Da Jini die “alles ist ein Service”-Philosophie verfolgt, wird selbst der Broker über einen derartigen Proxy angesprochen, den *Registrar*.

Abbildung 2.6 illustriert das Zusammenspiel der bekannten SOA-Komponenten mit Jini. Der Provider veröffentlicht die Service-Proxys seiner Services zusammen mit einigen Meta-Daten beim *Lookup-Service*, der die Broker-Rolle spielt. *Reggie*, die weit verbreitete Referenzimplementierung eines derartigen Services, kommt ebenfalls von Sun Microsystems. Da Jini vor allem die spontane Kooperation von Anwendungen anstrebt, muss die Broker-Adresse nicht im Vorfeld bekannt sein. Stattdessen können Provider und Consumer verfügbare Lookup-Services über einen IP-Multicast an die Multicast-Adresse `224.0.1.85` finden. Diese antworten mit

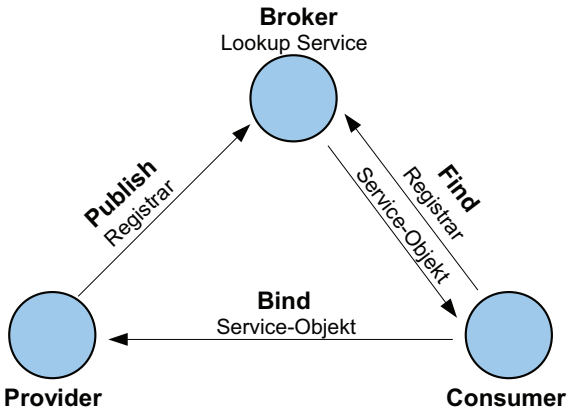


Abbildung 2.6: Gemäß der "Alles ist ein Service"-Philosophie von Jini ist auch der Broker als Service erreichbar und über einen Stub (Registrar) nutzbar.

dem Verweis auf ihren Registrar, der zur weiteren Kommunikation mit dem Lookup-Service genutzt wird.

Consumer können Registrars verfügbarer Lookup-Services ebenfalls per IP-Multicast finden. Um nach Services zu suchen, wird dem Registrar ein *Service-Template* übergeben, eine Suchmaske zum Filtern aus allen verfügbaren Services. Dieses kann beispielsweise Services fordern, die ein konkretes Java-Interface implementieren oder bestimmte Anforderungen an die Meta-Daten erfüllen. Der Lookup-Service antwortet mit Verweisen auf alle registrierten Service-Objekte, die dem Template entsprechen. Über diese Objekte wird anschließend der Service kontaktiert.

Jini hat seine Vorteile vor allem in der weitgehenden Kommunikations- und Ortstransparenz sowie seiner Unterstützung spontan entstehender SOAs ohne Vorwissen auf Provider- oder Consumer-Seite. Im Gegensatz zu UPnP gibt es auch ein umfassendes Sicherheitskonzept, welches auf dem Sicherheitsmodell des Java Development Kit (JDK) 1.2 mit Zertifikaten, Permissions und Signaturen basiert. Vor allem die Java-Bindung der Referenzimplementierung, die sehr komplexe Installation und Konfiguration von Jini und die schwache Lobby bremsen Jinis Verbreitung und begrenzen es heute auf sehr spezielle Anwendungsfelder. Im Gegensatz dazu erfreut sich *DNS Service Discovery* (DNS-SD) dank Apples verbreiteter Implementierung *Bonjour* wachsender Beliebtheit.

2.2.4 DNS-SD – SOA über vorhandene DNS-Infrastruktur

Wie bereits bei UPnP verdeutlicht, treten SOA-Implementierungen immer wieder in Kombination mit konfigurationsfreien Netzwerken auf. So ist auch *DNS-based Service Discovery* (DNS-SD) [Cheshire 08] ein Bestandteil der Spezifikation von *Zero Configuration Networking* (Zeroconf) [Steinberg 05]. DNS-SD ist verantwortlich für das Angebot und die Suche von Services in lokalen IP-Netzwerken und hat unter der Bezeichnung *Bonjour* vor allem Einzug in netzwerkfähige Hardware sowie Software der Firma Apple erhalten.

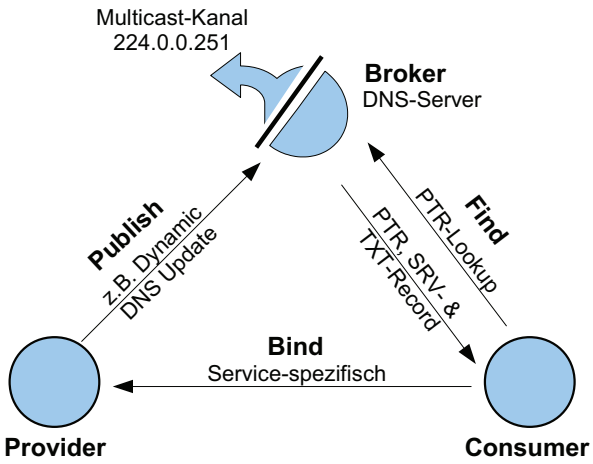


Abbildung 2.7: DNS-SD setzt auf die Nutzung ohnehin vorhandener DNS-Server als Broker. Doch auch ohne diese können in lokalen Netzen Services per IP-Multicast gefunden werden.

Im Gegensatz zu den bisherigen SOA-Implementierungen spezifiziert DNS-SD nur wie Services gefunden werden, nicht jedoch, wie diese von Providern veröffentlicht werden. Auch die Art der Service-Nutzung ist nicht vorgegeben und somit Service-spezifisch. Eine zentrale Rolle spielen hingegen vordefinierte Service-Typen, nach denen Consumer während des Service-Discovery suchen. Ein Service mit dem Typ `_ipp._tcp.example.com` wird beispielsweise unter der Internet-Domain `example.com` angeboten und setzt das *Internet Printing Protocol* (IPP) um. Bei seinem Provider könnte es sich um einen Drucker handeln.

Wie das Beispiel verdeutlicht, folgen DNS-SD-Services der Syntax `<Service>.<Domain>`, die mit der Syntax von DNS-Hostnamen (`<Host>.<Domain>`) vergleichbar ist. Somit können für DNS-SD DNS-Server als Broker eingesetzt werden. Da diese ohnehin in den meisten IP-Netzen vorhanden sind, erfordert DNS-SD keine zusätzlichen Netzwerk-Komponenten. Die Spezifikation sieht die Verwendung bestimmter *Ressource Records* für die SOA-Funktionalität vor. Diese werden von jedem DNS-Server akzeptiert, da sie ursprünglich für Verweise zwischen Hostnamen, IP-Adressen und anderen DNS-Servern genutzt werden. Relevant sind vor allem drei Record-Typen:

- *SRV-Records* werden zur Speicherung der wichtigsten Service-Informationen genutzt (z. B. Service-Name, -Anbieter und -Priorität).
- *TXT-Records* werden zur Speicherung Service-spezifischer Meta-Informationen verwendet (in Form von Schlüssel/Wert-Paaren).
- *PTR-Records* werden vom Consumer gesucht und daher zum Verweisen von Service-Instanzen auf SRV- und TXT-Records eingesetzt.

Wie Abbildung 2.7 verdeutlicht, nutzen Consumer sogenannte *PTR-Lookups*, um Services zu finden. Mit denen sendet ein Consumer seinem zuständigen DNS-Server eine Anfrage nach Instanzen des Service-Typs `<Service>.<Domain>`. Der Server antwortet mit den PTR-Records konkreter Service-Instanzen in der Form `<Instanz>.<Service>.<Domain>`. Diese werden vom Consumer genutzt sich für einen Service zu entscheiden und anschließend des-

sen SRV- und TXT-Records vom Server zu beziehen. Die Nutzung des Services ist Service-spezifisch. Dazu erforderliche Details sind im SRV-Record enthalten.

Für den Fall, dass kein DNS-Server verfügbar ist bzw. dessen Adresse noch unbekannt ist, kann *Multicast DNS* (mDNS) [Cheshire 09] zur Service-Suche eingesetzt werden. Dazu suchen Consumer per Multicast nach Services mit der Domain `.local`. Provider, die auf dem Multicast-Kanal lauschen und Instanzen des gesuchten Service-Typs anbieten, senden die entspechenden Records per Unicast an den Consumer zurück.

Mit der Nutzung von DNS-Servern als Broker geht DNS-SD einen vielversprechenden Weg zur Nutzung vorhandener Infrastrukturen für SOA-Overlays. Die zunehmende Verbreitung von Apple-Produkten hat diesem Discovery-Protokoll den Weg in die Wohnzimmer geebnet, da Bonjour ein fester Bestandteil der Zeroconf-Strategie von Apple ist. Auch das *Service Discovery Protocol* hat sich vor allem durch seine Integration in eine andere Technologie (Bluetooth) durchgesetzt.

2.2.5 Bluetooth SDP – SOA für Personal Area Networks

Das *Service Discovery Protocol* (SDP) ist fester Bestandteil des Protokoll-Stacks der Bluetooth-Technologie [Bluetooth SIG 09], welche vor allem für Kurzstreckenkommunikation in Personal Area Networks (PAN) eingesetzt wird. SDP ist zwar nicht an den Einsatz in Bluetooth-Netzen gebunden, besitzt darüber hinaus jedoch bisher kaum Relevanz. Da vor allem die große Gruppe der Mobiltelefone und Smartphones Bluetooth unterstützt, hat SDP auch eine Bedeutung für pervasive Umgebungen, die das Ziel haben derartige Geräte dynamisch zu integrieren.

Wie auch DNS-SD, spezifiziert SDP nicht das Binden bzw. Nutzen von Services. Diese werden als sogenannte *Profile* jeweils gemäß einer eigenen Spezifikation verwendet. Das Headset-Profil HSP wird beispielsweise von Geräten implementiert, die eine Schnittstelle zur Sprachausgabe besitzen. Aus SOA-Sicht sind diese Geräte Provider. Consumer können Audio-Informationen ausgeben, indem sie diese Implementierung über das HSP-Profil (Service) nutzen. Jedes Profil besitzt einen Universal Unique Identifier (UUID), der mittels SDP angeboten und gefunden werden kann. Er ist somit zusammen mit wenigen Meta-Informationen das Äquivalent zur Dienstbeschreibung anderer SOA-Implementierungen.

Durch seinen Fokus auf ad-hoc aufgebaute Kurzstreckenverbindungen und Gerätekommunikationen sind traditionelle SOA-Konzepte für Bluetooth eher ungeeignet. Vor allem die Verfügbarkeit eines zentralen Brokers oder Broker-Netzes kann nicht als gegeben angesehen werden. Multicast-basierte Konzepte wie bei DNS-SD sind ebenfalls nicht geeignet, da die Unicast-basierten Bluetooth-Netze diese nicht ohne einen zentralen Vermittler unterstützen. Aus diesen Gründen unterscheidet sich die SOA-Implementierung mittels SDP deutlich von den bisher behandelten Ansätzen, wie Abbildung 2.8 verdeutlicht.

Für die Implementierung eines Profiles sind auf Bluetooth-Geräten dedizierte Anwendungen verantwortlich, die die Provider-Rolle einnehmen (Gerät A in Abbildung 2.8). Sie registrieren ihre Profile (über eine API) als UUID mit Meta-Informationen beim SDP-Server des lokalen Bluetooth-Stacks. Dieser nimmt somit die Broker-Rolle ein. Die Besonderheit bei Bluetooth ist, dass der Broker immer direkt an das Gerät mit der Provider-Anwendung gebunden ist und

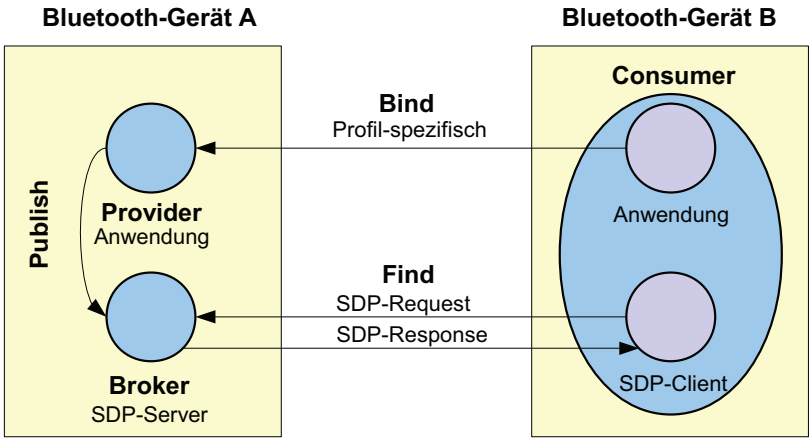


Abbildung 2.8: SOA zwischen Bluetooth-Geräten: Jedes Provider-Gerät hostet seinen eigenen Broker in Form eines *SDP-Servers* und Consumer-Geräte nutzen den *SDP-Client* ihrer lokalen Bluetooth-API, um Services zu finden. Der Binding-Prozess wird nicht durch SDP spezifiziert und ist abhängig vom Service (Bluetooth-Profil) selbst.

somit ausschließlich für die Profile dieses Gerätes verantwortlich ist. Wenn Anwendungen auf weiteren Geräten (z. B. Gerät B in Abbildung 2.8) diese Profile nutzen wollen, müssen sie diese ebenfalls über eine API ihres lokalen Bluetooth-Stacks finden. Der sogenannte SDP-Client ist für die iterative Suche von Profilen auf allen in Reichweite befindlichen Geräten zuständig, somit kann er zusammen mit der Anwendung als SOA-Consumer angesehen werden. Auf jedem Gerät kann mittels SDP-Request eine gezielte Suche nach einem konkreten Profil (*Searching*) oder nach allen vorhandenen Profilen (*Browsing*) durchgeführt werden. Der SDP-Response vom Broker enthält Verweise auf die passenden Profile. Wie bereits beschrieben ist die anschließende Nutzung ausschließlich vom jeweiligen Profil abhängig.

SDP ist bereits eine sehr spezielle SOA-Implementierung, die verdeutlicht, dass die SOA-Rollen und -Prozesse je nach Anwendungsbereich nicht immer ganz eindeutig identifizierbar sind. Durch ihrer Verbreitung im Rahmen des Bluetooth-Standards ist SDP dennoch relevant. Im Folgenden wird der Vollständigkeit halber kurz auf weitere Implementierungen eingegangen, die keine zusätzlichen Erkenntnisse über SOA-Konzepte bringen, für pervasive Umgebungen weniger geeignet sind oder eher ein Nischendasein fristen.

2.2.6 Weitere SOA-Umsetzungen

Da jede der bereits genannten SOA-Implementierungen seine Stärken und Schwächen besitzt, liegt die Entwicklung hybrider Lösungen nahe. Ein vielversprechender Ansatz ist *Device Profiles for Web Services* (DPWS) [Driscoll 09], eine Kombination aus Web Services und UPnP für eingebettete Systeme mit stark limitierten Ressourcen. Für pervasive Umgebungen wird DPWS vor allem durch die Kombination der ausgereiften Sicherheitsmechanismen der Web Services mit der UPnP-Unterstützung spontaner und ohne Vorwissen entstehender Gerätekooperationen interessant. Die Unterstützung durch Microsoft durch die DPWS-

Integration in aktuelle Windows-Betriebssysteme lässt außerdem eine hohe Verbreitung erwarten.

Die *Open Services Gateway initiative* (OSGi) [The OSGi Alliance 03] spezifiziert eine SOA-Umsetzung zur dynamischen Integration von Softwarepaketen in Anwendungen innerhalb einer Java Virtual Machine. Durch die damit erreichbare Modularisierung hat OSGi vor allem bei Frameworks zur Anwendungsentwicklung an Bedeutung gewonnen. OSGi wird heute beispielsweise von der populären und stark individualisierbaren Entwicklungsumgebung Eclipse verwendet, um Plugins zu integrieren.

Das *Service Location Protocol* (SLP) [Guttman 99b] ist ein einfaches Protokoll für das Auffinden von Services in TCP/IP-Netzen. SLP hält sich ähnlich wie Web Services streng an das dreiteilige Rollenkonzept des SOA-Dreiecks, spezifiziert allerdings nicht die Service-Nutzung. Da auch Multicast-Discovery unterstützt wird, ist SLP im Gegensatz zu Web Services auch für dynamische Gerätekooperationen ohne Vorwissen geeignet. Auch wenn SLP kaum verbreitet ist, hat es in einzelnen Anwendungen, wie das Common Unix Printing System (CUPS), Einzug gehalten.

Die in diesem Abschnitt vorgestellten SOA-Implementierungen basieren zwar auf demselben abstrakten Modell, setzen dieses aber jeweils sehr unterschiedlich um. Im folgenden Abschnitt werden sie technisch klassifiziert, um anschließend in Kapitel 3 Strategien für die Interoperabilität zwischen SOA-Klassen erarbeiten zu können.

2.3 Technische Klassifizierungen

Die SOA-Heterogenität führt zu einer Vielzahl von Unterschieden aber auch Gemeinsamkeiten zwischen den einzelnen Implementierungen, die bereits ausführlich in der Literatur diskutiert wurden [Zhu 05][Edwards 06]. Die im Rahmen der vorliegenden Arbeit entstandenen und in diesem Abschnitt vorgestellten Klassifizierungen dienen vor allem zur Feststellung von Charakteristika auf Prozessebene, da diese die größte Herausforderung bei der Realisierung von Interoperabilität zwischen verschiedenen SOA-Implementierungen darstellt.

2.3.1 Einsatz von Brokern

Die erste Klassifizierung lässt sich im Hinblick auf das Broker-Modell durchführen, welches für SOA-Implementierungen weit verbreitet ist. Während vor allem Web Services das klassische Broker-Modell umsetzen, arbeiten andere verbreitete Implementierungen wie UPnP komplett ohne eine Broker-Komponente. Somit lassen sich vor allem drei Klassen identifizieren:

Zu den *Broker-basierten SOAs* gehören Web Services und Jini. Während Web Services sehr unterschiedliche Arten von Brokern verwenden können (z. B. UDDI und WS-Inspection), die jedoch im Vorfeld bekannt sein müssen, unterstützt Jini nur einen dedizierten Broker-Service, der aber auch per IP-Multicast – und somit ohne Vorwissen – gefunden werden kann.

Im Gegensatz dazu verzichten *Broker-freie SOAs* wie UPnP und streng genommen auch Bluetooth SDP auf den Einsatz von dedizierten Broker-Komponenten. Stattdessen kommen auf Multicasts bzw. auf Sequenzen von Unicasts basierende Suchmechanismen zum Einsatz.

Die dritte Klasse kann als *SOAs mit Broker-Modus* bezeichnet werden und umfasst Technologien wie DNS-SD und SLP. Diese SOA-Klasse kann in Broker-basierten aber auch Broker-freien Modi arbeiten, wobei letzterer vor allem in lokalen SOA-Szenarien zum Einsatz kommt.

Es fällt auf, dass gerade bei Service-orientierten Architekturen, die für spontan entstehende Gerätekooperationen prädestiniert sind (z. B. UPnP, Bluetooth-SDP und für lokale SOAs auch DNS-SD und SLP) auf den Einsatz eines klassischen Brokers verzichtet wird. Der Grund dafür ist, dass diese SOA-Implementierungen möglichst geringe Ansprüche an die vorhandene Infrastruktur und das Vorwissen der beteiligten Komponenten stellen wollen. Nur so ist die maximale Spontanität und Flexibilität erreichbar, die auch bei spontanen, pervasiven Umgebungen angetrebt wird. Im Laufe dieser Arbeit wurde daher anstelle des Brokers der Begriff der *Service Busses* verwendet. Dieser wurde für den Rahmen der vorliegenden Arbeit wie folgt definiert:

Definition 3 *Als* ***Service Bus*** *werden Systeme zur transienten oder persistenten Übertragung von Service-Beschreibungen zwischen SOA-Providern und -Consumern bezeichnet.*

Persistente Service Busse sind meist klassische Broker, die Service-Beschreibungen über einen längeren Zeitraum speichern, während *transiente* Service Busse beispielsweise Multicast-Kanäle sind, die die Service-Beschreibungen nur für die Dauer eines Advertisements bzw. eines Requests enthalten. Neben der Persistenz unterscheiden sich die existierenden Service Busse vor allem im Hinblick auf ihre Aktivität bzw. Passivität.

2.3.2 Aktivität des Service Busses

Wenn Service-Beschreibungen von ihren Providern im Service Bus veröffentlicht werden, kann dieser sie auf zwei Arten Consumern bekannt machen. Zum ersten indem sie bei Veröffentlichung *aktiv* an den Consumer gesendet werden und zum zweiten *passiv*, indem sie gespeichert werden, um erst bei entsprechenden Requests des Consumers ausgeliefert zu werden. Da praktisch keine relevante SOA-Implementierung rein aktiv arbeitet, gibt es im Hinblick auf Bus-Aktivität lediglich zwei SOA-Klassen:

SOAs mit einem *Request-basierten Service Bus* wie Web Services, Jini und Bluetooth SDP verhalten sich ausschließlich passiv. Dies hat den Vorteil eines reduzierten Verwaltungsaufwands beim Consumer, da dieser erst bei Bedarf verfügbare Service-Beschreibungen abfragt. Nachteilig ist die beschränkte Reaktionsfähigkeit auf neu veröffentlichte Service-Beschreibungen, da diese eine erneute Abfrage des Consumers erfordern.

Um eine schnellere Reaktion auf neu veröffentlichte Service-Beschreibungen zu ermöglichen, verwenden einige SOA-Implementierungen wie UPnP und DNS-SD (mit mDNS-Unterstützung) einen *Service Bus mit Advertisement-Unterstützung*. Dessen Service-Beschreibungen können ebenfalls per Request gefunden werden. Zusätzlich besteht für Provider die Möglichkeit aktiv Advertisements zu verschicken, die von Consumern empfangen

werden können, die somit sofort auf neue Service-Beschreibungen reagieren können. Da Advertisements für gewöhnlich IP-Multicasts zugrunde liegen, ist diese Funktionalität auf lokale Netzwerke beschränkt, da die meisten Router IP-Multicasts nicht standardmäßig weiterleiten.

Sobald Service-Beschreibungen durch den Service Bus gefunden werden, stellt sich die Frage der Service-Nutzung. Auch hier unterscheiden sich vorhandene SOAs deutlich voneinander.

2.3.3 Spezifikation der Service-Nutzung

Die unterschiedliche Herangehensweise an die Nutzung von Services hat ihren Ursprung in dem unterschiedlichen Service-Verständnis der SOA-Implementierungen. Während beispielsweise Web Services den Anspruch haben das komplette SOA-Konzept zu unterstützen, sind andere Implementierungen wie DNS-SD eher als Service Discovery-Lösungen konzipiert. In dieser Arbeit wurden drei Klassifizierungen der Service-Nutzung identifiziert:

SOAs mit *SOAP-basierter Service-Nutzung* spezifizieren die Service-Nutzung über SOAP. Dazu gehören UPnP, bei dem die Nutzung von SOAP vorgeschrieben ist und Web Services, bei denen sich SOAP als Standardlösung durchgesetzt hat.

Andere Implementierungen verwenden eine *SOA-spezifische Service-Nutzung*. Jini setzt z.B. auf ein serialisiertes Java-Objekt, das der Consumer zur Kommunikation mit dem Service einbinden muss, während Bluetooth SDP-Services über eine Reihe von Bluetooth-Profilen auf sehr unterschiedliche Art und Weise angesprochen werden.

Außerdem gibt es viele Lösungen mit *nicht-spezifizierter Service-Nutzung* wie DNS-SD und SLP. Diese Implementierungen fokussieren ausschließlich den Discovery-Prozess, während die Service-Nutzung vollkommen unberücksichtigt bleibt. DNS-SD verweist zwar auf eine Vielzahl möglicher Service-Typen, doch deren eigentliche Nutzung kann selbst bei zwei Services des gleichen Typs unterschiedlich sein.

Neben dem Service-Verständnis ist es vor allem die Anwendungsdomäne der SOA-Implementierung, die im Hinblick auf die Technologiebindung für weitere Unterschiede sorgt.

2.3.4 Technologiebindung

Auch wenn das SOA-Konzept selbst unabhängig von konkreten unterliegenden Technologien wie Laufzeitumgebungen, Programmiersprachen oder Betriebssystemen ist, sind dies nur wenige konkrete SOA-Implementierungen. Da die Abhängigkeit von bestimmten Technologien auch beim Interoperabilitätskonzept zwischen SOA-Implementierungen berücksichtigt werden muss, wurden diese in drei Klassen eingeteilt:

Keine Technologiebindung haben SOA-Implementierungen wie Web Services. Da Web Services XML-basiert sind, ist theoretisch lediglich wichtig, dass die benötigten XML-Dokumente (z. B. WSDL- und SOAP-Dateien) zwischen den SOA-Komponenten ausgetauscht und ausgewertet werden, jedoch nicht, wie dies geschieht.

Trotzdem hat sich auch bei Web Services die Verwendung von HTTP als Übertragungsprotokoll etabliert, womit Web Services faktisch *technologiegebunden* sind.

	Web Services	**UPnP**	**Jini**	**DNS-SD**	**Bluetooth SDP**
Broker	ja	nein	ja	optional	nein
Service Bus	Request	Request und Advertisement	Request	Request und Advertisement	Request
Service-Nutzung	SOAP	SOAP	SOA-spezifisch	nicht spezifiziert	SOA-spezifisch
Technologiebindung	theoretisch keine Technologiebindung aber faktisch technologiegebunden (IP)	technologiegebunden (IP)	technologiegebunden (IP, Java)	technologiegebunden (IP)	technologiegebunden (Bluetooth)

Tabelle 2.2: Übersicht über die zur Interoperablilität relevanten technischen Merkmale der wichtigsten SOA-Implementierungen.

Diese Bindung wird durch die Verwendung des heute allgegenwärtigen Internet Protokolls sowie UDP bzw. TCP als Transportprotokoll geschaffen. Neben Web Services gehören dazu auch UPnP, SLP, DNS-SD sowie Jini und Bluetooth SDP. Jini findet heute nur zusammen mit Java und einer JVM Verwendung während Bluetooth SDP sogar auschließlich mit der nicht-IP-basierten Bluetooth-Technologie verwendet wird.

2.3.5 Übersicht der SOA-Klassifizierungen

Vor allem die umfangreicheren SOA-Implementierungen wie Web Services, UPnP und Jini bieten eine Reihe weiterer Klassifizierungsansätze wie beispielsweise unterstützte Sicherheitsmechanismen bzw. Eventing. Im Gegensatz zu den bisher in diesem Abschnitt vorgestellten und in Tabelle 2.1 zusammengefassten technischen Klassifizierungen spielen diese jedoch im Rahmen dieser Dissertationsschrift keine wichtige Rolle. Sie führen zwar für die jeweilige SOA-Implementierung eine Reihe zusätzlicher Funktionalitäten ein, sind aber im Hinblick auf die grundlegende Interoperabilität, die Ziel dieser Arbeit ist, eher irrelevant. Im Gegensatz dazu werden die bisher eingeführten Klassifizierungen in den folgenden Kapiteln eine wichtige Rolle bei der Erarbeitung von Interoperabilitätsstrategien spielen.

2.4 SOA-Kommunikationsmodell für pervasive Umgebungen

Das in diesem Kapitel eingeführte Kommunikationsmodell der Service-orientierten Architektur baut auf dem ebenfalls eingeführten Broker-Modell auf. Durch die Nutzung zusätzlicher Mechanismen wie Services sowie verschiedene Service Bus-Lösungen ist es jedoch mehr als

alle anderen Kommunikationsmodelle prädestiniert die infrastrukturelle Grundlage pervasiver Systeme zu bilden, da es den Großteil der in Abschnitt 2.1 aufgeführten Anforderungen erfüllt:

- **Hard- und Software-Abstraktion** ist in Form von Services gegeben.
- **Spontane Komponenteneinbindung** ist durch das Broker-Modell gegeben.
- **Spontane Komponentenanpassung** ist Aufgabe der Service-Anbieter.
- **Komponentenverteilung und -migration** ist durch das Broker-Modell gegeben.
- **Vorwissen** ist bei mehrerer SOA-Implementierungen durch Advertisements nicht erforderlich.
- **Software-Laufzeitumgebungen** sind nicht durch allgemeine SOA-Konzepte vorgesehen, jedoch für einige SOA-Implementierungen wie Web Services und Jini verfügbar.
- **Gute Skalierbarkeit** ist durch das Broker-Modell gegeben.
- **Integration existierender Komponenten** ist möglich, da viele der auf dem Markt befindlichen Geräte und Anwendungen bereits eine SOA-Implementierung unterstützen. Andere Komponenten können durch eine Service-Schnittstelle erweitert und dann ebenfalls als Service angeboten werden.
- **Autonome, kontrollierende Komponenten** sind in Form von Services, die völlig in der Kontrolle ihrer Anbieter verbleiben, gegeben.

Dennoch wirft die Nutzung Service-orientierter Architekturen das bereits angesprochene Problem der Vielfalt verfügbarer Implementierungen auf und bedarf Konzepte zur implementierungsübergreifenden SOA-Kommunikation. Vorhandene Interoperabilitätslösungen werden im folgenden Abschnitt detailliert betrachtet und bewertet sowie ein eigenes Modell vorgeschlagen.

Kapitel 3

SOA-Interoperabilität für pervasive Umgebungen

Die heute angestrebten pervasiven Umgebungen bestehen aus Geräten unterschiedlicher Anwendungsbereiche. Diese Heterogenität drückt sich auch auf der Ebene Service-orientierter Architekturen aus. So könnte das in Abschnitt 1.1.3 beschriebene Meeting-Szenario des MuSAMA-Projektes beispielsweise folgende Geräte und Anwendungen und somit verschiedene SOA-Implementierungen einbinden:

- mobile Nutzerendgeräte (z. B. Laptops, Smartphones), die für SOA-Implementierungen wie Apples DNS-SD oder Microsofts UPnP ausgelegt sind
- herkömmliche Mobiltelefone, die dank Bluetooth-Schnittstelle SDP unterstützen
- stationäre Rechner bzw. Webserver die Web Services anbieten bzw. nutzen
- Sensoren, die eine SOA-Implementierung für PANs (z. B. Bluetooth SDP oder ressourcenschonendere Implementierungen) benötigen
- kooperative Anwendungen, die auf Jini als Kommunikationsschnittstelle setzen

Um derartig unterschiedliche SOA-Implementierungen zusammenzuführen, sind verschiedene Ansätze zur Interoperabilität möglich, die in diesem Kapitel ausgeführt und bewertet werden. Weiterhin wird das in dieser Arbeit entstandene Interoperabilitätskonzept des General Purpose Access Points (GPAP) detailliert erläutert und eine Betrachtung der für Pervasivität wichtigen Kontextinformationen und deren Bedeutung für SOA vorgenommen.

3.1 Ansätze zur SOA-Interoperabilität

Jede SOA-Implementierung ist in dem Anwendungsbereich effektiv und verbreitet, für den sie entwickelt wurde. Somit schließt die Konzentration auf eine einzige SOA-Implementierung eine Reihe von Geräten von der Nutzung in der pervasiven Umgebung aus. Verwandte Arbeiten im Bereich der SOA-Interoperabilität können im Hinblick auf ihre generelle Heransgehensweise klassifiziert werden. Dafür wurden im Laufe dieser Arbeit vier Grundansätze identifiziert,

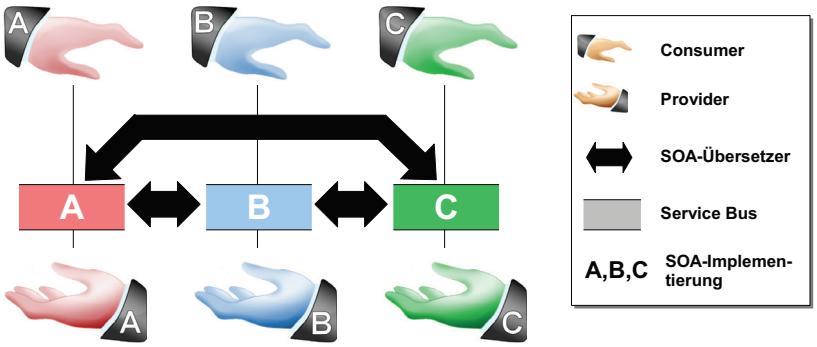


Abbildung 3.1: Mittels Technologiebrücken aus verwandten Arbeiten ist volle Interoperabilität erreichbar.

die in diesem Kapitel inklusive beispielhafter Vertreter näher erläutert werden. Dabei gibt es stets vier Komponenten:

- *Provider*, die Services in Form von Service-Beschreibungen ihrer jeweiligen SOA-Implementierung (A, B oder C) anbieten
- *Consumer*, die Services der von ihnen verwendeten Implementierung nutzen
- *Service-Busse*, die die Menge aller verfügbaren Service-Beschreibungen einer Implementierung beinhalten
- *SOA-Übersetzer*, die Service-Beschreibungen einer SOA-Implementierung in Beschreibungen desselben Services in einer anderen SOA-Implementierung übersetzen (z. B. von A zu B)

Die folgenden Interoperabilitätsansätze werden vor allem im Hinblick auf ihre Eignung bewertet, volle Interoperabilität zwischen verfügbaren SOA-Implementierungen zu realisieren:

Definition 4 ***Volle Interoperabilität*** *zwischen n SOA-Implementierungen $t_1, t_2, \ldots t_n$ ist dann erreicht, wenn für jeden Service s einer SOA-Implementierung $t_i \in \{t_1, t_2, \ldots t_n\}$ n Service-Beschreibungen $b_j(t_j, s)$ existieren oder zur Laufzeit erzeugt werden können, für die gilt: $t_j \in \{t_1, t_2, \ldots t_n\}$ und $b_j(t_j, s) \equiv b_i(t_i, s)$. D. h. dass für jeden Service mehrere Service-Beschreibungen existieren, mindestens eine pro SOA-Implementierung.*

3.1.1 Technologiebrücken

Die erste Klasse von Interoperabilitätsansätzen verwendet *Technologiebrücken*. Geboren aus der Notwendigkeit zwei SOA-Implementierungen und somit zwei Anwendungsbereiche miteinander zu verknüpfen, existieren derzeit viele duale Interoperabilitätslösungen. Wie in Abbildung 3.1 illustriert, bilden diese Services einer SOA-Implementierung A auf Services einer SOA-Implementierung B ab. Somit können Services, die von A-Providern angeboten werden, von B-Consumern genutzt werden und umgekehrt. Die Technologiebrücke arbeitet jeweils als Provider und Consumer sowohl in A- als auch in B-SOAs. Im Folgenden werden beispielhaft zwei Vertreter dieses Ansatzes aufgeführt.

Das *Jini/UPnP Interoperability Framework* [Allard 03] ist ein Beispiel für eine Technologiebrücke zwischen Jini und UPnP. Es verwendet einen speziellen UPnP-Consumer, der UPnP-Service-Beschreibungen findet und in Jini-Service-Beschreibungen übersetzt sowie einen speziellen Jini-Consumer für die entgegengesetzte Richtung. Dabei ist für jeden Service-Typ in Jini oder UPnP ein eigenes Plugin notwendig, um die Übersetzung durchzuführen, wodurch der semantische Unterschied zwischen einen Jini-Service und seinem UPnP-Äquivalent gering gehalten werden kann. Andererseits ist mit der Menge an möglichen Jini- und UPnP-Service-Typen ein erheblicher Aufwand zur Plugin-Erstellung verbunden.

Die *Jini/SLP Bridge* [Guttman 99a] ist ein weiteres Beispiel für eine Technologiebrücke. Sie wurde entwickelt, um *Thin Clients* (die nicht genügend Ressourcen für eine komplette JVM besitzen) das Anbieten von Jini-Services zu ermöglichen. Dafür wird ein spezieller Factory-Service in SLP angeboten, mit dem Thin Clients als SLP-Provider Jini-Service-Beschreibungen veröffentlichen können. Sie müssen dazu jedoch ein dediziertens Codefragment als Treiber beim Factory-Service hinterlegen, mit dem aus Java heraus auf Java-fremde Anwendungen zugegriffen werden kann. Somit ist diese Lösung nicht transparent für den anbietenden SLP-Provider. Da sie außerdem im Gegenzug nicht das Angebot von Jini-Services in Form von SLP-Service-Beschreibungen ermöglicht, ist sie nicht als vollwertige sondern nur als einseitige Technologiebrücke zu sehen.

Der Technologiebrückenansatz hat den Vorteil, dass er bereits vorhandene Lösungen zur Realisierung voller Interoperabilität verwendet. Problematisch ist, dass er bei einer hohen Anzahl unterstützter SOA-Implementierungen sehr schlecht skaliert, denn volle Interoperabilität zwischen t SOA-Implementierungen erfordert $\binom{t}{2}$ Übersetzer bzw. Technologiebrücken. Deutlich besser skalieren Interoperabilitätsansätze, die auf Abstraktion setzen.

3.1.2 Abstrakter Provider

Eine Vereinheitlichung von SOA-Implementierungen über **abstrakte Provider** bietet keine Services in einer konkreten SOA-Implementierung an, sondern verwendet abstrakte Service-Beschreibungen (Abbildung 3.2). Implementierungsspezifische SOA-Übersetzer generieren aus diesen Beschreibungen Service-Beschreibungen für alle unterstützten SOA-Implementierungen.

Ein Vetreter dieses Interoperabilitätsansatzes ist die *Unified Service*-Middleware [Uribarren 06]. Diese strebt die Integration unterschiedlicher Geräte in eine pervasive Umgebung an. Consumer-Anwendungen sollten diese durch beliebige SOA-Implementierungen wie UPnP und Web Services finden und nutzen können. Erreicht wird dieses Ziel durch eine Middleware, die für jede Geräteklasse einen dedizierten Treiber vorrätig hält, der für jedes verfügbare Gerät seiner Klasse eine abstrakte Service-Beschreibung erzeugt, der wiederum in konkrete SOA-spezifische Service-Beschreibungen umgewandelt werden kann.

Leider grenzt dieser Ansatz alle Geräte und andere Provider aus, die bereits mit einer konkreten SOA-Implementierung auf dem Markt sind, wie beispielsweise das breite Spektrum an DNS-SD-Druckern oder UPnP-fähigen Geräten. Er ist somit im Hinblick auf die Consumer- und -Provider-Transparenz nicht praktikabel für pervasive Umgebungen.

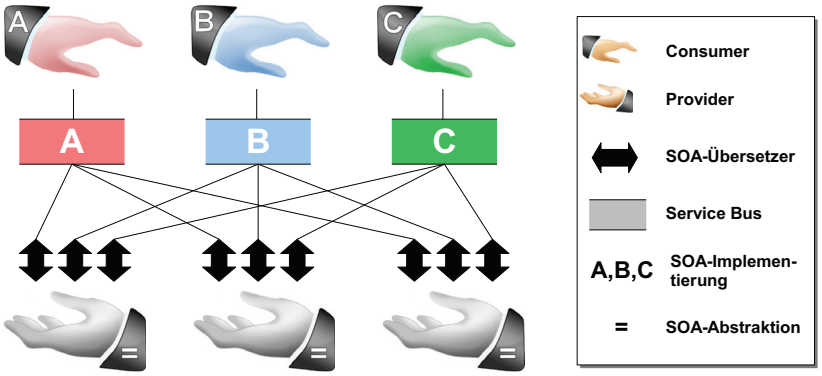


Abbildung 3.2: Abstrakte Provider bieten abstrakte Service-Beschreibungen an, aus denen von SOA-Übersetzern konkrete Service-Beschreibungen generiert werden.

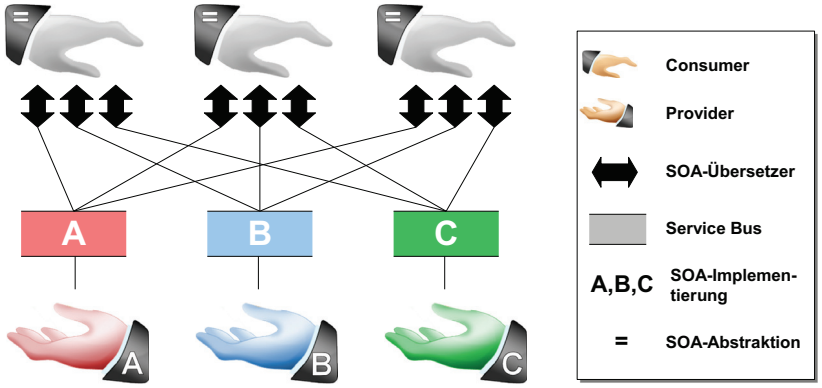


Abbildung 3.3: Abstrakte Consumer finden und nutzen abstrakte Service-Beschreibungen. Diese werden von SOA-Übersetzern aus konkreten Service–Beschreibungen abgeleitet.

Dennoch skaliert dieser Ansatz wesentlich besser als der vorhergehende. Da die SOA-Übersetzer auf den Providern betrieben werden, erfordert volle Interoperabilität zwischen t SOA-Implementierungen bei p Providern nur $p \cdot \ t$ Übersetzer. Allerdings müssen nur t Plugins entwickelt werden, die dann auf den p Providern wiederverwendet werden können. Weiterhin können Consumer zwar ihre ursprüngliche SOA-Implementierung nutzen, während Provider ihre Dienste jedoch auf abstrakte Art anbieten müssen. Analog dazu gibt es auch den Ansatz der Abstraktion auf Consumer-Seite.

3.1.3 Abstrakter Consumer

Wie in Abbildung 3.3 dargestellt, bieten Provider ihre Services bei diesem Ansatz in ihrer ursprünglichen SOA-Implementierung an, während die Consumer SOA-Übersetzer verwenden, um Services aller unterstützen SOA-Implementierungen einheitlich nutzen zu können.

Die *Unified Service Search Engine* [Pantazoglou 06] ist ein Beispiel für die Verwendung abstrakter Consumer. Mittels *Unified Service Query Language* (USQL) können abstrakte

Requests an einen USQL-Handler gestellt werden, der diese an die Plugins für konkrete SOA-Implementierungen weiterleitet. Diese wandeln die abstrakten Requests in konkrete Requests und senden sie an den jeweiligen Service-Bus. Die resultierenden konkreten Service-Beschreibungen werden wiederum in USQL-Responses umgewandelt und an den Consumer zurückgeliefert.

Auch hier besteht der Nachteil, dass Anpassungen bei den SOA-Teilnehmern (hier dem Consumer) vorgenommen werden müssen. Da es allerdings auf dem Markt bereits ein großes Spektrum SOA-spezifischer Consumer gibt und diese modifiziert werden müssten, ist dieser Ansatz ebenfalls nicht praktikabel, zumal viele Hersteller an der Durchsetzung ihrer favorisierten SOA-Implementierung gegenüber der Konkurrenz interessiert sind (z. B. Apple mit DNS-SD und Microsoft mit UPnP).

Analog zum vorhergehenden Ansatz müssen durch abstrakte Consumer bei c Consumern nur $c \cdot t$ wiederverwendbare Plugins eingesetzt werden. Bei einer vergleichbaren Anzahl von Providern, Consumern und SOA-Implementierungen skaliert dieser Ansatz somit ähnlich wie die Nutzung abstrakter Provider. Ganz anders verhält es sich beim vierten Interoperabilitätsansatz.

3.1.4 Zentrale Vereinheitlichung

Durch den Ansatz der **zentralen Vereinheitlichung** werden, wie in Abbildung 3.4 dargestellt, mit zentralen Übersetzern neue Komponenten eingeführt. Jeder zentrale Übersetzer bündelt mehrere einzelne SOA-Übersetzer. Alle Service-Beschreibungen, die SOA-spezifische Provider anbieten, empfängt ein SOA-Übersetzer. Er erzeugt daraus abstrakte Service-Beschreibungen, die der zentrale Übersetzer in einem abstrakten Service-Bus verwaltet. Diese können wiederum in SOA-spezifische Service-Beschreibungen umgewandelt und in allen unterstützten Implementierungen angeboten werden. Der abstrakte Service-Bus kann sowohl lokal auf einem einzigen zentralen Übersetzer existieren, als auch verteilt auf mehreren, wodurch Service-Beschreibungen auch über größere Distanzen hinweg ausgetauscht werden können.

Ein Beispiel für SOA-Interoperabilität mittels zentraler Vereinheitlichung ist die *Open Service Discovery Architecture* (OSDA) [Limam 07]. Diese nutzt sogenannte *Registration Advertisement Handler*, die sich während der Veröffentlichung eines Services zwischen seinen Provider und den Service-Bus schalten, um die Service-Beschreibung auszulesen, in ein abstraktes Format zu überführen und in einem abstrakten und verteilten Service-Bus zu speichern. Im Gegenzug schalten sich *Discovery Request Handler* zwischen Consumer und Service-Bus, um den Request des Consumers abzuhören. Dieser wird in einen abstrakten Request übersetzt und auf dem abstrakten Service-Bus angewendet, um passende Service-Beschreibungen herauszufiltern und dem Consumer zurückzuliefern. Auf diese Art können Service-Beschreibungen zwischen verschiedenen SOA-Implementierungen übermittelt werden und mittels verteiltem abstraktem Service-Bus sogar räumlich voneinander getrennte SOAs zusammengeführt werden. Leider ist dieser Ansatz in seiner Allgemeingültigkeit sehr beschränkt, da es nicht immer möglich ist, die Kommunikations zwischen Provider/Consumer und Service-Bus abzufangen. Insbesondere in Broker-basierten SOAs (wie Web Services oder DNS-SD mit DNS-Servern) kommunizieren Provider und Consumer per Unicast mit ihrem Broker. Diese Kommunikation abzufangen würde erhebliche Eingriffe in die SOA-Implementierung bedeuten. Vor allem

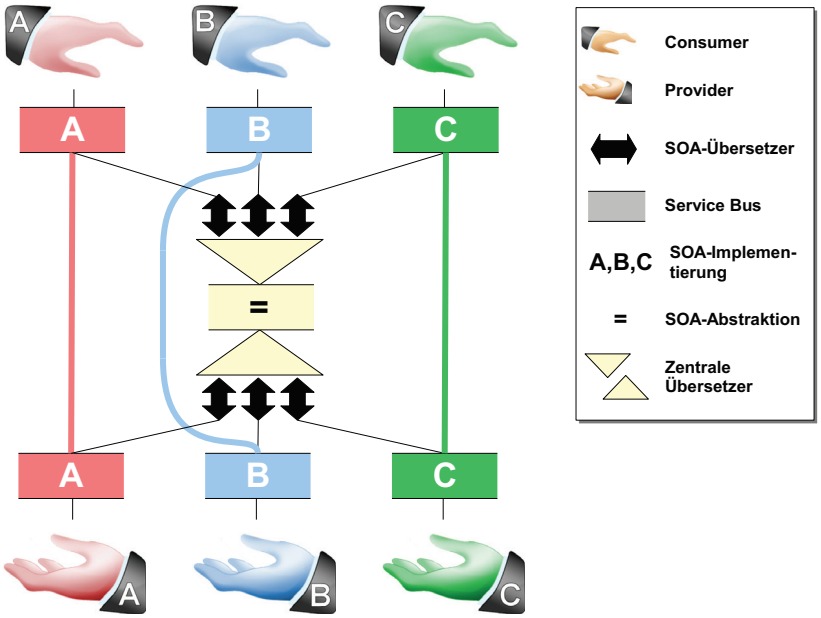


Abbildung 3.4: Bei der Zentralen Vereinheitlichung verbleiben Consumer und Provider in ihren SOA-Implementierungen, während sämtliche Übersetzungen durch zentrale Übersetzer durchgeführt werden.

wenn der Unicast verschlüsselt erfolgt, ist dieser Eingriff nicht nur sicherheitstechnisch bedenklich, sondern obendrein nicht trivial. In Broker-freien SOAs können zwar Multicasts mitgehört werden, doch bei SOAs wie Bluetooth SDP ist schon aus physikalischen Gründen kein Eingriff in die Bluetooth-interne Kommunikation möglich. Dennoch bietet OSDA innovative Teillösungen, wie die Nutzung von XML als abstraktes Format zur Service-Beschreibung, welches über alle denkbaren Netzwerke ausgetauscht werden kann.

Dieser Ansatz der zentralen Vereinheitlichung hat somit folgende Vorteile:

- Eine Modifikation existierender Provider oder Consumer ist nicht erforderlich.
- Volle Interoperabilität zwischen t SOA-Implementierungen erfordert bei z zentralen Übersetzern nur $z \cdot t$ Plugins, die wiederverwendbar sind. Da ein zentraler Übersetzer für ein ganzes lokales Netzwerk zuständig sein kann, welches mehrere Provider (p) und Consumer (c) beinhalten würde, gilt meist $z < p$ beziehungsweise $z < c$. Dieser Ansatz benötigt somit die geringste Anzahl von SOA-Übersetzern (t), die außerdem wiederverwendbar sind.
- Der abstrakte Service-Bus ist die Grundlage für eine Ausdehnung der Discovery-Reichweite.

Verglichen mit den anderen Ansätzen ist die zentrale Vereinheitlichung somit sowohl funktional als auch im Hinblick auf die Effizienz für pervasive Umgebungen zu bevorzugen. Zwar kann der zentrale Übersetzer zu einem *Single Point of Failure* werden, durch eine entsprechend robuste Komponente und/oder Reserve-Übersetzer kann dieser Fall jedoch im praktischen Einsatz verhindert oder behandelt werden. Aufgrund dieser Betrachtungen ist die zentrale

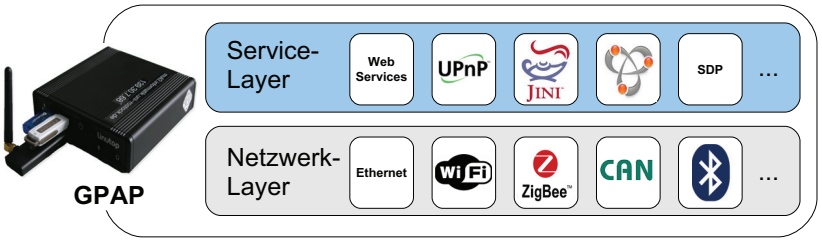


Abbildung 3.5: Die Aufgabe des GPAPs ist die Überwindung von Heterogenität auf dem Netzwerk- und dem Service-Layer.

Vereinheitlichung die Grundlage für den in dieser Arbeit entwickelten *General Purpose Access Point* (GPAP) als zentralen Übersetzer.

3.2 Der General Purpose Access Point (GPAP)

Die Basis der angestrebten dynamischen pervasiven Umgebungen ist ein interoperables und effizientes Kommunikationssystem. Heutige pervasive Umgebungen (z. B. Smart Homes) sind oftmals komplexe und statische Einzellösungen. Die Heterogenität sowohl auf dem Netzwerk- als auch auf dem Service-Layer verhindert die transparente Einbindung neuer Geräte in derartige Systeme. Der im Folgenden vorgestellte *General Purpose Access Point* (GPAP) überwindet diese Heterogenität systematisch auf diesen beiden kritischen Layern (Abbildung 3.5).

Die Entwicklung des GPAPs geschieht in Zusammenarbeit mit Herrn Enrico Dressler, der das Netzwerk-Layer bearbeitet, während das Service-Layer Kernthema der vorliegenden Arbeit ist. Die Grundlage beider Arbeiten ist ein dreiteiliges Netzwerkmodell, welches in Abbildung 3.6 beispielhaft dargestellt ist. Das heterogene Netzwerk und seine Teilnehmer werden in *Zellen*, *Ensembles* und die *Community* unterteilt [Dressler 08].

Die kleinste Netzwerkeinheit ist die *Zelle*. Sie beinhaltet alle Geräte, die aufgrund einer gemeinsamen Netzwerktechnologie (z. B. Bluetooth oder WLAN) miteinander kommunizieren können.

Definition 5 ***Zellen*** *sind lokale Sammlungen homogen miteinander vernetzter Geräte.*

Zur Vernetzung kommen sowohl drahtgebundene als auch drahtlose Technologien in Frage. Drahtlose Zellen sind dynamischer als drahtgebundene Zellen, da Geräte spontan die Zellreichweite verlassen oder betreten können. Weiterhin können unterschiedliche Zellen durch Geräte, die mehr als eine Kommunikationstechnologie verwenden, miteinander verbunden werden. Dies wird beispielsweise in der Arbeit von Herrn Dressler durch das Netzwerk-Layer des GPAPs errreicht [Dressler 09]. Durch die Verbindung von Zellen ist der Aufbau von *Ensembles* möglich.

Definition 6 ***Ensembles*** *sind lokale Sammlungen heterogen, permanent und/oder temporär vernetzter Geräte.*

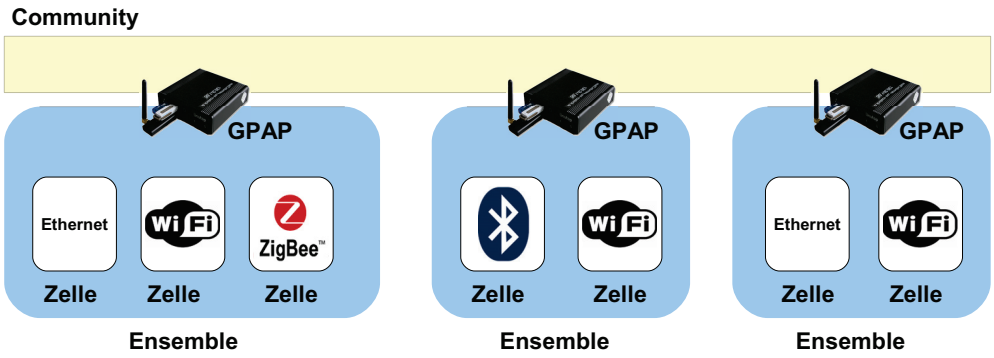


Abbildung 3.6: Dem GPAP-Konzept liegt eine Unterteilung pervasiver Umgebungen in Zellen, Ensembles und Communitys zugrunde.

Die Struktur eines Ensembles kann sich unvorhersehbar ändern, da Komponenten spontan hinzugefügt oder entfernt werden können. Dies gilt insbesondere für mobile Geräte. Die Geräte in einem Ensemble stellen anderen Ensemble-Mitgliedern ihre Fähigkeiten und Ressourcen in Form von Services zur Verfügung. Damit Services aus einem Ensemble in einem anderen Ensemble genutzt werden können, ist eine Vernetzung verschiedener Ensembles im Rahmen einer *Community* erforderlich.

Definition 7 ***Communitys*** *sind Sammlungen längerfristig miteinander verbundener Ensembles, in denen Service-Beschreibungen ausgetauscht werden, um diese über Ensemble-Grenzen hinweg zu finden und zu nutzen.*

Der GPAP stellt dabei die Schnittstelle zwischen Ensemble und Community dar. Durch Communitys sind Ensemble-Mitglieder in der Lage, Ressourcen und Fähigkeiten von Mitgliedern entfernter Ensembles zu nutzen. Das Community-Netzwerk ist statisch im Vergleich zum Netzwerk innerhalb eines Ensembles, da die GPAPs als stationäre und zuverlässige Netzwerkkomponenten angesehen werden können.

Bei der Umsetzung dieses Netzwerkmodells spielt der GPAP die Schlüsselrolle. Er hat mehrere unterschiedliche Netzwerkschnittstellen, so dass er ein breites Netzwerkspektrum (z. B. Ethernet, WLAN, Bluetooth) bedienen kann und verwendet zentrale Vereinheitlichung auf dem Software-basierten Service-Layer, um Interoperabilität zwischen SOA-Implementierungen zu erreichen. Zu seinen Aufgaben gehören:

- Paketvermittlung zwischen gleich- oder verschiedenartigen Zellen und damit Bildung von Ensembles (Netzwerk-Layer)
- Roaming für den Fall, dass ein mobiles Gerät ein Ensemble verlässt und einem anderen beitritt (Netzwerk-Layer)
- Übersetzung zwischen verschiedenartigen SOA-Implementierungen (Service-Layer)
- Bildung einer Community mit anderen GPAPs zur Verbindung von Ensembles (Service-Layer)
- Kontextbasierte Service-Auswahl als Ergebnis eines Service Discovery-Prozesses (Service-Layer)

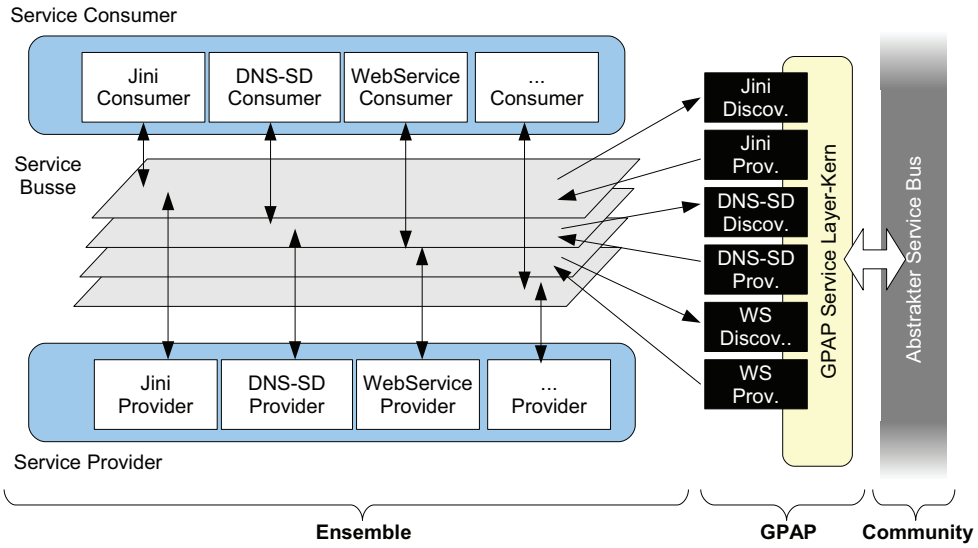


Abbildung 3.7: Auf seinem Service-Layer überwindet der GPAP SOA-Heterogenität mittels zentraler Vereinheitlichung.

Der GPAP ist somit die zentrale infrastrukturelle Komponente der pervasiven MuSAMA-Umgebungen. Dabei liegt ein besonderes Augenmerk auf der Transparenz für Geräte und Anwendungen. Diese sollen weiterhin ihre ursprüngliche Netzwerktechnologie und SOA-Implementierung nutzen können und trotzdem von den Vorteilen des GPAPs profitieren. Dadurch ist die Nutzung einer Vielzahl bereits auf dem Markt befindlicher Geräte möglich, ohne infrastrukturelle Anpassungen an einer pervasiven Umgebung vornehmen zu müssen. Die folgenden Abschnitte behandeln detailliert die Umsetzung der GPAP-Aufgaben der SOA-Interoperabilität und Community-Bildung, bevor Abschnitt 3.3 die Bedeutung von Kontextinformationen für die Service-Auswahl beleuchtet.

3.2.1 Zentrale Vereinheitlichung auf dem GPAP Service Layer

Um SOA-Interoperabilität zu erreichen, wird auf dem Service Layer des GPAPs der in Abschnitt 3.1.4 vorgestellte Ansatz der zentralen Vereinheitlichung umgesetzt. Wie Abbildung 3.7 illustriert, verbindet der GPAP jeweils ein lokales Ensemble mit einer Community. Er ist somit sowohl für die SOA-übergreifende Kommunikation innerhalb seines Ensembles als auch die Verbindung seines Ensembles mit einer Community aus abstrakten Diensten zuständig.

Innerhalb der Ensembles mit GPAP bedienen Provider und Consumer weiterhin ihre implementierungsspezifischen Service-Busse. Jini Provider veröffentlichen beispielsweise ihre Service–Beschreibungen im Jini Service-Bus (realisiert durch einen Broker) und Jini Consumer finden diese Beschreibungen in diesem. Der GPAP enthält eine Reihe von Plugins. Diese kapseln die Strategien zum Veröffentlichen und Finden von Service-Beschreibungen einer konkreten SOA-Implementierung, um den prozeduralen Unterschieden der einzelnen Implementierungen gerecht zu werden (siehe Abschnitt 3.2.3). Dabei gibt es drei Plugin-Arten: *Provision-Plugins* zum Veröffentlichen von Service-Beschreibungen, *Discovery-Plugins* zum

Finden von Service-Beschreibungen und *Broker-Plugins*, die diese beiden Arten kombinieren. In Abbildung 3.7 sind beispielhaft einige Provision- und Discovery-Plugins dargestellt (siehe Kapitel 4 für Implementierungsdetails). Zum Beispiel bedient ein Discovery-Plugin für Jini ebenfalls den Jini Service-Bus. Es ist dafür zuständig, Jini-Service-Beschreibungen zu finden und diese im Sinne der zentralen Vereinheitlichung in ein abstraktes Format zu überführen und die resultierenden Service-Beschreibungen in den abstrakten Service-Bus einzuspeisen. Im Gegenzug können Provision-Plugins (z. B. für die DNS-SD) Beschreibungen aus dem abstrakten Service-Bus beziehen, in konkrete Service-Beschreibungen umwandeln und diese in ihrem implementierungsspezifischen Service-Bus veröffentlichen.

Viele SOA-Implementierungen setzen auf Discovery-Verfahren ohne Vorkenntnisse und diese basieren meist auf IP-Multicasts. Da Multicasts von Routern für gewöhnlich nicht weitergeleitet werden, um die Netzlast zu reduzieren, ist der Wirkungsbereich eines GPAPs mitsamt seinen Plugins auf ein lokales Ensemble beschränkt. Um die Provider und Consumer seines Ensembles mit den Teilnehmern anderer Ensembles zu verbinden, ist eine Vernetzung mehrerer GPAPs in Form einer Community erforderlich, deren konkrete Realisierung in Abschnitt 4.1.6 diskutiert wird.

Der abstrakte Service-Bus miteinander vernetzter GPAPs ist auf diese verteilt. Der Ursprung der in den Bus eingespeisten Service-Beschreibungen, kann daher sowohl ein Discovery-Plugin eines lokalen als auch eines entfernten GPAPs sein. Im Gegenzug konkretisiert jedes Provision-Plugin abstrakte Service-Beschreibungen, deren Ursprung ein Service-Bus im lokalen als auch in entfernten Ensembles sein kann. Je nach Art des implementierungsspezifischen Discovery-Prozesses ist es für einige Provision-Plugins erforderlich Suchanfragen von Consumern entgegen zu nehmen und nur dazu passende Service-Beschreibungen aus dem abstrakten Service-Bus herauszufiltern. Aus den Anfragen müssen daher abstrakte Requests erstellt werden, die die passenden Service-Beschreibungen aus der Menge der im Bus verfügbaren Service-Beschreibungen zurückliefern. Eine einfache Sprache zur abstrakten Service-Beschreibung reicht demnach nicht aus.

Die abstrakte Sprache muss folgenden Anforderungen genügen:

- Die Sprache muss abstrakte **Service-Beschreibungen** sowie **filternde Requests** auf diesen Beschreibungen ausdrücken können.
- Sie muss möglichst **kompakt** sein, da in einer Community aus Ensembles sehr viele Dienste verfügbar sein werden. Weiterhin minimiert eine kompakte Sprache den Einarbeitungsaufwand für Plugin-Entwickler, wodurch wiederum die Akzeptanz gefördert wird.
- Um neben Beschreibungen und Requests weitere Konzepte, wie beispielsweise eine Community-weite Ereignisbeobachtung, zu realisieren, muss die Sprache **erweiterbar** sein.
- Die Sprache muss **unabhängig von einer konkreten Programmiersprache und Plattform** sein, da neben den GPAPs auch die Abstraktion und Konkretisierung durch andere Komponenten (z. B. Router bzw. Access Points oder sogar Sensor-Basisstationen) möglich sein soll. Weiterhin können Entwickler somit ihre favorisierte Programmiersprache verwenden, wodurch die Verbreitung gefördert wird.

Im Zuge der Recherchen zu dieser Arbeit konnte keine bereits existierende Sprache identifiziert werden, die diese Anforderungen erfüllt und so offen vorliegt, dass eine Integration in das GPAP-Modell problemlos möglich ist. Aus diesem Grund wurde mit der *Service Technology-independent Language* (STiL) eine eigene XML-basierte Sprache definiert, die diese Anforderungen erfüllt.

3.2.2 Die Service Technology-independent Language (STiL)

Die *Service Technology-independent Language* (STiL) bietet eine Abstraktion von Service-Beschreibungen konkreter SOA-Implementierungen und erlaubt die Ausführung von filternden Requests auf den resultierenden Abstraktionen. Somit kann vor allem der Discovery-Prozess implementierungsunabhängig durchgeführt werden. STiL wird ferner als Datenformat in der GPAP-basierten Community eingesetzt. Somit ist es möglich, vormals lokale und implementierungsspezifische Discovery-Prozesse auf eine globale Community auszudehnen und dabei nicht auf eine SOA-Implementierung zu beschränken. Da STiL komplett auf XML basiert, ist sie, wie gefordert, unabhängig von einer konkreten Programmiersprache und Plattform sowie erweiterbar. Durch die Optionalität der meisten Sprachelemente kann sie außerdem sehr kompakt gehalten werden. Im folgenden werden anhand von Beispielen die Funktionsweise und der Sprachumfang von STiL erläutert. Die komplette Spezifikation in Form des XML-Schemas kann Anhang A entnommen werden.

Ein STiL-Dokument entspricht einem der folgenden drei STiL-Basiselemente:

- *STiL-Description* zur abstrakten Beschreibung von Services
- *STiL-Request* zum Filtern von Service-Beschreibungen mit bestimmten Eigenschaften aus der Menge aller bekannten Descriptions
- *STiL-Response* als Antwort auf einen Request, die alle Descriptions enhält, die dem Filter des zugehörigen Requests entsprechen

Der Kern der STiL-Description ist der kleinste gemeinsame Nenner aus den in Kapitel 2 aufgeführten Service-Beschreibungen der SOA-Implementierungen Web Services, UPnP, Jini, DNS-SD und SLP. Jeder Service wird in irgendeiner Form mindestens durch drei Attribute beschrieben:

- Service-Typ
- Provider-Adresse
- Service-Beschreibung

Neben diesen obligatorischen Elementen werden in einer **STiL-Description** weitere optionale Elemente spezifiziert, die vor allem im Rahmen der Community-Bildung und GPAP-Funktionalität sinnvoll erscheinen (z. B. eine Gültigkeitsdauer, nach der der Dienst wieder entfernt wird oder eine Angabe der ursprünglichen SOA-Implementierung).

STiL-Description

Listing 3.1 zeigt eine beispielhafte STiL-Description, anhand der die einzelnen Elemente erläutert werden. Wie jedes STiL-Element wird die Description mit dem Tag `stil` eingeleitet. Durch das darauffolgende Tag `description` wird angezeigt, dass es sich um eine Service-Beschreibung handelt. Das einzige `description`-Attribut ist `provider`, welches die Adresse des Providers angibt. In den meisten Fällen ist dies ein *Uniform Resource Identifier* (URI). Für den Inhalt der Description gibt es zehn unterschiedliche Elementtypen, von denen einige optional sind oder gehäuft auftreten können.

```
<?xml version="1.0" encoding="UTF-8"?>

<stil version="0.2"
      xmlns="http://www.musama.de/STiL"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.musama.de/STiL STiL.xsd ">

 <description provider="bochum.informatik.uni-rostock.de:1234">
  <stilType>stil.pureOut.printer</stilType>
  <friendlyName>Druckdienst</friendlyName>
  <technology>Web Service</technology>
  <validity>10M</validity>
  <comProtocol>UDP</comProtocol>
  <comProtocol>TCP</comProtocol>
  <accessLocation>bochum.informatik.uni-rostock.de:1234/printService
   </accessLocation>
  <attribute name="location">Raum 126</attribute>
  <attribute name="color">true</attribute>
  <attribute name="dpi">4.800x1.200</attribute>
 </description>

</stil>
```

Listing 3.1: Beispiel: STiL-Description

Durch den obligatorischen `stilType` wird dem Service ein STiL-*ServiceType* zugewiesen. Dieser abstrahiert von dem Service-Typ einer implementierungsspezifischen Service-Beschreibung, und bildet die Grundlage für den semantischen Vergleich verschiedener Beschreibungen aus unterschiedlichen Implementierungen. Eine Übersicht über die aktuellen STiL-Typen kann Anhang B entnommen werden. Diese werden über eine in Abschnitt 4.1.2 beschriebene schlüsselwortbasierte Zuordnung aus den übrigen Service-Informationen abgeleitet. Eine vollständige Abbildung von Service-Typen konkreter Implementierungen auf STiL-Typen ist aufgrund der Komplexität vorhandener Implementierungen nicht möglich. Daher kann der urprüngliche Service-Typ in der Form `stil.realType?<realer Service-Typ>` codiert werden. In dem Beispiel wird durch den Service-Typ `stil.pureOut.printer` ein Druck-Service angeboten.

Durch den optionalen `friendlyName` wird eine kurze Beschreibung des Services angeboten. Diese ist für die Interpretation durch den Nutzer hilfreich.

Das ebenfalls optionale Element `technology` gibt an, in welcher SOA-Imlementierung der Service ursprünglich angeboten wird.

Die Gültigkeitsdauer der Service-Beschreibung ist durch das optionale Element `validity` codiert. Diese wird oft durch den Broker, bei dem die Service-Beschreibung hinterlegt wird, festgelegt. Die Gültigkeit wird als Dauer (`duration`) nach RFC 2445 (iCalendar) angegeben. Die Angabe einer Gültigkeitsdauer ist ebenfalls optional. Wird sie nicht angegeben, so kann von einer Gültigkeit auf unbestimmte Zeit ausgegangen werden. Der Druck-Service im Beispiel ist 10 Minuten gültig.

Um mit dem Service kommunizieren zu können, ist es wichtig, das gemeinsame Kommunikationsprotokoll festzulegen. Weit verbreitet sind die teilweise aufeinander basierenden Protokolle SOAP, HTTP, UDP und TCP. Das Kommunikationsprotokoll wird durch das Element `comProtocol` angegeben. Dieses Element ist optional, da das Protokoll auch durch die weiteren Elemente angegeben werden kann. Es ist wird jedoch empfohlen dieses Element zu nutzen, da dies dem Consumer erlaubt nach Services zu suchen, die ein bestimmtes Protokoll nutzen. `comProtocol` kann auch mehrfach verwendet werden, wenn unterschiedliche Kommunikationsprotokolle genutzt werden können.

Durch das Element `accessLocation` wird angegeben, unter welcher Adresse der Service erreicht werden kann. Dieses Element ist optional, wenn das Element `realDescription` existiert. `realDescription` enthält die Adresse, unter der eine implementierungsspezifische Service-Beschreibung abrufbar ist. Bei Web Services wird beispielsweise ein WSDL-Dokument als `realDescription` betrachtet. `realDescription` ist optional wenn `accessLocation` existiert, beide Elemente können aber auch gehäuft auftreten.

Das `extendedDescription`-Element ermöglicht den Verweis auf erweiterte Service-Informationen (z. B. als URI). Dazu gehören beispielsweise QoS-Vorraussetzungen des Services oder die Ereignisse, die der Service produziert oder konsumiert. Dieses Element hat das Attribut `name` für die Angabe der Beschreibungsbezeichnung und ist optional, kann aber auch mehrmals verwendet werden.

Durch das optionale und wiederholbare Element `attribute` können weitere Service-spezifische Attribute codiert werden. Für jedes `attribute` muss ein `name` angegeben werden. Im Beispiel wurden Standort, Farbdruckfähigkeit und Auflösung des zum Druck-Service gehörenden Druckers angegeben.

STiL-Request

Zur Auswahl einer oder mehrerer STiL-Descriptions aus einer größeren Menge bekannter STiL-Descriptions ist eine filternde Abfragesprache notwendig. An die Abfragesprache sowie den Abfrageprozess selbst werden folgende funktionsbedingten Anforderungen gestellt, die im Wesentlichen auf der Verwendung von SOA-spezifischen Discovery-Abfragen als Grundlage basieren:

- Die Abfragesprache muss **auf den STiL-Descriptions** basieren, da aus denen gefiltert werden soll.

- Als Ergebnis der Abfrage müssen **mehrere Services** möglich sein, da dies auch bei den meisten SOA-spezifischen Service Discovery-Prozessen der Fall ist.
- Wie SOA-spezifische Service Discovery-Prozesse, muss auch die STiL-Abfrage **in endlicher Zeit terminieren**.
- Die Abfragesprache muss **erweiterbar** sein, um zukünftige SOA-Neuentwicklungen integrieren zu können.

Da sie auf spezifischen Discovery-Prozessen basiert, die in den meisten Fällen sehr einfache Abfragen formulieren (bei SDP und Bonjour wird sogar nur der geforderte Service-Typ angegeben), ist somit keine syntaktisch komplexe Abfragesprache erforderlich. Ein einfacher Filter, der lediglich die geforderten Service-Attribute und ihre Belegung angibt, genügt im Hinblick auf die Discovery-Abfragen bisher untersuchter SOA-Implementierungen. Dennoch wurde die in dieser Arbeit entwickelte STiL-Abfragesprache (**STiL-Requests**) aufgrund der wichtigen Erweiterbarkeit nicht derart eingeschränkt, sondern unterstützt zusätzlich die Verknüpfung von Suchanfragen durch die aus der Aussagenlogik bekannten Disjunktionen, Negationen und somit implizit (durch Anwendung der De Morganschen Gesetze [Hackbusch 03]) auch Konjunktionen.

```
<?xml version="1.0" encoding="UTF-8"?>

<stil version="0.2"
      xmlns="http://www.musama.de/STiL"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.musama.de/STiL STiL.xsd ">

 <request timeout="1M" sender="toro.informatik.uni-rostock.de:1234"
   requestID="123456789">
  <serviceMask>
   <stilType>stil.pureOut.printer</stilType>
   <friendlyName restrict="contains">druck</friendlyName>
   <friendlyName restrict="contains">print</friendlyName>
   <technology>WebServices</technology>
   <technology>UPnP</technology>
   <validity restrict="minimum">2M</validity>
   <attribute name="color">true</attribute>
   <attribute not="true" name="busy">true</attribute>
  </serviceMask>
 </request>

</stil>
```

Listing 3.2: Beispiel: STiL-Request

Listing 3.2 zeigt beispielhaft einen STiL-Request, durch den Abfragen zu den Service-Eigenschaften einer Menge von STiL-Descriptions gestellt werden können. Jeder Request durch das `stil`-Kindelement `request` angezeigt und erwartet bis zu drei Attribute. Die obligatorische `requestID` kann beliebig belegt werden, sie wird im Response

wiederholt. Somit ist eine Zuordung jeder Antwort zu einer Anfrage möglich. Das ebenfalls obligatorische Attribut `sender` gibt die Adresse der anfragenden Instanz an (z. B. der GPAP des Ensembles mit dem anfragenden Consumer), an diese werden die Responses geschickt. Das optionale Attribut `timeout` gibt die Gültigkeit des Requests an, danach eintreffende Antworten werden von der anfragenden Instanz abgelehnt. Die Gültigkeit ist als `duration` gemäß RFC 2445 anzugeben. Wird kein Timeout angegeben, ist der Request bis auf unbestimmte Zeit gültig.

Jeder Request besteht wiederrum aus einer Reihe von Suchmasken für Service-Beschreibungen (`serviceMask`), um die gesuchten Service-Beschreibungen aus den verfügbaren Service-Beschreibungen heraus zu filtern. Da ein Request mehrere Suchmasken enthalten kann, kann auch das `serviceMask`-Element mehr als einmal vorkommen. Alle vorhandenen Suchmasken sind per Disjunktion miteinander verknüpft. Somit ist eine STiL-Description für einen STiL-Request gültig ($Match(Description, Request) = true$), sobald sie auch nur einer der angegebenen Suchmasken $Mask_1, Mask_2, \ldots Mask_n$ entspricht. Somit gilt für jede Description:

$$Match(Description, Request) = \\ Match(Description, Mask_1) \vee Match(Description, Mask_2 \vee \ldots \vee Match(Description, Mask_n)$$

Da jedes Tag `<ServiceMask>` ein `not`-Attribut erhalten kann, welches mit der Belegung `true` die angegebene Suchmaske negiert sowie eine Sequenz von Suchmasken als Ganzes über ein umschließende `<not>`-Tag negiert werden kann, ist nach De Morgan auch die konjunktive Verknüpfung von Suchmasken durch Disjunktionen und Negationen möglich:

$$Match(Description, Mask_1) \wedge Match(Description, Mask_2) \wedge \ldots \wedge Match(Description, Mask_n)) = \\ \rceil(\rceil Match(Description, Mask_1) \vee \rceil Match(Description, Mask_2) \vee \ldots \vee \rceil Match(Description, Mask_n))$$

Somit unterstützen STiL-Requests sowohl Disjunktion und Konjunktion als auch die Negation von Suchmasken für eine STiL-Description und sind daher auch für komplexere Service Discovery-Abfragen gewappnet als die, die in derzeit exitierenden SOA-Implementierungen eingesetzt werden.

Im Folgenden werden die einzelnen Elemente der `serviceMask` erläutert. Diese sind per Konjunktion miteinander verknüpft und können jeweils durch ein `not`-Attribut negiert werden. Um die Requests kompakt zu halten wurde zusätzlich eine weitere Regel eingeführt: Kommt ein Kindelement der `serviceMask` mehrfach vor, reicht es wenn die Service-Beschreibung einem der Elemente entspricht, um zur Anfrage zu passen. Hier liegt demnach eine Verknüpfung durch Disjunktion vor, durch die weitere Suchmasken, die sich lediglich in diesem Kindelement unterscheiden, erspart werden.

Damit ein Request nicht die gesamte Community flutet, wird mit `scope` eine numerische Reichweite angegeben. Diese kann je nach Art der Community genutzt werden, um beispielsweise die Anzahl der Hops, die der Request zurücklegen darf, zu beschränken. Ist das `scope`-Element nicht vorhanden, wird standardmäßig eine unbegrenzte Reichweite angenommen. Durch den bereits beschriebenen Timeout eines Requests kann dieser nach einer spezifizierten Zeit aus der Community verworfen und somit trotz

unbegrenzter Reichweite ein unkontrolliertes Fluten der Community mit Requests vermieden werden.

Der `stilType` ist das einzige obligatorische Element des Requests. Mit ihm kann der gesuchte Service-Typ, der im entsprechenden Element der Description festgehalten ist, spezifiziert werden. Dabei gilt standardmäßig: Wird nach einem Service-Type angefragt, der im Service-Typ-Baumgraphen kein Blattknoten ist, so sind auch alle tieferstehenden Service-Types mit eingeschlossen. Wird beispielsweise nach einem Service mit dem Service-Typ `stil.pureOut` gefragt, so passen sowohl `stil.pureOut`-Services als auch auch Services mit den Service-Type `stil.pureOut.video`, `stil.pureOut.audio` oder `stil.pureOut.printer` zur Anfrage. Soll diese Regel nicht gelten und somit tieferstehende Elemente nicht mit einbezogen werden, so ist das Attribut `restrict` zu verwenden und mit dem Wert `is` zu belegen.

Das Element `friendlyName`, welches das `stilType`-Element der Description prüft, besitzt ebenfalls das Attribut `restrict`. Ist dieses mit `contains` belegt, so muss der im Request angegebene String nur in der Description enthalten sein. Bei der Belegung mit `is` müssen beide Strings übereinstimmen, damit die Service-Beschreibung passt. Standardmäßig gilt `contains`. Die Prüfung erfolgt unabhängig von der Groß- und Kleinschreibung des Elementwertes.

Durch `technology` wird die ursprüngliche SOA-Implementierung der Services geprüft. Wurde die Service-Beschreibung in der hier angegebenen Implementierung angeboten, so ist sie für diese Anfrage gültig. Im Beispiel wird nach einem entweder als Web Service oder UPnP-Service angebotenen Service gefragt. Womöglich ist der Consumer in der Lage SOAP-Anfragen auf Services auszuführen, wodurch er diese beiden Implementierungen bedienen kann.

Das `validity`-Element erlaubt die Anfrage der Gültigkeitsdauer einer Service-Beschreibung. Wie auch in der Description wird dieser Wert als Dauer (`duration`) nach RFC 2445 angegeben. Das `restrict`-Attribut dieses Elementes kann mit `minimum` oder `maximum` belegt werden, um festzulegen, dass der `validity`-Wert im Request der Minimal- oder Maximalwert für die Gültigkeitsdauer passender Service-Beschreibungen ist.

Durch `comProtocol` wird geprüft, welche Kommunikationsprotokolle der Service unterstützt. Unterstützt der Service eines der hier angegebenen Protokolle, passt seine Beschreibung zu dieser Anfrage.

Im Element `extendedDescription` kann angegeben werden, zu welcher erweiterten Service-Beschreibung die gesuchten Service-Beschreibungen passen sollen. Dadurch kann beispielsweise nach Services gesucht werden, die bestimmte Ereignisse produzieren oder eine gesuchte Art von Kontext liefern. Das Attribut `name` wird genutzt, um die Art der erweiterten Beschreibung zu spezifizieren, die gesucht wird. Der Wert der `extendedDescription` ist in den meisten Fällen eine URL, die den Ort des beschreibenden Dokumentes angibt.

Service-spezifische Attribute können durch `attribute`-Elemente geprüft werden. Mit dem Attribut `name` wird der Name des Service-spezifische Attributs angegeben. Analog zu gleichnamigen Kindelementen sind `attribute`-Elemente nur dann innerhalb der Suchmaske per Disjunktion verknüpft, wenn ihr Attribut `name` den gleichen Wert hat.

Der Request in Listing 3.2 sucht somit eine beliebige STiL-Service-Beschreibung (`ServiceType` = `stil`), deren Service die Bezeichnung `print` oder `druck` enthält. Da die Service-Beschreibung aus Listing 3.1 den `ServiceType` `stil.pureOut.printer` und den `friendlyName` `Druckdienst` beschreibt, passt sie zunächst zum Request. Der Service soll laut Request als Web Service oder UPnP-Service angeboten werden. Die Beispiel-Beschreibung gehört zu einem als Web Service angebotenen Service, sie passt also weiterhin zur Abfrage. Außerdem wird gefordert, dass die Service-Beschreibung noch mindestens zwei Minuten gültig sein soll, diese Anforderung wird mit einer Gültigkeit von zehn Minuten ebenfalls erfüllt. Weiterhin soll das Service-spezifische Attribut `color` den Wert `true` haben. Da der Service der Beispiel-Beschreibung der eines Farbdruckers ist, erfüllt die Beschreibung auch dieses Kriterium. Zusätzlich wird durch das negierte Service-spezifische Attribut `busy` gefordert, dass der Drucker derzeit ungenutzt ist. Da dieses Attribut in der Beispiel-Description nicht exisitiert wird angenommen, dass dieser Punkt ebenfalls erfüllt ist. Die Service-Beschreibung passt somit in allen Punkten zur Abfrage und kann als Ergebnis an die fragende Instanz zurückgeliefert werden. Dies geschieht über einen Response.

```
<?xml version="1.0" encoding="UTF-8"?>

<stil version="0.2"
      xmlns="http://www.musama.de/STiL"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.musama.de/STiL STiL.xsd ">

 <response requestID="123456789">
  <service provider="bochum.informatik.uni-rostock.de">
   <stilType>stil.pureOut.printer</stilType>
   <friendlyName>Druckdienst</friendlyName>
   <technology>WebServices</technology>
   <validity>10M</validity>
   <comProtocol>UDP</comProtocol>
   <comProtocol>TCP</comProtocol>
   <accessLocation>bochum.informatik.uni-rostock.de:1234/printService
    </accessLocation>
   <attribute name="location">Raum 126</attribute>
   <attribute name="color">true</attribute>
   <attribute name="dpi">4.800x1.200</attribute>
  </service>
 </response>

</stil>
```

Listing 3.3: Beispiel: STiL-Response mit einem Service

STiL-Response

Listing 3.3 zeigt ein Beispiel für die Rückgabe einer gefundenen Service-Beschreibung. Jeder **STiL-Response** wird mit einem `response`-Element eingeleitet. Dieses enhält die `requestID` des Request, auf den sie sich bezieht.

Das Element `service` gibt an, dass dieser Response keine Fehlermeldung, sondern eine oder mehrere Service-Beschreibungen zurückliefert. Dabei wird für jede gefundene Beschreibung ein eigenes `service`-Element verwendet. Dieses enthält den Inhalt der bereits erläuterten Service-Description. Sie wird daher hier nicht weiter behandelt.

Während des Discovery-Vorganges können diverse Fehler auftreten. Auch diese werden der anfragenden Instanz durch einen Response mitgeteilt. Ein Beispiel dafür ist in Listing 3.4 dargestellt. Bei Fehlermeldungen besitzt das `response`-Element nur das Kindelement `error` mit den Attributen `code` und `reason`. Durch `code` wird ein Fehlercode angegeben, der an die HTTP-Statuscodes angelehnt ist. Das optionale `reason`-Attribut kann genutzt werden, um den Fehler näher zu beschreiben.

```
<?xml version="1.0" encoding="UTF-8"?>

<stil version="0.2"
      xmlns="http://www.musama.de/STiL"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.musama.de/STiL STiL.xsd">

 <response requestID="123456789">
  <error code="400" reason="Die Anfrage war syntaktisch falsch"/>
 </response>

</stil>
```

Listing 3.4: Beispiel: STiL-Response mit einer Fehlermeldung

Die in diesem Abschnitt detailliert beschriebene Meta-Sprache STiL ist die Grundlage der zentralen Vereinheitlichung durch GPAPs. Im folgenden Abschnitt wird eine Übersicht über die GPAP-Plugins gegeben, die STiL-basierte Abstraktionen und Konkretisierungen durchführen.

3.2.3 GPAP-Plugins - Strategien zur Einbindung von SOA-Klassen

Die Übersetzung heterogener Service-Beschreibungen und der Suchanfragen auf diesen Beschreibungen wird auf dem GPAP über implementierungsspezifische Plugins und eine übergeordnetes Plugin-Konzept erreicht. Für jede SOA-Implementierung, die vom GPAP unterstützt wird, sind Plugins der folgenden Klassen zu erstellen:

- **Discovery-Plugins** sind Service-Consumer jeweils einer SOA-Implementierung (z. B. *Jini* Discovery-Plugin). Diese Plugin-Klasse ist dafür zuständig Service-Beschreibungen der jeweiligen Implementierung im Ensemble zu finden und

äquivalente abstrakte STiL-Descriptions zu erstellen. Diese werden dann in den abstrakten STiL Service-Bus eingespeist.

- **Provision-Plugins** sind Service-Provider jeweils einer SOA-Implementierung. Sie entnehmen dem STiL Service-Bus abstrakte Service-Descriptions, erstellen äquivalente Service-Beschreibungen ihrer jeweiligen Implementierung und bieten diese stellvertretend im lokalen Ensemble an.
- **Broker-Plugins** sind Service-Broker jeweils einer SOA-Implementierung bzw. Erweiterungen vorhandener Broker dieser Implementierung. Sie sind Kombinationen aus Discovery- und Provision-Plugins der dieser Implementierung. Einerseits registrieren Provider ihre Services beim Broker-Plugin. Dieses erstellt äquivalente STiL-Service-Beschreibungen und speist sie in den STiL Service-Bus ein. Andererseits bieten Broker-Plugins Service-Beschreibungen der unterstützten SOA-Implementierung an, die aus abstrakten STiL-Descriptions abgeleitet wurden.

Diese grundlegenden Plugin-Klassen können je nach SOA-Implementierung unterschiedliche Strategien verfolgen, um ihre Aufgabe zu erfüllen. Daher wird im Folgenden eine weitere Untergliederung in Plugin-Unterklassen vorgenommen.

Analog zur in Abschnitt 2.3.2 vorgenommenen Klassifizierung nach Service-Bus-Aktivität, gibt es zur Realisierung von **Discovery-Plugins** theoretisch zwei Strategien: **Request-basiert** für Request-basierte Service-Busse (Consumer sucht aktiv nach Beschreibungen) und **Advertisement-basiert** für Service-Busse mit Advertisement-Unterstützung (Consumer lauscht passiv auf von Providern angebotene Beschreibungen). Die Interaktion beider Plugin-Varianten mit dem jeweiligen konkreten und dem abstrakten Service-Bus ist in Abbildung 3.8 in Form eines Sequenzdiagrammes dargestellt.

Während Request-basierte Discovery-Plugins durchaus in Reinform sinnvoll sind, würde ein rein Advertisement-basiertes Plugin im praktischen Einsatz nicht genügen, da es die Veröffentlichung aller Service-Beschreibungen in Form von Advertisements voraussetzt, diese aber selbst bei Service-Bussen mit Advertisement-Unterstützung für jeden Provider lediglich optional sind. Der Advertisement-basierte Ansatz ermöglicht jedoch eine schnellere Reaktion des GPAPs auf einen neuen Service, da nicht erst auf einen erneuten (z. B. periodisch angestoßenen) Suchvorgang des Plugins gewartet werden müsste, sondern direkt auf das Advertisement reagiert werden könnte. Daher ist neben einer Request-basierten Strategie ein Lauschen auf Advertisements sinnvoll, sofern der konkrete Service-Bus diese unterstützt.

Ebenso kann bei **Provision-Plugins** zwischen **Request-basierten** und **Advertisement-basierten** Lösungen unterschieden werden, wobei Advertisements auch Registrierungen von Service-Beschreibungen bei einem Broker einschließen, da beide Vorgänge auf der strategischen Ebene identisch sind. Das Vorgehen beider Varianten ist in Abbildung 3.9 illustriert.

Vor allem bei Broker-basierten SOA-Implementierungen und Implementierungen mit Broker-Modus ist die Verwendung von **Broker-Plugins** wünschenswert, da diese die

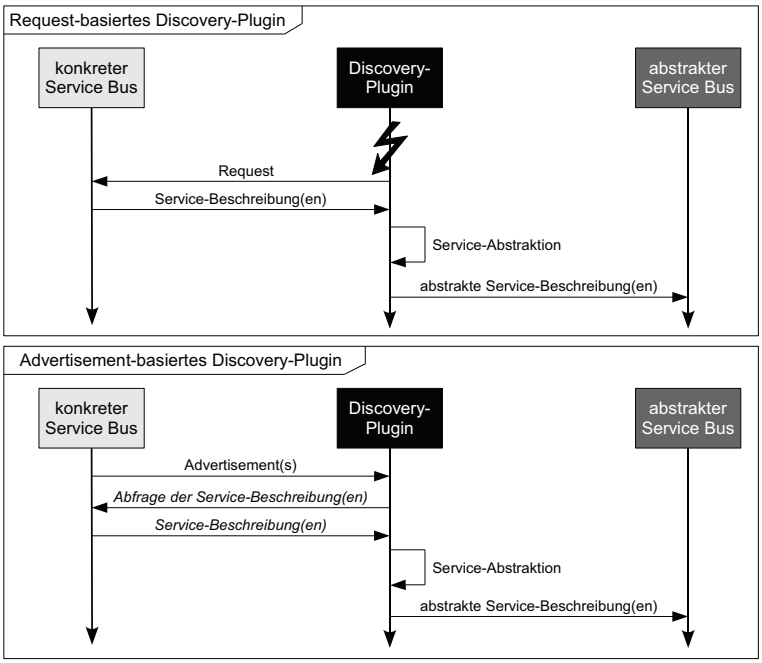


Abbildung 3.8: Varianten des Discovery-Plugins

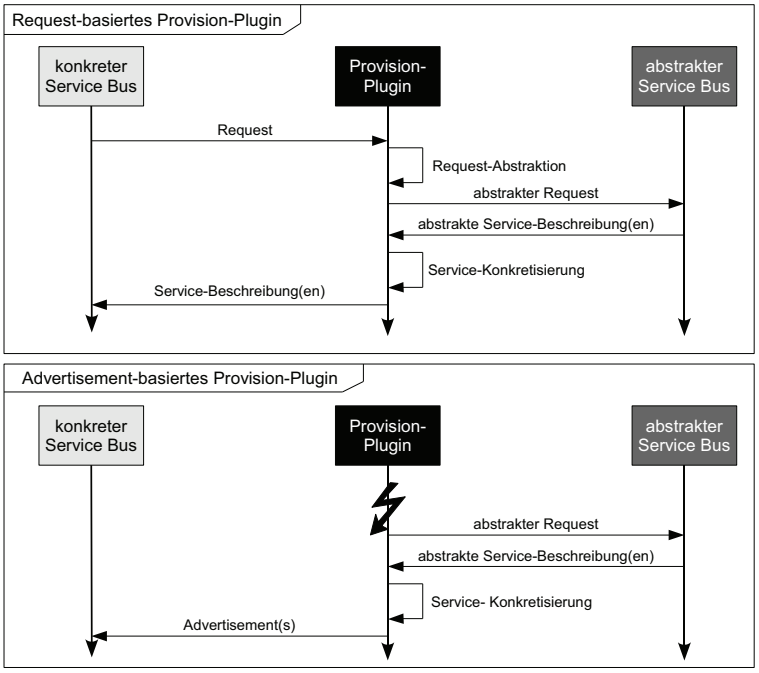


Abbildung 3.9: Varianten des Provision-Plugins

Funktionalität von Discovery- und Provision-Plugins in einer einzelnen Komponente kombinieren. Broker-Plugins sind jedoch nur dann einsetzbar, wenn Provider ihre Services bei mehreren Brokern registrieren oder vorhandene konkrete Broker des Ensembles erweitert werden können. Im ersten Fall kann das Broker-Plugin als **eigenständiger Broker** im Ensemble arbeiten. Im zweiten Fall wird der konkrete **Broker erweitert**, um mit dem Broker-Plugin zu kommunizieren. Die Interaktion beider Broker-Varianten mit Providern und Consumern sowie dem abstrakten Service-Bus ist in Abbildung 3.10 dargestellt.

Es wurde gezeigt, dass die Art der Plugins sowie dessen konkrete Umsetzung stark von der Natur des Service-Busses der jeweiligen SOA-Implementierung abhängt. Abbildung 3.11 fasst die erforderliche Entscheidungsfindung für die Auswahl der benötigten Plugin-Arten in Abhängigkeit von der Art des jeweiligen Service-Busses zusammen.

Neben den in diesem Abschnitt behandelten Plugin-Unterschieden gibt es je nach Klasse der SOA-Implementierung zusätzliche Anforderungen und Aufgaben für die implementierungsspezifischen Discovery-, Provision- und Broker-Plugins. Beispielsweise kann die Intensität der Technologiebindung Anforderungen an die Plugin-Hard- und -Software stellen (z. B. erfordern Plugins für Bluetooth SDP einen Bluetooth-Stack während Jini-Plugins eine Möglichkeit zur Einbindung von Java-Code bieten müssen). Weiterhin erfordern Plugins für SOAs, die die Service-Nutzung spezifizieren, Mechanismen zur Übersetzung zwischen den verschiedenen Nutzungstechniken.

Neben den unterschiedlichen Strategien für die GPAP-Plugins im lokalen Ensemble, gibt es auch bei der Realisierung der Ensemble-übergreifenden Community verschiedene Ansätze, die im folgenden Abschnitt näher betrachtet werden.

3.2.4 Topologien einer SOA-Community

Die Community ist eine Vernetzung von Ensembles in Form ihrer jeweiligen GPAPs. Ihre konkrete Realisierung ist durch die in dieser Arbeit entwickelte GPAP-basierte Infrastruktur nicht vorgeschrieben. Schon mit den klassischen Netzwerk-Topologien existieren mehrere Topologie-Kandidaten für die GPAP-Community. In diesem Abschnitt werden die wichtigsten Topologien kurz betrachtet und bewertet. Dabei wird aus praktischen Gründen davon ausgegangen, dass die GPAPs physikalisch über ein IP-basiertes Netzwerk wie das Internet miteinander kommunizieren. Die folgenden und in Abbildung 3.12 illustrierten Modelle sind daher nicht als physikalische Netzwerktopologien zu sehen, sondern als logische Overlays, über die GPAPs abstrakte Service-Beschreibungen anfragen und austauschen.

Bei der *Ring*-Topologie ist jeder GPAP mit genau zwei anderen GPAPs verbunden. Die Vorteile dieser Topologie sind ihre einfache Implementierbarkeit, die Vermeidung von bandbreitenunfreundlichen Kollisionen mehrerer Suchanfragen sowie die gute Skalierung der Rechenaufwandes einzelner GPAPs. Allerdings kann bei dieser Topologie bereits der Ausfall eines einzelnen GPAPs Community-Kommunikation erschweren und ab zwei Ausfällen sogar Community-Segmente völlig voneinander trennen. Zudem sind

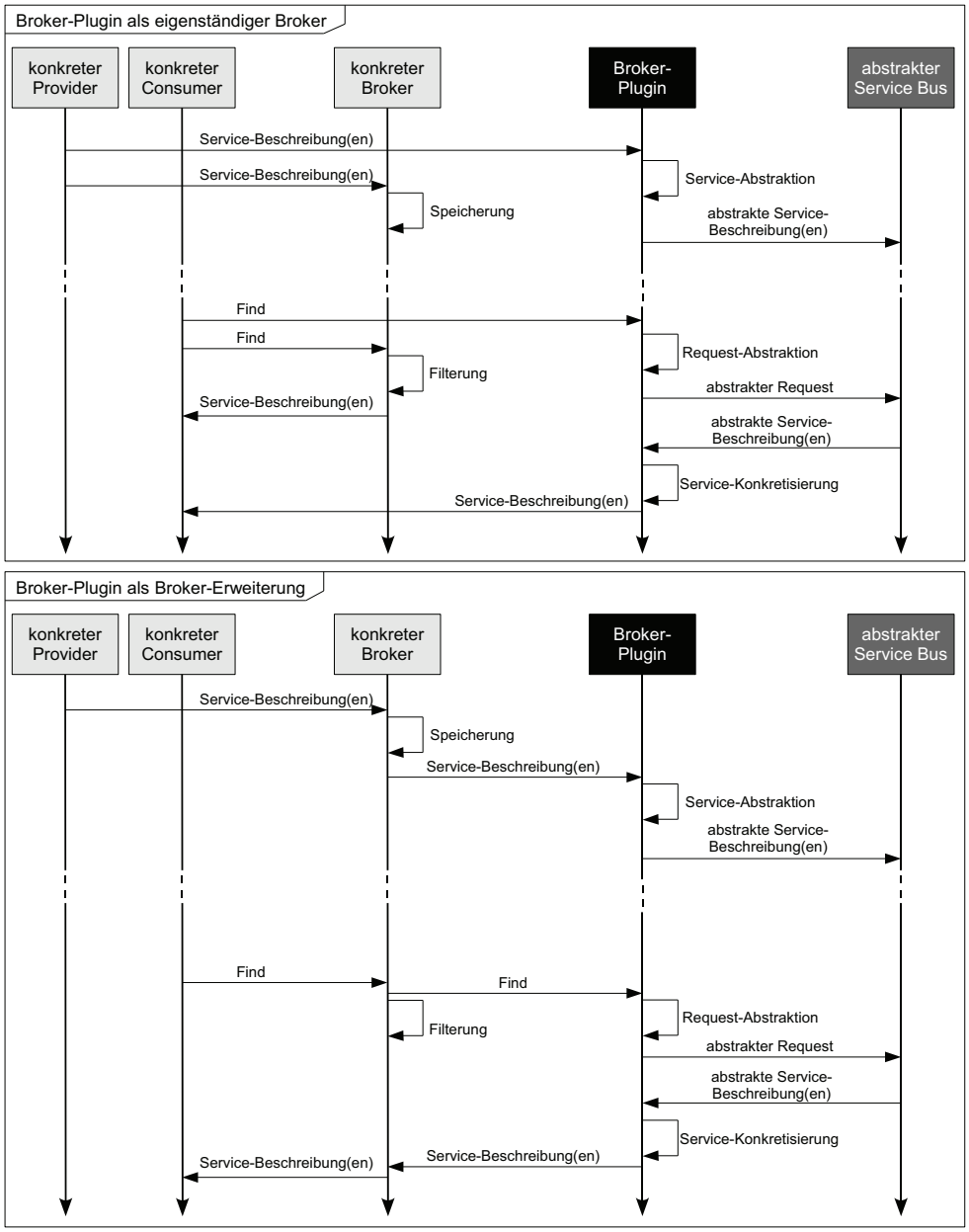


Abbildung 3.10: Varianten des Broker-Plugins

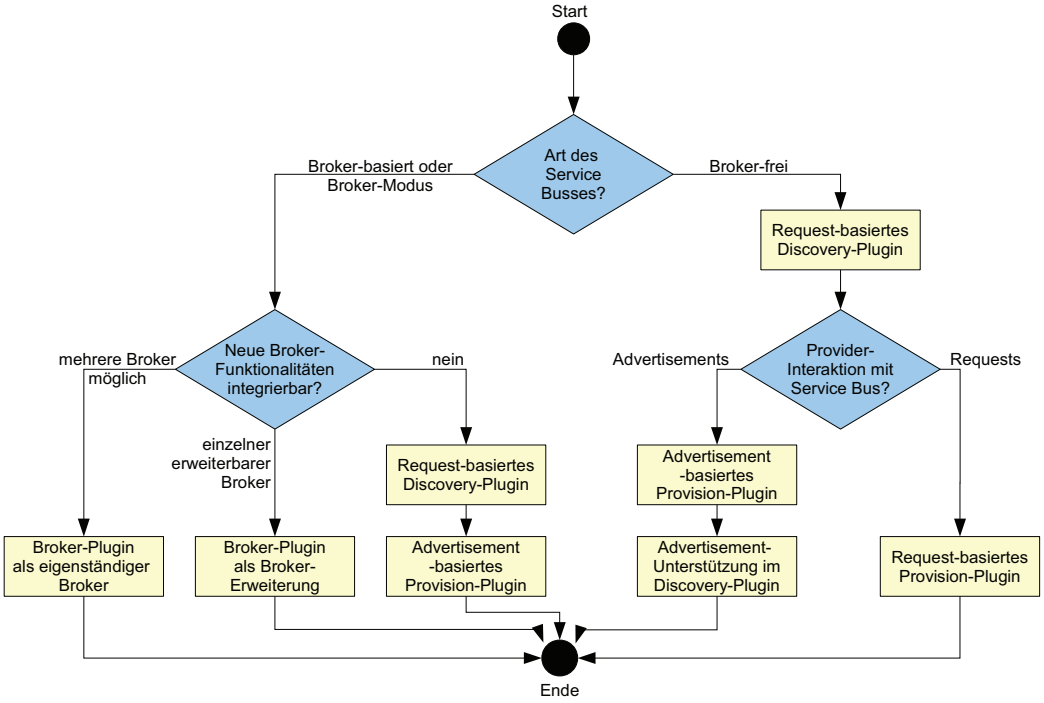


Abbildung 3.11: Entscheidungsbaum: Welche Plugin-Art und -Ausprägung ist für welchen Service-Bus sinnvoll?

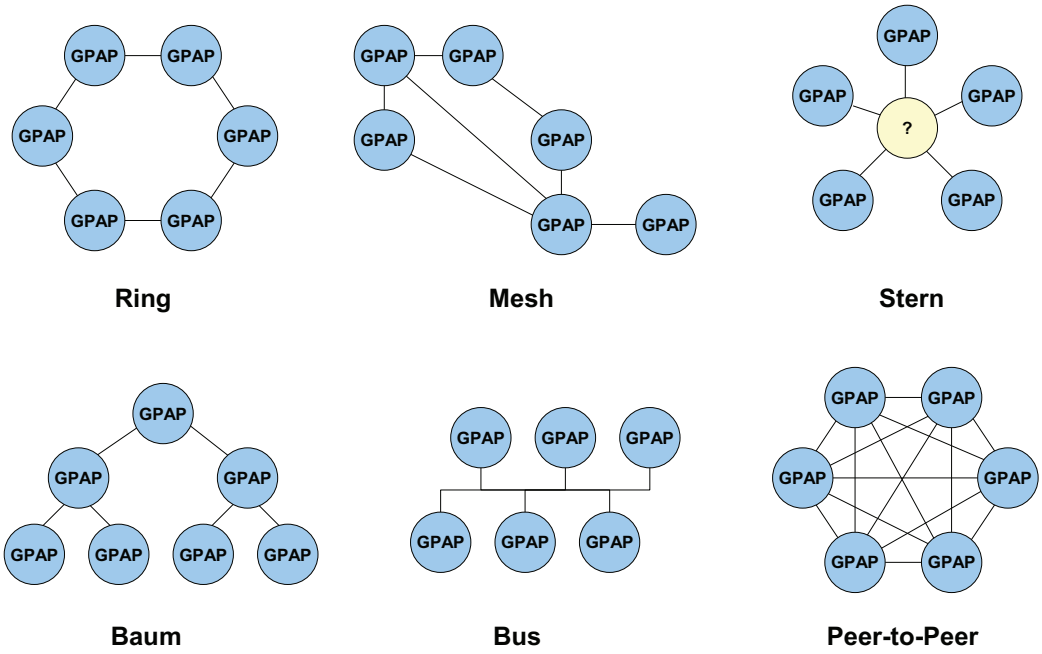


Abbildung 3.12: Die wichtigsten logischen Topologie zur Vernetzung von GPAPs in einer Community.

hohe Latenzen zwischen weit voneinander entfernten GPAPs zu erwarten. Daher eignet sich diese Topologie nur für einfache Implementierungen kleiner Communitys, die zudem aus sehr zuverlässigen GPAPs bestehen. Analoge Vor- und Nachteile gibt es bei einem offenen Ring, der als *Linie* bezeichnet wird.

Im Vergleich zum Ring ist die *Mesh*-Topologie nicht deutlich strukturiert, da jeder GPAP mit einem oder beliebig vielen anderen GPAPs verbunden sein kann. Da Mesh-Topologien sehr unterschiedlich strukturiert sein können, ist es schwer generelle Aussagen über ihre Community-Eignung zu treffen. Sie haben jedoch den Vorteil, dass ab einem bestimmten Vernetzungsgrad der Ausfall einer bestimmten Anzahl von Knoten das Netz nicht mehr zu stark behindert. Weiterhin ist es möglich, besonders aktiv kommunizierende GPAPs stärker zu vernetzen als eher passive Komponenten, wodurch der entstehende Netzwerkverkehr besser verteilt wird. Besonders dynamisch veränderbare Mesh-Strukturen haben daher ein hohes Optimierungspotential für die GPAP-Community. Leider ist der Routing-Aufwand innerhalb dieser Netze oft vergleichsweise komplex.

Die *Stern*-Topologie ist eine der einfachsten Strukturen für GPAP-Communitys. Hier ist jeder GPAP mit einem zentralen Knoten verbunden, der entweder selbst ein GPAP oder eine andere dedizierte Komponente sein kann. Diese Topologie ist von einer zentralen Komponente abhängig. Communitys mit Stern-Topologie funktionieren unabhängig vom Ausfall einzelner GPAPs (außer dem zentralen) und sind leicht implementierbar, wartbar sowie erweiterbar. Zudem ist diese Struktur prädestiniert für Multicasts und Broadcasts, wodurch die Verteilung von Service-Beschreibungen sowie die Service-Suche einfach umsetzbar ist. Latenz, Bandbreite und Ausfallsicherheit bestimmt jedoch der zentrale Knoten, ohne den die Stern-Community nicht funktionsfähig ist. Somit ist die Stern-Topologie nur dann einsetzbar wenn es eine entsprechend leistungsfähige und zuverlässige zentrale Komponente gibt.

Bei der *Baum*-Topologie sind die einzelnen GPAPs hierarchisch miteinander vernetzt. Je nachdem, wo sich ein ausfallender GPAP im Baum befindet, kann dies keine Auswirkungen auf die Commmunity haben (Blatt-GPAP) oder diese segmentieren (GPAP als Wurzel oder Gabelung). Somit bleibt die Community beschränkt einsatzfähig. Weiterhin gibt es in der Literatur eine Reihe effizienter Such- und Verteilungsalgorithmen für Baumstrukturen, die auch für die Service-Suche bzw. -Verteilung innerhalb der Community anwendbar sind. Vor allem hierarchische Organisationen profitieren von dieser Topologie. So könnten in einer Universität die Blatt-GPAPs für Ensembles eines Institutes verantwortlich sein, während die nächsthöheren GPAPs Fakultäts-Ensembles verwalten, usw. Somit können Services einer Organisationseinheit beispielsweise bequem innerhalb der Einheit verfügbar gemacht, jedoch von anderen Einheiten ausgeschlossen werden. Ein Nachteil dieser Topologie ist die oftmals komplexe Implementierbarkeit. Weiterhin kann es besonders bei sehr tiefen Bäumen (Anzahl der Gabelungen zwischen Wurzel und Blatt) zu hohen Latenzen zwischen einzelnen GPAPs kommen. Gerade diese Latenzen sind jedoch für hierarchische Organisationen vertretbar, da vor allem mit nahestehenden organisatorischen Einheiten angemessen kommuniziert werden kann.

Im Hinblick auf den abstrakten Service-Bus scheint gerade die *Bus*-Topologie prädestiniert für Communitys. Allerdings hat diese als eigenständige logische Topologie keine Relevanz, da sie nur nur mithilfe anderer Topologie-Ansätze realisierbar ist (z. B. Stern-Topologie mit der zentralen Komponente als Bus-Controller). Gerade in pervasiven Umgebungen werden zudem logische Busse in Form von verteilten Assoziativspeichern eingesetzt. Diese *Object Spaces* speichern Objekte (im vorliegenden Fall Service-Beschreibungen) in einem auf mehrere Komponenten verteilten Datenraum. Dieser ist wiederum oftmals als Peer-to-Peer-Netzwerk implementiert.

Peer-to-Peer-Topologien sind durch gleichberechtigte Knoten (Peers) charakterisiert, die mit allen anderen Peers kommunizieren können. Wie Bus-Systeme nutzen auch P2P-Netze andere Topologien (z. B. Ring oder Mesh) für eine Basis-Infrastruktur, die um ein Overlay mit Indizierungs- und Suchfunktionen sowie Mechanismen der Selbstorganisation erweitert wird. Eine P2P-Community mit GPAPs als Peers würde je nach zugrunde liegender P2P-Technologie (z. B. Chord, Kad, JXTA) spezielle Mechanismen zur Indizierung neuer und Suche existierender Service-Beschreibungen sowie zur Einbindung neuer GPAPs bereitstellen. Durch den dafür notwendigen Kommunikationsaufwand rentiert sich die P2P-Community jedoch erst bei einer hohen Anzahl beteiligter GPAPs.

Die in diesem Abschnitt behandelten logischen Topologien sind je nach Netzwerkinfrastruktur und Anwendungsszenario mehr oder weniger zur GPAP-Vernetzung geeignet. Sie können jedoch alle für einen Austausch von Service-Beschreibungen genutzt werden. In der resultierenden GPAP-Community werden sehr viele unterschiedliche Services angeboten, was die Consumer vor eine komplexere Auswahl eines konkreten Services stellt, als es bei einem einzelnen Ensemble der Fall wäre. Die Nutzung von Kontextinformationen, die im folgenden Abschnitt behandelt wird, hat das Potential diese Auswahl zu erleichtern und zusätzliche Service-Funktionalitäten zu realisieren.

3.3 Nutzung von Kontextinformationen

Wenn wir Menschen miteinander kommunizieren und interagieren, nutzen wir implizit und meist unbewusst Informationen, die unsere aktuelle Situation beschreiben. So können wir beispielsweise mit dem Ausdruck “Ich fühle mich hier wohl”während einer Bahnfahrt eine allgemeine Zuneigung zum Verkehrsmittel Bahn ausdrücken. Äußern wir diesen Satz auf einer Feier oder einer Konferenz, steht vermutlich eher die Sympathie für anwesende Personen im Vordergrund. Während wir so einerseits die Bandbreite unseres Informationsaustausches erhöhen, sind wir andererseits in der Lage unsere offensichtlichen Informationen semantisch zu konkretisieren oder klassifizieren. Sobald wir jedoch mit Computern kommunizieren, müssen wir uns auf die limitierte Aussagekraft von einfachen Ein- und Ausgabeformaten beschränken. Dadurch wird ein wirklich (oder zumindest scheinbar) intelligentes Verhalten der uns umgebenden IT stark behindert. Aus diesem Grund ist die Erfassung und Auswertung von situationsbeschrei-

benden Informationen, dem so genannten *Kontext*, ein wichtiges Gebiet der Pervasive Computing-Forschung.

Aus den vielen in der Literatur verwendeten Kontextdefinitionen ist vor allem die von G. D. Abowd und A. K. Dey aufgrund ihrer Allgemeingültigkeit sehr verbreitet (frei übersetzt aus [Abowd 99]):

Definition 8 ***Kontext*** *ist jede Information, die zur Beschreibung der Situation einer Entität genutzt werden kann. Entitäten sind dabei Personen, Orte oder Objekte, die für die Interaktion zwischen Nutzern und Anwendungen relevant sein könnten. Dies beinhaltet auch die Nutzer und Anwendungen selbst.*

Da diese Definition sehr umfassend ist, erfordert eine nähere Kontextbetrachtung ein konkretes Anwendungsfeld, welches im folgenden die Realisierung pervasiver Umgebungen darstellt. Insbesondere diese haben den Anspruch, Nutzer möglichst automatisch aus dem Hintergrund heraus proaktiv zu unterstützen und erfordern daher ein Kontextbewusstes System [Satyanarayanan 01].

3.3.1 Kontextarten und -Modelle

Analog zu den unterschiedlichen Kontext-Definitionen existieren auch eine Reihe von Kontextklassifizierungen, beispielsweise anhand der Kontext-Semantik, -Lieferanten oder der konkreten Anwendungsdomäne. Da pervasive Umgebungen einerseits keiner dieser Klassen zugeordnet werden können und andererseits keine dieser Klassifizierungen allgemein anerkannt ist, wird innerhalb dieser Arbeit eine weit verbreitete Klassifizierung anhand der Kontext-Komplexität bevorzugt (angelehnt an [Park 06]). Diese unterteilt Kontext in **Rohen Kontext** (*Raw Context*), **normalisierten Kontext** (*Low-level Context*) und **Komplexen Kontext** (*High-level Context*).

Roher Kontext beschreibt informative Rohdaten, die direkt von Sensoren oder anderen Messgeräten geliefert werden (z. B. Signallaufzeit von GPS-Satelliten). Dieser wird anhand von Algorithmen, Tabellen, Formeln oder anderen Transformationsgrundlagen in normalisierten Kontext umgewandelt (z. B. GPS-Koordinaten). Normalisierter Kontext kann dank seiner einheitlichen Datenstruktur und -Semantik mit anderen (Kontext)Daten verglichen und ausgewertet werden. Das Ergebnis dieses Prozesses wird als Komplexer Kontext bezeichnet (z. B. Entfernung zu einem Point-of-Interest) und kann selbst wieder zur Erzeugung von komplexerem Kontext genutzt werden.

Vor allem normalisierter und komplexer Kontext benötigen ein einheitliches Datenformat. Leider existieren derzeit keine Standards zur Kontextbeschreibung, so dass sich in der Fachliteratur eine Reihe konkurrierender Modelle für diesen Zweck etabliert haben. Derzeit sind sechs Ansätze identifizierbar [Strang 04]:

- *Schlüssel/Wert-Modelle* drücken Kontext textuell in Form von Schlüssel/Wert-Paaren aus.

- *Markup-Schema-Modelle* beschreiben Kontext durch hierarchische Datenstrukturen (z. B. XML-Dokumente).
- *Grafische Modelle* verwenden Diagramme zur Kontext-Visualisierung.
- *Objekt-orientierte Modelle* nutzen Kapselung und Wiederverwendbarkeit, um Kontext in Form von Objekten auszudrücken.
- *Logik-basierte Modelle* beschreiben Kontext formell durch Fakten, Ausdrücke und Regeln.
- *Ontologie-basierte Modelle* verwenden Ontologien (z. B. Web Ontology Language-Dokumente) zur Kontextbeschreibung.

Dieselbe Kontextinformation kann in einem oder mehreren dieser Modelle ausgedrückt werden, wobei sich insbesondere die nicht-grafischen Modelle auch als Austauschformat innerhalb einer SOA eignen. Der folgende Abschnitt beschreibt die Ansätze zur Integration von Kontext in Service-orientierte Architekturen und dessen Potential für GPAP-Ensembles und -Communitys.

3.3.2 Kontextinformationen in service-orientierten Architekturen

Im Rahmen dieser Arbeit wurden drei Ansätze zur Integration von Kontext in SOA-basierte Infrastrukturen identifiziert. Der erste Ansatz sind **kontextbasierte Services** von denen drei elementare Ausprägungen existieren:

- *Kontext-Anbieter* sind Services, die für den Zugriff auf normalisierte oder komplexe Kontextdaten genutzt werden können (z. B. Atomuhr- und Wettervorhersage-Services).
- *Kontext-Transformatoren* sind Services, die zur Übersetzung zwischen unterschiedlichen Kontext-Formaten oder -Modellen dienen (z. B. Celsius/Fahrenheit-Umrechnungsservice).
- *Kontext-Ableiter* sind Services, die Komplexen Kontext aus anderen Kontextdaten und ggf. zusätzlichen Informationen ableiten (Services zur Intentionserkennung oder ISBN-basierten Büchersuche).

Kontextbasierte Services sind die Grundlage vieler pervasiver Anwendungen [Venezia 09] [Panganelli 09]. Wie alle anderen Services müssen sie innerhalb von Ensembles und auch der Community ausgetauscht werden. Die Wenigsten dieser Services dienen direkt der Unterstützung der GPAP-Infrastruktur.

Im Gegensatz dazu kann die GPAP-Infrastruktur durch eine **Kontextnutzung für Service-Angebot und -Suche** direkt profitieren. Bei diesem Ansatz werden mit der Service-Schnittstelle auch normalisierte oder komplexe Kontextinformationen über den Service veröffentlicht, die während des Discovery-Prozesses wiederum von Consumern bei der Service-Auswahl helfen. Dazu gehören beispielsweise:

- Quality-of-Service (QoS)-Parameter (z. B. Netzwerkanbindung des Providers, Service-Verfügbarkeit, Service-Auslastung)
- Semantische Informationen (z. B. Service-Typ, ontologische Klassifikation, Vor- und Nachbedingungen)
- Beschreibung des Service-Providers (z. B. Physischer Standort, Firmen- oder Abteilungszugehörigkeit)
- Nutzungsbedingungen und -regeln (z. B. Zugriffsbeschränkungen, Preise für die Service-Nutzung)

Während einige dieser Kontextinformationen vom Provider bei der Service-Veröffentlichung angegeben werden müssen (z. B. ontologische Klassifikation, Zugriffsbeschränkungen, Preise) [Lee 03][Mostéfaoui 04], können andere nach der Veröffentlichung automatisch von der Infrastruktur hinzugefügt werden (z. B. Netzwerkanbindung, Verfügbarkeit) [Al-Masri 07], wobei mit dem GPAP eine zentrale Komponente für diese Aufgabe zur Verfügung steht. Von den in Abschnitt aufgeführten Modellen kommen im Allgemeinen die textuellen Formate zur Kontextbeschreibung in Frage, während in speziellen Fällen auch exotische Formate eingesetzt werden können (z. B. das Objekt-orientierte Modell beim Java-basierten Jini).

Weiterhin können derartige Kontextinformationen vom GPAP genutzt werden, um aus der Vielzahl der in der Community verfügbaren und teils funktional identischen Services diejenigen herauszufiltern die im eigenen Ensemble am geeignetsten erscheinen (z. B. aufgrund der QoS-Eigenschaften oder Kosten dieser Services). Somit würde der GPAP eine Vorauswahl treffen, die durch die Consumer-spezifischen Discovery-Prozesse zusätzlich gefiltert werden kann.

Kontextbewusste Service-Anpassungen bilden den dritten Ansatz zur Kontext-Anreicherung in SOA-basierten Infrastrukturen. Hier veröffentlichen Provider Beschreibungen adaptiver Services und Consumer senden während des Bind-Prozesses Kontextinformationen, um diese für sich zu individualisieren. So kann ein Service beispielsweise für ein konkretes Gerät angepasst werden (z. B. minimalistische Nutzeroberfläche für ein Mobiltelefon und komplexere Schnittstelle für einen herkömmlichen PC). Derartige SOAs können die SOA-eigene Flexibilität durch flexible Services zusätzlich steigern. Da dieser Ansatz sehr Service-spezifisch ist und Kontextinformationen für gewöhnlich direkt zwischen Providern und Consumern ausgetauscht werden, kann die GPAP-Infrastruktur hier nur durch die Verteilung derartiger Service-Beschreibungen unterstützen. Aufgrund des hohen Aufwands bei der Erstellung adaptiver Services, neigen Provider jedoch eher dazu mehrere statische Service-Versionen (z. B. ein Service pro Geräteklasse) als einen adaptiven Service zu veröffentlichen [Zuidweg 03][Corradi 04].

Von den vorgestellen Ansätzen zur Nutzung von Kontext in Service-orientierten Architekturen, ist vor allem die Kontextnutzung für Service-Angebot und -Suche für die GPAP-Ensembles und Community relevant. Durch sie ist eine Vorauswahl von Service-Beschreibungen durch den GPAP möglich und einzelne Consumer werden nicht mehr mit der Gesamtheit der verfügbaren Service-Beschreibungen konfrontiert. Da einige der

für einen Kontext-bewussten Discovery-Prozess benötigten Kontextinformationen direkt von GPAPs generiert, veröffentlicht und ausgewertet werden können (z. B. Service-QoS), ist es möglich diese Optimierungsprozesse im Hintergund und ohne Modifikationen der einzelnen Provider und Consumer durchzuführen.

Die in diesem Abschnitt vogestellten Konzepte und Lösungen zur SOA-Interoperabilität und zusätzlichen Optimierung durch Kontextinformationen wurden in großen Teilen im Laufe dieser Arbeit implementiert und darauf aufbauend verifiziert. Das folgende Kapitel erläutert detailliert die daraus resultierenden Ergebnisse.

Kapitel 4

Implementierung des GPAP Service Layers

Der im vergangenen Kapitel vorgestellte Ansatz der SOA-übergreifenden Interoperabilität durch zentrale Vereinheitlichung wurde im Rahmen der vorliegenden Arbeit in verschiedenen, größtenteils E-Learning-bezogenen, Anwendungen an der Universität Rostock implementiert und evaluiert. Die Grundlage dieser in Kapitel 5 näher beschriebenen Arbeiten ist die Implementierung der zentralen Vereinheitlichung auf dem Service-Layer des General Purpose Access Points (GPAP), die im folgenden Kapitel ausführlich behandelt wird.

4.1 GPAP Service-Layer

Abbildung 4.1 zeigt ein vereinfachtes Klassendiagramm des GPAP Service-Layers, der in der Programmierprache Java implementiert wurde. Begonnen mit der zentralen `ServiceLayer`-Klasse werden im Folgenden die wichtigsten Klassen und Interfaces dieser Übersicht näher erläutert, da diese den Kern des Frameworks zur SOA-übergreifenden Interoperabilität des GPAPs darstellen.

4.1.1 `ServiceLayer`-Klasse

Die zentrale Klasse `ServiceLayer` ist für die Initialisierung des gesamten Frameworks zuständig. Sie verwaltet die Manager-Klassen für die einzelnen Discovery-, Provision- und Broker-Plugins und initialisiert die Anbindung an die jeweiligen Implementierungen des abstrakten Service Busses. `ServiceLayer` hat somit vor allem folgende Aufgaben:

- Konfiguration des Frameworks (über Konfigurationsdateien)
- Installation vorhandener Discovery-, Provision- und Broker-Plugins

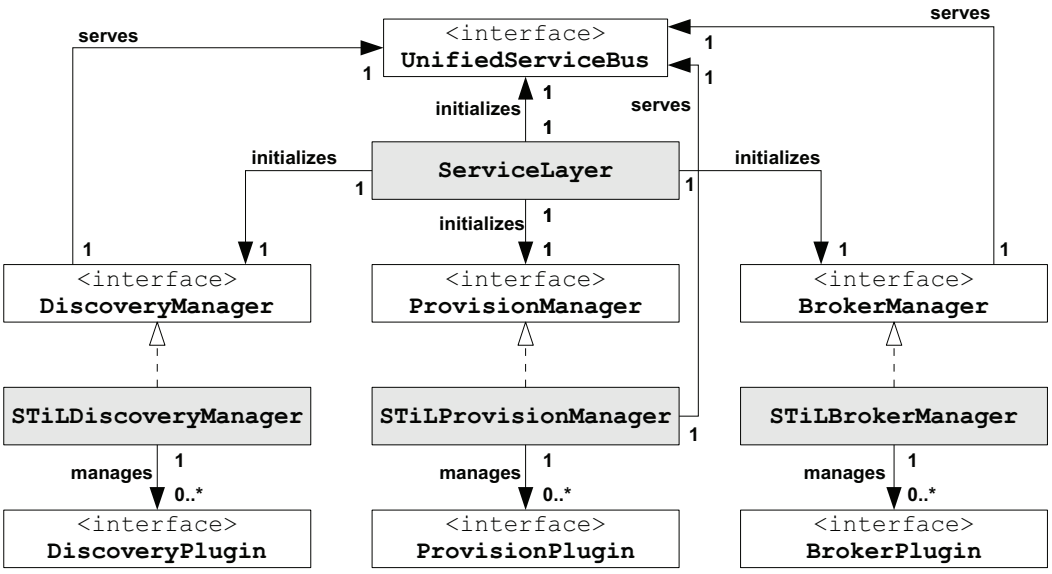


Abbildung 4.1: Vereinfachtes Klassendiagramm des GPAP Service Layers

ServiceLayer

```
void main()
void addDiscoveryPlugin(String name, DiscoveryPlugin plugin)
void addProvisionPlugin(String name, ProvisionPlugin plugin)
void addBrokerPlugin(String name, BrokerPlugin plugin)
void startServiceLayer()
void stopServiceLayer()
```

Abbildung 4.2: Die zentrale `ServiceLayer`-Klasse

- Initialisierung des abstrakten Service Busses
- Initialisierung und Terminierung der Plugin-Manager (und somit auch der Plugins)

Abbildung 4.2 zeigt die Klasse `ServiceLayer` in Form einer UML-Klasse. Während die `main()`-Methode die Startmethode für das gesamte Framework darstellt und sowohl den abstrakten Service Bus als auch die Plugin-Manager initialisiert, werden mit den Methoden `addDiscoveryPlugin()`, `addProvisionPlugin()` und `addBrokerPlugin()` bei der Inititalisierung Plugins installiert. Nach der Initialisierung des Frameworks kann dieses über die Methoden `startServiceLayer()` und `stopServiceLayer()` gestartet bzw. beendet werden.

Während der Laufzeit hat die Klasse nur die Aufgabe auf externe Signale zur Terminierung (z. B. bei Fernwartung) zu reagieren. Diese können das Framework pausieren, auf den Startzustand zurücksetzen oder terminieren. Beim Zurücksetzen und Terminieren werden alle Service-Beschreibungen, die diese GPAP-Instanz dem abstrakten Service Bus hinzugefügt hat, wieder aus diesem entfernt. Zur Laufzeit hingegen wird

ihre Einspeisung in den und Entfernung aus dem Bus vom `DiscoveryManager` und seinen `DiscoveryPlugin`s geleistet.

4.1.2 Discovery-Manager und -Plugins

Discovery-Plugins finden Service-Beschreibungen einer konkreten Service-Implementierung im lokalen Ensemble. Da in diesem mehrere SOA-Implementierungen aktiv sein können, ist auch die Verwendung mehrerer Discovery-Plugins in dem für dieses Ensemble zuständigen GPAP sinnvoll. Die Koordination der verschiedenen Plugins übernimmt der `DiscoveryManager`. Dieser hat folgende konkrete Aufgaben:

- **Installation und Deinstallation der vom ServiceLayer übergebenen Discovery-Plugins**: Während der Service Layer die Plugins bei der Initialisierung aber auch zur Laufzeit in das Framework einbindet bzw. aus diesem entfernt, muss der Manager sie nur noch in seiner internen Liste registrieren bzw. aus dieser entfernen.
- **Initialisierung und Terminierung der installierten Discovery-Plugins**: Mit der Initialiserung beginnen die individuellen Discovery-Prozesse in den Plugins. Die durch ein Plugin gefundenen Services werden beim Terminieren des Plugins wieder aus dem abstrakten Service Bus entfernt. Im Fall einer unkontrollierten Terminierung, wie etwa einem Systemabsturz, ist ein Entfernen der Services durch den abstrakten Service Bus erforderlich, da jeder abstrakte Service eine Referenz auf seinen GPAP enthält und dessen Ausfallen leicht erkennbar ist, beispielsweise mittels *Ping*.
- **Vermittlung zwischen Discovery-Plugins und dem Service Bus**: Der Discovery-Manager bietet den Plugins Methoden, um abstrakte Services in den abstrakten Service einzutragen bzw. aus diesem zu entfernen.
- **Tunneling**: Diese Methode kann genutzt werden, um einen Tunnel zwischen zwei Ensembles zu erstellen und ist vor allem dann notwendig, wenn sich beide in eigenen privaten Subnetze befinden. Dem Prinzip der *Network Address Port Translation* (NAPT) [Srisuresh 01] folgend, lauscht der lokale GPAP auf bestimmten Ports auf Anfragen für Services einer der beiden Ensembles und leitet diese dann an den eigentlichen Service weiter. In die abstrakte Service-Beschreibung wird die Adresse am GPAP als `accessLoaction` eingetragen. Zusammen mit einem weiteren Tunnel-Enpunkt in dem anderen Ensemble (durch ein `ProvisionPlugin` erstellt) kann somit Ensemble-übergreifend eine Verbindung (Tunnel) zwischen Service und Consumer implementiert werden.

Abbildung 4.3 zeigt das Interface eines Discovery-Managers zusammen mit dem eines Discovery-Plugins. Die Methode `isActive()` wird durch die `ServiceLayer`-Klasse genutzt, um festzustellen, ob der Manager in Betrieb ist. Mit `getPluginNames()` wird eine Liste der beim Manager registrierten Plugins erfragt. Einzelne Plugins können dann über Ihren Namen direkt mittels `getDiscoveryPlugin()` bezogen wer-

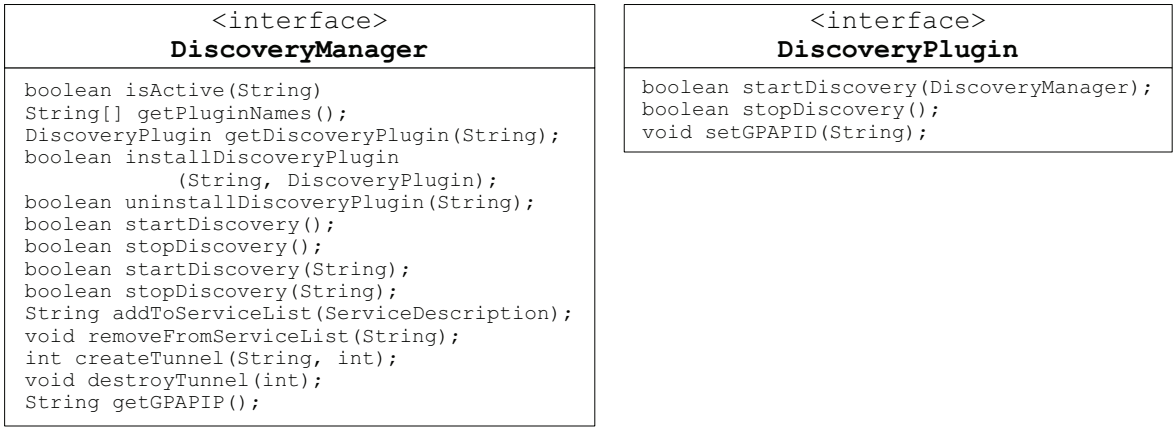


Abbildung 4.3: Die Interfaces `DiscoveryManager` und `DiscoveryPlugin`

den. `installDiscoveryPlugin()` und `uninstallDiscoveryPlugin()` dienen zur Installation und Deinstallation von Plugins durch die `ServiceLayer`-Klasse und mit den Methoden `startDiscovery()` und `stopDiscovery()` kann der Discovery-Prozess für alle installierten oder auch einzelne Discovery-Plugins gestartet und terminiert werden. Während die Plugins in Betrieb sind, nutzen Sie `addToServiceList()`, um die erstellten abstrakten Service-Beschreibungen in den abstrakten Bus einzuspeisen. Als Resultat erhalten sie einen einmalige ID der eingespeisten Service-Beschreibung. Mit dieser können sie diese wieder per `removeFromServiceList()` aus dem Bus entfernen. Weiterhin besitzt die Klasse die Methoden `createTunnel()` und `destroyTunnel()`, um Tunnel zu erstellen bzw. zu löschen. Weiterhin benutzen die Discovery-Plugins die `getGPAPIP()`-Methode, um die korrekte IP des lokalen GPAPs zu ermitteln. Diese wird beispielsweise verwendet, um den korrekten Tunnel-Endpunkt in die abstrakte Service-Beschreibung eintragen zu können.

Im Vergleich zum Discovery-Manager ist das Interface eines Discovery-Plugins sehr einfach. Es definiert lediglich die Methoden `startDiscovery()` und `stopDiscovery()` zum Starten und Terminieren des Discovery-Prozesses eines Plugins und die Methode `setGPAPID()`. Mit dieser wird dem Plugin bei der Installation eine Community-weit eindeutige ID des lokalen GPAPs mitgeteilt, die sie als Wert eines Attributes `stilDiscovered` in die abstrakte Service-Beschreibungen gefundener Services eintragen. Ein lokales Provision-Plugin derselben SOA-Implmentierung würde eine derartige abstrakte Beschreibung nicht konkretisieren da es an dem Attribut und der ebenfalls codierten SOA-Implementierung erkennt, dass es bereits eine derartige Beschreibung im lokalen Ensemble gibt.

Die eigentliche Funktionalität des Plugins ist von der jeweiligen SOA-Implementierung abhängig und folgt den in Abschnitt 3.2.3 definierten Strategien. Vor allem ein Problem vereint jedoch die unterschiedlichen Plugin-Realisierungen: Die Ableitung eines oder mehrerer STiL-Typen aus den jeweiligen SOA-spezifischen Service-Typen. Diese konnte aufgrund der limitierten Zeit für die Realisierung der vorliegenden Arbeit nur sehr ru-

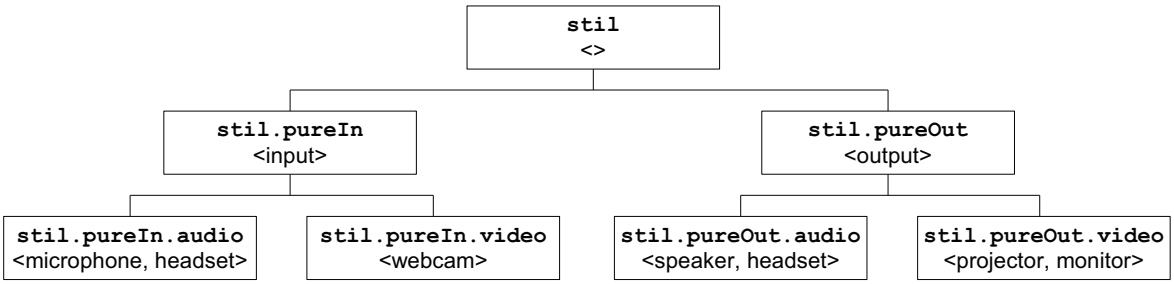


Abbildung 4.4: Auszug aus der Hierarchie der STil-ServiceTypen, versehen mit beispielhaften Schlüsselwörtern

dimentär über ein Schlüsselwort-basiertes Verfahren gelöst werden, welches anhand des in Abbildung 4.4 dargestellten Auszug aus der Hierarchie der STiL-Typen beispielhaft erläutert wird. Zu allen Blattknoten des durch die Hierarchie aufgespannten Baumes sowie einigen Elternknoten wurden eine Reihe von Schlüsselwörtern definiert. Sobald ein Schlüsselwort im Namen oder dem SOA-spezifischen Service-Typ des jeweiligen Services auftaucht, besitzt dessen STiL-Description den entsprechenden STiL-Typen (z. B. hat eine Webcam durch ihren Namen "Webcam" den Typ `stil.pureIn.video`). Es gibt allerdings auch Fälle, in denen mehrere Schlüsselwörter aus unterschiedlichen Knoten für einen Service zutreffen. Für diese wird der STiL-Typ zugeordnet, der die beiden Blattknoten vereint (z. B. `stil.pureOut` für einen "Monitor with speaker"). Allerdings kann diese Vorgehensweise auch zu extremen semantischen Verlusten führen, da beispielsweise ein "Headset" nur den grundlegenen STiL-Typen `stil` besitzt, wodurch wenig Aussagekraft durch den Service-Typen gegeben ist. Für die durchgeführten und in Kapitel 5 erläuterten Szenarien war diese Vorgehensweise ausreichend, zumal auch der `friendlyName` einer STiL-Description aussagekräftig ist. Für weitere Arbeiten empfiehlt sich jedoch ein effektiveres Verfahren zu wählen, welches beispielsweise die Schlüsselwörter gewichtet oder Schlussfolgerungen basierend auf ontologischen Formulierungen der STiL-Typen-Hierachie trifft.

Gemäß den in diesem Abschnitt erläuterten Java-Interfaces wurden bisher Discovery-Plugins für Web Services, DNS-SD (basierend auf Apples Implementierung *Bonjour*), Jini und Bluetooth implementiert und durch die in Kapitel 5 aufgeführten Szenarien evaluiert.

Im folgenden Abschnitt wird auf das Gegenstück zum Discovery-Manager und seinen Plugins eingegangen: Den Provision-Manager sowie die Provision-Plugins.

4.1.3 Provision-Manager und -Plugins

Durch Provision-Plugins werden abstrakte Service-Beschreibungen für eine bestimmte SOA-Implementierung konkretisiert und im lokalen Ensemble angeboten. Analog zu den Discovery-Plugins ist auch die Verwendung mehrerer Provision-Plugins in dem für

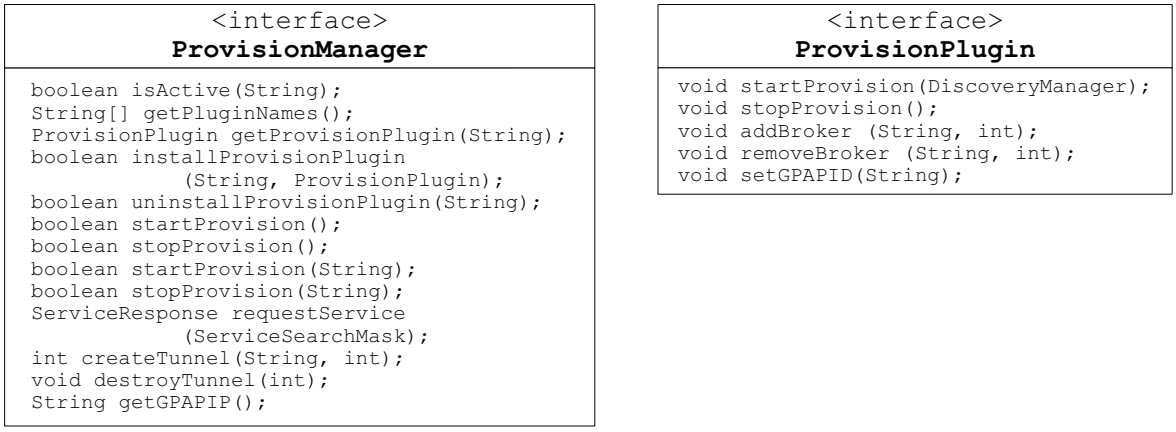


Abbildung 4.5: Die Interfaces `ProvisionManager` und `ProvisionPlugin`

dieses Ensemble zuständigen GPAP sinnvoll und ein `DiscoveryManager` koordiniert diese. Seine konkreten Aufgaben sind:

- **Installation und Deinstallation der vom ServiceLayer übergebenen Provison-Plugins** (analog zum Discovery-Manager)
- **Initialisierung und Terminierung der installierten Provison-Plugins**: Nach ihrer Initialisierung beginnen die Provision-Plugins je nach Provision-Strategie (siehe Abschnitt 3.2.3) mit dem Angebot der im abstrakten Service Bus gefundenen Services oder nehmen Requests von lokalen SOA-spezifischen Consumern entgegen. Die durch ein Plugin angebotenen Services werden beim Terminieren des Plugins wieder aus dem SOA-spezifischen Service Bus entfernt.
- **Vermittlung zwischen Provision-Plugins und dem Service Bus**: Der Provision-Manager bietet den Plugins Methoden, um abstrakte Services aus dem abstrakten Service Bus zu beziehen.
- **Tunneling**: Der Provision-Manager erstellt die Gegenstücke zu den durch den Discovery-Manager erstellten Tunnel-Endpunkten.

Abbildung 4.5 stellt das Interface eines Provision-Managers zusammen mit dem eines Provision-Plugins dar. Die Methoden `isActive()`, `getPluginNames()`, `getProvisionPlugin()`, `installProvisionPlugin()`, `uninstallProvisionPlugin()`, `startProvision()`, `stopProvision()`, `createTunnel()`, `destroyTunnel()` und `getGPAPIP()` funktionieren analog zu ihren Äquivalenten bei der `DiscoveryManager`-Klasse. Anstelle von Methoden zum Einspeisen oder Entfernen von abstrakten Services aus dem abstrakten Bus besitzt die Klasse `ProvisionManager` jedoch die Methode `requestService()`, mit der ein Provision-Plugin seinem Manager eine der in Abschnitt 3.2.2 beschriebenen Suchmasken übergeben kann. Dieser leitet sie an den abstrakten Service Bus weiter, der wie in Abschnitt 4.1.5 beschrieben die zu der Maske passenden abstrakten Service-Beschreibungen ermittelt und an den Provision-Manager zurückgibt.

Analog zu den Discovery-Plugins besitzen die Provision-Plugins Methoden zum Starten und Stoppen ihrer individuellen Provision-Prozesse (`startProvision()`, `stopProvision()`). Weiterhin besitzen diese Plugins Methoden zum Registrieren und Entfernen von SOA-spezifischen Brokern, indem dem Plugin ihre Adresse übergeben wird (`addBroker()`, `removeBroker()`). Bei den angegebenen Brokern werden die generierten konkreten Service-Beschreibungen registriert. Außerdem wird die Methode `setGPAPID()` wiederum genutzt, um die angebotenen Beschreibungen mit dem Attribut `stilProvided` und dem Wert der GPAP-ID zu versehen. Ein Discovery-Plugin derselben SOA-Implementierung wird eine derartige Service-Beschreibung nicht abstrahieren, da es anhand dieses Attributs erkennt, dass ihr Ursprung bereits eine abstrakte Beschreibung war.

Auch bei Provision-Plugins stellt sich das in Abschnitt 4.1.2 bereits beschriebene Problem der Ableitung von Service-Typen. Es wurde für diesen Prototypen über eine Plugin-eigene Liste gelöst, die jedem abstrakten Service-Typ einen konkreten Service-Typ zuordnet. Diese Lösung ist analog zu dem in Abschnitt 4.1.2 beschriebenen Verfahren für die Praxis nicht ausreichend, genügte jedoch für die durchgeführten Szenarien. Auch hier empfiehlt sich für spätere Arbeiten die Entwicklung eines effektiveren Verfahrens, beispielsweise auf der Basis von Service-Typ-Ontologien.

Für den enwickelten Prototypen wurden bisher Discovery-Plugins für Web Services, DNS-SD (ebenfalls basierend auf Apples Implementierung *Bonjour*), Jini und Bluetooth implementiert.

Der folgenden Abschnitt behandelt kurz die Kombinationen der beiden bisher vorgestellten Plugin-Arten: Broker-Plugins und ihren Broker-Manager.

4.1.4 Broker-Manager und -Plugins

Broker-Plugins kombinieren die Aufgaben von Discovery- und Provision-Plugins und können eingesetzt werden, wenn die jeweilige SOA-Implementierung den gleichzeitigen Betrieb mehrerer lokaler Broker oder die Modifikation vorhandener Broker unterstützt. Die Aufgaben von Broker-Plugins werden in dieser Arbeit nicht separat aufgeführt, da sie sich aus den Aufgaben der beiden vorhergehenen Plugins ergeben.

Wie Abbildung 4.6 zeigt ist auch das Interface der Klasse `BrokerManager` eine Kombination der Interfaces `DiscoveryManager` und `ProvisionManager`. Das Interface der `BrokerPlugin`-Klassen hat nur die Methoden zum Initialisieren und Terminieren der jeweiligen Plugins sowie die bereits vorgestellte Methode `setGPAPID`.

Broker-Plugins wurden bisher in Anbetracht der begrenzten Zeit für diese Arbeit nicht entwickelt, jedoch wurde mit dem Broker-Manager sowie dem Plugin-Interface die Möglichkeit für deren zukünftige Implementierung und Integration in das Framework geschaffen.

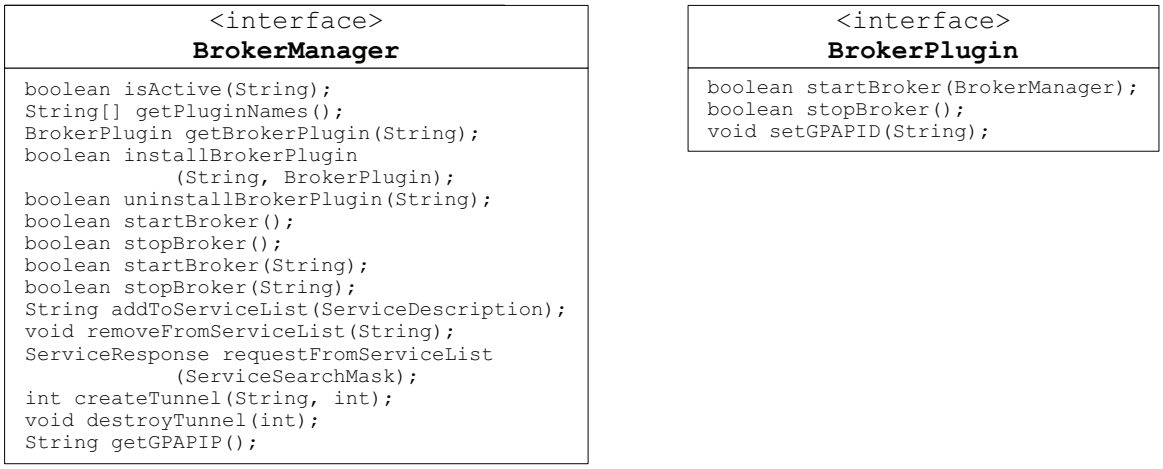


Abbildung 4.6: Die Interfaces `BrokerManager` und `BrokerPlugin`

Sämtliche Plugins bedienen sich eines gemeinsamen abstrakten Services Busses, in den sie abstrakte Service-Beschreibungen einspeisen oder aus dem sie diese auslesen. Die prototypische Implementierung dieses Busses ist Thema des folgenden Abschnitts.

4.1.5 Service Bus

Der abstrakte Service Bus ist die logische Verbindung zwischen den GPAP-Service-Layern verschiedener GPAPs und enhält alle STiL-Service-Beschreibungen. Darauf aufbauen sind seine konkreten Aufgaben:

- **Persistente Speicherung von abstrakten Service-Beschreibungen von Discovery- und Broker-Managern**: Die konkrete Art der Speicherung hängt von der spezifischen Bus-Implementierung ab. Abschnitt 4.1.6 beschreibt zwei Speicherungsansätze.
- **Annahme von Service-Beschreibungen und von Anfragen auf diesen**: Discovery- und Broker-Manager speisen ihre Service-Beschreibungen in den Bus ein und Provision- sowie Broker-Manager können diese über Suchmasken abfragen.
- **Logisches Verbinden verschiedener GPAP-Service-Layer**: Um Service-Beschreibungen und -Abfragen zwischen GPAPs übertragen zu können sowie einen gemeinsamen, verteilten Raum für Service-Beschreibungen zu gewährleisten, müssen die einzelnen Service-Layer über den Service Bus miteinander kommunizieren können.

Das in Abbildung 4.7 dargestellte Java-Interface `UnifiedServiceBus` muss von abstrakten Service Bussen des aktuellen Service-Layers implementiert werden. Zum Einspeisen von Service-Beschreibungen bietet es die Methode `addService()`, die von Discovery- und Broker-Managern direkt genutzt wird.

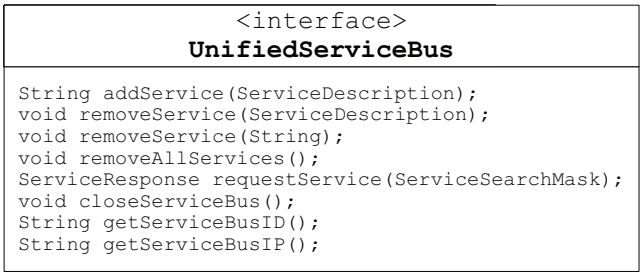


Abbildung 4.7: Das Interface `UnifiedServiceBus`

Das Entfernen von Service-Beschreibungen aus dem Bus kann über eine der beiden `removeService()`-Methoden erfolgen, die entweder eine konkrete Service-Beschreibung erwarten oder einen String, der dem jeweiligen Manager beim Einspeisen der Service-Beschreibung als Referenz auf die Beschreibung vom Service Bus übergeben wurde. Weiterhin steht die Methode `removeAllServices()` zur Verfügung, die alle von diesem GPAP in den Bus eingespeisten Beschreibungen wieder aus dem Bus entfernt. Dies ist beispielsweise beim Terminieren des lokalen GPAPs sinnvoll.

Die Methode `requestService()` dient der Suche von Service-Beschreibungen anhand von Suchmasken und liefert eine STiL-Response zurück. Die Auswahl der zu einer Suchmaske passenden Service-Beschreibungen für die Response geschieht wie in Abschnitt 3.2.2 erläutert. Eine in einer studentischen Arbeit prototypisch implementierte *Matching Unit* [Lenz 08] ist für die Umsetzung der Auswahl-Regeln zuständig.

Während sämtliche Anmeldeprozesse eines GPAPs beim Service Bus in dem `UnifiedServiceBus`-Konstruktor vorgenommen werden können, ist je nach konkreter Realisierung des Service Busses bei Terminierung eines GPAPs möglicherweise ein Abmeldesprozess notwendig (z. B. Abmeldung aus einem P2P-basierten Service Bus und somit Aktualisierung der Verweise anderer Peers). Die Methode `closeServiceBus()` ist zur Aufnahme dieses Prozesses gedacht und wird vom `ServiceLayer` bei dessen Terminierung aufgerufen.

Die beiden restlichen Methoden `getServiceBusID()` und `getServiceBusIP()` werden von den Manager-Klassen genutzt, um die in den vorangegangenen Abschnitten erläuterte GPAP-IP und GPAP-ID zu ermitteln. Da der GPAP im Netzwerk durch seine Service Bus-Anbindung identifizierbar ist, liefert diese auch IP und ID.

Wie bereits in Abschnitt 3.2.4 erläutert, gibt es konkurrierende Ansätze zur Implementierung einer Community aus GPAPs, die aus Sicht der Implementierung nicht anderes als eine Verbindung der GPAPs über den gemeinsamen Service Bus darstellt. Im folgenden Absatz werden zwei dieser Community- und somit Service Bus-Implementierungen näher betrachtet, die sich im Laufe der vorliegenden Arbeit als vielversprechend erwiesen haben.

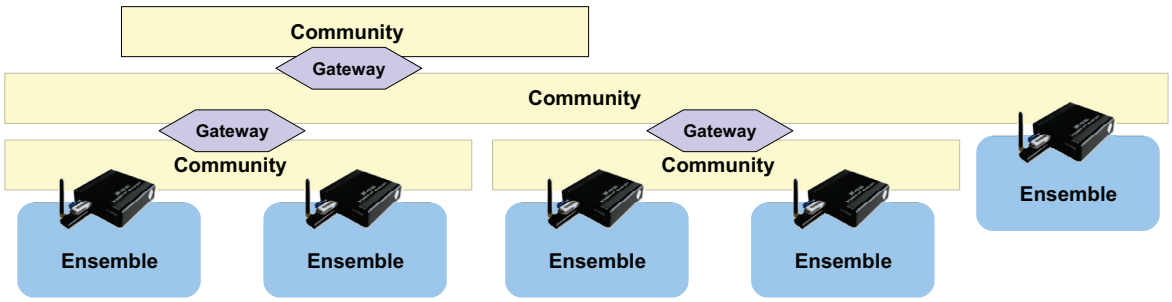


Abbildung 4.8: Eine Community-Hierarchie verbindet Communitys mit speziellen Gateway-Knoten.

4.1.6 Community-Implementierungen

Die Topologie einer GPAP-Community hängt stark von ihrem Einsatzgebiet ab. Wie in Abschnitt 3.2.4 erläutert, eignet sich beispielsweise eine Baum–Topologie gut für den Einsatz in hierarchischen Organisationen wie einer Universität. Aus diesem Grund wurde im Rahmen der vorliegenden Arbeit zunächst eine Community-Hierachie zur Verbindung der einzelnen GPAPs angestrebt und beispielhaft umgesetzt [Heinemann 08]. Die Grundstruktur der GPAP-Hierarchie ist in Abbildung 4.8 dargestellt.

Die Community-Hierarchie besteht aus einer Reihe von eigenständigen Communitys zur Verbindung untergeordneter Communitys oder Ensembles. Während die GPAP-Knoten für die Integration eines Ensembles in eine Community zuständig sind, verbinden *Gateways* Communitys wiederum zu höherwertigen Communitys.

Wird von einem GPAP eine Service-Beschreibung in die Community eingespeist, so wird diese an alle Gateways der Community weitergeleitet. Jedes Gateway entscheidet selbstständig, ob es diese der jeweils höheren Community anbieten möchte. Diese Entscheidung kann beispielsweise anhand des Service-Typs erfolgen. So kann es Service-Arten (z. B. Druck-Service eines Instituts) geben, die nur in der niederen Community zur Verfügung stehen (z. B. dem Institut), jedoch nicht der nächsthöheren Community angeboten werden sollen (z. B. keine institutseigenen Druck-Services durch die gesamte Fakultät nutzbar). Auf diese Art und Weise können Nutzungsrechte einer organisatorischen Einheit gewahrt werden. Soll ein Service der nächsthöheren Community angeboten werden, so speichert das entsprechende Gateway selbst die Beschreibung und sendet sie außerdem an alle Gateways der höheren Community.

Sendet ein GPAP eine Service-Suchmaske in die Community, so kann deren `scope`-Element als *Hop Count* genutzt werden, um zu spezifizieren, wie viele Gateways die Abfrage durchlaufen darf. Dadurch ist es möglich, eine Abfrage auf organisatorisch nahe Service-Beschreibungen einzugrenzen (z. B. Service-Beschreibungen aus dem eigenen Institut). Weiterhin ist es jedem einzelnen Gateway überlassen, ob es die Abfrage in die jeweils höhere Community weiterleitet oder nicht. Die in der Hierachie gefunden Service-Beschreibungen werden an den anfragenden GPAP weitergeleitet, der diese zu einer gemeinsamen STiL-Response zusammensetzt.

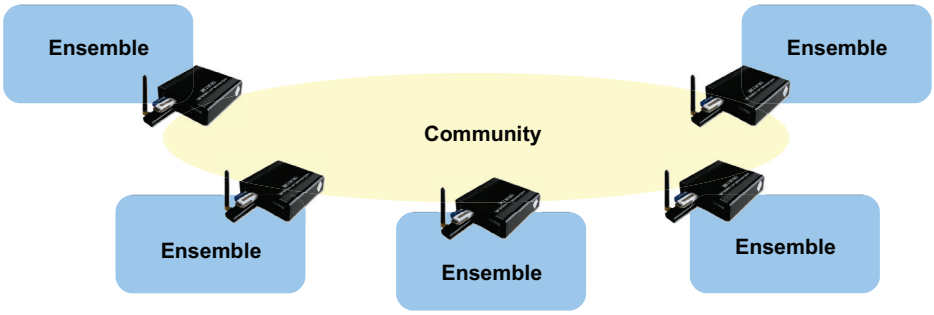


Abbildung 4.9: In einer flachen Community (z. B. als P2P-Netzwerk) gehören alle Ensembles einer einzigen Community an.

Von den in Abschnitt 3.2.4 geschilderten Vor- und Nachteilen gestaltete sich in dieser Implementierung insbesondere die Integration neuer GPAPs oder gar Gateways in die Infrastruktur als sehr aufwendig, da diese je nach Position in der Hierarchie mit den Adressinformationen ihrer Geschwister-, Eltern- und Kinderknoten konfiguriert werden müssen. Dennoch ist dieser Ansatz für sehr statische Communitys vor allem durch die natürliche Einbeziehung organisatorischer Hierachien sinnvoll und effizient [Heinemann 08].

Um auch für dynamischere GPAP-Communitys eine topologische Lösung anzubieten, wurde als Alternative zur Baum-Topologie eine P2P-basierte GPAP-Vernetzung implementiert [Zender 10b]. Die dadurch entstandene "flache" Community-Struktur illustriert Abbildung 4.9.

In dieser Community-Variante sind alle GPAPs Peers eines flachen P2P-Netzes. Dieses wurde in Form eines als *Distributed Hash Table* (DHT)-basierten Ringes der P2P-Technologie Chord umgesetzt [Stoica 01], da für Chord eine Java-Bibliothek (OpenChord [Kaffille 07]) zur Verfügung stand, die unkompliziert in den Java-basierten Service-Layer eingebunden werden konnte. In dem DHT-Ring wird jedem Peer ein Bereich in einem Raum von Hashwerten zugewiesen. Für in dem Netz registrierte Daten werden ebenfalls Hashwerte desselben Raumes gebildet. Jeder Peer verwaltet Verweise auf die Datensätze innerhalb seines Hashwertbereiches sowie auf direkte und einige entferntere Nachbar-Peers (siehe [Stoica 01] für weitere Details).

In das Netzwerk eingespeiste Service-Beschreibungen werden wie gewöhnliche Datensätze behandelt und unter Angabe eines Schlüssels (Service-Typ, aus diesem berechnet sich der Hashwert) in dem P2P-Netzwerk registriert. Unter Angabe desselben Schlüssels können die Service-Beschreibungen von allen Peers/GPAPs wiedergefunden werden.

Für die P2P-Community gelten die in Abschnitt 3.2.4 aufgeführten Vor- und Nachteile. Durch das minimale Vorwissen, das ein GPAP benötigt, um Mitglied des P2P-Netzes zu werden (Adresse eines einzigen anderen GPAPs), ist insbesondere die Integration neuer GPAPs in das Netzwerk unkompliziert.

Während der Arbeit mit dieser Community-Lösung kristallisierte sich jedoch auch ein zusätzlicher Nachteil heraus: Da bei Chord (so wie bei vielen P2P-Implementierungen) nur ein einziges, eindeutiges Schlüsselwort zur Suche nach einem Datensatz verwendet werden kann, wurden Service-Beschreibungen nur unter Angabe ihres Service-Typs als Schlüssel gespeichert. Dadurch kann ein GPAP allerdings auch nur den Service-Typ als Suchwort verwenden und erhält als Resultat alle Service-Beschreibungen dieses Typs. Die Auswertung des restlichen Suchmaske muss lokal auf dem GPAP erfolgen. Gerade bei vielen Service-Beschreibungen eines Typs muss der GPAP demnach einen hohen Arbeitsaufwand leisten. Dieses Problem kann beispielsweise durch die Wahl komplexerer P2P-Technologien (z. B. JXTA, Kad) als Community-Grundlage gelöst werden. Dies sollte bei den weiteren Arbeiten an einer Community-Lösung berücksichtigt werden.

Die P2P-Community ist gut für dynamische Communitys geeignet, leistet jedoch keine Abbildung von organisatorischen Hierachien. Für die weiteren Arbeiten an der GPAP-Community erscheint daher eine hybride Lösung sinnvoll, die die beiden vorgestellten Ansätze kombiniert. Beispielsweise könnte eine grundlegend hierarchische Community P2P-Netze zur Realisierung von GPAP-nahen Teil-Communitys nutzen. Die zuständigen Gateways wären dann für die GPAP-nahe Community Peers und würden so die Integration von GPAPs (ebenfalls als Peers) erleichtern. Auch dieser Ansatz sollte in folgenden Arbeiten untersucht werden.

Für Evaluierung dieser Implementierung und die damit verbundene prototypische Realisierung der Lehr- und Lernarragements einer pervasiven Universität wurde die P2P-Lösung aufgrund ihrer Flexibilität und Robustheit bevorzugt.

4.2 Evaluierung und Kontext

Die korrekte Funktionsweise der in diesem Kapitel erläuterten Implementierung wurde im Laufe der vorliegenden Arbeit einerseits anhand diverser Komponententests verifiziert. Andererseits wurden konkrete Fragestellungen aus dem Gebiet der Service-orientierten Universität als Infrastruktur für die Vision einer Pervasiven Universität genutzt, um das korrekte Zusammenspiel der einzelnen Komponenten und somit der Funktionstüchtigkeit des Gesamtsystems in Fallstudien zu demonstrieren und verifizieren.

In der in diesem Abschnitt beschriebenen Implementierung des GPAP-Service-Layers wurde noch keine systematische Integration und Nutzung von Kontextinformationen implementiert, da dies nicht im direkten Fokus der Arbeit lag und deren Rahmen gesprengt hätte. Dennoch wurden eine Reihe von eigenständigen Arbeiten zu ausgewählten Apekten durchgeführt, um Erkenntnisse über die Nutzbarkeit von und den Umgang mit Kontextinformationen in pervasiven Systemen zu untersuchen. Aufbauend auf diesen Arbeiten, die ebenfalls im folgenden Kapitel betrachtet werden, kann die systematische Integration eines Kontextbewusstseins in die GPAP-Implementierung in der weiterführenden Forschung im Graduiertenkolleg MuSAMA durchgeführt werden.

Kapitel 5

Evaluierung: Die Universität der Zukunft

Wie bereits in Abschnitt 1.2 dargelegt, müssen Hochschulen sich heute sowohl neuen organisatorischen Herausforderungen (z. B. wachsende Mobilität der Studenten) als auch daraus oftmals resultierenden technologischen Herausforderungen (z.B. Mobile Computing, Open Content) stellen. Da die dadurch entstehenden Problemstellungen (z. B. Bedarf nach einer flexiblen und dynamischen Infrastruktur) bereits im Forschunggebiet Pervasive Computing behandelt werden, ist eine zunehmende Integration von Technologien des Pervasive Computing in Lehr- und Lernprozesse erkennbar [Tavangarian 09].

Dieses Kapitel betrachtet den Weg einer traditionellen Universität zur *Pervasive University*, der Universität der Zukunft, und fokussiert dabei vor allem infrastrukturelle Aspekte. Die in der vorliegenden Arbeit motivierte Eignung Service-orientierter Architekturen als Infrastruktur für moderne Bildungseinrichtungen wird weiterhin anhand ausgewählter Fallstudien vertieft und verifiziert. Außerdem dienen SOA-übergreifende und kontextbewußte Szenarien der Evaluierung der in dieser Arbeit vorgestellten Konzepte und Implementierungen.

5.1 Die Pervasive Universität

5.1.1 Der Weg zur Pervasiven Universität

Mit der breiten Verfügbarkeit mobiler Netzwerktechnologien wie WLAN und *Universal Mobile Telecommunications System* (UMTS), smarter und kontextbewußter Nutzerendgeräte wie das iPhone oder Android-basierte Smartphones sowie freier und nutzergenerierter Inhalte haben Pervasive Computing-Technologien Einzug in das öffentliche Leben erhalten. So ist es heute beispielsweise fast selbstverständlich, orts- und zeitunabhängig über das mobile Internet Nachrichten zu konsumieren oder mit anderen Nutzern zu interagieren. Natürlich ist auch die Universität als Ort öffentlichen Lebens von

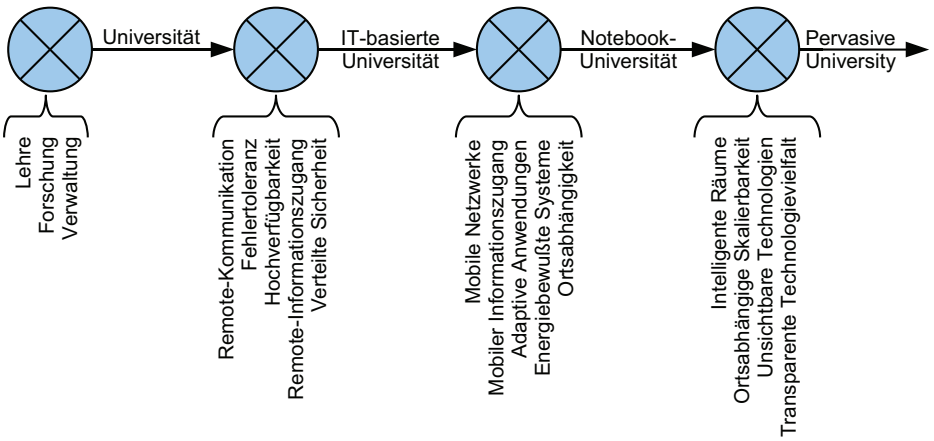


Abbildung 5.1: Analog zur Evolution moderner Computing-Paradigmen führt die Entwicklung von Hochschulen (frei übersetzt aus [Tavangarian 09]) von der klassischen, dreigeteilten Einrichtung über die IT-basierte und Notebook-Universität zur Pervasive University.

dieser Entwicklung nicht ausgeschlossen. Wie Abbildung 5.1 illustriert, hat sich parallel zur in Abschnitt 1.1.1 betrachteten Evolution moderner Computing-Pradigmen auch ein Wandel an den Universitäten vollzogen [Tavangarian 09]. Aus der klassischen Universität mit ihren traditionellen Lehr-, Forschungs- und Verwaltungsaufgaben wurde durch die zunehmende IT-Durchdringung (z. B. in Form des E-Learning) eine IT-basierte Universität und wenige Jahre später durch den wachsenden Markt an mobilen Technologien und drahtlosen Netzwerken die Notebook-Universität [Tavangarian 01].

Durch die Einführung von Technologien und Strategien aus der Pervasive Computing-Forschung, die helfen die neue Mobilität und allgegenwärtige Vernetzung zu beherrschen, entwickeln sich Hochschulen weiter zu pervasiven Bildungseinrichtungen (Pervasive University).

> Eine **Pervasive University** ist eine Bildungseinrichtung, die in zielgerichteter Art und Weise Mechanismen und Artefakte des Pervasive Computing einsetzt. Aus Anwendungssicht ist sie eine Universität mit nahtloser IT-Unterstützung in all ihren Aktivitätsfeldern: Der elektronisch gestützten Lehre, Forschung und Verwaltung. Aus technischer Sicht ist sie eine Pervasive Computing-Umgebung, deren Komponenten und Interaktionsmuster auf die Charakteristika einer Universität zugeschnitten sind.
>
> (frei übersetzt aus [Tavangarian 09])

Die Weiterentwicklung zur Pervasive University müssen Bildungseinrichtungen somit nicht nur technisch sondern auch sozial vollziehen. Analog zum Pervasive Computing im allgemeinen würde sich eine derartige Evolution nicht ohne die Akzeptanz durch die betroffenen Nutzer durchsetzen [Plociennik 10]. Der in dieser Arbeit im Vordergrund stehende technische Aspekt einer flexiblen und dynamischen Infrastruktur in Form einer Service-orientierten Architektur führt zur Service-orientierten Universität

als infrastrukturelle Grundlage der Pervasive University. Diese wird im kommenden Abschnitt näher betrachtet.

5.1.2 Service-orientierte Universität

Wie bereits in Kapitel 2 begründet, eignen sich vor allem Service-orientierte Architekturen als Kommunikationsmodell für die Infrastruktur pervasiver Umgebungen. Da dies auch für die Pervasive University zutrifft, ist eine service-orientierte Universität deren infrastrukturelle Grundlage.

Die potentiellen Services einer derartigen Universität lassen sich direkt aus den Ressourcen, Werkzeugen und anderen Diensten ableiten, die schon in einer Notebook-Universität in den drei universitären Aktivitätsfeldern Anwendung finden [Zender 09c]:

- **Dienste der Aus- und Weiterbildung** umfassen beispielsweise universitärer Kommunikationsdienste, das Angebot von Lehrmaterial für Lehrveranstaltungen, den Zugriff auf zusätzliche Ressourcen wie E-Books und Multimedia sowie die Verteilung von Vorlesungsaufzeichnungen. Weiterhin nutzen Lehrende und Lernende kollaborative Dienste wie etwa den Zugriff auf Chats und virtuelle Welten sowie persönliche Dienste wie Blogs und eigene Webseiten.
- Die **Dienste der Forschung** überschneiden sich teilweise mit denen der Aus- und Weiterbildung (z. B. bei Wissensdiensten wie den Zugriff auf E-Books und Multimedia), umfassen jedoch auch klar forschungsspezifische Dienste wie den Zugriff auf Spezialequipment (z. B. Rechencluster und Simulatoren), die Fernsteuerung industrieller Maschinen und Geräte und die Nutzung von Grid- und Cloud-Diensten.
- **Dienste der Verwaltung** einer Universität sind bereits heute schon oft als netzwerkfähige Anwendungen implementiert. Sie dienen beispielsweise der Koordination der universitären Infrastruktur (z. B. Raumbelegungspläne), der Studentenverwaltung (z. B. Online-Immatrikulation und Überwachung der Studienergebnisse), dem Personal- und Finanzmanagement (z. B. Leistungsüberwachung und Lohnverwaltung) und weiteren administrativen Aufgaben (z. B. Zugriff auf den Mensa-Speiseplan und universitätsinterne News-Kanäle).

Auch wenn diese Dienste bereits als netzwerkfähige Komponenten implementiert sind, handelt es sich doch oft um Einzellösungen. Es fehlt in den meisten Fällen eine allgemeine, systematische Infrastruktur und viele dieser Dienste sind noch keine Services im Sinne einer SOA. Die im Rahmen dieser Arbeit vorgestellten SOA-Implementierungen unterstützen die Kapselung der Funktionalitäten der genannten Dienste hinter Service-Schnittstellen und deren Einbindung in entsprechende SOAs. Dieser Schritt muss für alle Dienste vorgenommen werden, die im Rahmen der Service-orientierten Universität angeboten werden sollen. Er ist zwar in Anbetracht der Masse universitärer Dienste langwierig, aber unumgänglich, um die heute existierenden Einzellösungen in eine systematische Architektur zu überführen. Die bisherigen Infrastrukturen an Hochschulen

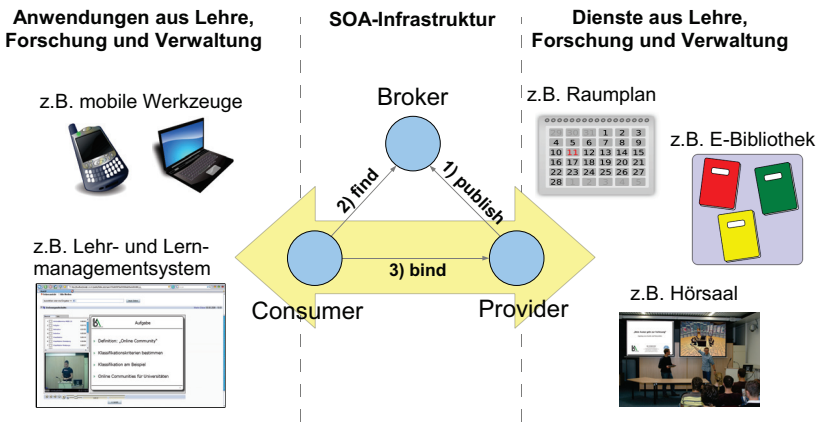


Abbildung 5.2: Service-orientierte Architekturen können universitäre Dienste systematisc in Form von SOA-Services universitären Anwendungen zur Verfügung stellen.

nehmen – beschleunigt durch hochschulübergreifende Herausforderungen – rasch an Komplexität zu und es werden neue organisatorische Rahmenbedingungen benötigt [Lucke 07], wie eine Service-orientierte Infrastruktur (Abbildung 5.2). In Abschnitt 5.2 wird exemplarisch gezeigt, wie die Überführung eines Dienstes in einen SOA-Service erfolgen kann.

Das im Rahmen dieser Arbeit entwickelte und in Abbildung 5.3 dargestellte Modell einer Service-orientierten Institution basiert auf der bereits vorgestellten Einteilung der Infrastruktur in Ensembles und Communities. Ein Ensemble könnte beispielsweise die Infrastruktur einer Abteilung der Institution (in einer Hochschule z. B. Institute, Lehrstühle, Labore oder Mensen) umfassen. In ihm gibt es Hardware-Ressourcen und dazu gehörige Anwendungen, die von Nutzern über Benutzungsschnittstellen genutzt werden. Die Anwendungen können in zwei Arten unterteilt werden: hardwarefokussierte Anwendungen, die direkt der Kontrolle der Hardware dienen (z. B. Fernbedienungen für industrielle Maschinen) und Software-fokussierte Anwendungen, die lediglich auf Hardware gehosted werden (z. B. HTTP-Server und andere Web-Anwendungen). Entsprechend dieser Einteilung gibt es auch zwei Service-Arten: Services, die Hardware kontrollieren und Services, die Anwendungen nutzen [Zender 09c]. Die Service-Beschreibungen der Services werden wie in Abbildung 5.3 dargestellt stets von Anwendungen angeboten.

Um die SOA-Heterogenität der angebotenen Services zu überwinden wird die bereits beschriebene GPAP-Komponente eingesetzt. Die resultierenden abstrakten Service-Beschreibungen des Ensembles werden in den abstrakten Service Bus eingespeist, der Ensemble-übergreifend die Community bildet und an Universitäten beispielsweise die Fakultät, die gesamte Universität oder gar hochschulübergreifende Organisationen umfasst. Durch die in Abschnitt 3.2.4 beschreibenen hierachischen Communitys ist außerdem eine Abbildung komplexerer organisatorischer Hierarchien möglich.

Abbildung 5.4 konkretisiert dieses Modell in Bezug auf eine Service-orientierte Universität. Während die Transformation heterogener sowie die Verwaltung abstrakter

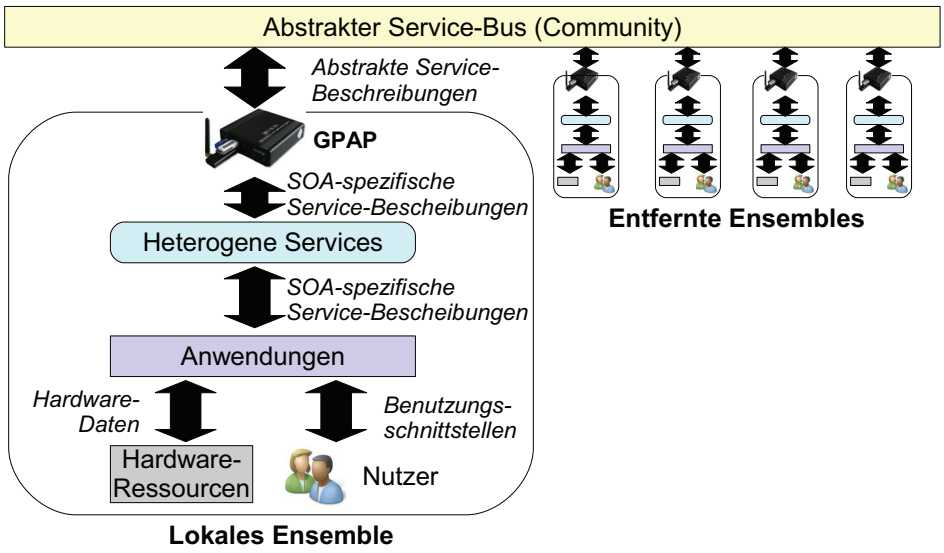


Abbildung 5.3: Abstraktes Modell einer Service-orientierten Institution

Services an einer Hochschule und selbst über Hochschulgrenzen hinweg (Integration externer Umgebungen) im Wesentlichen den in den vergangenen Kapiteln erarbeiteten Vorgehensweisen entspricht, wird im folgenden auf die hochschulspezifischen Hardware- und Software-Anteile des Modells eingegangen.

Hardware-Ressourcen und Nutzerendgeräte

Heutige Universitäten nutzen eine Vielzahl unterschiedlicher Hardware. Das Spektrum reicht von Hochleistungsrechentechnik in den Rechenzentren über Spezialequipment für naturwissenschaftliche Experimente bis hin zu RFID-Tags, die der Identifizierung von Studenten und Mitarbeitern dienen. Weiterhin nutzen viele Universitätsangehörige persönliche Geräte wie Mobiltelefone, Laptops und Smartphones oder öffentliche Workstations, um mit der universitären Infrastruktur zu interagieren. Ein Auszug aus der Hardwarevielfalt wird mitsamt seiner Herausforderungen für die Service-orientierte Universität im folgenden betrachtet.

Spezielle und allgemeine **Server** sind für die Infrastruktur einer Bildungseinrichtung essentiell. Sie hosten beispielsweise persönliche und dienstliche Webseiten, E-Mail-Systeme, Lehr- und Lernmanagementsysteme, verschiedene Anwendungen und Lehrmedien. Diese Systeme sind bereits mit Netzwerken wie dem Internet oder dem internen Hochschulnetzwerk verbunden. Somit können die zugehörigen Anwendungen nach Auswahl einer SOA-Implementierung relativ einfach mit Provider- und Consumer-Logik erweitert werden. Dasselbe gilt für die breite Masse der Multimedia-Arbeitsplätze und -Labore, mit denen heutige Hochschulen ausgestattet sind [Lucke 07].

Im Gegensatz dazu fehlt es **Spezialequipment** oftmals an einer Netzwerverbindung. Daher sind Mediatoren notwendig, die die Hardware in ein Netzwerk integrieren und zwischen der SOA und der Hardware vermitteln. Am Lehrstuhl für

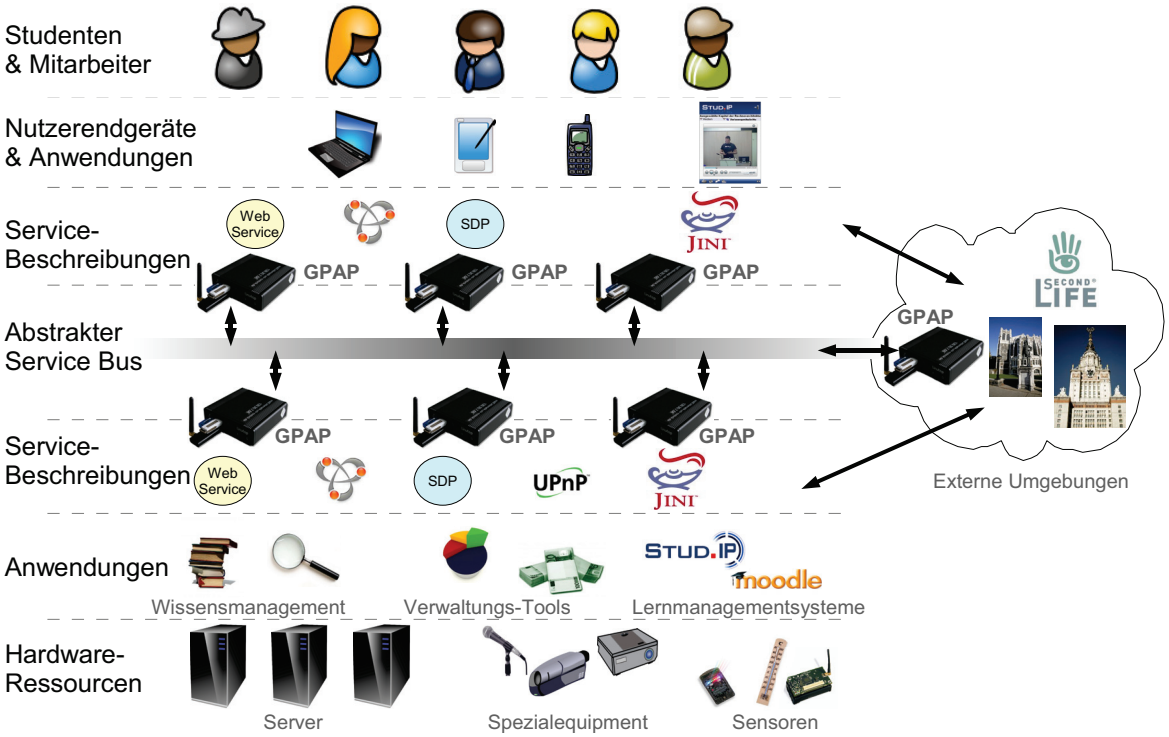


Abbildung 5.4: Konkretisiertes Modell einer Service-orientierten Universität: Hardware-Ressourcen, Nutzerendgeräte, Anwendungen sowie externe Umgebungen werden systematisch und flexibel in die universitäre Infrastruktur integriert.

Rechnerarchitektur der Universität Rostock wurde im Rahmen dieser Arbeit beispielsweise eine derartige Mediatorenkomponente implementiert, die die E-Learning-Werkstatt – eine Präsenzlehrumgebung mit vielfältigem E-Learning-Equipment – in eine SOA-Infrastruktur integriert und somit die Remote-Steuerung und eine werkstattübergreifende Lehrmedien-Übertragung ermöglicht [Zender 09b].

Für pervasive Umgebungen spielen insbesondere **Sensoren** eine wichtige Rolle. Diese messen Rohkontext über die Umgebung (z. B. Luftfeuchtigkeit, Helligkeit, Temperatur) und den Nutzer (z. B. Identität, Position, Geschwindigkeit). Um diese Sensoren in größere Netzwerke zu integrieren werden üblicherweise ausgewählte spezielle Sensorknoten mit zusätzlichen Ressourcen genutzt. Im Rahmen dieser Arbeit wurden beispielhaft Sensoren als Services in eine SOA integriert. Die Ergebnisse werden in Abschnitt 5.4.1 geschildert.

Neben den universitären Hardware-Ressourcen nutzen Studenten und Mitarbeiter ihre persönlichen Nutzerendgeräte, die ebenfalls in die universitäre Infrastruktur integriert werden müssen. Es ist vorteilhaft, dass diese heute die Kommunikation in den Vordergrund stellen und sich über UMTS, WLAN oder auch *General Packet Radio Service* (GPRS) mit größeren Netzwerken wie dem Internet verbinden können. Durch das Netzwerk-Layer des GPAP [Dressler 10] werden diese Geräte mit der GPAP-

Infrastruktur verbunden und müssen analog zu den Multimedia-Arbeitsplätzen nur noch um SOA-Logik erweitert werden.

Anwendungen und Services

Neben Hardware-Ressourcen verwenden Universitäten eine Reihe unterschiedlicher Anwendungen zur Unterstützung von Lehre, Forschung und Verwaltung. Dazu gehören beispielsweise Werkzeuge zum **Wissensmanagement** (z. B. digitale Bibliotheken) und **Verwaltungs-Tools** die beispielsweise der Organisation der universitären Infrastruktur (z. B. Raumplaner) und der Studierendendaten (z. B. Lösungen der Hochschul-Informations-System GmbH) dienen. In den Folgenden Abschnitten spielen vor allem *Lernmanagementsysteme* eine wichtige Rolle als universitäre Anwendungen. Diese dienen neben der Organisation von Veranstaltungen und der Verbreitung von Lehrmaterialen mehr und mehr auch der sozialen Interaktion und Präsentation von Studenten.

Auf dieser Ebene ist die zentrale Herausforderung die Erweiterung existierende Anwendungen um SOA-Mechanismen. Traditionelle Anwendungen als Ganzes und/oder ihre einzelnen Komponenten müssen um Provider- und Consumer-Funktionalitäten erweitert werden, um die angestrebte flexible, transparente und adaptive Interaktion mit anderen Entitäten zu erzielen. Dabei ist es hilfreich, dass viele Anwendungen bereits SOA-Funktionlität mitbringen (z. B. soziale Netze, Lehr- und Lernmanagementsysteme und Blogs, die bereits Web Services anbieten). Die in diesem Kapitel aufgeführten Fallstudien demonstrieren die erforderliche SOA-Erweiterung für wichtige Anwendungen einer modernen Hochschule wie der Universität Rostock.

5.1.3 Ausgangssituation der Universität Rostock

Die Rostocker Universität ist ein Beispiel für eine Bildungseinrichtung auf dem Weg zur Pervasive University. Indem sie als erste europäische Universität eine campusweite WLAN-Versorgung einführte, vollzog sie den Wandel von der IT-basierten Universität zur Notebook-Universität [Tavangarian 01].

Durch die breite Verfügbarkeit von internetfähigen Multimedia-Arbeitsplätzen, eines hochschulweit eingesetzten Lehr- und Lernmanagementsystems (Stud.IP [data-quest GmbH 05]) und weitreichenden Erfahrungen mit dem Im- und Export von aufgezeichneten Lehrveranstaltungen ist orts- und zeitunabhängiges Lehren und Lernen an der Universität Rostock Alltag. Zahlreiche Projekte mit dem Schwerpunkt auf Infrastrukturen für modernes E-Learning schaffen außerdem den idealen Nährboden für innovative Infrastrukturen wie der Service-orientieren Universität.

Die im folgenden aufgeführten Fallstudien und Implementierungen wurden daher an den Universität Rostock unter Berücksichtigung der lokalen Bedingungen und Möglichkeiten realisiert.

5.2 Service-orientierte Lehr- und Lernarrangements

Die in diesem Abschnitt aufgeführten Fallstudien und Implementierungen verdeutlichen wie Basisdienste einer Universität als Services im Rahmen einer SOA umgesetzt werden können. Nach der beispielhaften Erläuterung der Service-Implementierung eines Basisdienstes (Druckdienst) werden mit den Fallstudien *Lecture as a Service* und *Immersive Learning* auch zwei komplexere Service-basierte Lehr- und Lernarragements betrachtet.

5.2.1 Basis-Services

Die Anbindung vorhandener universitärer Dienste an eine Service-orientierte Architektur erfolgt auf Provider-Seite nach folgendem generellen Vorgehen auf Providerseite:

1. Auswahl einer SOA-Implementierung für die geplante Anbindung
2. Zerlegung des Dienstes in Teildienste (falls praktikabel)
3. Auswahl einer Provider-Instanz für jeden Teildienst
4. Implementierung von Services für jeden Teildienst
5. Deployment der implementierten Services

In den weiteren Fallstudien wird dieses Vorgehen fallspezifisch konkretisiert, soweit die Studie so komplex ist dass eine derartige Beschreibung unverhältnismäßig umfangreich wird.

Auf Consumer-Seite lässt sich kein generelles Vorgehen erkennen. Die bestehenden Anwendungen müssen um eine Consumer-Schnittstelle erweitert werden. Der Aufwand und das Vorgehen bei dieser Erweiterung bzw. Modifikation ist stark von der gewählten SOA-Implementierung sowie der konkreten Anwendung abhängig und kann daher nicht generalisiert werden.

Im Rahmen der vorliegenden Arbeit wurde in einem studentischen Projekt beispielhaft der Zugriff auf einen handelsüblichen Drucker als universitärer Basis-Service implementiert. Der in dem Projekt entstandene Prototyp wird im folgenden kurz als Referenzimplementierung betrachtet.

Angestrebt wurde ein Druck-Service, der es Consumern ermöglicht Dokumente auf einen herkömmlichen Drucker auszudrucken ohne über Vorwissen (z. B. Netzwerkadresse des Druckers) und Druckerspezifische Software (z. B. Druckertreiber) zu verfügen. Das Vorgehen zur SOA-Anbindung auf Provider-Seite konkretisiert das generelle Vorgehen:

1. Als SOA-Implementierung für die Anbindung wurde Jini gewählt, da die Benutzungsoberfläche auf dem Druck-Client (Consumer) mit Java einfach zu implementieren war und die Studenten zudem bereits Java-Erfahrungen besaßen.

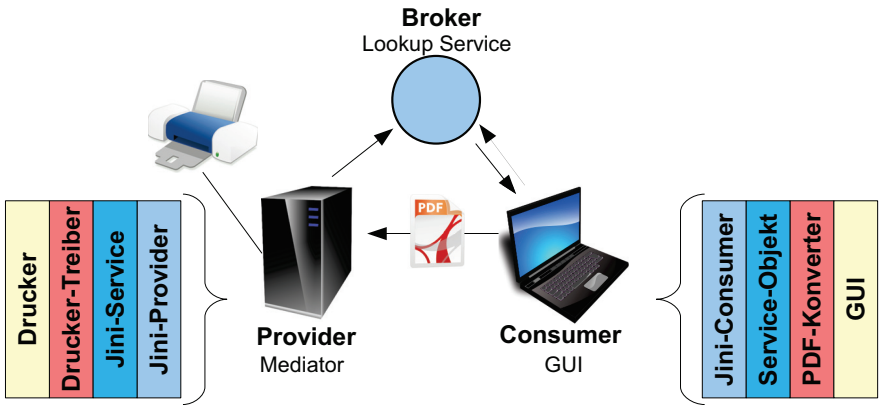


Abbildung 5.5: SOA-Anbindung am Beispiel eines Druckers als Jini-Service.

2. Eine Zerlegung des bereits sehr einfachen Druckdienstes in weitere Teildienste war nicht erforderlich.

3. Der Drucker selbst konnte nicht als Provider des Services fungieren, da es zu aufwendig gewesen wäre, dessen herstellerspezifische Firmware, um SOA-Funktionalitäten zu erweitern. Daher wurde der in Abschnitt 5.1.2 erläuterte Ansatz eines SOA-Mediators verfolgt, für den der Drucker an einem PC (Mediator) angeschlossen wurde. Der netzwerkfähige Mediator konnte den Drucker über seinen Treiber ansprechen und war aus SOA-Sicht der Provider des angestrebten Drucker-Services.

4. Der Jini-Service konnte über die Java-VM auf den Drucker-Treiber zugreifen und somit den Drucker ansprechen. Der eigentliche Service wurde so implementiert, dass er eine Dokument im verbreiteten PDF-Format erwartet und dieses dann auf dem Drucker ausgibt.

5. Der Jini-Service wurde über die lokale Jini-Software wie in Abschnitt 2.2.3 beschrieben beim Lookup-Service registriert und konnte somit von möglichen Consumern gefunden werden.

Wie Abbildung 5.5 illustriert, benötigt der Druck-Client als Consumer ebenfalls ein lokales Jini-System. Über eine extra zu diesem Zweck entwickelte Benutzungsoberfläche kann der Client einen der verfügbaren Drucker und sein zu druckendes Dokument auswählen. Dieses wird ggf. in ein PDF-Dokument umgewandelt und an den Service bzw. den Drucker gesendet. Für diesen Prototyp ist auf Consumer-Seite neben der entwickelten Software kein weiterer Treiber notwendig und verfügbare Drucker-Services können ohne weiteres Vorwissen gefunden und genutzt werden. Dies sind klare Vorteile gegenüber herkömmlichen Print-Mechanismen wie etwa denen der Windows-Betriebssysteme. Während der Aufwand bei einer geringen Druckeranzahl noch hoch erscheinen mag, skaliert er wiederum gut bei einer großen Druckeranzahl, wie sie an einer Universität vorliegt, zumal ein Mediator für mehrere Drucker zuständig sein kann.

Der Prototyp verdeutlicht beispielhaft, wie einfache Basisdienste als Services in eine SOA integriert werden. Neben diesem Beispiel sind weitere grundlegende Services entstanden, wie etwa ein Messaging-Service, über den dem Provider (z. B. Mobiltelefon eines Studenten) eine einfache Textnachricht gesendet werden kann. Derart einfache Services verdeutlichen die grundlegende Vorgehensweise bei der "SOAfizierung" einer Infrastruktur. Das wahre SOA-Potential offenbart sich jedoch erst bei der Einbettung dieser Services in komplexere Anwendungen. So ist beispielsweise der Messaging-Service Teil eines größeren *Immersive Messaging*-Lösung zur individuellen Kommunikation zwischen heterogenen Lehr-/Lernumgebungen und wird in Abschnitt 5.3.2 näher erläutert. Im Folgenden wird mit den Fallstudien *Lecture as a Service* und *Immersive Learning* die Implementierung einer komplexen SOA für konkrete Anwendungen moderner Hochschulen tiefer thematisiert.

5.2.2 Fallstudie: Lecture as a Service

Neben der Forschung ist die Ausbildung das wichtigste Standbein jeder Universität. Während die Didaktik der universitären Lehre je nach Lehrinhalten, Lernfortschritt, indiviuellen Vorlieben und anderen Kriterien unterschiedlich ausgeprägt ist, sind Vorlesungen dennoch weiterhin ein essentieller Bestandteil der Wissensvermittlung. An modernen Universitäten werden diese durch digitalisierte Lehr- und Lernmedien bereichert (z. B. Powerpoint-Folien, Videos sowie digitalisierte Skripte und Literatur). Oftmals werden diese Medien den Studenten zur individuellen Vor- und Nachbereitung einer Vorlesung als Download im Rahmen einer dedizierten Webseite oder vorlesungsübergreifenden Lehr- und Lernmanagementsystemen (LLMS) wie Stud.IP, Moodle oder ILIAS angeboten. Insbesondere LLMS sind inzwischen weit verbreitet, da sie zusätzlich Werkzeuge zur Organisation und Kooperation von Lehrenden und Lernenden bieten. Neben den vorlesungsergänzenden Medien kann heute auch die Vorlesung selbst durch entsprechende Hard- und Software als Video aufgezeichnet, für das Internet aufbereitet und online konsumiert werden. Derart aufbereitete Inhalte sind in Kombination mit LLMS die Grundlage moderner Fernstudiumslösungen wie dem Juniorstudium der Universität Rostock [Thomanek 09]. Leider ist ihre Einbindung in LLMS und andere Portale ein aufwendiger und auf Einzellösungen basierender Prozess. Eine systematische Lösung würde nicht nur diesen Aufwand minimieren, sondern auch die dynamische Einbindung hochschulübergreifender Vorlesungsaufzeichnungen ermöglichen. Medienbrüche würden reduziert, der Lernprozess somit vereinfacht und dadurch der Lernerfolg gesteigert. Weiterhin fördert eine derartige Lösung die in Abschnitt 1.2 beschriebene Mobilität heutiger Studenten sowie die Nutzung hochwertiger Inhalte anderer Hochschulen für die lokale Lehre.

Im Rahmen dieser Arbeit wurde daher die prototypische Nutzung Service-orientierter Architekturen zur Integration von Vorlesungsaufzeichnungen und -Übertragungen in LLMS und andere Lernumgebungen untersucht. Die Fallstudie *Lecture as a Service* zeigt ein Ergebnis dieser Untersuchungen. Durch eine moderne Umgebung für Präsenzlehre werden dort stattfindene Vorlesungen sowohl als Live-Stream (synchron)

1. Kameras
2. Leinwände und Projektoren
3. Mikrofone
4. Lautsprecher
5. Laptop / PC
6. Mobile Mediensteuerung
7. Zusätzliches Multimedia-Equipment

Abbildung 5.6: Die E-Learning-Werkstatt des Rostocker Lehrstuhls für Rechnerarchitektur bietet mit ihrem großen Gerätespektrum hervorragende Bedingungen für moderne Lehr- und Lernarrangements.

als auch als Aufzeichnung (asynchron) aufbereitet und als Service angeboten. Diese wurden wiederum exemplarisch in einem um SOA-Consumer-Funktionalität erweiterten Stud.IP eingebunden.

Abbildung 5.6 zeigt die für diese Fallstudie genutzte Präsenzumgebung, die E-Learning-Werstatt des Lehrstuhls für Rechnerarchitektur der Universität Rostock, mitsamt ihres breiten Gerätespektrums. Zur Aufzeichnung von Vorlesungsbestandteilen stehen verschiedene Kanäle zur Verfügung:

- Videokameras (zeichnen Videos des Vortragenen bzw. des Publikums auf)
- Mikrofone (zeichnen Sprache des Vortragenden bzw. Fragen oder Kommentare aus dem Publikum auf)
- PC (zeichnet seinen Bildschirminhalt auf, z. B. Powerpoint-Folien oder Videos)

Weiterhin gibt es diverse Kanäle zur Medienwiedergabe (z. B. Beamer und Lautsprecher). Zusätzlich kann ein Streaming-Server für die Zusammenführung von Video und Audio sowie den Export des resultierenden Multimedia-Streams genutzt werden. Eine zentrale Mediensteuerung ist außerdem für die Vermittlung zwischen Datenquellen und -senken sowie weitere Umgebungssteuerung zuständig. Basierend auf dieser Umgebung und dem Ziel, der Integration von Lehrmedien in LLMS, wurde folgendes Vorgehen verfolgt:

1. Die Wahl der SOA-Implementierung fiel auf Web Services, da deren Nuzung durch ein web-basiertes LLMS wie Stud.IP den geringsten Aufwand erzeugt, zumal Stud.IP selbst Provider diverser Web Services ist.
2. Als Teildienste wurden der Zugriff auf Live- und aufgezeichnete Multimedia-Streams (Video und Audio des Vortragenden und des Publikums sowie Bildschirminhalt) sowie eine Bedienung der Mediensteuerung zur Zuweisung der Datenquellen des Streams identifiziert.

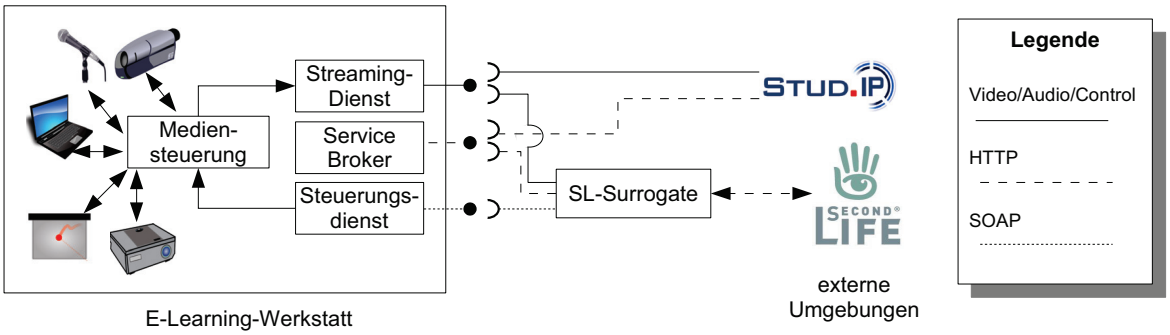


Abbildung 5.7: Grobe Architekturübersicht für die SOA-basierte Verbindung der E-Learning-Werkstatt mir externen Umgebungen wie Stud.IP und Second Life

3. Der Streaming-Server konnte als Provider der Streaming-Services eingesetzt werden und die Mediensteuerung bot den Zugriff auf die Medientechnik als Service an.
4. Während die Live-Streaming-Services im Quicktime-Format und die Aufzeichnungen als HTML-Komposition (Lecturnity-Format [IMC AG 09]) angeboten wurden, diente eine SOAP-Schnittstelle als Service-Interface für die Mediensteuerung.
5. Die implementierten Services wurden über einen WSIL-Broker bubliziert und standen somit externen Consumern zur Verfügung.

Abbildung 5.7 stellt die Grobarchitektur dar, die somit im Rahmen dieser Fallstudie realisiert wurde. Als externe Umgebung wird neben Stud.IP auch die virtuelle Welt Second Life dargestellt, diese Anbindung wird im nächsten Abschnitt diskutiert.

Stud.IP wurde als SOA-Consumer erweitert, so dass die im WSIL-Broker registrierten Streaming-Services zu einer konkreten Lehrveranstaltung automatisch eingebunden werden konnten [Gläser 08]. Wie in Abbildung 5.8 illustiert, wurde der *Dateien*-Bereich einer Lehrveranstaltung zum Bereich *Medien* ausgebaut (1). Wählt der Nutzer diese Sektion, wird nach vorhandenen Streams gesucht, die mit dieser Lehrveranstaltung assoziiert wurden. Als Ergebnis dieser Suche erhält er eine Auflistung der aufgezeichneten (2) und live übertragenden Streams. Nach Auswahl eines Streams kann dieser konsumiert werden (4).

Diese Fallstudie zeigt, wie durch eine – im Vergleich zum Drucker-Service – komplexeren SOA ein grundlegender Bestandteil universitärer Ausbildung flexibel in ein LLMS eingebunden werden kann. Das lokale Lehr- und Lernarrangement der Präsenzlehre (E-Learning-Werkstatt) wird somit um einen virtuellen Bestandteil erweitert. Während das erweiterte Lehr- und Lernarrangement zur unidirektionalen Einbindung entfernte Teilnehmer einer Lehrveranstaltung genutzt werden kann, wird das Potential eines SOA-basierten Arrangement erst durch bidirektionale Ansätze deutlich. Die folgende Fallstudie behandelt exemplarisch einen derartigen Ansatz, der zur infrastrukturellen Realisierung einee neuen E-Learning-Paradigmas beiträgt: *Immersive Learning*.

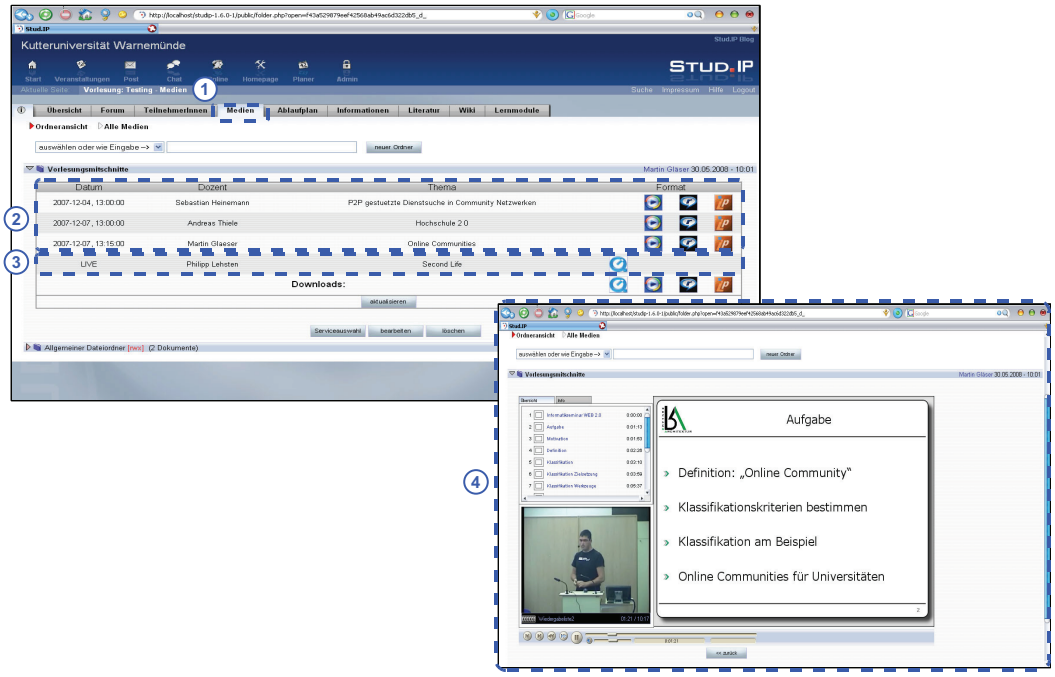


Abbildung 5.8: Die als Services angebotenen Vorlesungen können dnamisch in das Lehr- und Lernmanagementsytem Stud.IP eingbunden werden.

5.2.3 Fallstudie: Immersive Learning

Neben der klassischen rechnergestützten Präsenzlehre, bei der Lehrende und Lernende zum Zeitpunkt der E-Learning-Veranstaltung am selben Ort sind, hat sich mit der Online-Lehre auch die Verwendung des Internets zur Überbrückung räumlicher Entfernungen durchgesetzt. So können beispielsweise Kollaborationsplattformen (z. B. Shared Whiteboard, Chat-Systeme und Videokonferenzen) für synchrone Lehr- und Lernszenarien eingesetzt werden. Auch asynchrone Szenarien, in denen die Lehrinhalte zeitunabhängig konsumiert werden, haben sich dank LLMS und weiteren Systemen zur Inhaltsarchivierung inzwischen ethabliert. Eine besondere Form der Online-Lehre hat im vergangenen Jahrzehnt durch die öffentliche Verfügbarkeit virtueller Welten (allen voran Second Life) an Bedeutung gewonnen: *Virtual Learning*.

Auch wenn virtuelle Lehrumgebungen (*Virtual Learning Enwironments*, VLE) [Totkov 03] bereits vor öffentlichen virtuellen Welten in der Lehre eingesetzt wurden, hat deren freie und breite Verfügbarkeit zu einem regelrechten Virtual Learning-Hype geführt. In dieser Arbeit wird das Potential und der Einsatz derartiger Welten am Beispiel Second Life, als verbreitetste virtuelle Welt, verdeutlicht. Nutzer bewegen sich in Second Life über sogenannte *Avatare*, virtuelle Repräsentaten über die mit anderen Nutzern interagiert und die virtuelle Welt gestaltet werden kann. Durch die Avatare wird weiterhin eine soziale Präsenz und somit ein emfundenes "Eintauchen"(Immersion) in die virtuelle Umgebung erzielt.

Während Lernende vor allem ihr eigener Spieltrieb und der Neuwert an der Teilnahme virtueller Lehrveranstaltungen motiviert, ist ist aus Sicht der Lehrenden die Zusammenführung von weltweit verteilten Lernenden ein wichtiger Vorteil [Hainsworth 08]. Daher ist es kaum verwunderlich, dass – trotz der Vielseitigkeit virtueller Welten – oftmals Szenarien aus der Präsenzlehre nachgebildet werden, die sich lediglich durch die räumlichen Verteilung der Teilnehmer von diesen unterscheiden. So gibt es eine Vielzahl virtueller Hörsääle und Seminarräume, in denen sich die Avatare von Lehrenden und Lernenden zu synchronen Lehrveranstaltungen zusammenfinden. Doch gerade die (teils konstruktive) Erschließung von Lehrinhalten (z. B. durch virtuelle 3D-Modelle) hat hohes Potential den individuellen Lernprozess zu unterstützen.

Bereits heute sind beim Second Life-Anbieter Linden Lab rund 50 Lehrveranstaltungen pro Tag registriert. Die wahre Anzahl lässt sich leider kaum abschätzen, da die meisten Veranstaltunen keineswegs offiziell registriert werden. Allein die Tatsache, dass fast alle namhaften Bildungseinrichtungen eine virtuelle Präsenz mit Hörsaal und/oder Seminarraum besitzen, lässt jedoch eine große Anzahl virtueller Lehrveranstaltungen vermuten.

Sowohl rechnergestützte Präsenzlehre als auch Virtual Learning haben jeweils Vor- und Nachteile. Aus den in Tabelle 5.2.3 ausgeführten Charakteristika der beiden Paradigmen, ist vor allem die Ortsgebundenheit der Präsenzlehre und die oftmals komplexe Einrichtung virtueller Lehrveranstaltungen und nicht-triviale Teilnahme an diesen hervorzuheben. Die Virtual Learning-Nachteile resultieren vor allem aus der Neuheit des Mediums und dem daraus entstehenden Einarbeitungsaufwand für alle Teilnehmer.

rechnergestützte Präsenzlehre	**Virtual Learning**
ortsgebunden	ortsunabhängig
synchron	synchron/asynchron
erweiterte Lehrmaterialen (z. B. Folien und Multimedia)	neue Lehrmaterialien (z.B. virtuelle 3D-Modelle)
einfache Einrichtung und Teilnahme (keine besonderen Kenntnisse nötig)	komplexe Einrichtung und Teilnahme (Erfahrungen mit virtueller Welt nötig)

Tabelle 5.1: Vergleich rechnergestützter Präsenzlehre und des Virtual Learning

Durch die genannten Vor- und Nachteile sowie die Analogien zwischen Lehr- und Lernszenarien der Präsenzlehre und des Virtual Learning lag im Rahmen dieser Arbeit eine systematische Kombination beider Paradigmen zur gegenseitigen Bereicherung nahe: *Immersive Learning* [Zender 09b]

Definition 9 ***Immersive Learning*** *bezeichnet die zielgerichtete und adaptive Verschmelzung von virtueller und Präsenzlehre. Lehrende und Lernende nehmen dabei vor Ort in einer Präsenzlernumgebung oder über eine virtuelle Lehrumgebung teil, die miteinander gekoppelt sind. Das Lehrmaterial kommt entweder aus der virtuellen oder der Präsenzlernumgebung.*

Ein Lehr-/Lernarragement des Immersive Learnings beinhaltet demnach sowohl Umgebungen der Präsenzlehre als auch virtuelle Lehr-/Lernumgebungen. Die Umgebungen der Präsenzlehre werden um die Integration entfernter Teilnehmer und die Integration virtueller Lehrmaterialen (z.B. virtuelle 3D-Modelle von Lehrobjekten) bereichert. Im Gegenzug können virtuelle Lehrveranstaltungen um Teilnehmer einer Präsenzlehrumgebung erweitert werden (die ohne technische Hürden wie gewohnt eine Veranstaltung der rechnergestützten Präsenzlehre besuchen) und virtuelle Teilnehmer an realen Lehrveranstaltungen teilnehmen. Im Vergleich zu den beiden ursprünglichen Paradigmen hat Immersive Learning somit folgende Charakteristika:

- ortsunabhängig (durch Integration lokaler und entfernter Teilnehmer)
- wahlweise synchron oder asynchron (da Live-Lehrveranstaltungen sowie die zeitunabhängige Arbeit mit virtuellen Inhalten wie etwa 3D-Modellen möglich sind)
- Nutzung von erweiterten und neuen Lehrmaterialien (aufgrund der Einbeziehung beider Lehr-/Lernumgebungen)
- implizit einfache Einrichtung und Teilnahme (da Teilnehmer aus der virtuellen Welt mit dieser umgehen können, während Präsenzteilnehmer diese Hürde nicht überwinden müssen)

Die Menge der genutzten Lehrmedien setzt sich aus den Lehrmedien der virtuellen und der Präsenzlehre zusammen. Sie umfasst beispielsweise:

- Videobild und Sprache realer Lehrender und Lernender (live und aufgezeichnet)
- Avatarbild und Sprache virtueller Lehrender und Lernender (live und aufgezeichnet)
- online verfügbare Folien und Skripte
- online verfügbare Multimediainhalte (z. B. Lehrvideos)
- individuelle Kommunikations- und Kollaborationsmedien (z. B. Mobiltelefon, virtueller Chat, E-Mail)
- virtuelle 3D-Modelle von Lernobjekten

Die Vielfältigkeit und Dynamik dieser Medien erweckt den Bedarf nach einer transparenten und flexiblen Infrastruktur. Daher wurde, wie bereits in Abbildung 5.7 dargestellt, eine Service-orientierten Architektur zur prototypischen Medienintegration in die virtuelle oder Präsenzumgebung genutzt. Während auch hier die E-Learning-Werkstatt mitsamt ihrer exportierten Services als Präsenzumgebung diente, wurde die virtuelle Welt Second Life als virtuelle Lehr- und Lernumgebung eingesetzt. Folgendes Vorgehen diente der erweiterung der bereits behandelten Architektur und der Realisierung der Immersive Learning-Infrastruktur:

1. Da die E-Learning-Werkstatt bereits im Rahmen der *Lecture as a Service*-Fallstudie und der zuvor beschriebenen Implementierung über Web Services an-

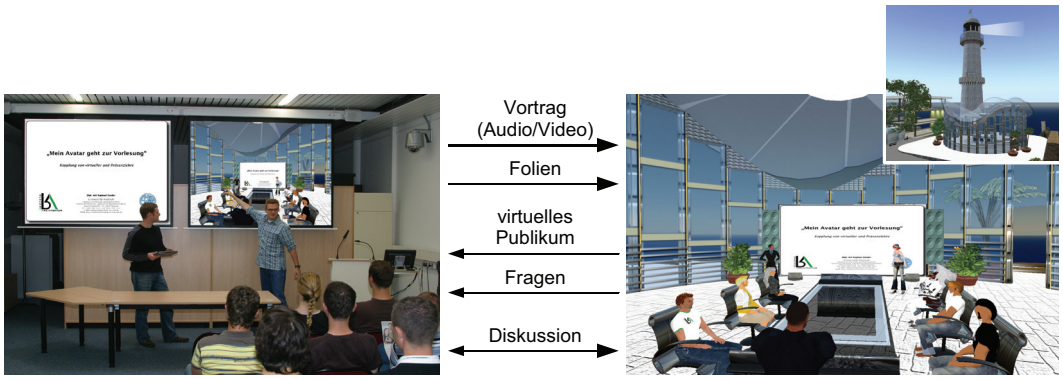


Abbildung 5.9: In diesem Immersive Learning-Szenario wird eine Lehrveranstaltung aus der E-Learning-Werkstatt in die virtuelle Welt Second Life eingebunden.

gebunden wurde, fiel auch bei dieser Fallstudie die Wahl auf Web Services als SOA-Implementierung.

2. Die bereits aus der E-Learning-Werkstatt angebotenen Services wurden genutzt, um Multimedia aus der Werkstatt in Second Life einzubinden. In der Gegenrichtung wurde die Einbindung der Multimedia-Daten aus Second Life in die E-Learning-Werkstatt zunächst nur über den kostenlosen Second Life-Client realisiert. Eine SOA-basierte Multimedia-Einbindung erschien im Rahmen dieser Fallstudie als nicht praktikabel, da der Client bereits über einen ausreichenden Funktionsumfang verfügt. Im Sinne der Systematik ist jedoch eine SOA-basierte Multimedia-Anbindung in Zukunft erstrebenswert. Eine bidirektionale SOA exisietert dennoch, indem aus Second Life Services zum Nachrichtenversand an Avatare exportiert wurden. Diese Implementierung wird im Rahmen der *Immersive Messaging*-Fallstudie in Abschnitt 5.3.2 beschrieben.

3. - 5. Als Provider dienten die bereits beschriebenen Instanzen aus der E-Learning-Werkstatt. Auch deren Implementierung und Deployment wurde bereits erläutert.

Die besondere Herausforderung dieser Fallstudie stellte die Consumer-Seite dar, denn Second Life ist wie viele andere virtuelle Welten nicht nativ SOA-fähig. Daher wurde im Rahmen dieser Arbeit ein Mediator entwickelt, der per HTTP mit Second Life kommuniziert und als Web Service-Consumer dient (für andere Szenarien gleichzeitig als Web Service-Provider) [Zender 09b]. Dieser sogenannte *SL-Surrogate* ist weiterhin in der Lage, den bereits beschriebenen WSIL-Broker per HTTP anzufragen und so Informationen über verfügbare Services zu erhalten. Für jeden Service, den der SL-Surrogate als Consumer oder Provider bedient, gibt es ein eigenes Surrogate-Plugin, dass die SOA-Interaktion auf die SL-Interaktion abbildet und umgekehrt. So wird beispielsweise die Adresse eines als Web Service angebotenen Videostream einfach als HTTP-Nachricht an SL-Objekte weitergeleitet, die den Stream wiedergeben können. Ein Beispiel für die Gegenrichtung ist in Abschnitt 5.3.2 beschrieben.

Anhand des in Abbildung 5.9 illustrierten Anwendungsszenarios wird im folgenden das Potential des Immersive Learning verdeutlicht. In dem Szenario findet eine Vorlesung in der E-Learning-Werkstatt statt und einige Teilnehmer nehmen über Second Life an dieser Veranstaltung teil. Aus der E-Learning-Werkstatt werden über den beschrieben Protoypen Video und Audiosignale angeboten (z. B. vom Vortragenden oder dem Publikum). Ferner wird auch Lehrmaterial wie die Vortragsfolien als Streaming-Services übertragen. Das virtuelle Publikum kann aus den (über den SL-Surrogate in Second Life angebotenen) Streams denjenigen auswählen, den es für sinnvoll erachtet. Während der Vorlesung ist beispielsweise die Anzeige der Folien sinnvoll. Während einer Diskussion kann könnte stattdessen das Video des Vortragenden oder das Publikum gewählt werden. Das Audio-Signal des Vortragenden wir auf seinen Avatar übertragen, um die gefühlte soziale Präsenz zu steigern. In der Gegenrichtung wird durch die Nutzung des Second Life-Clients in der E-Learning-Werkstatt das virtuelle Publikum angezeigt. Über den Second Life-Voice Chat kann dieses sich auch aktiv an der Vorlesung beteiligen (z. B. bei einer Diskussion).

Diese SOA-basierte Kopplung von Szenarien der Präsenzlehre mit der virtuellen Lehre geht weit über bisherige Ansätze hinaus. So gibt es bereits Vernetzungen zwischen Second Life und traditionellen LLMS [Kemp 06], die sich jedoch auf das Setzen von Querverweisen (Links) bzw. eine gemeinsame Datenbank beschränken. Es handelt sich dabei um keinen systematischen Ansatz zur Verzahnung von synchronen und asynchronen Prozessen beider Lernwelten. Auch die Durchführung von Lehrveranstaltungen im Second Life wird schon intensiv praktiziert [Müller 07]. Allerdings handelt es sich hierbei um Veranstaltungen, die explizit für diese virtuelle Umgebung konzipiert und durchgeführt werden.

Dennoch kann der Prototyp noch an einigen Stellen erweitert und verbessert werden. So fehlen derzeit noch SOA-basierte Rückkanäle aus der virtuellen in die Präsenzumgebung. Diese sind noch nicht zwingend notwendig – der Second Life-Client erfüllt diese Aufgabe zufriedenstellend – aber im Sinne erhöhter Flexibilität und Systematik wünschenswert. Auch eine unmittelbare Integration von SOA-Mechanismen in virtuelle Welten wie Second Life würde die Flexibilität des Ansatzes deutlich steigern, ebenso wie eine Erweiterung der virtuellen Umgebungen um weitere Medien (z. B. HTML und PDF).

Die in diesem Abschnitt aufgeführten Fallstudien und Implementierungen verdeutlichen das generelle Vorgehen sowie fallspezifische Besonderheiten bei der SOA-Integration von Basisdiensten und komplexeren Service-basierten Lehr- und Lernarragements. Das Fernziel dieser Bemühungen ist, dass nicht nur Lehr- und Lernmedien als Services angeboten werden, sondern letzlich die gesamte IT-Infrastruktur aus Lehre, Forschung und Verwaltung auf Service-orientierte Architekturen umgestellt werden. Dafür ist die Konzentration auf einzelne SOA-Implementierungen jedoch nicht zielführend, da die Komponentenvielfalt dieser Infrastruktur auch eine SOA-Vielfalt mitbringt, die SOA-Interoperabilitätsstrategien erfordert. Basierend auf den in den vergangenen Ka-

piteln dieser Arbeit beschriebenen Ergebnissen werden daher im Folgenden SOA-übergreifende Lehr- und Lernarrangements diskutiert.

5.3 SOA-übergreifendes Lehr- und Lernarrangement

Analog zu anderen pervasiven Umgebungen setzen sich heutige Lehr- und Lernarragements aus einer Vielzahl unterschiedlicher Geräteklassen und somit auch SOA-Implementierungen zusammen. Es ist keine Seltenheit, dass Studenten und Dozenten individuelle Mobilgeräte wie Mobiltelefone, Smartphones oder Laptops zum Lernen und Lehren nutzen. Auch die neue Generation hochmobiler Tablet-Computer wie das *iPad* ist prädestiniert für mobile Lehr- und Lernszenarien (z. B. als Lesegerät für elektronische Bücher). Durch leistungsfähige Rechentechnik sowie diverse Kontext-Schnittstellen (z. B. GPS und Bewegungssensorik) ist sie auch ideal als Komponente pervasiver Umgebungen. Zusätzlich zu den Nutzerendgeräten ist die Interaktion mit festinstallierter Technik moderner Seminarräume sowie der restlichen universitären Infrastruktur essentiell für pervasive Lehr- und Lernarragements. Die bisher erläuterten SOA-basierten Arragements, die durch die Verwendung der gleichen SOA-Implementierung auf Provider- und Consumer-Seite homogen sind, können somit nicht generell an modernen Hochschulen erwartet werden. Daher fokussiert dieser Abschnitt SOA-übergreifende Szenarien am Beispiel der folgenden Zusammenführung zweier Druck-Services und der anschließenden Fallstudie *Immersive Messaging*.

5.3.1 Zentrale Vereinheitlichung ohne STiL

Vor der Konzeption und Implementierung der Service Technology-independent Language wurde die Brauchbarkeit vorhandener Formate am Beispiel von Service-Beschreibungen in einem UDDI-Verzeichnis (siehe Abschnitt 2.2.1) untersucht [Machuska 07]. Als Service-Typ wurde erneut ein Druck-Service gewählt.

Wie Abbildung 5.10 illustriert, wurden Beschreibungen von Druck-Services der SOA-Implementierungen Jini und DNS-SD über einen zentralen Controller gesammelt und in das standardisierte Service-Beschreibungsformat eines UDDI-Verzeichnisses transformiert. Über eine grafische Benutzungsschnittstelle konnten die erzeugten UDDI-Beschreibungen angezeigt und somit der Transformationserfolg überwacht werden.

Das Ergebnis dieser Untersuchung war zwar eine erfolgreiche Transformation der jeweiligen Services, jedoch unter erheblichen Performance-Kosten aufgrund der Komplexität einer UDDI-Beschreibung sowie der standardisierten UDDI-Schnittstelle. So benötigte beispielsweise allein die Transformation einer Jini- oder DNS-SD-Beschreibung in das UDDI-Format durchschnittlich 8 Sekunden, was neben der vorhergehenden Abfrage der Services und dem nachfolgenden Eintrag in das Verzeichnis ca. 80% der Gesamtzeit entsprach. Dieses Ergebnis ist jedoch nicht verwunderlich, da UDDI sowie der gesamte

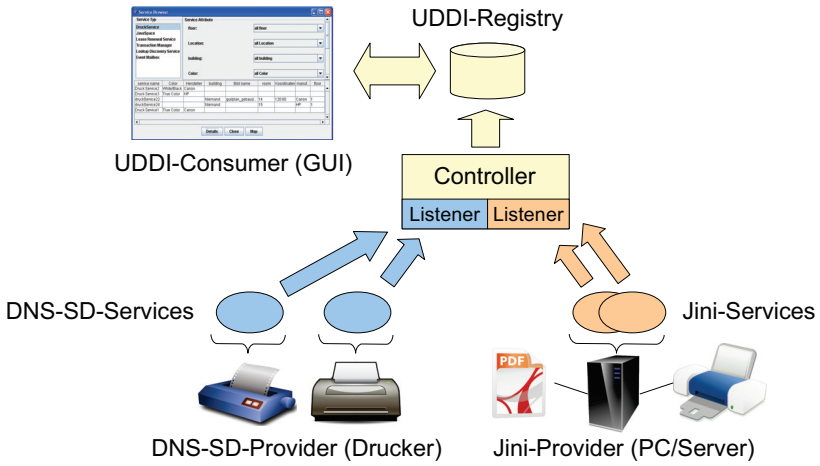


Abbildung 5.10: Druck-Service-Beschreibungen unterschiedlicher SOA-Implementierungen wurden in einem UDDI-Verzeichnis vereinheitlicht.

Web Service-Stack nicht auf hochdynamische SOAs ausgerichtet ist (im Gegensatz zu Jini oder DNS-SD) und somit nicht auf zeiteffiziente Funktionalität optimiert wurde.

Mit der Entwicklung des deutlich kompakteren STiL-Formates sowie der Beschreibungstransformation über das einfache Füllen bzw. Auslesen von XML-Dateien wurde eine deutlich bessere Performance erreicht. Die Transformation zwischen konkreten Service-Beschreibungen und der abstrakte Sprache war von der konkreten SOA-Implementierung sowie der Service-Beschreibung abhängig, lag jedoch für gewöhnlich unter einer Sekunde (z. B. zwischen 200 und 300ms in [Lenz 09]).

Die folgende Fallstudie betrachtet STiL-basierte SOA-Interoperabilität in dem komplexeren Lehr- und Lernarragement des Immersive Learning. Dieses wird über eine umgebungsübergreifende Lösung zur individuellen Kommunikation erweitert und somit um ein rudimentäres Werkzeug zur Kollaboration bereichert.

5.3.2 Fallstudie: Immersive Messaging

Das in Abschnitt 5.2.3 beschriebene und durch die Kopplung einer Präsenz- und einer virtuellen Lehr-/Lernumgebung realisierte Szenario ermöglicht die Durchführung einer umgebungsübergreifenden Vorlesung. Daneben beinhaltet eine Lehrveranstaltung jedoch auch deren – oftmals kollaborative – Vor- und Nachbereitung sowie Kommunikation zwischen Lernenden während der Veranstaltung selbst (z. B. für Phasen der Gruppenarbeit). Um diese auf einfache, aber effektive Weise zu unterstützen, wurde der vorgestellte Prototyp um einen bidirektionalen Versand von Textnachrichten zwischen den lokalen und virtuellen Teilnehmern ergänzt.

Im Rahmen dieser Fallstudie sollte transparente Individualkommunikation mit den Werkzeugen ermöglicht werden, die den Lernenden in den jeweiligen Umgebungen ohnehin zur Verfügung standen. In der Präsenzumgebung fiel die Wahl auf die bereits

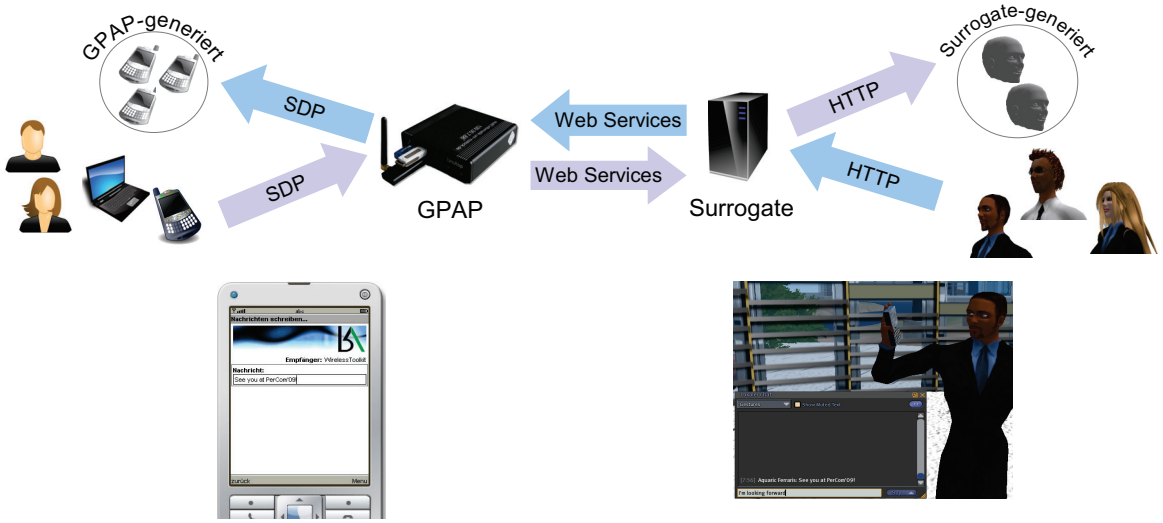


Abbildung 5.11: Durch eine GPAP-basierte Übersetzung zwischen den SOA-Implementierungen Web Services und SDP ist eine transparente Individualkommunikation zwischen unterschiedlichen Immersive Learning-Umgebungen möglich.

allgegenwärtigen Mobiltelefone der Anwesenden. Als Netzwerkschnittstelle wurde Bluetooth gewählt, da durch Bluetooth-basierte Kommunikation keine zusätzlichen Verbindungskosten für die Kommunikationspartner entstehen. Auch wenn viele Netzbetreiber inzwischen attraktive Datentarife für die internetbasierte Kommunikation anbieten und viele Geräte bereits WLAN-fähig sind, können diese Möglichkeiten noch nicht bei den Lernenden vorrausgesetzt werden. Weiterhin unterstützen auch ältere Mobiltelefone schon die Bluetooth-basierte Kommunikation, wodurch die angestrebte Lösung nicht ausschließlich für finanzkräftige Studenten anwendbar ist.

Auf der Seite des Second Life stand neben einem simplen Text-Chat auch die Kommunikation via E-Mail oer Voice-Chat zur Verfügung. Im Sinne der Einfachheit wurde für diese rudimentäre Lösung zunächst der Text-Chat als Kommunikationskanal gewählt. Andere Lösungen sind auf ähnliche Weise realisierbar. Die Aktivierung eines sogenannten Listeners, der den gesendeten Text an den Surrogate übertragen muss, erfolgt dabei über ein virtuelles 3D-Objekt, welches dem Avatar angehängt wird (z. B. virtuelles Mobiltelefon).

Durch die Auswahl von Bluetooth auf der einen Seite und die Nutzung des entwickelten IP-basierten Surrogates auf der anderen Seite wurden neben unterschiedlichen Netzwerktechnologien auch die jeweiligen SOA-Implementierungen SDP und Web Services implizit für die Kommunikationspartner festgelegt. Somit ist der GPAP zur Interoperabilität auf Netzwerk- sowie Service-Ebene fester Bestandteil der Lösung. Abbildung 5.11 zeigt einen Überblick über das Gesamtsystem und das Zusammenspiel der einzelnen Komponenten, welches im Rahmen dieser Dissertation konzipiert und prototypisch implementiert wurde [Zender 09b].

In der Präsenzumgebung bieten Mobiltelefone ihre Fähigkeit Textnachrichten zu empfangen als SDP-Services im Bluetooth-Netz an. Der GPAP erfasst deren Service-Beschreibungen, generiert entsprechende STiL-Beschreibungen sowie konkretisierte Beschreibungen in den durch entsprechende Plugins unterstützten SOA-Implementierungen. Zu diesen gehören auch Web Services. Der Surrogate – als Web Service-Consumer – empfängt diese Service-Beschreibungen und generiert virtuelle Chat-Endpunkte, die Teilnehmern in der virtuellen Welt angeboten werden. Über diese Endpunkte können die Teilnehmer aus der virtuellen Welt Text-Nachrichten in die Präsenzumgebung senden. Sie nutzen dabei den Second Life-eigenen Text-Chat. Im Sinne höchster Transparenz macht es für die virtuellen Teilnehmer keinen Unterschied, ob sie mit anderen virtuellen Teilnehmern oder vom Surrogate generierten Chat-Endpunkten realer Teilnehmer kommunizieren. Dies trifft analog für die realen Teilnehmer zu.

In der Gegenrichtung bietet der Surrogate die virtuellen Chat-Endpunkte als Web Services an, die durch den GPAP in der Präsenzumgebung als SDP-Services angeboten werden. Auch die Teilnehmer aus der Präsenzumgebung kommunizieren dabei gleichermaßen mit anderen lokalen oder virtuellen Teilnehmern über die Chat-Anwendung ihres Mobiltelefons.

Die in diesem Abschnitt beschriebenen SOA-übergreifenden Szenarien zeigen das Potential einer Berücksichtigung verschiedener SOA-Implementierungen in modernen Lehr- und Lernarragements. Die Konzentration auf eine einzige Implementierung würde entweder auf Kosten der Transparenz gehen, da zusätzliche Modifikationen vorhandener Komponenten erforderlich wären, oder sogar Komponenten von vornerein ausschließen.

Die bisher vorgestellten Ansätze und Lösungen lassen sich als Bestandteile einer service-orientierten Umgebung wie etwa einer service-orientierten Universität klassifizieren. Der Schritt zur Pervasivität dieser Umgebungen kann erst durch weitere Konzepte vollzogen werden, allem voran die Einbeziehung von Kontextinformationen. Vor allem, um den Bedarf dieser Funktionalitäten für auf dieser Schrift aufbauende Arbeiten zu motivieren und erste Lösungsansätze zu präsentieren, wird im folgenden Abschnitt die Erstellung kontext-basierter Arrangements demonstriert und diskutiert.

5.4 Kontextbasierte Arrangements

Die durch pervasive Umgebungen angestrebte proaktive Unterstützung eines Nutzers beim Erreichen seiner Ziele erfordert eine Einbeziehung von Kontextinformationen wie beispielsweise die Intention des Nutzers. Wie bereits in Abschnitt 3.3 diskutiert gibt es vor allem drei Ansätze zur Integration von Kontext in SOA-basierte Infrastrukturen: kontextbasierte Services, Kontextnutzung für Service-Angebot und -Suche sowie kontextbewusste Service-Anpassungen. Obwohl eine umfassende Bearbeitung dieses komplexen Themas nicht im Fokus dieser Dissertationsschrift werden zur Unterstützung auf dieser Schrift aufbauender Arbeiten im Folgenden drei Fallstudien betrachtet, die das

Vorgehen zur Einbeziehung von Kontextinformationen für die ersten beiden Ansätze demonstrieren.

5.4.1 Fallstudie: Kontext als Service

In dieser ersten kontextbasierten Fallstudie wird exemplarisch – in Form kontextbasierter Services – die Integration von Rohkontextlieferanten in eine SOA gezeigt [Lindemann 08]. Als Lieferanten dienen dabei *SunSPOTs* [Sun Microsystems 07], mobile Sensorknoten die über eine an ZigBee angelehnte drahtlose Kommunikationsverbindung miteinander kommunizieren. Es existieren zwei SunSPOT-Varianten, die untereinander kommunizieren, die Basisstation und der Sensor. Während die Sensoren Helligkeit, Temperatur und Beschleunigung in drei Dimensionen messen, dient die Basisstation als Schnittstelle zwischen Sensoren und anderen Systemen (z. B. handelsüblichen PCs).

Der Entwurf eines Systems zur Integration der Kontextdaten in eine SOA folgt auf Provider-Seite wieder dem bereits vorgestellten Vorgehen:

1. Da die SunSPOTs über eine Java-Platform programmiert werden können, wurde im Sinne geringer Komplexität die Java-basierte SOA-Implementierung Jini verwendet.
2. Als Teildienste wurden die einzelnen Sensorinformationen identifiziert: Somit gibt es pro Sensor-SunSPOT zunächst drei Dienste: Lieferant für Helligkeits-, Temperatur- und Beschleunigungskontext. Zusätzlich wurde für jeden Sensor-SunSPOT ein weiterer Dienst identifiziert, der die Signalstärke zwischen dem Sensor und der Basisstation liefert, wodurch sehr grobe Ortsinformationen abgeleitet werden könnten.
3. Sensortypisch sind SunSPOTs ressourcenarm. Daher wurde ein handelsüblicher Windows-PC als Mediator und somit Provider gewählt, der über eine angeschlossene SunSPOT-Basisstation mit den einzelnen Sensoren kommunizieren konnte.
4. Die von den verfügbaren Sensor-SunSPOTs gelieferten Sensordaten wurden durch den Mediator normalisiert und der Zugriff auf den normalisierten Kontext in Form dreier Jini-Services implementiert. Zusätzlich wurde von der Basisstation die Signalstärke zu jedem der Sensoren ebenfalls über einen Jini-Service verfügbar gemacht. Somit existierten zu jedem Sensor vier Services.
5. Die Jini-Services wurde über die Jini-Software des Mediators wie in Abschnitt 2.2.3 beschrieben beim Lookup-Service registriert und konnten somit von möglichen Consumern gefunden werden.

Als Consumer wurde eine Java-Anwendung implementiert, die beim Lookup-Service registrierte Services findet, konsumiert und die konsumierten Information in einer grafischen Benutzungsschnittstelle darstellt. Diese GUI ist mitsamt der Architektur und Funktionsweise des prototypischen Gesamtsystems in Abbildung 5.12 dargestellt.

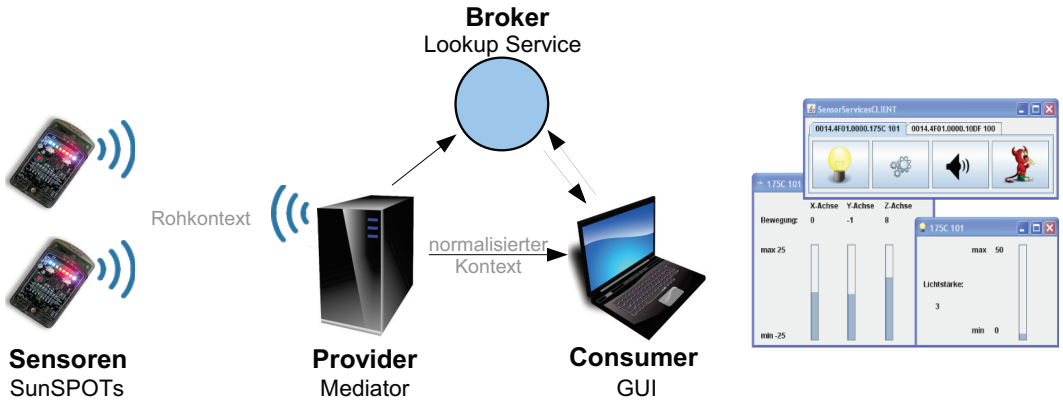


Abbildung 5.12: Die Integration von speziellen Kontextlieferanten (z. B. Sensoren) erfolgt über Mediatoren, die als Provider fungieren.

Diese Fallstudie zeigt, dass der Zugriff auf Sensordaten in Form von Services über das gleiche Vorgehen realisiert werden kann, wie die Erstellung anderer Services. Die besondere Stärke von kontextbewussten und auf SOA-basierten Infrastrukturen aufbauenden Anwendungen wird in den beiden folgenden Fallstudien deutlich.

5.4.2 Fallstudie: Kontextbasierte Stud.IP-Anpassung

Die situationsabhängige Visualisierung von Informationen ist ein Aspekt pervasiven Verhaltens. Insbesondere in mobilen Szenarien, in denen auch Kleingeräte mit (im Vergleich zu PCs oder Laptops) sehr beschränkten Anzeigefähigkeiten eine wichtige Rolle spielen, können Kontextinformationen über die entsprechende Geräteklassen genutzt werden, um die angezeigten Daten an das jeweilige Gerät anzupassen.

Im Umfeld einer pervasiven Universität spielen Lehr- und Lernmangementsysteme eine wichtige Rolle für die Organisation von Lehrmedien und -veranstaltungen. Diese Systeme sind selten für die Nutzung über mobile Endgeräte wie Smartphones optimiert. Sie können zwar auf derartigen Geräten dargestellt werden, ihre Bedienbarkeit hält sich jedoch in Grenzen und gestaltet sich insbesondere für ungeübe Smartphone-Nutzer schwierig. Beispielsweise erfordert die Bedienung der auf höhere Auflösungen optimierten Webseiten Erfahrungen mit dem Zoomen und das Trainieren von Link-Betätigungen per Finger. In der folgenden Fallstudie wird die Nutzung des Lehr- und Lernmanagementsystems Stud.IP auf mobilen Geräten vereinfacht, indem Kontextinformationen über die entsprechende Geräteklassen genutzt werden, um den Umfang und die Darstellung von Stud.IP an das jeweilige Gerät anzupassen.

Stud.IP unterstützt bereits die gezielte Abfrage von Informationen (z. B. Übersicht über die nächsten Lehrveranstaltungen) über Web Services. Somit muss nicht das gesamte Stud.IP-System monolithisch an das Smartphone übermittelt werden, wie es für PCs und Laptops üblich ist. Es genügt, die in der aktuellen Situation relevanten Informationen über Web Services abzufragen, diese für die mobile Nutzung aufzubereiten

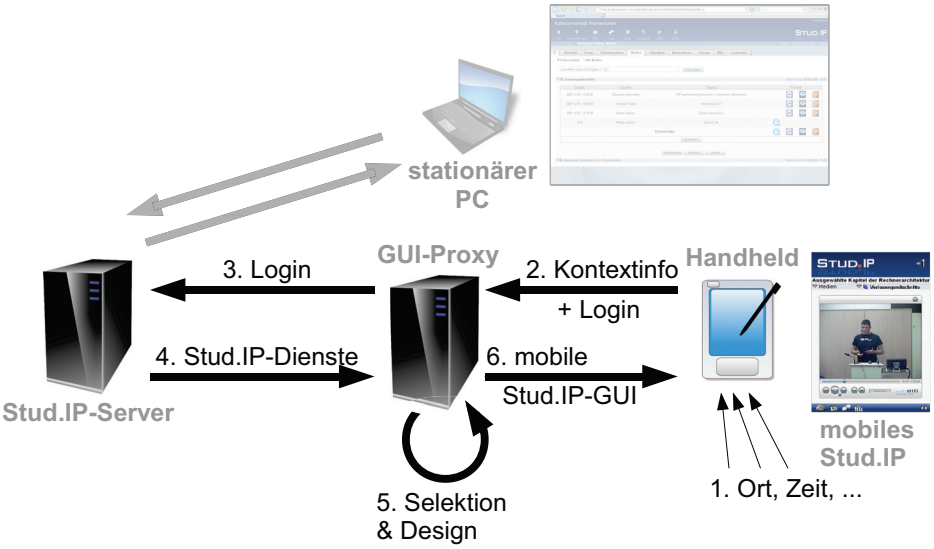


Abbildung 5.13: Eine kontextbasierte Service-Auswahl ist die Grundlage für die kontextbasierte Stud.IP-Optimierung für mobile Geräte.

und an das anfragende Gerät zu übertragen. Abbildung 5.13 illustriert die Funktionsweise eines im Rahmen dieser Arbeit entstandenen Systems zur kontextbasierten Stud.IP-Nutzung auf mobilen Geräten [Lehsten 10].

Für diese Fallstudie wurde angenommen, dass die Nutzer an einer Universität über internetfähige Mobilgeräte (Handheld) verfügen, die in der Lage sind, ihren Ort sowie die aktueller Uhrzeit als Kontextinformationen zu erfassen. Eine Browser-Erweiterung, die zunächst für das G1-Smartphone entwickelt wurde, sammelt diese Informationen zusammen mit der Nutzeridentität (über einen Login erfasst) und bietet den gesammelten Kontext als Service an. Ein Proxy konsumiert die angebotenen Informationen, loggt sich mit den Nutzerdaten beim traditionellen Stud.IP-System ein und konsumiert die von Stud.IP angebotenen Services, die einen Bezug zu den registrierten Veranstaltungen des Nutzers haben. Weiterhin bestimmt er abhängig von den Ortsinformationen des Nutzers dessen Position in der Universität (z. B. Gebäude). Abhängig von der aktuellen Uhrzeit, den registrierten Veranstaltungen und der Nutzerposition leitet der Proxy die vermutliche Nutzeraktivität bzw. -intention ab und liefert eine reduzierte und für den mobilen Einsatz otpimierte Stud.IP-Version inklusive weiterer hilfreicher Informationen an den Handheld. Zwei Beispiele für ein derartiges Verhalten sind:

- Der Nutzer ist in dem Raum einer laufenden oder in Kürze beginnenden Veranstaltung: Das System zeigt Informationen über die aktuelle Veranstaltung (z. B. Lehrmedien).
- Der Nutzer befindet sich auf dem Weg zu einer laufenden oder in Kürze beginnender Veranstaltung: Das System zeigt aktuelle Nahverkehrsverbindungen zur Veranstaltung sowie die Adresse eines zur Veranstaltung gehörenden Live-Streams an (falls vorhanden).

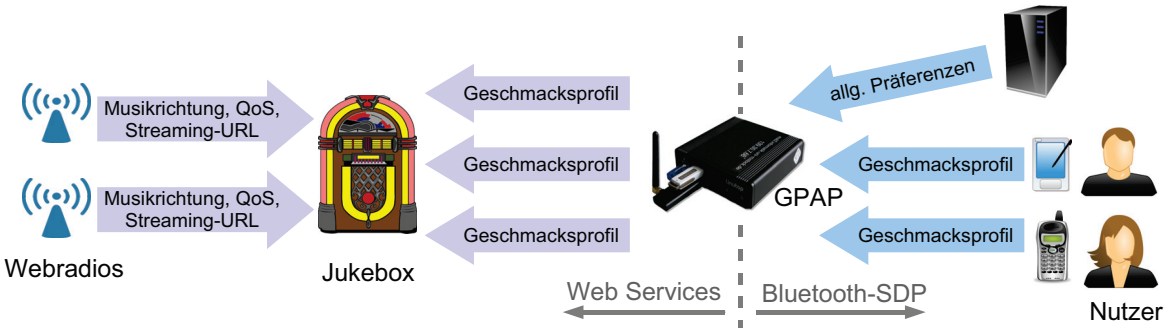


Abbildung 5.14: Die kontextbasiere Jukebox konsumiert Geschmacksprofile von Nutzern, ermittelt einen gemeinsamen Musikgeschmack wählt dann ein dazu passendes Webradio zur Wiedergabe aus.

Diese Fallstudie zeigt, wie SOA und Kontext für ein konkretes Szenario an einer pervasiven Universität genutzt werden können, um den Nutzer zielgerichtet und intelligent zu unterstützen ohne ihn mit einer unnötigen Funktionsvielfalt zu überfordern. Sie dient weiterhin als erster Schritt in Richtung einer pervasiven Universität auf Grundlage einer SOA-basierten Infrastruktur, wie sie von auf dieser Arbeit aufbauenden Forschungen im Graduiertenkolleg MuSAMA angestrebt wird.

Auch außerhalb des universitären Umfeldes bietet die Kombination einer SOA-basierten Infrastruktur mit kontextbewusstem Verhalten ein großes Potential für nutzerzentrierte, smarte Umgebungen, wie folgende Fallstudie eines SOA-übergreifenden Systems verdeutlicht.

5.4.3 Fallstudie: Kontextbasierte Jukebox

Die Fallstudie der kontextbasierten Stud.IP-Anpassung hat bereits gezeigt, wie kontextbasierte Anwendungen einzelne Nutzer beim Erreichen ihrer Ziele unterstützen können. In pervasiven Umgebungen kommen jedoch häufig mehrere Nutzer zusammen und oftmals hat ein kontextbasiertes System es daher nicht mit einem einzelnen Nutzerkontext zu tun, sondern zeitgleich mit mehreren die sich zudem voneinander unterscheiden können. Die Fallstudie der kontextbasierten Jukebox ist ein Beispiel für eine kontextbasierte Multiuser-Anwendung, die obendrein eine SOA-übergreifende Interoperabilität erfordert [Lenz 09].

Wie Abbildung 5.14 verdeutlicht, ist die kontextbasierte Jukebox in der Lage Webradios, die als Web Services zur Verfügung stehen, zu nutzen und den angebotenen Audio-Stream abzuspielen. Die Radio-Services enthalten neben der Streaming-URL, über die der Audio-Stream aufgerufen werden kann, diverse Kontextinformationen über den angebotenen Stream. Dazu gehören:

- Name der Radiostation
- Format des Audio-Streams (z. B. Mp3, AAC+)
- Bitrate des Audiostreams (z. B. 32 kbps, 96 kbps, 128 kbps)
- bedientes Musikgenre (z. B. Klassik, Rock, Schlager, Techno)

Die Jukebox ist für Broadcast-Szenarien ausgelegt, sie kann somit beispielsweise in einem Supermarkt, einem Wartezimmer oder einer Flughafenlounge betrieben werden. Ihre Aufgabe ist es, die Menschen im Einzugsgebiet bestmöglich zu unterhalten. Dafür muss sie den jeweiligen Musikgeschmack der Anwesenden abrufen und auswerten können. Wie Abbildung 5.14 illustriert, wurde davon ausgegangen, dass dieser in Form eines Geschmacksprofils auf den Blueooth-fähigen Mobiltelefonen von Kunden bzw. Gästen mitgeführt wird. Bluetooth-fähige Endgeräte nutzen die SOA-Implementierung SDP. Somit ist die Jukebox als Web Service Consumer nicht in der Lage, diese nativ zu konsumerien. Das Konzept des GPAPs wurde daher genutzt, um SDP-Dienste auf Web Services abzubilden. Diese konnten dann von der Jukebox verwendet werden, um basierend auf dem Musikgeschmack der Anwesenden einen Webradio-Sender auszuwählen, der den meisten Personen gefallen sollte. Diese Entscheidung ist das Ergebnis eines Vergleichs der Webradio-Informationen mit den Geschmacksprofilen der Nutzer sowie ggf. generellen Präferenzen (z. B. beruhigende Musik in der Arztpraxis).

Zur Entscheidungsfindung wurden die bekannten Musikrichtungen in einem gerichteten Baum organisiert. So hatte beispielsweise die Musikrichtung Rock als Subgenre unter anderem die Musikrichtungen Classic Rock sowie Hard Rock und war selbst ein Subgenre der unbenannten Baumwurzel. Durch diese Aufteilung konnten die Geschmacksrichtungen ebenfalls in einen korrespondierenden Baum einsortiert werden. Durch jeden Nutzer mit einer bestimmten Geschmacksrichtung wurde der entsprechenden Musikrichtung eine zusätzliche Stimme erteilt. Nachdem Auszählen aller Stimmen wurde der Baum rekursiv durchlaufen und das Genre mit der höchsten Anzahl an Stimmen als gewünschte Musikrichtung ausgewählt. Die Stimmen für ein Genre ergeben sich aus den Stimmen für es selbst, sowie einer gewichteten Summe der Subgenres. Durch dieses Verfahren wurde beispielsweise generelle Rockmusik ausgewählt, wenn die meisten Nutzer ein Rock-Subgenre gewählt haben, sich aber nicht einem gemeinsamen Subgenre zuordnen ließen. Weitere Mechanismen, beispielsweise zur spontanen Einbeziehung neuer Nutzer, rundeten diesen Algorithmus ab [Lenz 09].

Diese kontextbasierte Fallstudie zeigt, dass neben reinen Schlussfolgerungen auf individuelle Nutzerintentionen auch Entscheidungen getroffen werden müssen, die bei der individuellen Betrachtung eines einzelnen Nutzers nicht optimal wären. Durch die verschiedenen Geschmacks-Services aus dieser Fallstudie wurde ein systematischer Weg aufgezeigt, verschiedene Kontextquellen zu konsumieren und gegenüberzustellen. Die Entscheidung für eine gemeinsame Musikrichtung ist zwar im Vergleich zu den komplexeren, mehrdimensionalen Nutzerintentionen pervasiver Umgebungen trivial, verdeutlicht jedoch dass die verschidenen Kontextquellen in einer einheitlichen Form (Services) zur Verfügung stehen müssen. Durch die verschiedenartigen Kontextlieferanten einer

pervasiven Umgebung erfordert dies auch SOA-übergreifende Interoperabilität, wie sie in dieser Fallstudie herrschte.

Anhand ausgewählter Fallstudien an der Universität Rostock wurden in diesem Kapitel exemplarische Aspekte der SOA-Einführung für moderne Bildungseinrichtungen vertieft und verifiziert. Es wurde gezeigt, wie grundlegende universitäre Dienste als Services implementiert werden, die SOA-Modularität und -Flexibilität moderne Lehr- und Lernszenarien fördert und sogar neue Lehr- Lernparadigmen wie das Immersive Learning entstehen. Durch SOA-übergreifende Mechanismen, wie den in dieser Arbeit vorgestellten GPAP, ist auch die transparente Integration heterogener Lehr und -Lernnarragements sowie neuer Werkzeuge realisierbar. Abschließend wurde durch exemplarische Kontexteinbeziehung der wichtige Schritt vom Service-orientierten zum pervasiven Arrangement vollzogen und der Weg für darauf aufbauende Arbeiten geebnet.

Kapitel 6

Zusammenfassung und Ausblick

Mit der wachsenden Bedeutung von Mobilität, allgegenwärtigem IT-Zugang und dynamischer Allokation anpassbarer Inhalte ist eine zunehmende Integration von Technologien des Pervasive Computing in Lehr- und Lernprozesse erkennbar. Moderne Bildungseinrichtungen sind mehr und mehr auf dem Weg pervasive Institutionen zu werden, eine Ausprägung pervasiver Umgebungen mit auf Hochschulen zugeschnittenen Komponenten und Interaktionsmustern. Somit müssen sich die mit dem Begriff *Pervasive Universität* [Tavangarian 09] bezeichneten Hochschulen neben speziellen organisatorischen Anforderungen den gleichen Herausforderungen auf infrastruktureller Ebene stellen, wie andere pervasive Umgebungen.

6.1 Zusammenfassung und erzielte Ergebnisse

Eingebettet in das Graduiertenkolleg *Multimodal Smart Appliance Ensembles for Mobile Applications* (MuSAMA) analysierte diese Dissertation die infrastrukturellen Herausforderungen für pervasive Umgebungen im Allgemeinen und pervasive Hochschulen im Speziellen. Sie bewertete basierend auf dieser Analyse die Eignung heutiger Kommunikationsmodelle als infrastrukturelle Basis zur Erfüllung der speziellen Anforderungen derartiger Umgebungen. Als Ergebnis dieses Vergleiches wird der infrastrukturelle Einsatz Service-orientierter Architekturen (SOA) zur Erfüllung der wichtigsten Anforderungen motiviert. Dafür waren die modellgegebenen und implementierungsspezifischen SOA-Charakteristika entscheidend:

- Hard- und Software-Abstraktion
- spontane Komponenteneinbindung
- spontane Komponentenanpassung
- Komponentenverteilung und -migration
- kein Vorwissen über Infrastruktur und Komponenten erforderlich

- Laufzeitumgebungen teilweise verfügbar
- gute Skalierbarkeit
- Integration existierender Komponenten möglich
- autonome, kontrollierende Komponenten

Diese Arbeit zeigte jedoch auch die Schwächen und offenen Probleme Service-orientierter Architekturen auf. Eines dieser Probleme, das insbesondere für pervasive Umgebungen hochaktuell ist, ist SOA-Heterogenität. Pervasive Umgebungen setzen sich heute aus einer Vielzahl unterschiedlicher Komponenten aus verschiedenen Anwendungsbereichen zusammen. Da sich in den jeweiligen Anwendungsbereichen unterschiedliche SOA-Implementierungen wie Web Services, UPnP und SDP etabliert haben, müssen pervasive Umgebungen Interoperabilitätsmechanismen zur Überwindung der SOA-Heterogenität bereithalten. Nach einer Betrachtung der relevanten SOA-Implementierungen wurden in dieser Arbeit die bekannten Interoperabilitätsansätze einer von drei Klassen zugeordnet:

- Technologiebrücken
- Abstrakte Provider
- Abstrakte Consumer

Deren Eignung zur Interoperabilität in pervasiven Umgebungen wurde anschließend diskutiert und bewertet.

Ergebnis 1: Konzeption und prototypische Implementierung eines systematischen Interoperabilitätskonzept

Da die bekannten Ansätze aufgrund unzureichender Systematik vor allem Nachteile in der Skalierbarkeit und Komplexität haben, wird mit dem Konzept der *Zentralen Vereinheitlichung* [Zender 10a] ein systematischer Interoperabilitätsansatz vorgestellt, der sich auf die Nutzung einer integrierten Meta-Sprache stützt. Da dieser Ansatz die systematische Integration SOA-spezifischer Syntax- und Prozesscharakteristika erfordert, sind entsprechende Strategien für SOA-spezifische Plugins essentieller Bestandteil des systematischen Interoperabilitätskonzeptes. Die entwickelten Strategiegen wurden im Rahmen des, in dieser Dissertation entwickelten, Service-Layers für die zentrale infrastrukturelle GPAP-Komponente implementiert. Dieser ist als infrastruktureller Koordinator einer pervasiven Umgebung in der Lage, zwischen verschiedenen Geräten und Anwendungen sowohl auf Netzwerk- als auch auf Service-Ebene zu vermitteln und steigert somit die Zuverläsigkeit, Robustheit und Effizienz pervasiver Kommunikation.

Ergebnis 2: SOA-Klassifikation und Entscheidungshilfe zur Pluginentwicklung

Im Rahmen dieser Dissertation wurden die wichtigsten SOA-Implementierungen im Hinblick auf für die Interoperabilität relevante Kriterien klassifiziert. Für die einzelnen Klassen wurden Strategien zu deren Integration in das entwickelte Konzept konzipiert. Diese werden in Form dedizierter Plugins umgesetzt. Um die zukünftige Entwicklung dieser Plugins zu erleichtern und somit zu fördern, wurde eine Entscheidungshilfe zur Klassifikation einer konkreten SOA-Implementierung und der Entscheidung für eine entsprechende Plugin-Variante erarbeitet.

Ergebnis 3: Konzeption und Durchführung richtungsweisender Fallstudien an einer pervasiven Universität

Die Bedeutung und das Potential der erzielten Ergebnisse wurde im Rahmen dieser Arbeit für moderne Hochschulen aufgezeigt. Im Rahmen mehrerer konkreter und durch prototypische Implementierungen unterlegter Fallstudien erarbeitete diese Dissertation den Mehrwert einer Service-orientierten Universität als infrastrukturelle Grundlage der Pervasiven Universität und evaluierte die vorgestellten Konzepte und Lösungen [Zender 09a]. Neben der Bereicherung traditioneller Lehr- und Lernarrangements ergeben sich durch die in dieser Arbeit entstandenen Zuwächse hinsichtlich Dynamik und Transparenz völlig neue, innovative Arrangements wie die *Immersive Lehre* [Zender 10a].

6.2 Ausblick und weiterführende Fragestellungen

Diese Dissertation ist ein erster Schritt in Richtung systematischer und auf Pervasivität ausgerichteter SOA-Interoperabilität. Es gibt in vielen Bereichen dieser Arbeit noch Baustellen und offene Fragestellungen, die weiterer Aufmerksamkeit bedürfen und diese im Rahmen des Graduiertenkollegs MuSAMA auch erhalten. Im folgenden werden die wichtigsten Themen heraus gegriffen, um sowohl den Forschungsbedarf als auch die entscheidende Ingenieursarbeit zielführend zu lenken.

Neben der Integration weiter SOA-Implementierungen in die Service-Ebene das GPAPs ist in den kommenden Jahren ein besonderer Fokus auf die Interoperabilität der Service-Nutzung und deren Berücksichtigung durch die Zentrale Vereinheitlichung zu richten. Dies war aufgrund des geringen Spektrums der Nutzungsmechanismen (meist nicht spezifiziert, in den meisten anderen Fällen SOAP-basiert) im Vergleich zu den Discovery-Mechanismen nicht Schwerpunkt dieser Arbeit. Dennoch darf dieses Thema auch im Sinne der Erweiterbarkeit um weitere SOA-Implementierungen und der damit verbundenen Zukunftsfähigkeit des entwickelten Konzeptes nicht völlig vernachlässigt werden.

Der zentrale gemeinsame Nenner der Service-Beschreibungen aller SOA-Implementierungen ist die Verwendung eines Typen als Grundlage für den semantischen Vergleich von Services sowie deren Suche. Im Rahmen der vorliegenden Arbeiten wurde auch in der STiL ein Attribut `stilType` eingeführt. Dieser wird anhand einer schlüsselwortbasierten Zuordnung aus den implementierungsspezifischen Service-Typen abgeleitet. Diese Lösung ist in Anbetracht des sematischen Interpretationsspielraumes leider nur unzureichend praktikabel. Im Zuge weiterer Arbeiten sollte daher in eine semantisch ausgereiftere Methode konzipiert und implementiert werden. Dabei erscheinen vor allem Ontologie-basierte Ableitungen als vielversprechend.

Weiterhin wurden in dieser Dissertation die Bedeutung und das Potential von Kontextinformationen für SOA-basierte pervasive Umgebungen betrachtet. Diese Informationen könnten beispielsweise genutzt werden, um während des Service Discovery-Prozesses die gefundenen Service-Beschreibungen im Hinblick auf ihre Performance, Robustheit oder allgemeine Eignung für das jeweilige Anwendungsszenario zu bewerten. Als Ergebnis könnte nur die Beschreibung des "besten" Services oder etwa ein Service-Ranking zurückgeliefert werden. Allerdings erfordern derartige Funktionalitäten eine weitere und möglichst automatische Anreicherung der STiL-Descriptions mit Kontextinformationen. Die Fragen nach der Art der Kontext-Allokation, der kontextuellen Service-Beschreibung und der Auswertung dieser Information während des Service-Discovery sollte Thema weiterer Arbeiten auf diesem Gebiet sein.

Die vorliegende Arbeit behandelt vor allem die grundlegenden Fragen, Probleme und Herausforderungen SOA-basierter Umgebungen im Allgemeinen und deren Unterstützung durch SOA-Interoperabilität im Speziellen. Darauf aufbauende und begleitende Fragen wie etwa die Sicherheitsmechanismen in derartigen Umgebungen lagen nicht im Fokus dieser Arbeit, sind jedoch Thema im Graduiertenkolleg MuSAMA. Zwar bringen einige SOA-Implementierungen eigene Sicherheitsmechanismen mit, die Frage nach der Übertragbarkeit dieser Lösungen auf SOA-übergreifende und pervasive Szenarien bedarf jedoch weiterer Forschung und sollte in weiterführenden Arbeiten mit hoher Priorität betrachtet werden.

Der Weg traditioneller Hochschulen zur Pervasiven Universität führt auf infrastruktureller Ebene über die SOA-Einführung. Diese gestaltet sich aufgrund der oftmals proprietären IT-Einzellösungen im universitären Umfeld nicht immer einfach. Hier ist in den nächsten Jahren ein Aufbrechen der monolithischen Softwarekomponenten erforderlich. Im Einzelfall muss dabei geklärt werden, welche Teilkomponenten wie als Services angeboten werden sollten und ob einzelne Funktionalitäten und Inhalte auch von externen Dienstleistern (z.B. anderen Bildungseinrichtungen) integriert werden können. Neben neuen Prozessmodellen für die Interaktion zwischen internen und externen Services sowie SOA-freien Komponenten ist auch ein Umdenken und Kooperationsbereitschaft von Seiten der Verantwortlichen für universitäre Lösungen erforderlich. Der Paradigmenwechsel von der Notebook-Universität zur Pervasiven Universität steht nach wie vor noch am Anfang. Dennoch ist dieser Web zwingend erforderlich und klar erkennbar. Er wird weiterhin durch diverse Projekte an den einzelnen Hochschulen und

durch die neue Mobilität und Flexibbilität der Studenten und Mitarbeiter gefördert und gefordert.

Thesen

1. Mit der wachsenden Bedeutung von Mobilität, allgegenwärtigem IT-Zugang und dynamischer Allokation anpassbarer Inhalte ist eine zunehmende Integration von Technologien des Pervasive Computing in Lehr- und Lernprozesse erkennbar.
2. Die (teils konstruktive) Erschließung von Lehrinhalten durch virtuelle Welten hat ein hohes Potential, den individuellen Lernprozess zu unterstützen.
3. Immersive Lehre verbindet systematisch die Lehr- und Lernparadigmen der virtuellen und der Präsenzlehre und bereichert somit beide Paradigmen um Werkzeuge und Medien des jeweils anderen.
4. Das Service-orientierten Architekturen zugrunde liegende Broker-Modell ist eine geeignete infrastrukturelle Grundlage für pervasive Umgebungen, da es dem Großteil der aus Dynamik und Mobilität resultierenden Anforderungen gerecht wird.
5. Dynamische pervasive Umgebungen bestehen aus Komponenten unterschiedlicher Anwendungsbereiche. Mit der Vielzahl der Anwendungsfelder stehen derartige Umgebungen vor einer SOA-Heterogenität, die systematische Interoperabilitätsansätze erfordert.
6. Von den existierenden Interoperabilitätsansätzen ist die Zentrale Vereinheitlichung sowohl funktional als auch im Hinblick auf die Effizienz für pervasive Umgebungen zu bevorzugen.
7. Die Strategien zur funktionalen und syntaktischen Integration verschiedener SOA-Implementierungen in das Konzept der zentralen Vereinheitlichung können in Form vom Discovery-, Provision-, und Broker-Plugins umgesetzt werden.
8. Die Nutzung von Kontextinformationen hat das Potential, die Service-Auswahl nach dem Service Discovery-Prozess einer SOA zu erleichtern und zusätzliche Service-Funktionalitäten zu realisieren.
9. Welche Informationen als Kontext für eine Anwendung relevant sind, hängt wiederum vom Kontext der Anwendung ab. Daher führt diese Dissertation keine allgemeingültige Blaupause sondern ein generelles Vorgehensmodell ein.

10. Zur Integration von Kontext in SOA-basierte Infrastrukturen stehen kontextbasierte Services, eine Kontextnutzung für Service-Angebot und -Suche sowie kontextbewusste Service-Anpassungen zur Verfügung.

11. Peer-to-Peer-basierte Communitys sind Hierarchie-basierten Communitys in Bezug auf Flexibilität und Robustheit überlegen.

Anhang A

XML-Schema der Service Technology-independent Language (STiL)

```
<?xml version="1.0" encoding="UTF-8"?>
<schema targetNamespace="http://www.musama.de/STiL"
   elementFormDefault="qualified"
   xmlns="http://www.w3.org/2001/XMLSchema"
   xmlns:tns="http://www.musama.de/STiL">

 <complexType name="stil">
  <choice maxOccurs="1" minOccurs="1">
   <element name="request" type="tns:serviceRequest"></element>
   <element name="response" type="tns:serviceResponse"></element>
   <element name="description"
   type="tns:serviceDescription"></element>
  </choice>
  <attribute name="version" type="string"></attribute>
 </complexType>

 <element name="stil" type="tns:stil"></element>

 <complexType name="serviceRequest">
  <sequence>
   <element name="serviceMask" type="tns:serviceSearchMask"
    maxOccurs="unbounded" minOccurs="1">
   </element>

  </sequence>
  <attribute name="requestID" type="string"
   use="required"></attribute>
  <attribute name="sender" type="string" use="required">
   <annotation>
   </annotation>
```

```
 </attribute>
 <attribute name="timeout" type="string" use="optional">
  <annotation>
  </annotation>
 </attribute>

</complexType>

<complexType name="serviceResponse">
 <choice maxOccurs="1" minOccurs="1">
  <element name="error" type="tns:error" maxOccurs="1"
  minOccurs="1"></element>
  <element name="service" type="tns:serviceDescription"
  maxOccurs="unbounded" minOccurs="1"></element>
 </choice>
 <attribute name="requestID" type="string"
  use="required"></attribute>
  <annotation>
   <documentation>
    Distanz (in Hops) zu dem antwortendem E2C-Gateway.
   </documentation>
  </annotation></attribute>
</complexType>

<complexType name="serviceDescription">
 <sequence>
  <element name="stilType" type="string" maxOccurs="unbounded"
   minOccurs="1">
   <annotation>
    <documentation>STiL service type</documentation>
   </annotation>
  </element>

  <element name="friendlyName" type="string" maxOccurs="1"
   minOccurs="0">
  </element>
  <element name="technology" type="string" maxOccurs="1"
   minOccurs="0">
   <annotation>
    <documentation>
     SOA technology, that is used to provide this
     service
    </documentation>
   </annotation>
  </element>
  <element name="validity" type="string" maxOccurs="1"
   minOccurs="0">
   <annotation>
    <documentation>
     period of time, the service is valid - DURATION
     (RFC 2445)
    </documentation>
```

```
   </annotation>
  </element>

  <element name="comProtocol" type="string" maxOccurs="unbounded"
  minOccurs="0">
   <annotation>
    <documentation>
     Protocol, that can be used to communicate with
     this service (e.g. UDP, TCP, SOAP, HTTP, ...)
    </documentation>
   </annotation>
  </element>
  <choice maxOccurs="unbounded" minOccurs="1"><element
  name="accessLocation" type="string" minOccurs="1"
  maxOccurs="unbounded">
   <annotation>
    <documentation>
     Address (e.g. URL), to access the service.
    </documentation>
   </annotation>
  </element><element name="realDescription" type="string"
  minOccurs="1" maxOccurs="unbounded">
   <annotation>
    <documentation>
     Address of a service description document (e.g.
     WSDL).
    </documentation>
   </annotation>
  </element></choice>
  <element name="extendedDescription" type="tns:namedString"
  maxOccurs="unbounded" minOccurs="0">
   <annotation>
    <documentation>
     To describe the service for special scopes (e.g.
     context orientation, service composition)
    </documentation>
   </annotation>
  </element>
  <element name="attribute" type="tns:namedString"
  maxOccurs="unbounded" minOccurs="0">
   <annotation>
    <documentation>
     service specific name/value pair
    </documentation>
   </annotation>
  </element>
 </sequence>
 <attribute name="provider" type="string" use="required">
  <annotation>
   <documentation>
    Address of the device, providing this Service (e.g.
    IP-Adress of the GPAP and port corresponding to the
```

```
    provider).
   </documentation>
  </annotation></attribute>
</complexType>

<complexType name="error">
 <attribute name="code" type="int">
  <annotation>
   <documentation>Fehlercode</documentation>
  </annotation></attribute>
 <attribute name="reason" type="string"></attribute>
</complexType>

<complexType name="serviceSearchMask">
 <sequence>
  <element name="scope" type="int" minOccurs="0"
  maxOccurs="1"></element>
  <element name="stilType" type="tns:restrictedString"
   maxOccurs="unbounded" minOccurs="1">
   <annotation>
   </annotation>
  </element>

  <element name="friendlyName" type="tns:restrictedString"
   maxOccurs="unbounded" minOccurs="0">
   <annotation>
   </annotation>
  </element>
  <element name="technology" type="string"
   maxOccurs="unbounded" minOccurs="0">
  </element>
  <element name="validity" type="tns:restrictedString"
   maxOccurs="1" minOccurs="0">
   <annotation>
    <documentation>
     Restriction for this element: minimum, maximum
    </documentation>
   </annotation>
  </element>
  <element name="comProtocol" type="string"
   maxOccurs="unbounded" minOccurs="0">
  </element>

  <element name="extendedDescription" type="tns:namedString"
   maxOccurs="unbounded" minOccurs="0">
  </element>
  <element name="attribute" type="tns:namedString"
   maxOccurs="unbounded" minOccurs="0">
  </element>

 </sequence>
```

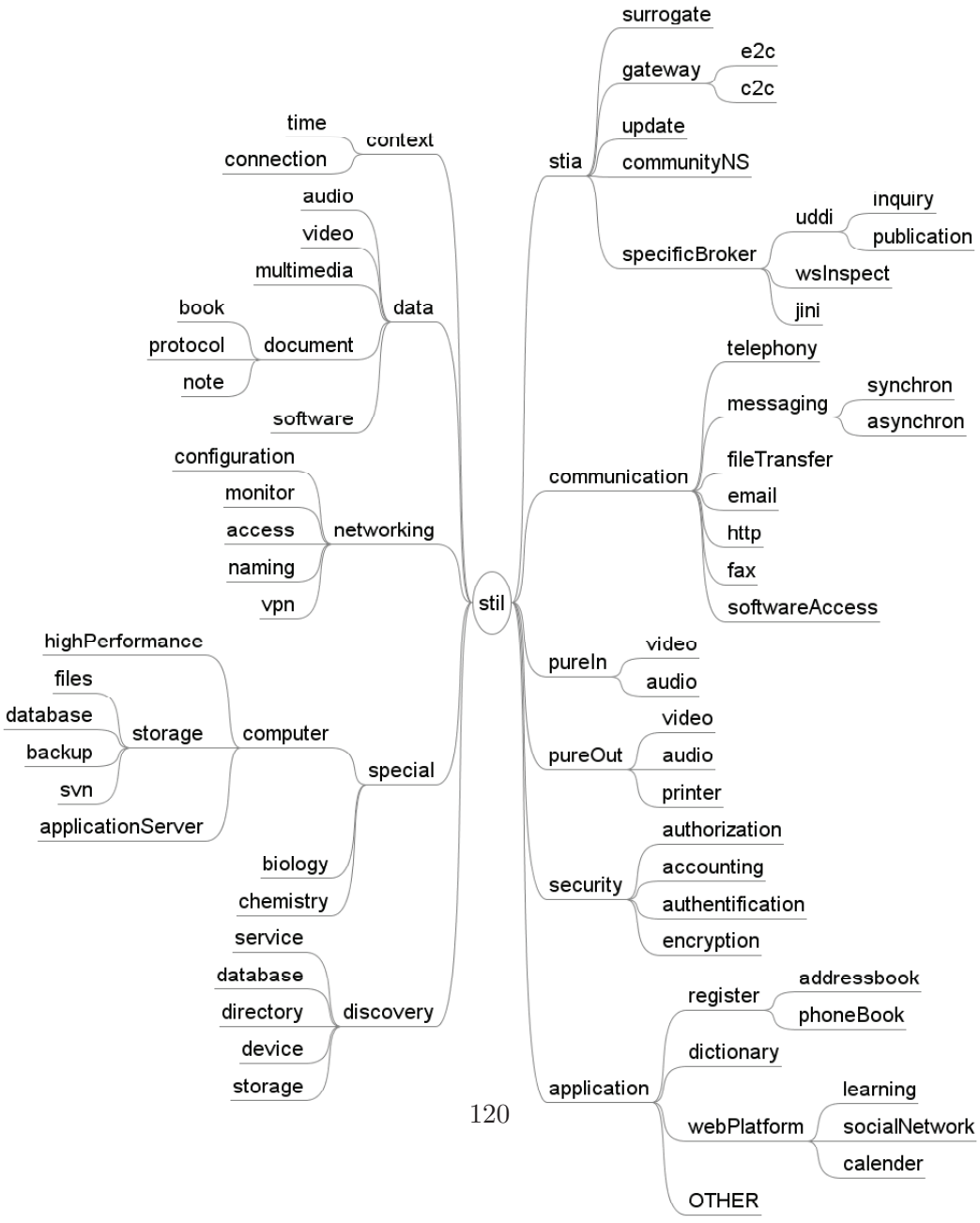
```
 </complexType>

 <complexType name="namedString">
  <simpleContent>
   <extension base="string">
    <attribute name="name" type="string" use="required"></attribute>
   </extension>
  </simpleContent></complexType>

 <complexType name="restrictedString">
  <simpleContent>
   <extension base="string">
    <attribute name="restrict" type="string"
   use="optional"></attribute>
   </extension>
  </simpleContent>
 </complexType>
</schema>
```

Anhang B

Service-Typen der STiL



Abkürzungsverzeichnis

AAC Advanced Audio Coding

API Application Programming Interface

CORBA Common Object Request Broker Architecture

DFG Deutsche Forschungsgemeinschaft

DNS Domain Name System

DNS-SD Domain Name System Service Discovery

DPWS Device Profiles for Web Services

GENA General Event Notification Architecture

GPAP General Purpose Access Point

GPRS General Packet Radio Service

GPS Global Positioning System

GRK Graduiertenkolleg

GUI Graphical User Interface

HTML Hypertext Markup Language

HTTP Hypertext Transfer Protocol

IP Internet Protocol

IPP Internet Printing Protocol

ISBN International Standard Book Number

ISO International Organization for Standardization

IT Information Technology

LLMS Lehr- und Lernmanagementsystem

JDK Java Development Kit

JVM Java Virtual Machine

mDNS Multicast Domain Name System

MP3 MPEG-1 Audio Layer 3

MuSAMA Multimodal Smart Appliance Ensembles for Mobile Applications

OSDA Open Service Discovery Architecture

OSGi Open Services Gateway initiative

OSI Open Systems Interconnection

P2P Peer-to-Peer

PAN Personal Area Network

PDF Portable Document Format

PTR Pointer (DNS)

RMI Remote Method Invocation

SDP Service Description Protocol

RFC Requests for Comments

SL Second Life

SLP Service Location Protocol

SOA Service-oriented Architecture

SOAP Simple Object Access Protocol

SRV Service (DNS)

SSDP Simple Service Discovery Protocol

STIA Service Technology Independence Architecture

STiL Service Technology-independent Language

SunSPOT Sun Small Programmable Object Technology

TCP Transmission Control Protocol

TXT Text (DNS)

UDP User Datagram Protocol

UDDI Universal Description, Discovery and Integration

UMTS Universal Mobile Telecommunications System

UPnP Universal Plug and Play

UPnP-AV UPnP Audio and Video

URI Uniform Resource Identifier

USQL Unified Service Query Language

UUID Universal Unique Identifier

VLE Virtual Learning Enwironment

W3C World Wide Web Consortium

WLAN Wireless Local Area Network

WSDL Web Service Description Language

WSI Web Service Inspection

WSIL Web Service Inspection Language

XML Extensible Markup Language

Literaturverzeichnis

[Abowd 99] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, P. Steggles. Towards a Better Understanding of Context and Context-Awareness. Tagungsband: Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing, Seiten 304–307, Karlsruhe, Germany, Sept. 1999.

[Al-Masri 07] E. Al-Masri, Q. H. Mahmoud. QoS-based Discovery and Ranking of Web Services. Tagungsband: Proceedings of the 16st International Conference on Computer Communications and Networks, Seiten 529–534, Honolulu, USA, 2007. IEEE.

[Allard 03] J. Allard, V. Chinta S., Gundala, G. G. Richard. Jini Meets UPnP: An Architecture for Jini/UPnP Interoperability. Tagungsband: SAINT '03: Proceedings of the 2003 Symposium on Applications and the Internet, Seiten 268–275, Washington, DC, USA, Jan. 2003.

[Ballinger 01] K. Ballinger, P. Brittenham, A. Malhotra, W. A. Nagy, S. Pharies. Web Services Inspection Language (WS-Inspection) 1.0. IBM, Nov. 2001.

[Bieberstein 06] N. Bieberstein, S. Bose, M. Fiammante, K. Jones, R. Shah. Service-Oriented Architecture (SOA) Compass - Business Value, Planning, and Enterprise Roadmap. IBM Press, Upper Saddle River, NJ, USA, 2006.

[Bluetooth SIG 09] Bluetooth SIG. Specification of the Bluetooth System (Version 4.0). Bluetooth SIG, Inc., Dez. 2009.

[Brumitt 00] B. Brumitt, B. Meyers, J. Krumm, A. Kern, S. Shafer. Easyliving: Technologies for intelligent environments. Seiten 12–29. Springer., 2000.

[Cheshire 08] S. Cheshire, M. Krochmal. DNS-Based Service Discovery. Apple Inc., Sept. 2008.

[Cheshire 09] S. Cheshire, M. Krochmal. Multicast DNS. Apple Inc., Sept. 2009.

[Chinnici 07] R. Chinnici, J. Moreau, A. Ryman, S. Weerawarana. Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language. W3C, Juni 2007.

[Christensen 01] E. Christensen, F. Curbera, G. Meredith, S. Weerawarana. Web Services Description Language (WSDL) 1.1. W3C, Maerz 2001.

[Clement 04] L. Clement, A. Hately, C. von Riegen, T. Rogers. UDDI Version 3.0.2 - UDDI Spec Technical Committee Draft. OASIS, Okt. 2004.

[Colan 04] M. Colan. Future of Web Services: SOA and Business Processes (Talk). IBM Corporation, 2004.

[Cooperstock 97] J. R. Cooperstock, S. S. Fels, W. Buxton, K. C. Smith. Reactive environments: Throwing away your keyboard and mouse. Communications of the ACM, 40:65–73, 1997.

[Corradi 04] A. Corradi, R. Montanari, D. Tibaldi. Context-Based Access Control Management in Ubiquitous Environments. Tagungsband: Third IEEE International Symposium on Network Computing and Applications (NCA'04), Seiten 253–260, Boston, MA, USA, 2004. IEEE.

[data-quest GmbH 05] data-quest GmbH. Produktbroschüre Stud.IP für Hochschulen. Juni 2005.

[Dressler 08] E. Dressler, R. Zender, U. Lucke, D. Tavangarian. A new Architecture for Heterogeneous Context Based Routing. Tagungsband: Proceedings of the 13th International CSI Computer Conference, Kish Island, Iran, Maerz 2008.

[Dressler 09] E. Dressler, R. Zender, U. Lucke, D. Tavangarian. A Multi-Layer Approach for Cross-Technology Communication in a Pervasive Community. Tagungsband: Third International Workshop on Information Fusion and Dissemination in Wireless Sensor Networks (SensorFusion'09), Toronto, Canada., Juli 2009. IEEE Press.

[Dressler 10] E. Dressler. Zuverlässige, kontextbezogene Geräte-Sensoren-Netzwerke in horizontalen und vertikalen drahtlosen Kommunikationsarchitekturen (Dissertationsschrift). Universität Rostock, Rostock, Germany, 2010.

[Driscoll 09] D. Driscoll, Antoine Mensch. Devices Profile for Web Services Version 1.1. OASIS, Mai 2009.

[Edwards 06] W. K. Edwards. Discovery systems in ubiquitous computing. IEEE Pervasive Computing, 5/2:70–77, April 2006.

[Encarnação 05] J. L. Encarnação, T. Kirste. Ambient Intelligence: Towards Smart Appliance Ensembles. In M. Hemmje et al, Hrsg., Tagungsband: From Integrated Publication and Informations Systems to Virtual Information and Knowledge Environments, Seiten 261–270. Springer LNCS 3379, 2005.

[Freeman 02] E. Freeman, S. Hupfer, K. Arnold. Javaspaces Principles, Patterns, and Practice. Addison-Wesley Longman, Amsterdam, 2002.

[Gläser 08] M. Gläser. Service-basierte Integration von dynamischen, interaktiven Medienströmen in die Lernplattform Stud.IP. Universität Rostock, Studienarbeit, 2008.

[Gudgin 07] M. Gudgin, M. Hadley, N. Mendelsohn, J. Moreau, H. F. Nielsen, A. Karmarkar, Y. Lafon. SOAP Version 1.2 Part 1: Messaging Framework (Second Edition). W3C, April 2007.

[Guttman 99a] E. Guttman, J. Kempf. Automatic discovery of thin servers: SLP, Jini and the SLP-Jini Bridge. Tagungsband: IECON '99 Proceedings of the 25th Annual Conference of the IEEE, Seiten 722–727, San Jose, CA, USA, 1999.

[Guttman 99b] E. Guttman, C. Perkins, J. Veizades, M. Day. Service Location Protocol, Version 2 (RFC2608). The Internet Society, Juni 1999.

[Hackbusch 03] W. Hackbusch, H. R. Schwarz, E. Zeidler. Teubner - Taschenbuch der Mathematik: TEIL 1. Vieweg+Teubner, 2. Auflage Auflage, Nov. 2003.

[Hainsworth 08] R. Hainsworth. Exploring Second Life as a beneficial virtual learning environment. The University of Edinburgh, Studie, 2008.

[Heider 06] T. Heider, T. Kirste. Building smart environments: The Ensemble Challenge. Information Management and Consulting, 4/06:6–12, 2006.

[Heinemann 08] S. Heinemann. P2P-gestützte Dienstsuche in Community-Netzen. Universität Rostock, Masterarbeit, 2008.

[Henricksen 01] K. Henricksen, J. Indulska, A. Rakotonirainy. Infrastructure for Pervasive Computing: Challenges. Tagungsband: Workshop on Pervasive Computing at INFORMATIK 01, Seiten 214–222, Vienna, Austria, Sept. 2001.

[IMC AG 09] IMC AG. Lecturnity Campus - Rapid Authoring Tool. Dez. 2009.

[Jeronimo 03] M. Jeronimo, J. Weast. UPnP Design by Example. Intel Press, Hillsboro, USA, Mai 2003.

[Johnson 09] L. Johnson, A. Levine, R. Smith. The 2009 Horizon Report. Technischer Bericht, The New Media Consortium, Austin, TX, USA, 2009.

[Johnson 10] L. Johnson, A. Levine, R. Smith, S. Stone. The 2010 Horizon Report. Technischer Bericht, The New Media Consortium, Austin, TX, USA, 2010.

[Kaffille 07] S. Kaffille, K. Loesing. Open Chord version 1.0.4 - User's Manual. Otto-Friedrich-Universität Bamberg, Distributed and Mobile Systems Group, Okt. 2007.

[Kemp 06] J. Kemp, D. Livingstone. Putting a Second Life 'Metaverse' Skin on Learning Management Systems. Tagungsband: Second Life Education Workshop at SL Community Convention, San Francisco, CA, USA, Aug. 2006.

[Kirste 08] T. Kirste. GRK 1424 MuSAMA: Multimodal Smart Appliance Ensembles for Mobile Applications. In M. Diehl, H. Libskoch, R. Meyer, C. Storm, Hrsg., Tagungsband: Proceedings des gemeinsamen Workshops der Graduiertenkollegs 2008, Seiten 75–76, Berlin, Germany, 2008. GITO-Verlag.

[Langheinrich 03] M. Langheinrich, F. Mattern. Digitalisierung des alltags - was ist pervasive computing? Bundeszentrale für politische Bildung, B 42:6–12, 2003.

[Lee 03] C. Lee, S. Helal. Context Attributes: An Approach to Enable Context-awareness for Service Discovery. Tagungsband: In Proceedings of the 2003 Symposium on Applications and the Internet, Seiten 22–30, Gainesville, FL, USA, Jan. 2003.

[Lehsten 08] P. Lehsten, A. Thiele, R. Zilz, E. Dressler, R. Zender, U. Lucke, D. Tavangarian. Dienste-basierte Kopplung von virtueller und Präsenzlehre. Tagungsband: DeLFI 2008: Die e-Learning Fachtagung Informatik, Lübeck, Germany, Sept. 2008.

[Lehsten 10] P. Lehsten, R. Zender, U. Lucke, D. Tavangarian. A Service-oriented Approach towards Context-aware Mobile Learning Management Systems. Tagungsband: PerEL 2010, Workshop im Rahmen der 8th IEEE International Conference on Pervasive Computing and Communications (PerCom), Mannheim, Deutschland, April 2010. IEEE Computer Society.

[Lenz 08] D. Lenz. Realisierung einer Matching Unit für den Vergleich von XML-Daten der Service Technology-independent Language (STiL). Universität Rostock, Studienarbeit, 2008.

[Lenz 09] D. Lenz. Kontextorientierte SOA-Interoperabilität für Broadcast-Szenarien. Universität Rostock, Diplomarbeit, 2009.

[Limam 07] N. Limam, J. Ziembicki, R. Ahmed, Y. Iraqi, D. T. Li, R. Boutaba, F. Cuervo. Osda: Open service discovery architecture for efficient cross-domain service provisioning. Elsevier Computer Communications, 30:546–563, Feb. 2007.

[Lindemann 08] S. Lindemann. Dienstebasierter Zugriff auf Sensordaten. Universität Rostock, Studienarbeit, 2008.

[Lucke 07] U. Lucke, D. Tavangarian. Aktueller Stand und Perspektiven der eLearning-Infrastruktur an deutschen Hochschulen. Tagungsband: Die 5. e-Learning Fachtagung Informatik DeLFI 2007, Siegen, Germany, Sept. 2007.

[Machuska 07] I. Machuska. Konzeption und Implementierung eines Adaptersystems zur Integration spezieller Service-Discovery-Protokolle in den UDDI-Standard. Universität Rostock, Masterarbeit, 2007.

[Melzer 07] I. Melzer. Service-orientierte Architekturen mit Web Services: Konzepte - Standards - Praxis. Elsevier, München, 2007.

[Müller 07] A. Müller, M. Leidl. Virtuelle (Lern-)Welten: Second Life in der Lehre. Portal e-teaching.org, Dez. 2007.

[Mostéfaoui 04] S. K. Mostéfaoui, B. Hirsbrunner. Context Aware Service Provisioning. Tagungsband: Proceedings of the IEEE/ACS International Conference, Seiten 71–80, Beirut, Libanon, 2004. IEEE.

[Newmarch 06] J. Newmarch. Foundations of Jini 2 Programming. Apress, New York, 2006.

[Panganelli 09] F. Panganelli, D. Parlanti, N. Francini, D. Giuli. A SOA-Based Mobile Guide to Augment Tourists Experiences with User-Generated Content and Third-Party Services. Tagungsband: Proceedings of the Fourth International Conference on Internet and Web Applications and Services, Seiten 435–442, Tokyo, Japan, 2009. IEEE.

[Pantazoglou 06] M. Pantazoglou, A. Tsalgatidou, G. Athanasopoulos, T. Pilioura. A Unified Approach for the Discovery of Web and Peer-to-Peer Services. Tagungsband: Proceedings of the IEEE International Conference on Web Services (ICWS'06), Seiten 901–902, Chicago, Illinois, USA, Sept. 2006.

[Park 06] S. Park, D. Kim, B. Kang. Context-aware Middleware Architecture for Intelligent Service in Mobile Environment. Tagungsband: Proceedings of the Sixth IEEE International Conference on Computer and Information Technology, Seiten 240–245, Washington D.C., USA, 2006.

[Plociennik 10] C. Plociennik, H. Wandke, T. Kirste. What Influences User Acceptance of Ad-hoc Assistance Systems? – A Quantitative Study. Tagungsband: Proceedings of MMS, Göttingen, Germany, Feb. 2010.

[Satyanarayanan 01] M. Satyanarayanan. Pervasive computing: vision and challenges. IEEE Personal Communications, 8:10–17, Aug. 2001.

[Scholler 08] D. Scholler. 2008 SOA User Survey: Adoption Trends and Characteristics. Technischer Bericht, Gartner Research, Sept. 2008.

[Schulte 96] W. R. Schulte, Y. V. Natis. 'Service Oriented' Architectures, Part 1, Research Note SPA-401-06. Technischer Bericht, Gartner Research, April 1996.

[Seungyong 06] L. Seungyong, L. Younglok, L. Hyunghyo. Jini-Based Ubiquitous Computing Middleware Supporting Event and Context Management Services. Tagungsband: Ubiquitous Intelligence and Computing (LNCS 4159/2006), Seiten 786–795, Berlin, Aug. 2006. Springer.

[Srisuresh 01] P. Srisuresh, K. Egevang. Traditional IP Network Address Translator (RFC3022). The Internet Society, Jan. 2001.

[Steinberg 05] D. Steinberg, S. Cheshire. Zero Configuration Networking: The Definitive Guide. O'Reilly Media, Dez. 2005.

[Stoica 01] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. Tagungsband: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications, Seiten 149–160, San Diego, CA, USA, 2001.

[Strang 04] T. Strang, C. Linnhoff-Popien. A Context Modeling Survey. Tagungsband: Workshop on Advanced Context Modeling, Reasoning and Management at The Sixth International Conference on Ubiquitous Computing, Nottingham, Grossbritannien, 2004.

[Sun Microsystems 07] Sun Microsystems. Sun Small Programmable Object Technology (Sun SPOT) - Developers Guide. April 2007.

[Tavangarian 01] D. Tavangarian, F. Burchert, U. Lucke, S. Malo, K. Nölting, G. Pöplau, H. R. Vatterott. Untersuchung der Einsatzmöglichkeiten von Notebooks in Lehre und Ausbildung an Hochschulen, Studie im Auftrag des BMBF. Universität Rostock, 2001.

[Tavangarian 09] D. Tavangarian, U. Lucke. Pervasive University - A Technical Perspective (Die Pervasive University aus technischer Perspektive). it - Information Technology, 51(1):6–13, Jan. 2009.

[The OSGi Alliance 03] The OSGi Alliance. OSGi Service Platform. IOS Press, 2003.

[Thomanek 09] A. Thomanek, C. Schönfeldt, W. Schwelgengräber, M. Donick, D. Tavangarian. Media-based Junior Studies (MbJS) in Context of the New Learning Culture. Tagungsband: eLearning Baltics 2009. Proceedings of the 2nd International eLBa Science Conference, Seiten 27–36, Stuttgart, 2009.

[Totkov 03] G. Totkov. Virtual learning environments: towards new generation. Tagungsband: Proceedings of the 4th international conference conference on Computer systems and technologies: e-Learning, Seiten 8–16, Rousse, Bulgarien, 2003.

[Uribarren 06] A. Uribarren, J. Parra, J. P. Uribe, K. Makibar, I. Olalde, N. Herrasti. Service Oriented Pervasive Applications Based On Interoperable Middleware. Tagungsband: Proceedings of the 1st International Workshop on Requirments and Solutions for Pervasive Software Infrastructures, Dublin, Ireland, Mai 2006.

[Venezia 09] C. Venezia, N. Taylor, H. Williams, H. Doolin, I. Roussaki. Novel Pervasive Computing Services Experienced through Personal Smart Spaces. Tagungsband: Proceedings of the Tenth International Conference on Mobile Data Management: Systems, Services, and Middleware, Seiten 484–489, Taipei, Taiwan, 2009. IEEE.

[W. Isserstedt and J. Link 08] W. Isserstedt and J. Link. Internationalisierung des Studiums - Ausländische Studierende in Deutschland - Deutsche Studierende im Ausland: Ergebnisse der 18. Sozialerhebung des Deutschen Studentenwerks durchgeführt durch HIS Hochschul-Informations-System. Bundesministerium für Bildung und Forschung, Bonn/Berlin, 2008.

[Weiser 91] M. Weiser. The computer for the 21st century. Scientific American, 09/91:94–102, Sept. 1991.

[Zender 09a] R. Zender, E. Dressler, U. Lucke, D. Tavangarian. Multi-level Interoperability for Pervasive Communication Networks. Tagungsband: Proceedings of the 7th IEEE International Conference on Pervasive Computing and Communications (PerCom), Galveston, TX, USA, Maerz 2009. IEEE Computer Society.

[Zender 09b] R. Zender, E. Dressler, U. Lucke, D. Tavangarian. Pervasive Media and Messaging Services for Immersive Learning Experiences. Tagungsband: Proceedings

of PerEL 2009, Workshop at 7th IEEE International Conference on Pervasive Computing and Communications (PerCom), Galveston, TX, USA, Maerz 2009. IEEE Computer Society.

[Zender 09c] R. Zender, D. Tavangarian. Service-oriented University: Infrastructure for the University of Tomorrow. Tagungsband: International Conference on Intelligent Interactive Assistance and Mobile Multimedia Computing 2009 (IMC 2009), Seiten 73–84, Rostock, Germany, Nov. 2009. Springer.

[Zender 10a] R. Zender, U. Lucke, D. Tavangarian. Integrating Context Awareness with SOA: Case Studies for Service-oriented and Context-based Architectures. Tagungsband: Workshop on Context-Systems Design, Evaluation and Optimisation (CoSDEO) im Rahmen der 23rd International Conference on Architecture of Computing Systems (ARCS), Feb. 2010.

[Zender 10b] R. Zender, U. Lucke, D. Tavangarian. SOA Interoperability for Large-scaled Pervasive Environments. Tagungsband: 5th IEEE Int. Workshop on Service Oriented Architectures in Converging Networked Environments (SOCNE) im Rahmen der IEEE 24th Int. Conference on Advanced Information Networking and Applications (AINA), Perth, Australia, April 2010. IEEE Computer Society.

[Zhu 05] F. Zhu, M. W. Mutka, L. M. Ni. Service discovery in pervasive computing environments. IEEE Pervasive Computing, 4/4:81–90, Okt. 2005.

[Zuidweg 03] M. Zuidweg, F.J. Goncalves, M.J. van Sinderen. Using P2P in a Web Services-based Context-aware Application Platform. Tagungsband: EUNICE 2003 9th Open European Summer School and IFIP WG6.3 Workshop on Next Generation Networks, Seiten 238–243, Balaton-fured, Ungarn, 2003. Budapest University of Technology and Economics.