Service-Interoperabilität in Ubiquitous Computing Umgebungen

Dissertation

an der

Fakultät für Mathematik, Informatik und Statistik
der
Ludwig-Maximilians-Universität München

vorgelegt von

Thomas Strang

Datum der Einreichung: 20. Oktober 2003

Service-Interoperabilität in Ubiquitous Computing Umgebungen

Dissertation

an der

Fakultät für Mathematik, Informatik und Statistik
der
Ludwig-Maximilians-Universität München

vorgelegt von

Thomas Strang

Datum der Einreichung: 20. Oktober 2003
Datum des Rigorosums: 22. Dezember 2003

Erster Berichterstatter: **Prof. Dr. Claudia Linnhoff-Popien**, LMU München
Zweiter Berichterstatter: **Prof. Dr. Martin Wirsing**, LMU München
Auswärtiger Gutachter: **Prof. Dr. Alexander Schill**, TU Dresden

Zusammenfassung

Die vorliegende Arbeit präsentiert einen Ansatz zur Modellierung, Spezifikation und Auswertung von kontextuellem Wissen. Dieser Ansatz ermöglicht die Bestimmung kontextueller Interoperabilität von Diensten in Ubiquitous Computing Umgebungen. Grundlegend ist dabei das Konzept der kontextadaptiven Dienstnutzung, die sowohl die kontextadaptive Dienstauswahl als auch Dienstausführung umfaßt.

Kern des Ansatzes ist die Verwendung von *Ontologien*. Dadurch können einerseits die Konzepte des Modells, sowie darauf basierende Unterkonzepte und Fakten in einheitlicher Weise als verteiltes, kontextuelles Wissen definiert werden. Andererseits können ontologische Reasoner gewinnbringend zur Auswertung von kontextuellem Wissen eingesetzt werden.

Das Modell und verschiedene Integrationselemente werden mit ausgewählten Ontologiesprachen umgesetzt. Daraus resultiert unter dem Oberbegriff der *Context Ontology Language* eine nicht monolithische Sprache zur Spezifikation von kontextuellem Wissen und dessen Integration in verteilte Dienstumgebungen.

Kontextuelle Interoperabilität zwischen den an einer Dienstinteraktion beteiligten Komponenten kann durch Anwendung dieser Sprache erreicht werden. Mit ihr erstellte Spezifikationen bilden insbesondere eine Grundlage zur Bestimmung von kontextueller Kompatibilität und Ersetzbarkeit. Dies wird u.a. detailliert am Beispiel *kontextbasierter Service Handover* vorgestellt. Darüber hinaus werden weitere Anwendungsmöglichkeiten der Sprache z.B. im Bereich der Zusammenführung der Interoperabilitätsebenen oder als Meta-Sprache zur Abbildung zwischen verschiedenen Kontextmodellen vorgestellt.

Summary

This thesis presents a novel context modelling approach. Its implementation is used as a key element for determining contextual service interoperability in ubiquitous computing environments.

The approach is based on ontologies, providing a uniform way for specifying the model's core concepts as well as an arbitrary amount of subconcepts and facts, altogether describing the contextual knowledge in a distributed system. This contextual knowledge is evaluated using ontology reasoners.

The model has been implemented applying selected ontology languages. These implementations build up the core of a non monolithic *Context Ontology Language*, which is supplemented by integration elements such as schema extensions for Web Services and others.

Beyond determination of service interoperability in terms of contextual compatibility and substitutability, this language is used to support context-awareness in distributed service frameworks for various applications. A novel approach of contextual motivated *non-carrier service handovers* is presented as one of the applications.

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

In verteilten Systemen ist die *Interoperabilität* eine Grundvoraussetzung für jede Interaktion zwischen den dezentralen Komponenten. Interoperabilität wurde von P. Wegner im Jahre 1996 definiert als die „Fähigkeit von zwei oder mehr Softwarekomponenten, trotz unterschiedlicher Programmiersprachen, Interfaces oder Ausführungsplattformen, zu kooperieren“ [172], basierend auf Vereinbarungen zwischen den Interaktionspartnern über Form und Nutzen der ausgetauschten Daten und erbrachten Dienste [80]. Mitte/Ende der neunziger Jahre wurden die Untersuchungen zum Thema Interoperabilität intensiviert und vor allem auf der Ebene der Anwendungen und Dienste unter dem Oberbegriff *Service-Interoperabilität* verfeinert. In der Literatur (z.B. [80, 164]) wird bei der Service-Interoperabilität üblicherweise auf die drei Module *signature level*, *protocol level* und *semantic level* verwiesen (eine vertiefende Einführung der Terminologie erfolgt in Kapitel 2). In jedem dieser Module wird Interoperabilität durch Anwendung eines gemeinsamen Verständnises erreicht. Dieses gemeinsame Verständnis, *shared understanding* oder *shared knowledge* genannt, repräsentiert das Wissen über die Umstände, unter denen die Interoperabilität von zwei oder mehr Komponenten in einem verteilten System gegeben ist.

Eine der modernsten Formen von verteilten Systemen sind die Ubiquitous Computing Systeme [173, 137]. Das Ubiquitous Computing Paradigma steht dabei im wesentlichen für drei neue Facetten der Nutzung von mobilen, verteilten Systemen: Spontane Vernetzung (*ad-hoc networks*), intelligente Kleinstgeräte (*smart devices*) und kontextadaptive Dienste (*context-awareness*). Mit Etablierung des Ubiquitous Computing entsteht u.a. das Problem, daß der *Kontext* [61] einer Anwendung oder eines Dienstes in den Interoperabilitätsbetrachtungen bisher nur unzureichend berücksichtigt wird.

Dieses Problem ist Gegenstand dieser Arbeit. Sie basiert auf der Idee, ein neues Modul der Service-Interoperabilität einzuführen – die *Kontextebene (context level)*. Dazu erfolgt in Kapitel 2 eine detaillierte Aufbereitung der Thematik der Interoperabilität, um nach einigen grundlegenden Betrachtungen zum Kontextbegriff eine Einordnung der neuen Interoperabilitätsebene in die bestehenden Ebenen der Interoperabilität vornehmen zu können.

Die Spezifikation von kontextuellem Wissen (*contextual shared knowledge*) auf dieser neuen Ebene der Interoperabilität ermöglicht die Auswertung der kontextuellen Abhängigkeiten

der verschiedenen an einer Dienstinteraktion beteiligten Komponenten sowohl zur Entwurfszeit als auch zur Laufzeit vor und während einer Dienstinteraktion. Zur Spezifikation von kontextuellem Wissen wird ein geeignetes Modell benötigt, dessen Semantik kontextuelle Zusammenhänge auf Konzepte des Modells projiziert. Solch ein Modell wird im Rahmen der hier vorliegenden Arbeit entwickelt und in Kapitel 3 vorgestellt.

Aus diesem Modell wird anschließend eine Sprache zur Spezifikation kontextueller Interoperabilität abgeleitet, die als *Context Ontology Language* bezeichnet wird. Mit den Elementen dieser Sprache können kontextuelle Fakten und kontextuelle Beziehungen spezifiziert werden. Kontextuelle Interoperabilität kann durch Anwendung der so erstellten Spezifikationen ermöglicht werden. Die Sprache selbst setzt auf Ontologien [163] auf, so daß neben einer Taxonomie von kontextuellen Fakten auch implizites Wissen über den Kontext in Form von Inferenzregeln [60] explizit angegeben werden kann.

Die Bereitstellung von Kontextinformationen und ihre Anwendung in Architekturen zur Dienstsuche und Dienstausführung (*context-aware service framework*) bietet enorme Vorteile. Die Context Ontology Language und damit das ihr zugrundeliegende Modell dienen als Basis für verschiedene Komponenten und Aufgaben der vorgestellten Gesamt-Systemarchitektur. Durch ihre Anwendung kann die kontextuelle Service-Interoperabilität erreicht und zur Laufzeit aus verschiedenen Perspektiven überprüft werden. Das Kapitel 4 zeigt das Potential der Context Ontology Language als Instrument zur Spezifikation und Abfrage von kontextuellem Wissen, eingebettet in ontologisches Reasoning und mit einem Fokus auf der *Kompatibitlität* als Perspektive der Interoperabilität.

Der Vorteil kontextadaptiver Dienstinteraktionen wird insbesondere auch bei der Betrachtung von Handover-Verfahren deutlich. Durch die Berücksichtigung des Kontextes einer Dienstinteraktion können Erfahrungen von klassischen Kommunikationsdiensten [4] auf Handover-Verfahren bei höherwertigen Diensten übertragen und aus diesem Bereich bekannte Protokolle angewandt werden. In Kapitel 5 wird dazu gezeigt, wie das jeweilige Handover-Verfahren in allgemeinen Dienstumgebungen insbesondere durch die Einbeziehung der kontextuellen Interoperabilitätsebene als Entscheidungsgrundlage davon profitiert. Voraussetzung für Handover-Verfahren aller Art ist die Interoperabilität verschiedener Komponenten des verteilten Systems insbesondere aus der Perspektive der *Ersetzbarkeit*, die daher einen Schwerpunkt der Betrachtungen in diesem Kapitel darstellt. Die notwendigen Erweiterungen des Service Frameworks werden vorgestellt und anhand ausgewählter Beispiele demonstriert.

Kapitel 2

Service-Interoperabilität

Moderne IT-Infrastrukturen sind in der Regel nach dem Prinzip der verteilten Systeme entworfen, bei dem eine Menge voneinander unabhängiger Computer miteinander vernetzt sind, die als kohärentes Gesamtsystem sowohl den Anwendern als auch verschiedenen Applikationen zur Verfügung stehen [159]. Eine der wichtigsten Eigenschaften verteilter Systeme ist die Kooperation verschiedener Komponenten zwecks Erreichung gemeinsamer Ziele [105]. Dabei ist ein besonderes Merkmal, daß die Unterschiede der beteiligten Komponenten bezüglich der Hardware, des Netzwerks oder der Software, sowie die Art der Kommunikation zwischen den Komponenten durch das System angeglichen und damit vor dem Benutzer versteckt werden. Durch Anpassung der Unterschiede der beteiligten Komponenten bezüglich der Hardware, des Netzwerks oder der Software in der Middleware können Anwender und Applikationen mit/in einem verteilten System in konsistenter und einheitlicher Weise interagieren.

Jede Kooperation erfordert von den beteiligten Partnern ein „gemeinsames Verständnis“, der sogenannten *Interoperabilität*, über Form und Nutzen der ausgetauschten Daten und erbrachten Dienste. Interoperabilität wird dabei in der Regel durch die Anwendung einer gemeinsamen Spezifikation gewährleistet. Nach [172] ist Interoperabilität die Fähigkeit von zwei oder mehr Softwarekomponenten, trotz unterschiedlicher Programmiersprachen, Interfaces oder Ausführungsplattformen, miteinander kooperieren zu können. Dies ist allerdings nur eine sehr allgemeine Definition der Interoperabilität, wie im Folgenden deutlich wird.

Dieses Kapitel setzt sich detailliert mit der Interoperabilität von Diensten auseinander. Zunächst erfolgt ein kurzer Abriß über die Historie der Forschungen im Bereich der Interoperabilität von Komponenten in verteilten Systemen (Abschnitt 2.1). Anschließend werden die Betrachtungen auf der Ebene der Service Interoperabilität intensiviert, wobei insbesondere die Einführung einer neuen Unterebene motiviert wird (Abschnitt 2.2). Diese neue Unterebene berücksichtigt speziell die kontextuellen Gesichtspunkte einer Dienstnutzung, welche dann in Abschnitt 2.3 herausgearbeitet werden. Es zeigt sich, daß diese kontextuellen Gesichtspunkte durch das Paradigma des Ubiquituous Computing impliziert werden, bzw. daß Dienstinteraktionen in Ubiquitous Computing Umgebungen enorm von der Anpassung an die kontextuellen Gegebenheiten profitieren. Deshalb werden diese in Abschnitt 2.4 aufgearbeitet und ihre Integrationsmöglichkeiten in Abschnitt 2.4.3 gezeigt.

2.1 Interoperabilität in verteilten Systemen

Interoperabilität ist ein Ziel, das bereits frühzeitig bei der Entwicklung und Erweiterung von verteilten Systemen angestrebt wird, und erfordert in der Regel eine Middleware- oder Plattform-Komponente zur Laufzeit. Sie erleichtert die Entwicklung in Bezug auf die Wiederverwendbarkeit von Komponenten, die gewöhnlich an unterschiedlichen Orten von verschiedenen Leuten zu unterschiedlichen Zeiten entwickelt werden.

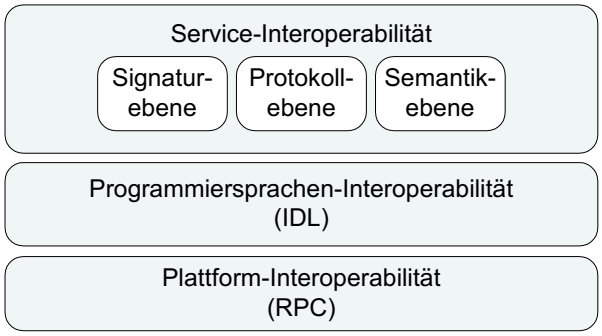

Abbildung 2.1: Klassische Ebenen der Interoperabilität

Innerhalb der letzten 20 Jahre wurde die Interoperabilität auf vielen verschiedenen Ebenen betrachtet und analysiert (siehe Abbildung 2.1). Trotz einer Vielzahl von Untersuchungen zur Interoperabilität sind die vorgeschlagenen Ansätze zur Lösung der dabei auftretenden Probleme auch heute noch unvollständig. Insbesondere der Kontext einer Applikation oder eines Dienstes wird nur unzureichend in den Interoperabilitätsbetrachtungen berücksichtigt. Dies begründet im Folgenden die Motivation zur Einführung einer neuen Ebene der Interoperabilität, der Kontextebene, in Abschnitt 2.2.6. Zunächst erfolgt jedoch eine Betrachtung der „klassischen“ Ebenen der Interoperabilität.

2.1.1 Interoperabilität auf Plattformebene

Bereits zu Beginn der Entwicklung weg von zentralistischen Ansätzen hin zu verteilten Systemen bestehend aus vernetzten leistungsfähigeren Einzelrechnern Mitte der achtziger Jahre wurde mit dem Vorschlag des *Remote Procedure Calls (RPC)* der erste Schritt auf der Interoperabilitäts-Evolutionsleiter erklommen. Der RPC ermöglicht einen rechnerübergreifenden Funktionsaufruf nach folgendem Schema: Ein Prozeß ruft eine Subroutine auf, die tatsächlich auf einem entfernten Rechner ausgeführt werden soll. In diesem Fall wird der aufrufende Prozeß auf dem lokalen Rechner vorübergehend unterbrochen, die Aufrufparameter werden an den entfernten Rechner übertragen, die Subroutine wird auf dem entfernten Rechner ausgeführt, das Ergebnis der Subroutine wird wieder auf den lokalen Rechner zurück übertragen und dem dann reaktivierten Prozeß übergeben. Für den Anwender bleibt diese Vorgehensweise im Normalfall transparent.

Die Anwendung von

- normierten Verfahren zur Codierung (*marshalling*) bzw. De-Codierung (*unmarshalling*) der Aufrufparameter
- einheitliche Verfahren zum dynamischen Binden und zur Fehlerbehandlung

ermöglichen den Einsatz völlig unterschiedlicher Hardware und/oder Betriebssysteme auf der Seite des Clients und des Servers. Somit stellt der RPC die Interoperabilität zwischen Hardwarekomponenten mit unterschiedlicher Basistechnologie sicher. Man spricht von einer sogenannten *Plattform-Interoperabilität*.

Zur Entwicklungszeit helfen in der Regel geeignete Tools, um aus einer formalen Spezifikation der Signatur der Prozedur, die zur Laufzeit aufgerufen werden soll, sogenannte *Stubs* sowohl für den Client- als auch für den Server-Teil zu generieren. Diese übernehmen zur Laufzeit das marshalling/unmarshalling sowie die Kommunikation zwischen den Komponenten über das Netzwerk. Da der Programmierer diese Stubs zur Entwicklungszeit statisch hinzubinden kann, bleiben die RPC-spezifischen Details in der Regel auch für den Programmierer transparent, und die Plattform-Interoperabilität bleibt gewährleistet.

Selbst zur Laufzeit kann eine Middleware-Komponente im Rahmen des sogenannten dynamischen Bindens die Plattform-Interoperabilität sicherstellen. Beim dynamischen Binden wird die Kompatibilität einer Client- und einer Server-Anwendung zur Laufzeit festgestellt. Die dabei ursprünglich verwendeten Techniken zur Lokalisierung und Vermittlung von geeigneten Partnern waren jedoch in der Regel nicht standardisiert und hingen ebenfalls stark von der verwendeten Programmiersprache ab. Diesem Problem ist man mit der Interoperabilität auf Programmiersprachenebene entgegen getreten.

2.1.2 Interoperabilität auf Programmiersprachenebene

Ende der achtziger Jahre wurde mit dem Konzept der *Interface Definition Language (IDL)* die nächste Ebene auf der Interoperabilitäts-Evolutionsleiter erreicht.

Diese ist beispielsweise Bestandteil der CORBA Architektur. CORBA ist die Spezifikation einer offenen Architektur, welche die Interoperabilität von Anwendungen in einer verteilten Umgebung zum Ziel hat [120, 106]. Neben einer zentralen Komponente der CORBA Architektur, die u.a. den RPC Mechanismus als Middleware-Komponente zur Verfügung stellt (dem sogenannten *Object Request Broker (ORB)*), ist eines der wesentlichen Merkmale von CORBA die Trennung zwischen Schnittstellenbeschreibung und Implementierung beim zugrundeliegenden Objektmodell. Zur Beschreibung der Schnittstellen eines Objekts dient die eben erwähnte Interface Definition Language, die unabhängig von einer bestimmten Programmiersprache abstrakt die Signatur der Schnittstellen deklariert. Die Abbildung von IDL auf verschiedene Programmiersprachen wie Java oder C++ (*language mapping*) stellt dabei zusammen mit der Spezifikation der IDL selbst das „gemeinsame Verständnis“ dar, welches Grundlage der *Programmiersprachen-Interoperabilität* ist. Diese Art der Interoperabilität ermöglicht die Kooperation zwischen Objekten, die in verschiedenen Programmiersprachen implementiert sein können.

Auch in anderen Architekturen besteht die Möglichkeit der Interaktion von Komponenten, die in unterschiedlichen objektorientierten Programmiersprachen implementiert sind, z.B. DCOM [2]. Den meisten Architekturen ist dabei gemeinsam, daß sie die verteilten Technologien als Mittel der Abstraktion von Kommunikationsschichten und die Objektorientierung als Konzept der Kapselung und Wiederverwendbarkeit miteinander verknüpfen. Dadurch wird erreicht, daß Anwendungen zueinander interoperabel sind, die in unterschiedlichen Programmiersprachen geschrieben wurden, auf verschiedenen Betriebssystemen laufen und unterschiedliche Netzwerkprotokolle nutzen. Diese Merkmale finden sich in der Definition der Interoperabilität nach Wegner [172], die somit im wesentlichen als Definition der Programmiersprachen-Interoperabilität verwendet werden kann.

In diesem Sinne setzt die Programmiersprachen-Interoperabilität auf der Plattform-Interoperabilität auf und vereinigt diese beide Ebenen der Interoperabilität. Plattform-Interoperabilität und Programmiersprachen-Interoperabilität sind eine Grundvoraussetzung für die Kooperation von Anwendungen in verteilten Systemen, der Interoperabilität auf Anwendungsebene.

2.1.3 Interoperabilität auf Anwendungs- und Dienstebene

In den neunziger Jahren wurden die Untersuchungen zum Thema Interoperabilität weiter intensiviert und vor allem auf der Ebene der Anwendungen und Dienste verfeinert. Da ein Schwerpunkt dieser Arbeit die Interoperabilität auf dieser Ebene ist, wird diese im folgenden Abschnitt eingehender vorgestellt.

2.2 Der Begriff der Service-Interoperabilität

Das Dienstkonzept ist ein Ansatz zur Lösung von komplexen Aufgabenstellungen durch das Zusammenwirken verteilter Komponenten. Die im Rahmen dieser Arbeit verwendete allgemeine Definition eines Dienstes ist gemäß [150]:

> **Definition 1 (Dienst / Service)** *Ein Dienst (Service) ist eine eindeutig identifizierbare Instanz, die für die Beschaffung von Informationen oder für die Ausführung von Aktionen mit spezifischen Merkmalen verantwortlich ist.*

Bei der Nutzung von Diensten steht in der Regel weniger die Interaktion mit dem Menschen als die Interaktion zwischen verschiedenen Diensten im Vordergrund. Der Zugriff auf einen Dienst erfolgt über einen Dienstendpunkt (*port*), der vom einem Server bereitgestellt wird, welcher als Prozeß den spezifischen Service implementiert und für einen oder mehrere Clients zur Verfügung stellt.

Dienste sind individuell anwendbar und benötigen ein Framework zur Dienstsuche und Dienstausführung. Ein Framework ist nach [68] eine Menge kooperierender Klassen, die wiederverwendbare Designs für eine spezifische Klasse von Software darstellen. Anders ausgedrückt

stellt ein Service Framework eine Reihe von Basiskomponenten (Libraries, Services etc.) sowohl zur Entwurfszeit als auch zur Laufzeit bereit, welche die Gesamtarchitektur einer (verteilten) Anwendung sehr stark beeinflussen bzw. festlegen. Einige konkrete, existierende Service Frameworks mit Schwerpunkt auf Architekturen für mobile Dienste und Endgeräte werden in Abschnitt 2.4.3 vorgestellt.

In verteilten Dienst-Architekturen wie z.B. Jini [29] erfolgt eine konsequente Dienstorientierung, die als Paradigma hat, daß alles (Hardware, Software, Benutzer etc.) ein Dienst ist. Dienste werden über ihre Interfaces definiert und stellen über diese ihre Funktionalität zur Verfügung. Hier lassen sich viele Parallelen zur Objektorientierung feststellen, wo alles als Objekt interpretiert wird und Objekte durch ihre Klasse mitsamt ihrer Attribute und Methoden charakterisiert werden. Ähnlich wie Klassen die Interna der Implementierung kapseln, sind die Interna einer Dienstimplementierung nach außen nicht sichtbar, da lediglich das Interface eines Dienstes öffentlich ist. Analog dazu, wie eine Klasse mehrere auf das Objekt anwendbare Methoden zusammenfaßt, kann ein Service mehrere „atomare“ Operationen gruppieren, wobei diese atomaren Operation z.B. in Form von individuellen, per Remote Procedure Call (RPC) aufrufbaren Funktionen bereitgestellt werden. Diese Kapselung hat gerade in verteilten Systemen viele Vorteile, die vor allem in der Austauschbarkeit und Wiederverwendbarkeit liegen. Weitere Vorteile bietet die Kapselung z.B. hinsichtlich der Verwendung verschiedener Versionen der gleichen Softwarekomponenten, der Zugriffssteuerung, sowie dem Load-Balancing. Die Kapselung ist bei Interoperabilitätsbetrachtungen sehr nützlich, da die Trennung zwischen Interface und Implementierung bereits eine Abstraktion der Zugriffsmöglichkeiten und -parameter auf die Implementierung darstellt. Die Trennung ermöglicht eine Fokussierung eines Großteils der Interoperabilitätsbetrachtungen auf die Interfaces der Dienste. Sie erfordert aber auch eine Erweiterung der Betrachtung, inwiefern eine Dienstimplementierung und/oder eine Dienstnutzerimplementierung interoperabel zum dazugehörigen Interface ist.

In [161] wird ein zweistufiges Verfahren zur Zertifizierung der Interoperabilität vorgeschlagen, das zum Beispiel auch in [109] angewandt wird: Im ersten Schritt, der logischen Zertifizierung, werden die Spezifikationen der Komponenten auf Konsistenz, Vollständigkeit etc. geprüft. Im zweiten Schritt, der physischen Zertifizierung, wird überprüft, ob eine Implementierung den gegebenen Spezifikationen folgt. Während die logische Zertifizierung in der Regel eher durch menschlische Interaktion erfolgt, kann die physische Zertifizierung häufig sehr gut durch Simulation oder testweise Integration in bestehende Systeme erfolgen.

In der Literatur (z.B. [80, 118, 165]) wird bei der Service-Interoperabilität üblicherweise aus Gründen der Modularisierung auf die drei Unterebenen *Signaturebene* (signature level, siehe Abschnitt 2.2.2), *Protokollebene* (protocol level, siehe Abschnitt 2.2.3) und *Semantikebene* (semantic level, siehe Abschnitt 2.2.4) verwiesen (siehe auch Abbildung 2.1 auf Seite 4). Diese werden in den genannten Abschnitten eingehender vorgestellt, differenziert nach zwei unterschiedlichen Perspektiven der Interoparabilität, die in Abschnitt 2.2.1 erläutert werden.

2.2.1 Kompatibilität und Ersetzbarkeit von Diensten

Vallecillo, Hernández und Troya identifizieren in [164] zwei Eigenschaften, durch die sich Komponenten der Dienstebene auszeichnen müssen, um interoperabel zu sein: Kompatibili-

tät (*compatibility*) und Ersetzbarkeit (*substitutability*). Während die Kompatibilität für die korrekte Durchführbarkeit einer Kommunikation zwischen zwei Entitäten sorgt, ergibt sich aus der Ersetzbarkeit, daß eine Entität durch eine andere ausgewechselt werden kann.

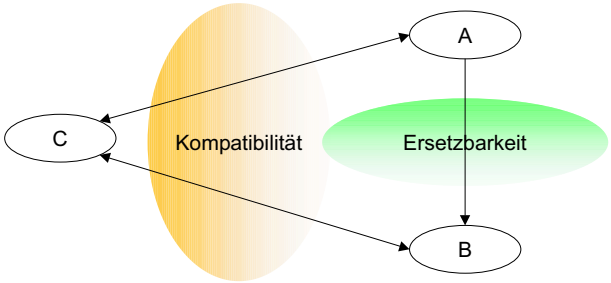

Abbildung 2.2: Kompatibilität vs. Ersetzbarkeit

Abbildung 2.2 verdeutlicht die Zusammenhänge zwischen Kompatibilität und Ersetzbarkeit. In dieser Abbildung ist die Entität C jeweils kompatibel zu den beiden Entitäten A und B, und die Entität A ist ersetzbar durch die Entität B.

Die in diesem Abschnitt vorgenommenen Kompatibilitäts- und Ersetzbarkeits-Betrachtungen gelten nicht nur für Dienste, sondern ganz allgemein für Komponenten verteilter Systeme. Um diesem Tatbestand Rechnung zu tragen, wird im Folgenden häufig der Begriff Entität verwendet, für den stellvertretend Dienst-Instanz oder Dienstnutzer-Instanz eingesetzt werden kann.

Kompatibilität

Vallecio et al. [164] beschreiben Kompatibilität als die Fähigkeit von zwei Entitäten, korrekt zusammenzuarbeiten. D.h. daß alle zwischen diesen Entitäten ausgetauschten Nachrichten und Daten von der Entität, die diese empfängt, verstanden werden. Kompatibilität ist in dieser Betrachtungsweise eine symmetrische Relation, dargestellt in Abbildung 2.2 durch den beidseitig gerichteten Pfeil. Dies ist zum einen darin begründet, daß an dieser Stelle bei der Kommunikation in der Regel von einem Dialog und nicht von einem Monolog zwischen den Entitäten ausgegangen wird. Zum anderen geht es bei der Kompatibilität vor allem um das eingangs erwähnte gemeinsame Verständnis (shared understanding), welches von allen beteiligten Entitäten identisch interpretiert wird, wodurch die Kompatibilität keine Vorzugsrichtung erhält.

Kompatibilität ist die Grundlage für jede Kommunikation zwischen Entitäten. Deshalb wird im Folgenden eine Kompatibilität immer bezüglich einer Kommunikation zwischen zwei Entitäten betrachtet. Um Kompatibilität zwischen beteiligten Entitäten zu erreichen, müssen die Entitäten bezüglich der zugrundeliegenden Kommunikation in allen vier, im Folgenden aufgeführten Ebenen kompatibel sein. Kompatibilität in einer Ebene bedeutet, daß die beteiligten Entitäten bezüglich dieser Ebene ein gemeinsames Verständnis haben.

- Dazu müssen die Entitäten grundsätzlich eine korrespondierende **Signatur (Syntax)** besitzen. Eine korrespondierende Syntax ergibt sich aus einem syntaktisch passenden gegenseitigen Verständnis. Am Beispiel der Entitäten A und C aus Abbildung 2.2 bedeutet dies, daß A alle von C zur Kommunikation mit A benutzten Ausdrücke als syntaktisch korrekt erkennt und, da es sich um eine symmetrische Relation handelt, auch C alle von A zur Kommunikation mit C benutzten Ausdrücke als syntaktisch korrekt erkennt.
- In der Ebene des **Protokolls** wird das gemeinsame Verständnis als Liste gültiger Zustandsübergänge verstanden. Hierbei nutzen alle an der Kommunikation beteiligten Entitäten eine bestimmte Anwendungsreihenfolge der in der Ebene der Signatur festgelegten Syntax. Entität C muss dafür die zur Kommunikation mit A verwendbaren Ausdrücke in der Reihenfolge benutzen, in der sie von A erwartet werden. Entsprechendes gilt für die von C zur Kommunikation mit A eingesetzten Ausdrücke.
- Desweiteren müssen die bei der Kommunikation beteiligten Entitäten der zur Kommunikation verwendeten Syntax die gleiche Bedeutung zumessen, d.h. deren **Semantik** kennen und identisch interpretieren, um ein gemeinsames Verständnis zu erlangen. Folglich müssen die Entitäten A und C ein identisches semantisches Verständnis der bei der Kommunikation zwischen A und C eingesetzten Syntax und dem Zweck der Interaktion besitzen.
- Letztlich besitzen die bei der Kommunikation beteiligten Entitäten ein gemeinsames Verständnis bezüglich des **Kontextes**, wenn sie ein identisches Verständnis der gegenseitigen externen und internen Einflußfaktoren besitzen. Die aktuellen kontextuellen Rahmenbedingungen des Dienstnutzers (Dienst und/oder Mensch) bei der Suche oder Ausführung eines Dienstes müssen also den Vorgaben des Dienstangebots entsprechen und umgekehrt.

Äquivalente Aussagen bezüglich der Kompatibilität gelten aufgrund der Symmetrie der Zeichnung auch für die Entitäten B und C aus Abbildung 2.2.

Die Feststellung der Kompatibilität ist eine der Hauptaufgaben der Dienstvermittlung. Diese wird in Abschnitt 2.4.3 eingehend diskutiert.

Ersetzbarkeit

Der Begriff der Ersetzbarkeit hat viele Facetten. Vallecio et al. [164] bezeichnen die Ersetzbarkeit von Komponenten in verteilten Systemen recht unscharf als die Fähigkeit einer Entität, eine andere zu ersetzen. Für eine detaillierte Definition der Ersetzbarkeit reicht dieser ausschließliche Bezug auf die ersetzende Entität einerseits und die zu ersetzende Entität andererseits jedoch nicht aus. Vielmehr bezieht sich die Ersetzbarkeit grundsätzlich auf zwei Entitäten bezüglich einer Interaktion mit einer dritten Entität. Dementsprechend läßt sich die Ersetzbarkeit einer Entität B definieren als die Fähigkeit dieser Entität, äquivalent zu einer anderen Entität A bezüglich einer Kommunikation zu einer dritten Entität C zu interagieren. Zur Veranschaulichung ist die Ersetzbarkeit in Abbildung 2.2 durch einen Pfeil von A nach B dargestellt. Die Ersetzbarkeit ist im Gegensatz zur Kompatibilität eine nicht symmetrische Relation, und wird daher in Abbildung 2.2 durch einen einseitig gerichteten Pfeil

dargestellt. Ist eine Entität durch eine andere ersetzbar, so gilt dadurch nicht automatisch auch die Umkehrung. In Abbildung 2.2 ist die Entität A ersetzbar durch die Entität B, B aber ist nicht ersetzbar durch A.

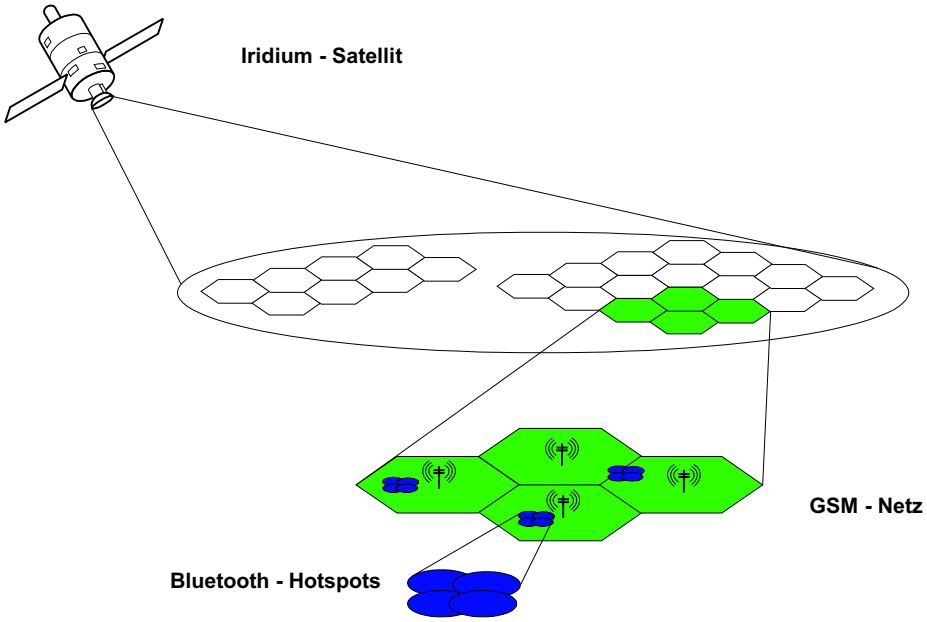

Abbildung 2.3: Überlagerung verschiedener Kommunikationssysteme

Durch eine Ersetzung, den sogenannten *Handover*, ist es für C möglich, die ursprüngliche Kommunikation zu A durch eine äquivalente Kommunikation zu B fortzusetzen [96, 95]. Es stellt sich also die Frage, wann eine Entität durch eine andere ersetzbar ist. Im allgemeinen müssen die ersetzende und die zu ersetzende Entität dazu eine äquivalente Funktionalität anbieten, z.B. als Mobilfunk-Basisstationen dem gleichen GSM-Standard folgen, siehe Abbildung 2.3 in der Mitte. In einigen Fällen ist auch eine Ersetzung von einer Entität A durch eine Entität B erlaubt, obwohl sich z.B. die Qualität der Dienste der Entitäten oder gar die gesamte Funktionalität unterscheiden, was zur Unterscheidung zwischen einer *horizontalen Ersetzung* im Falle gleicher Funktionalität und einer *vertikalen Ersetzung* im Falle unterschiedlicher Funktionalität führt (z.B. vertikaler Handover zwischen GSM und Iridium in Abbildung 2.3). Aber auch (oder gerade) bei einer vertikalen Ersetzung ist es notwendig, die Ersetzbarkeit als Perspektive der Interoperabilität auf ein gemeinsames Verständnis zu stützen, unter welchen Umständen welche Entität durch eine andere ersetzt werden darf oder kann.

In der Regel setzt eine Ersetzbarkeit einer Entität A durch eine Entität B die individuelle Kompatibilität beider Entitäten zu einer dritten Entität C voraus, mit der sie jeweils vor und nach einer Ersetzung interagieren. Soll A durch B bezüglich der Kommunikation zu C ersetzbar sein, muß C also einerseits kompatibel zu A und andererseits kompatibel zu B sein, um mit A sowie mit B eine Kommunikationsbeziehung eingehen zu können.

In bestimmten Fällen setzt eine Ersetzbarkeit einer Entität A durch eine Entität B zusätzlich eine Kompatibilität zwischen der ersetzenden und der zu ersetzenden Entität voraus. Diese Kompatibilität ist Grundlage für den direkten Informationsaustausch zwischen den Entitäten A und B, z.B. beim Austausch von Sessioninformationen bei einer zustandsbehafteten Interaktion.

Insbesondere bei einer vertikalen Ersetzung, d.h. bei einer Ersetzung zwischen nicht äquivalenten Entitäten, hat in der Regel das Service Framework eine Adaptionsaufgabe. Diese Adaptionsaufgabe erstreckt sich bei einer Ersetzung auf alle vier Ebenen der Service-Interoperabilität:

- Auf der Ebene der **Signatur** muß eine Abbildung des Vokabulars, das von der zu ersetzenden Entität verwendet wird, auf das von der ersetzenden Entität verwendete Vokabular erfolgen.
- Für die Adaption auf der Ebene des **Protokolls** sind zwei Dinge erforderlich: Zum einen die Anpassung auf die möglicherweise veränderte Anforderung bzgl. der Reihenfolge von Protokollschritten und zum anderen das Management des Zeitpunkts, zu dem eine Ersetzung evtl. nur möglich ist.
- Auf der Ebene der **Semantik** besteht in der Regel nur dann Adaptionsaufwand, wenn es sich um eine vertikale Ersetzung handelt. In diesem Fall muß die Adaption dafür sorgen, daß notwendige Vor- und Nachbedingungen eingehalten werden.
- Für den **Kontext** als vierte Ebene muß durch die Adaption eine Abbildung der externen Einflüsse der zu ersetzenden Entität auf äquivalente externe Einflüsse der ersetzenden Entität durchgeführt werden.

Die Ersetzbarkeit spielt eine wichtige Rolle im Rahmen der Service Handover. Diese werden in Kapitel 5 ausführlich diskutiert.

2.2.2 Service-Interoperabilität auf Signaturebene

Auf Signaturebene erfolgt eine exakte Beschreibung der Syntax der nutzbaren Dienste. Dies umfaßt in der Regel den Namen der Operationen sowie Typ und Reihenfolge aller Parameter eines Service-Interfaces als gemeinsames Verständnis zwischen Dienstnutzer und Dienstanbieter. Die Standardisierung der Signaturebene wird heute als am weitesten fortgeschritten angesehen.

Service-Interoperabilität auf der Signaturebene im Sinne der Kompatibilität ist durch den Bezug auf eine Menge von Interface-Beschreibungen gegeben, da diese das gemeinsame Verständnis zwischen Dienstnutzer und Dienstanbieter festlegen. Direkte (horizontale) Ersetzbarkeit auf der Signaturebene ist gegeben, wenn der ersetzende Dienst (Nachfolger) mindestens die Menge an Methoden und darin verwendete Input- und Output-Parameter kennt, die der ersetzte Dienst (Vorgänger) für die Erfüllung einer bestimmten Aufgabe anbietet.

Populäre Sprachen zur Interface Spezifikation sind CORBA's *Interface Definition Language (IDL)* [120] oder die *Web Service Definition Language (WSDL)* [55] bei Web Services.

Mit CORBA IDL kann die Signatur der Methoden eines Dienstes programmiersprachenähnlich als Interface definiert werden. Die Beschreibung des Interfaces steht anschließend vollständig zur Verfügung und zwar zeitlich vor der Entwicklung von Client- und Server-Komponente. Diese Beschreibung kann dazu verwendet werden, automatisch und toolgestützt durch einen IDL-Compiler sogenannte Client-Stubs und Server-Skeletons in einer bestimmten Programmiersprache wie C oder Java zu erstellen, welche die Entwicklung einer Client- bzw. Serverimplementierung zum einen vereinfachen und beschleunigen und zum anderen die Fehleranfälligkeit erheblich senken. Mit der Einführung des *Dynamic Invocation Interface (DII)* bzw. *Dynamic Skeleton Interface (DSI)* wurden die statischen IDL-Dateien durch eine dynamische Komponente im CORBA-Framework ergänzt. Diese Komponente des Framworks ermöglicht das Erweitern bzw. Benutzen von IDL-Interfacebeschreibungen zur Laufzeit.

WSDL basiert hingegen auf einem XML Schema, so daß Interface-Beschreibungen in WSDL immer XML Instanzdokumente sind. Auch bei WSDL ist die toolgestützte Erstellung von Client-Stubs und Server-Skeletons in einer bestimmten Programmiersprache möglich, wobei auf eine Vielzahl von XML-Parsern zurückgegriffen werden kann.

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

   targetNamespace="http://context-aware.org/wsdl/PhotoShop.wsdl"
   xmlns:this="http://context-aware.org/wsdl/PhotoShop.wsdl"
   xmlns:apachesoap="http://xml.apache.org/xml-soap"
   xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
   xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:wsdlcb="http://context-aware.org/cool/wsdl-cb.xsd">


   <!-- ================ TYPES section ================== -->

   <wsdl:types/>

   <!-- ================ MESSAGE section ================ -->

   <wsdl:message name="beginUploadRequest">
      <wsdl:part name="userId" type="xsd:int"/>
   </wsdl:message>

   <wsdl:message name="beginUploadResponse">
      <wsdl:part name="orderId" type="xsd:int"/>
   </wsdl:message>
   ...

   <!-- ================ PORTTYPE section ================ -->

   <wsdl:portType name="PhotoShopPortType">

      <wsdl:operation name="beginUpload"
             parameterOrder="userId">
         <wsdl:input
            message="this:beginUploadRequest"
            name="beginUploadRequest"/>
         <wsdl:output
            message="this:beginUploadResponse"
            name="beginUploadResponse"/>
      </wsdl:operation>
      ...
   </wsdl:portType>
```

```
   <!-- ================ BINDING section ================ -->

   <wsdl:binding name="PhotoShopBinding" type="this:PhotoShopPortType">

      <wsdlsoap:binding style="rpc"
          transport="http://schemas.xmlsoap.org/soap/http"/>

      <wsdl:operation name="beginUpload">
        <wsdlsoap:operation
          soapAction=
            "http://context-aware.org/wsdl/PhotoShop.wsdl#beginUpload"/>
        <wsdl:input name="beginUploadRequest">
          <wsdlsoap:body
            encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
            namespace="http://context-aware.org/wsdl/PhotoShop.wsdl"
            use="encoded"/>
        </wsdl:input>
        <wsdl:output name="beginUploadResponse">
          <wsdlsoap:body
            encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
            namespace="http://context-aware.org/wsdl/PhotoShop.wsdl"
            use="encoded"/>
        </wsdl:output>
      </wsdl:operation>
      ...
   </wsdl:binding>

   <!-- ================ SERVICE section ================ -->

   <wsdl:service name="PhotoShopService">

     <wsdl:port binding="this:PhotoShopBinding" name="PhotoShopSP1">
       <wsdlsoap:address
         location="http://foo.bar.de:8079/soap/servlet/rpcrouter"/>
     </wsdl:port>
     <wsdl:port binding="this:PhotoShopBinding" name="PhotoShopSP2">
       <wsdlsoap:address
         location="http://context-aware.org:80/soap/servlet/rpcrouter"/>
     </wsdl:port>
     ...
   </wsdl:service>

</wsdl:definitions>
```

Abbildung 2.4: Ausschnitt einer WSDL Beschreibung der Signaturebene

Eine WSDL-Beschreibung der Signaturebene eines PhotoShop-Dienstes ist in Abbildung 2.4 skizziert. Jede WSDL-Definition besteht nach [55] aus den fünf Elementen `types`, `message`, `portType`, `binding` und `service`. Hiervon beschreiben die ersten vier das Interface des Dienstes sowie deren Bindung an die verwendeten Kommunikationsprotokolle. Mit dem letzten Beschreibungselement kann eine Liste von Dienstzugangspunkten angegeben werden. WSDL als Signaturbeschreibungssprache dient im Verlauf dieser Arbeit noch häufiger als Beispiel, um die Verankerung bestimmter Strukturen bei der Signatur eines Dienstes zu demonstrieren.

2.2.3 Service-Interoperabilität auf Protokollebene

Interoperabilität wird auf Protokollebene durch folgende Aktivitäten angestrebt: Zum einen ist eine relative Ordnung festzulegen, in der die Methoden eines Dienstes aufgerufen werden, bzw. in der ein Dienst seinerseits Funktionen anderer Dienste aufruft. Zum anderen sind Blockierungsbedingungen zu definieren. Während also auf der Signaturebene die einzelnen syntaktischen Elemente einer Dienstinteraktion das gemeinsame Verständnis darstellen, steht auf der Protokollebene der Zustand eines Dienstes stark im Vordergrund.

Die ersten Ideen zur Interoperabilität auf Protokollebene wurden bereits 1997 von Yellin und Strom [178] auf der Basis von *finite state machines* veröffentlicht. Lea und Marlowe schlagen in [101] eine Erweiterung von CORBA's IDL namens PSL vor. Diese erlaubt es, die Methoden eines Dienstes mit dem Zustand des Dienstes zu verknüpfen und somit das Protokoll als gemeinsames Verständnis auf der Basis von Zuständen und logischen sowie zeitlichen Regeln zu definieren. Andere Ansätze, wie beispielsweise der von Bastide et al. [34], verwenden Petri-Netze zur Spezifikation eines Protkolls bzw. zur Überprüfung der Kompatiblität mit einer Protokoll-Spezifikation.

Bei Web Services erfahren Ansätze zur Spezifikation des gemeinsamen Verständnisses auf der Protokollebene zur Zeit eine Renaissance. Ein Beispiel hierfür sind die Aktivitäten im Bereich des *Web Service Choreography Interface (WSCI)* [28, 117]. WSCI erlaubt die Beschreibung des sichtbaren Verhaltens eines Web Services während einer Interaktion. Einen ähnlichen Ansatz verfolgt IBM mit der *Web Service Flow Language (WSFL)* [103], ohne jedoch nach [11] das zustandsbehaftete Zusammenwirken von Web Services besonders zu berücksichtigen. Dieses ist jedoch essentiell für Interoperabilitätsbetrachtungen auf der Protokollebene. Ein Vergleich der Verfahren zur Beschreibung der Protokollebene bei Web Services ist z.B. bei Wohed et al. [175] oder Mohan [117] zu finden.

Interoperabilität im Sinne der Kompatibilität ist auf der Protokollebene dann gegeben, wenn alle einschränkenden Bedingungen bezüglich des Ablaufs einer Dienstinteraktion beachtet werden und die Kommunikation insgesamt verklemmungsfrei ist. Eine einschränkende Bedingung ist z.B., daß die eingesetzten Protokolle aller bei einer Dienstinteraktion beteiligten Parteien zu der Rolle (*role*) passen, welche die jeweilige Komponente bei der Interaktion einnimmt.

Die direkte (horizontale) Ersetzbarkeit auf Protokollebene erfordert insbesondere die Überprüfung, ob die relative Ordnung der Sequenz der Eingabe- und der Ausgabewerte sowie der Zeitpunkt einer Ersetzung von einem Dienst durch einen anderen konsistent mit der Protokollspezifikation ist. Da bei den meisten Dienstinteraktionen eine Ersetzung nicht zu jedem

beliebigen Zeitpunkt ohne Datenverlust durchgeführt werden kann, müssen die Zustände auf der Protokollebene beschrieben werden, zu denen ein „stabiler“ Zwischenzustand (*Checkpunkt*) erreicht wird, bei dem eine Ersetzung - wenn überhaupt - möglich ist.

Eine ausführliche Analyse der Protokollebene wird u.a. von Vallecillo et al. in [164] vorgenommen. Viele Probleme der Service-Interoperabilität auf Protokollebene (z.B. Dynamik der Zustände) sind auch bereits aus dem Bereich des Workflow Managements bekannt. Eine Literaturrecherche mit diesen Schlagworten ergibt zahlreiche Veröffentlichungen, die ebenfalls zur Vertiefung der Problemstellungen der Protokollebene hinzugezogen werden können.

2.2.4 Service-Interoperabilität auf Semantikebene

Auf der Semantikebene wird versucht, dem Problem des unterschiedlichen Verständnisses entgegenzutreten, da Informationen über die Semantik einer Komponente durch die Beschreibung ihrer Interfaces nicht erfaßt werden. Von Heiler [80] wird dazu das Beispiel angeführt, daß nach einer Studie die Wahrscheinlichkeit, daß zwei Datenbankdesigner den gleichen Namen für identische Datenelemente verwenden, nur bei etwa 7% bis 18% liegt. Oftmals ist es so, daß Entwickler und Nutzer einer bestimmten Komponente unterschiedliche Ansichten von deren Einsatzmöglichkeiten und dem Funktionsumfang der Komponente haben.

Auf der Semantikebene ist die Service-Interoperabilität im Sinne der Kompatibilität dann gegeben, wenn das Verhalten einer Komponente mit dem Verhalten übereinstimmt, welches die mit dieser Komponente in Interaktion stehenden anderen Komponenten erwarten. Damit ist primär die Erwartungshaltung von Service Clients an eine Service Implementierung gemeint, aber auch für die umgekehrte Richtung gilt eine äquivalente Erwartungshaltung. Üblicherweise wird auf semantische Kompatibilität geprüft, indem definierte *pre-conditions* der Methoden eines Dienstes vom Framework gegen entsprechende Eigenschaften eines Dienstnutzers vor der eigentlichen Nutzung des Dienstes gecheckt werden, bzw. indem wiederum durch das Service Framework überprüft wird, ob vom Client erwartete *post-conditions* vom Dienst gehalten werden können. Weitere Möglichkeiten, das Verhalten eines Dienstes auszudrücken, bzw. die Konformität zu einer solchen Verhaltensbeschreibung zu zeigen, besteht nach [164] u.a. durch die Verwendung von temporaler Logik, Prozeßalgebren, Petrinetze und analytischer Verfeinerung (*refinement calculus*).

Ein Ansatz, die Ersetzbarkeit auf der Semantikebene zu ermöglichen, ist *semantisches Ableiten (behavioural subtyping)*. Beim semantischen Ableiten, welches erstmalig von P. America in [22] erwähnt wurde, wird das Verhalten von ersetzbaren Komponenten auf eine Vererbungsbeziehung zurückgeführt, d.h. das Verhalten einer Instanz einer abgeleiteten Klasse ist konsistent mit dem Verhalten einer Instanz der Oberklasse. Das Verhalten bezieht sich hier auf eine Spezifikation, wie die Methoden eines Objekts die Attribute des Objekts manipulieren. Sogenannte semantische Typen (*behavioural types*) sind als Erweiterung der Signaturtypen definiert und dienen als Verbindung zwischen Signaturbeschreibung und Verhaltensbeschreibung. Der Ansatz von P. America wurde in vielen anderen Arbeiten aufgegriffen und auf die eine oder andere Weise modifiziert und erweitert. Dazu gehören beispielsweise die Arbeiten von Liskov und Wing [108], Zaremski und Wing [180] sowie Dhara und Leavens [62]. In diesen Arbeiten basiert die Spezifikation des Verhaltens einer Komponente auf pre- und

post-conditions sowie auf Invarianten. Diese werden dazu verwendetet, zumindest partielle semantische Ersetzbarkeit und damit Interoperabilität zu zeigen.

In [114] wird das der semantischen Interoperabilität zugrundeliegende Entwicklungsmuster passenderweise „design by contract“ genannt. Denn in der Tat ist es wieder ein „contract“, der z.B. die pre- und post-conditions festhält und damit ein gemeinsames Verständnis von Komponenten ausdrückt, die an einer Dienst-Interaktion beteiligt sind.

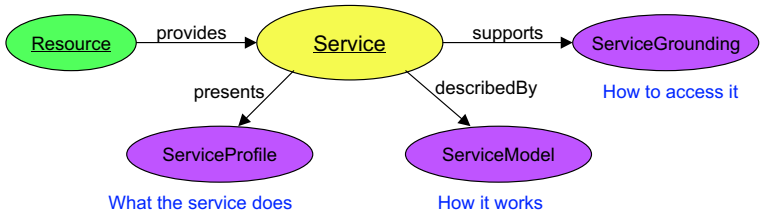

Abbildung 2.5: Kernkonzepte von DAML-S

Ein Ansatz, das gemeinsame Verständnis durch ausschließliche Verwendung einer festgelegten Terminologie zur Beschreibung der semantischen Bedeutung zu erreichen, ist der Einsatz von Ontologien [163], auf die in Abschnitt 3.1.1 im Detail eingegangen wird. Ein solcher Ansatz ist die *ontology of services* namens DAML-S [27, 124]. Damit können computer-interpretierbare semantische Beschreibungen von Diensten aus drei verschiedenen Perspektiven erstellt werden, siehe Abbildung 2.5[1]. Diese drei Typen werden gemäß [27] durch die Antwort auf folgende drei Fragen grob charakterisiert:

- Was sind die Anforderungen eines Dienstes an die aufrufende Komponente, und was bietet der Dienst dieser Komponente? (*ServiceProfile*)
- Wie funktioniert der Dienst? (*ServiceModel*)
- Wie wird der Dienst benutzt? (*ServiceGrounding*)

Insgesamt ist die Spezifikation von DAML-S noch in einem recht frühen, unausgereiften Stadium.

Im Rahmen der Verwirklichung von Tim Berners-Lee's Vision [37] des *Semantic Web* wurden Sprachen wie DAML+OIL [82] oder die *Web Ontology Language (OWL)* [126] entworfen. Diese werden im Abschnitt 3.1.2 näher vorgestellt. Sie sind bei Vorhandensein entsprechender Reasoner möglicherweise geeignet, einen Teil der Anforderungen der semantischen Ebene zu erfüllen. Allerdings werden auf der semantischen Ebene der Interoperabilität auch heute noch zahlreiche Fragestellungen als ungelöst angesehen.

[1] Die Wahl der Farben in dieser Abbildung orientiert sich an der Darstellung in der Literaturquelle.

2.2.5 Alternative Gliederungen der Service-Interoperabilität

Nicht alle Veröffentlichungen ordnen bestimmte Interoperabilitätsbetrachtungen klar einer der drei genannten Ebenen Signatur, Protokoll und Semantik zu. Stattdessen werden zum Teil eigene Ebenen zur Klassifikation herangezogen.

So unterscheidet Miller in [116] z.B. nach

1. Technical Interoperability
2. Semantic Interoperability
3. Political/Human Interoperability
4. Inter-community Interoperability
5. Legal Interoperability
6. International Interoperability

Während in diesem Fall 1. und 2. mehr oder weniger 1:1 auf die Signaturebene bzw. Semantikebene abgebildet werden könnte, befassen sich 3. bis 6. mit Vorbedingungen, die den Zugriff auf einen Dienst unter bestimmten politischen, ethischen, rechtlichen oder regionalen Voraussetzungen ermöglichen. So gesehen, könnte man 3. bis 6. auch der Protokollebene, zum Teil aber auch der noch einzuführenden Kontextebene zuordnen.

Murer et al. [118] betrachten eine Dreiteilung der Interoperabilität nach

1. Interface Level
2. Originator Level
3. Semantic Level

Gemäß dieser Aufteilung kann 1. auf die Signaturebene und 3. auf die Semantikebene der Darstellungsform in dieser Arbeit abgebildet werden. Das neu eingeführte Originator Level beschäftigt sich mit der Frage, welche Typen und Versionen von Komponenten zusammenarbeiten. Eine solche Aufteilung bzw. die Einführung des Originator Level scheint aber für verteilte Systeme wenig geeignet zu sein, da die Integration bisher unbekannter Dienste durch eine Beschreibung auf dem Originator Level nicht sehr hilfreich ist.

Da sich die Mehrheit der Literaturstellen der vorgestellten Dreiteilung nach Signatur, Protokoll und Semantik anschließt, soll diese Dreiteilung als Basis der weiteren Betrachtungen in dieser Arbeit gelten.

2.2.6 Service-Interoperabilität auf Kontextebene

Es zeigt sich, daß eine Betrachtung der Service-Interoperabilität auf den drei Ebenen Signatur, Protokoll und Semantik nicht ausreicht. Bereits S. Heiler hat in [80] angemerkt, daß „interessante semantische Informationen kontextabhängig sind“. Diese werden jedoch von der semantischen Ebene nur unzureichend erfaßt [164]. Denn auch wenn auf semantischer Ebene das (interne) Verhalten eines Dienstes z.B. durch *pre/post conditions* [107, 27] oder *abstract type frameworks* [93] beschrieben werden kann, bleiben Bezüge auf die (externe) Situation bisher bei Interoperabilitätsbetrachtungen unberücksichtigt. Außerdem bedarf die semantische Ebene einer Konzentration auf die reine Semantik, da sie in bisherigen Betrachtungen bereits zu „breit“ genutzt wurde [165].

Durch die Betrachtung kontextueller Abhängigkeiten – wenn überhaupt – als Teil einer der drei genannten Ebenen Signatur, Protokoll oder Semantik kann es passieren, daß zwei oder mehr an einer Dienstinteraktion beteiligte Komponenten auf allen drei Ebenen als interoperabel gelten, aber nicht aus dem Blickwinkel der Problemstellung, die dieser Arbeit zugrundeliegt.

Als Beispiel für diese Konstellation können die Premiumdienste [104, 142] des europäischen Satellitennavigationssystems Galileo dienen. Obwohl die Software eines Empfängers auf allen drei klassischen Ebenen interoperabel zum Galileo-System ist, ist die Nutzung der Premiumdienste (höhere Genauigkeit der Daten etc.) nur im „kommerziellen“ Kontext möglich (d.h. gegen Bezahlung), im „gebührenfreien“ Kontext jedoch nicht.

Ein anderes Beispiel für diesen Fall sind zwei elektronische Fahrplanauskunfsdienste (EFA-Dienste) für München und Berlin. Obwohl die EFA Berlin möglicherweise auf allen drei klassischen Ebenen interoperabel zur EFA München ist, gilt dies nicht, wenn z.B. ein ortsbezogener Kontext von München für die EFA-Abfrage in Berlin zugrunde gelegt wird.

Daher wird gemäß [152, 151] eine neue, vierte Ebene der Service-Interoperabilität, die *Kontextebene (context level)*, vorgeschlagen (siehe Abbildung 2.6). In dieser Ebene werden alle Fragestellungen zum Kontext gebündelt und getrennt von den übrigen drei Ebenen betrachtet. Ein Hauptziel im Rahmen dieser Arbeit besteht darin, ein auf der kontextuellen Ebene geschaffenes Modell dahingehend zu verwenden, möglichst automatisiert auf Veränderungen des Kontextes im Sinne von kontextadaptiver Dienstnutzung zu reagieren.

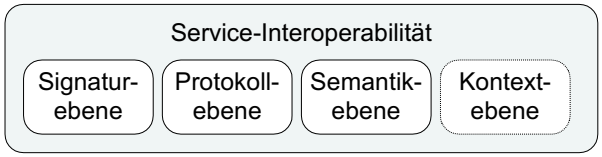

Abbildung 2.6: Erweiterte Ebene der Service-Interoperabilität

Im folgenden Abschnitt wird auf die Vorteile, die sich durch die Einführung der Kontextebene ergeben, näher eingegangen.

2.3 Kontext und Service-Interoperabilität

In diesem Abschnitt wird zunächst eine ausführliche Analyse des Kontextbegriffs vorgenommen (2.3.1). Ausgehend von Literaturrecherchen zum Kontextbegriff wird die in dieser Arbeit verwendete Terminologie hergeleitet. Es wird gezeigt, wie der Kontext die Service-Interoperabilität beeinflußt (2.3.2), was zur Einführung der Kontextebene als Unterebene der Service-Interoperabilität führt. Der Abschnitt endet mit einer Betrachtung, wie die hier gefundene Definition des Kontextes und seine Anwendbarkeit im Dienstumfeld in ein bestehendes allgemeines Dienstmodell integriert werden kann (2.3.3).

2.3.1 Analyse des Kontextbegriffs

Der Begriff des Kontextes und das Konzept der Kontextadaptivität gewannen Anfang der neunziger Jahre im Forschungsumfeld verteilter Systeme durch den Trend zum mobile Computing zunehmend an Bedeutung. Man hat im Kontext einen vielversprechenden Lösungsansatz für Probleme erkannt, die sich bei der mobilen Benutzung von Computersystemen aus den sich ständig ändernden physikalischen Umgebungsbedingungen ergeben. Die Kontextabhängigkeit ist daher auch eines der Kernthemen aktueller Forschungsarbeiten im Umfeld der Ubiquitous Computing Umgebungen, die eine Spezialisierung mobiler, verteilter Systeme sind und auf deren Besonderheiten im Zusammenhang mit dieser Arbeit im Abschnitt 2.4 detailliert eingegangen wird.

Frühe Untersuchungen zum Kontext wurden am Olivetti Research Lab mit der Entwicklung des Active Badge Systems durchgeführt, wo die mittels Sensoren erfaßte örtliche Position der mobilen Benutzer Einfluß auf die Applikation des Benutzers hatte [170]. Parallel dazu entstanden am Palo Alto Research Center von Xerox erste Veröffentlichungen, in denen der Kontext in allgemeiner Form betrachtet wurde, und die zum ersten Mal den Begriff der Kontextadaptivität betrachteten [138].

In den Arbeiten von Schilit (z.B. [139]) wird der Kontext vornehmlich nutzerbezogen definiert. Schilit versteht zum einen unter dem Begriff Kontext den momentanen Aufenthaltsort des Nutzers. Zum anderen ist der Kontext nach Schilit eine Liste von Identitäten von Personen und Objekten in der örtlichen, unmittelbaren Umgebung des Nutzers. Eine weitergehende Definition des Kontextes nach Schilit klassifiziert den Kontext in drei verschiedene Dimensionen:

Computing Environment bezeichnet die Menge der verfügbaren Ressourcen, die dem Benutzer zu einem bestimmten Zeitpunkt zur Verfügung stehen. Dazu gehören Geräte in der örtlichen Umgebung[2] des Benutzers, wie etwa Drucker, aber auch netzwerkspezifische Informationen, wie z.B. Angaben über die Kapazität des Netzwerkes oder die Kosten für die Nutzung bereitgestellter

[2] Diese Ausdrucksweise könnte als Tautologie verstanden werden, wurde aber bewußt gewählt. Sie wird im weiteren Verlauf der Arbeit in dieser oder ähnlicher Form verwendet, um auszudrücken, daß Begriffe wie *Umgebung* oder *Nähe* über räumliche oder zeitliche Aspekte hinaus definiert sein können.

Dienste. Das Computing Environment bezieht sich also auf die ortsbezogene Ausführungsumgebung für Benutzerdienste.

User Environment gibt den Aufenthaltsort eines Benutzers und die Anwesenheit anderer Personen in der örtlichen Umgebung des Benutzers an.

Physical Environment umfaßt physikalisch weitere meßbare Umgebungsinformationen wie Angaben zur Lautstärke oder der Lichtverhältnisse.

In diesen ersten Arbeiten wurden als kontextueller Parameter vornehmlich Positionsinformationen betrachtet. Diese Betrachtungsweise führte z.B. zu Leonhardt's formalem Lokalitätsmodell, von dem ein Modell der Ortsadaptivität abgeleitet werden konnte [102]. Nur wenige Arbeiten wiesen darauf hin, daß ein allgemeines Kontextmodell auch andere Parameter berücksichtigen müßte. Dazu zählen z.B. die Arbeit von Pascoe [125] (*„Kontext ist die Untermenge von physikalischen und konzeptionellen Stati, die für eine bestimmte Entität von Interesse sind“*) wie auch die von Schmidt et al. [140]. Letztere definiert den Kontext wie folgt:

> *Der Kontext beschreibt eine Situation und die Umgebung, in der sich ein Gerät oder ein Anwender befindet. Ein Kontext wird identifiziert durch einen eindeutigen Namen. Für jeden Kontext ist eine Menge von Merkmalen relevant. Für jedes relevante Merkmal ist ein Wertebereich implizit oder explizit durch den Kontext festgelegt.*

Chen und Kotz [51] unterscheiden bei ihrer Definition des Kontextes noch zwischen der bestimmenden und der interessanten Art des Kontextes. Letztere wird zwar als relevant, nicht aber als kritisch für eine Anwendung eingestuft, d.h. die Anwendung muß sich auf interessante Kontextzustände nicht automatisiert anpassen, sondern lediglich dem interessierten Anwender zur Ansicht anbieten:

> *Der Kontext ist die Menge von Zuständen und Ausstattungen der Umgebung, die entweder das Verhalten einer Applikation bestimmen oder in denen ein Anwendungsereignis auftritt, das interessant für den Anwender ist.*

Die in der Literatur am häufigsten zitierte Definition des Kontextes ist allerdings die Definition nach Dey [61]:

> *Kontext ist jede Information, die dazu benutzt werden kann, die Situation einer Entität zu charakterisieren. Eine Entität ist eine Person, ein Ort oder ein Objekt, welches bzgl. der Interaktion zwischen einem Benutzer und einer Anwendung als relevant zu betrachten ist, einschließlich dem Benutzer und der Applikation selbst.*

Stellt man die beiden Definitionen von Dey und Schmidt et al. gegenüber, dann kann man feststellen, daß beide für sich genommen nicht vollständig sind, sich aber gegenseitig recht gut ergänzen. Einerseits ist in der Definition von Schmidt et al. bereits ein Hinweis auf die

notwendige Festlegung eines Wertebereiches für kontextuelle Merkmale zu finden, der in der Definition von Dey fehlt. Andererseits trifft die Formulierung von Dey die Eigenschaft der Kontextinformationen besser, den Status einer bestimmten Entität zu charakterisieren.

Beide Definitionen haben aber den Schwachpunkt, daß der Kontext in Abhängigkeit des Begriffs der Situation definiert wird. Während Schmidt et al. sich nicht darauf festlegen, was sie unter der Situation verstehen, definiert Dey zunächst den Begriff des Kontextes in Abhängigkeit von der Situation, und wenig später die Situation in Abhängigkeit vom Kontext. Diese rekursive Vorgehensweise ist wissenschaftlich nicht optimal.

Ein weiterer Nachteil in beiden Definitionen ist die Abgrenzung der Aussage auf die Interaktion zwischen Anwender und Gerät oder Applikation. Es spricht eigentlich nichts gegen die Verwendung von Kontextinformationen bei Dienst/Dienst-Interaktionen bzw. Interaktionen zwischen Plattform und Dienst oder Applikation, wie sie auch schon Chen und Kotz vorsehen, indem sie auf den bestimmenden Charakter von Kontextinformationen hinweisen, deren Auswertung einer Applikation und nicht dem Menschen obliegt.

Aus den genannten Gründen werden im weiteren Verlauf dieser Arbeit die nun folgenden Definitionen des Kontextbegriffs verwendet, die sich in wesentlichen Punkten an den Definitionen von Schmidt et al. und Dey orientieren, diese aber an den identifizierten Schwachstellen modifizieren bzw. ergänzen. Diese Definitionen werden insbesondere Grundlage der Sprache zur Modellierung kontextueller Interoperabilität sein, die in Kapitel 3 eingeführt wird.

Zunächst beziehen sich Kontextinformationen immer auf Entitäten. Diese sind ganz allgemein wie folgt definiert:

> **Definition 2 (Entität)** *Eine Entität ist eine Person, ein Ort oder allgemein ein Objekt.*

Wie bei Chen und Kotz ist der Begriff des Zustands wichtig zur Definition des Kontextbegriffs. Gemäß Schwickert und Fischer [144] *„wird unter einem Zustand $(x_1, .., x_n)$ die Belegung von Zustandsvariablen (ZV) mit einem konkreten Wert verstanden, wobei die Anzahl der Zustandsvariablen in einem Zustand nicht festgelegt ist. Eine Zustandsvariable x_i ist eine elementare Größe, die gewisse definierte Werte annahmen kann. Sind beispielsweise x_1 und x_2 Zustandsvariablen mit den Ausprägungen $x_1 = 1$ und $x_2 = 5$, so bezeichnet das Tupel (1,5) einen Zustand. Werden alle Zustände, die eine Menge von Zustandsvariablen annehmen kann, zusammenfaßt, erhält man den sogenannten Zustandsraum.“*

Zur Beschreibung des Zustands einer Entität wird in Analogie dazu folgende Definition herangezogen:

> **Definition 3 (Zustand)** *Der Zustand einer Entität wird beschrieben durch eine Menge von Zustandsvariablen. Jeder Zustand ist gegeben durch eine Belegung der Zustandsvariablen mit konkreten Werten. Der Zustandsraum ist gegegen durch die Menge aller Zustandsvariablen.*

Ein aktueller Zustand der Entität „Mobiltelefon“, die durch die IMEI=88364... identifiziert wird, könnte also anschaulich wie folgt dargestellt werden:

$$Zustand(Mobiltelefon_{IMEI=88364...}) = \begin{pmatrix} ZV_{Zeit} & = & 20.10.2003\ 11:55 \\ ZV_{Ort} & = & 367032\ 533074 \\ ZV_{Temperatur} & = & 27\ Grad\ Celsius \\ ZV_{Eigentümer} & = & Thomas\ Strang \\ ZV_{Gespräch} & = & nicht\ aktiv \\ ZV_{Ladestand} & = & 31\% \\ \ldots & = & \ldots \end{pmatrix}$$

Jede Zustandsvariable hat dabei eine individuelle Semantik. Die jeweilige Semantik wird in dieser Darstellung durch den Namen der Zustandsvariable ZV_{Name} ausgedrückt, durch den die einzelnen Zustandsvariablen unterscheidbar werden. Obwohl als Unterscheidungsmerkmal sehr wichtig, reicht ein eindeutiger Name zur Beschreibung der Semantik oft nicht aus. So wünscht sich ein menschlicher Betrachter eine umfassende Beschreibung hinsichtlich der Bedeutung der durch die jeweilige Variable erfaßten Zustände. Dies kann verglichen werden mit einem Lexikon, wo zu jedem Eintrag (Zustandsvariable) eine mehrzeilige Passage (semantische Beschreibung) mit erläuterndem, oft vergleichendem Fließtext zugeordnet ist. Eine computergestützte Auswertung erfordert in der Regel jedoch einen höheren Grad an Formalität zur Beschreibung der Semantik. Hierbei erfolgt die Beschreibung der Semantik unter Verwendung des Vokabulars eines Modells mit wenigen vordefinierten Konzepten und Relationen. Die computergestützte Auswertung der Semantik wird auf eine Auswertung der dem Computer bekannten Konzepte und Relationen des Modells zurückgeführt. Ein solches Konzept ist der im Folgenden definierte Aspekt:

> **Definition 4 (Aspekt)** *Ein Aspekt ist eine Dimension des Zustandsraumes mit zugehöriger Beschreibung des semantischen Typs.*

Ein Aspekt ist also Identifikator und Beschreibung einer Klassifikation, Symbol- oder Wertemenge des identischen semantischen Typs. Aspekte werden insbesondere zur Beschreibung der Semantik von Zustandsvariablen verwendet.

Die durch einen Aspekt identifizierten Informationen identischen semantischen Typs (z.B. Temperaturangaben) können ihrerseits in mehrere *Skalen* (z.B. Grad Fahrenheit Skala und Grad Celsius Skala) gruppiert sein, die ineinander überführbar sind.

Im Sinne des Kontextbegriffs ist der Aspekt der Schlüssel zur Definition der Relevanz:

> **Definition 5 (Relevanz)** *Ein Aspekt ist relevant für eine Aufgabe, wenn während der Erfüllung dieser Aufgabe auf Werte dieses Aspekts zugegriffen wird, bzw. Werte dieses Aspekts den Ablauf der Aufgabe beeinflussen.*
>
> *Eine Entität ist relevant für eine Aufgabe (Task, Goal, Perspective, Interaktion, Dienstnutzung), wenn ihr Zustandsraum einen für diese Aufgabe relevanten Aspekt enthält.*

Im vorgenannten Beispiel wird die Entität „Mobiltelefon mit der IMEI=88364...“ z.B. dadurch zu einer relevanten Entität, daß zur Erfüllung der Aufgabe „Bestimmung des Aufenthaltortes von Thomas Strang“ auf den Aspekt *Ort* und damit auf die Zustandsvariable ZV_{Ort} zugegriffen wird.

Bezogen auf den Kontextbegriff kann eine Kontextinformation mit der Belegung einer Zustandsvariablen gleichgesetzt werden, was unter Berücksichtigung der Definitionen von Zustand und Relevanz zu folgender Definition des Kontextes führt:

> **Definition 6 (Kontextinformation)** *Eine Kontextinformation ist eine Information, die dazu benutzt werden kann, den Zustand einer Entität bzgl. eines Aspekts zu charakterisieren. Sie ist gegeben durch ein Element des Wertebereichs eines Aspekts. Eine Kontextinformation ist relevant für eine Aufgabe, wenn sie die Belegung einer Zustandsvariable eines relevanten Aspekts ist, der zu einer für diese Aufgabe relevanten Entität gehört.*

> **Definition 7 (Kontext)** *Ein Kontext ist die Menge aller für eine Aufgabe relevanten Kontextinformationen.*

In der Literatur ist die Trennung zwischen den Begriffen *Kontext* und *Situation* sehr unscharf. Der eine Begriff findet häufig als Synonym für den anderen Verwendung. Beide Begriffe haben jedoch in dieser Arbeit eine klare Unterscheidung.

> **Definition 8 (Situation)** *Eine Situation ist die Menge aller bekannten Kontextinformationen.*

Die Situation ist nach Definition 8 relevanzfrei. Sie umfaßt somit die Menge *aller bekannten* Kontextinformationen. Relevanzfreiheit bedeutet, daß die Verwendung dieser Kontextinformationen nicht a priori an eine bestimmte Aufgabe oder Entität gebunden ist. Kontextinformationen dieser Art können z.B. aus Übergangswahrscheinlichkeiten gewonnen werden. Der Situation gegenüber steht der Begriff des Kontextes nach Definition 7 als Menge der *für eine Aufgabe relevanten* Kontextinformationen, die eine Teilmenge aller bekannten Kontextinformationen darstellt. Demzufolge ist implizit die Angabe von Relevanzkriterien gefordert, wenn im Folgenden vom Kontext gesprochen wird.

> **Definition 9 (Kontextadaptivität / Context-Awareness)** *Ein System ist kontextadaptiv (context-aware), wenn es den Kontext vor oder während der Erfüllung einer Aufgabe berücksichtigt.*

Die o.g. Definitionen 2, 6 und 7 der Terminologie sind sehr nahe derjenigen von Dey [61]. Verglichen mit Dey wird bei der *Relevanz* zwischen einer relevanten Entität und einem relevanten Aspekt unterschieden. Die oben genannten Definitionen unterscheiden sich von den meisten anderen Definitionen des Kontextes (siehe oben, z.B. [61, 141, 139]) durch die Einführung

des *Aspekts* (Definition 4). Jede Aspekt-Instanz stellt eine Dimension des Situationsraumes[3] dar, und ist gleichermaßen Aggregator aller möglichen Zustände dieser Dimension, wie auch Sammelbegriff für Informationen eines semantischen Typs.

Ein Aspekt ist eine Menge von artverwandten Skalen. Jede Skala ist wiederum eine Menge von Objekten, die den Wertebereich gültiger Kontextinformationen aus dieser Skala festlegen. Mit anderen Worten ist eine gültige Kontextinformation bezüglich eines Aspekts ein Element aus einer der Skalen aus diesem Aspekt. Zum Beispiel könnte ein Aspekt „GeographicCoordinateAspect“ zwei verschiedene Skalen haben, „WGS84Scale“ und „GaussKruegerScale“. Eine gültige Kontextinformation daraus ist eine Objekt-Instanz, die in einer objektorientierten Programmiersprache wie Java mit `new GaussKruegerCoordinate("367032", "533074")` erzeugt wurde.

Für eine einzelne *Entität* kann die Relation ihres *Kontextes* zu mehreren *Aspekten* mit Couderc's Metapher einer „Art Cursor in einem multi-dimensionalen Informationsraums“ [33, 58] verglichen werden.

Eine Unterklasse von Kontextinformationen stellen Qualitätsparameter dar, welche die Qualität von Kontextinformationen im Sinne von Meta-Informationen charakterisieren. Meta-Kontextinformationen sind insofern Kontextinformationen höherer Ordnung, welche die Qualität der Kontextinformationen geringerer Ordnung beschreiben, siehe auch Abbildung 2.7. Meta-Informationen dieser Art und ihre Integrationsmöglichkeiten in Dienstumgebungen wurden in zahlreichen Publikationen untersucht (u.a. in [174, 112, 38]). Dabei handelt es sich allgemein um nicht funktionale Parameter, die z.B. die Performanz, Zuverlässigkeit, Verfügbarkeit oder Sicherheit eines Dienstes charakterisieren.

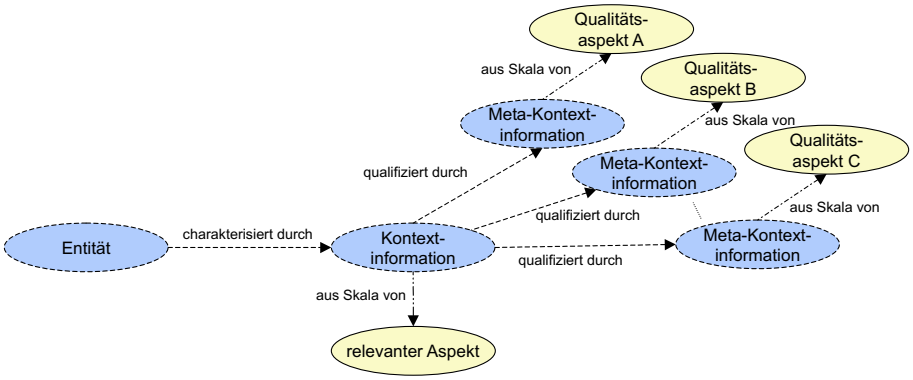

Abbildung 2.7: Meta-Kontextinformationen

Auf diesen Parametern aufbauend kann z.B. innerhalb eines Service Level Agreements festgelegt werden, welche Aktion im Fall des Unter- oder Überschreitens eines vertraglich definierten Grenzwertes von wem ausgelöst werden soll. Exemplarisch können hier die *Web Service Level Agreements (WSLA)* [109] genannt werden, mit denen die Vertragsparteien (*parties*) in

[3] Der Begriff *Situationsraum* wird hier in Analogie zu Schwickert und Fischer's [144] Begriff des *Zustandsraums* verwendet.

WSLA Dokumenten festlegen, welche Grenzwerte (*service level*) für bestimmte Web Services garantiert werden (*obligations*), und was im Falle eines Unter- oder Überschreitens zu tun ist (*actions*). Die in WSLA verwendeten Sprachelemente sind zum Teil sehr gut geeignet, um Aspekte und Relevanz-Bedingungen zu definieren. Allerdings ist das Gesamtkonzept von WSLA vor allem sowohl aufgrund des statischen Aufbaus eines solchen „Vertrages“ im Form eines einzigen WSLA Dokuments, als auch aufgrund der beschränkten Gültigkeit zwischen genau zwei Vertragspartnern, für die Zielrichtung dieser Arbeit ungeeignet. Hier bedarf es einer flexibleren Möglichkeit der spontanen Komposition und Überprüfung der momentanen Situation, u.a. ausgehend von Kontextinformationen, deren Struktur und Existenz zum Zeitpunkt der Erstellung eines WSLA Dokuments noch nicht bekannt ist.

Die nicht funktionalen Parameter wie o.g. Qualitätsmerkmale werden in der Literatur nicht eindeutig einer der drei „klassischen“ Service Interoperabilitätsebenen zugeordnet. Mani und Nagarajan [112] sehen diese beispielsweise als Erweiterung einer Interface Spezifikation und ordnen sie somit der Signaturebene zu. Weller [174] schlägt vor, auf Qualitäts-, Payment- und Deadlock-Informationen parallel zur Dienstausführung zuzugreifen und siedelt diese dabei auf der Protokollebene an. Meistens jedoch werden Informationen, die nicht unmittelbar die Funktion eines Dienstes beschreiben, wenn überhaupt, auf der semantischen Ebene behandelt. Ankolekar et. al. [27] haben z.B. einige wenige als *functional attributes* bezeichnete Charakteristiken wie *geographicRadius* oder *degreeOfQuality* definiert, mit denen nach ihrer Aussage das nicht funktionale Verhalten, beziehungsweise die Bedingungen zur Ausführung von Diensten beschrieben werden können. Es macht jedoch viel mehr Sinn, die Interoperabilität von Diensten bezüglich kontextueller Abhängigkeiten und Möglichkeiten auf einer eigenen Ebene, eben der Kontextebene, zu betrachten.

Eine für den Menschen sehr wichtige Größe stellt eine interessante Kontextinformation dar: Die örtliche Position in Bezug auf ein Referenzsystem (z.B. WGS84). Durch Verknüpfung mit symbolischen Informationen („Gebäude A, Raum 008“) und relationalen Informationen („in der Nähe von“) wird aus einer Position eine *Location* [26]. Dienste, die auf den aktuellen Aufenthaltsort als Kontextinformation zugreifen, werden unter dem Oberbegriff *Location Based Services (LBS)* zusammengefaßt. Allerdings ist der Ort bei weitem nicht die einzige wichtige Kontextinformation [140].

Eine andere sehr wichtige Art von Kontextinformation ist die Zeit. So kann ein Dienst sehr unterschiedliche Ergebnisse liefern, wenn er vor, während oder nach einem bestimmten Zeitpunkt aktiviert wird. Allgemein kann jede von einem Sensor erfaßte Größe als Kontextinformation dienen (Position, Zeit, Temperatur, Lichtstärke, Momentangeschwindigkeit etc.), siehe Abschnitt 2.4.2.

Bei den sogenannten primären Kontextinformationen wird dabei auf die Rohdaten des jeweiligen Referenzsystems (WGS84-Koordinate, Aktuelle Weltzeit, Temperatur in Grad Celsius, Helligkeit in Lux, Geschwindigkeit in km/h etc.) zugegriffen. Im Gegensatz dazu durchlaufen die Rohdaten bei sekundären Kontextinformationen weitere Verarbeitungsstufen. Aus dieser Überlegung leitet sich unmittelbar ab, daß der Wertebereich eines Aspekts einer primären Kontextinformation immer das Referenzsystem selbst ist. Bei sekundären Kontextinformationen ist im Gegensatz dazu die Definition einer Klassifikation, Symbol- oder Wertemenge als eine zu einem Aspekt gehörende Skala zwingend erforderlich, um die Menge der möglichen

Zustände der sekundären Kontextinformation geschlossen angeben zu können. Ein Beispiel: Die Menge

$$\mathcal{W}_{TempCelsius} = \{t | t \in \mathbb{R}, t > -273\}$$

definiert den Wertebereich (Menge der möglichen Zustände) einer primären Kontextinformation, die den Zustand eines Temperatursensors darstellt. Dagegen definiert die Menge

$$\mathcal{W}_{TempFeeling} = \{warm, cold\}$$

den Wertebereich einer sekundären Kontextinformation, die den Zustand bezüglich eines anderen Temperatur-Aspekts charakterisiert. Allerdings obliegt es dem für die Kontextinformation zuständigen Sensor (*context observer*), zu entscheiden, welchen Zustand die Kontextinformation repräsentiert, beispielsweise anhand einer Schwellwertentscheidung. Ein Sensor kann als Funktion f verstanden werden, der Daten aus n Sensorquellen (Definitionsbereich $\mathcal{D}$) fusioniert und als Ergebnis einen gültigen Zustand aus dem Wertebereich $\mathcal{W}$ eines Aspekts zurückliefert:

$$f : \mathcal{D} \rightarrow \mathcal{W} \times]0, 1)$$

So liefert die Funktion $f(quelle_1, .., quelle_n) = (w, p)$ die Kontextinformation w als Ergebnis der Datenfusion. Gleichzeitig gibt p mit $p \in \mathbb{R}, 0 < p \leq 1$ in diesem Beispiel das Maß der Sicherheit an, mit der ein Sensor von sich behauptet, den korrekten Zustand bestimmen zu können, was z.B. als Meta-Kontextinformation ausgedrückt werden kann.

Entscheidend ist, daß innerhalb der Definition eines Aspekts festgelegt werden muß, unter welchen Bedingungen welcher Zustand eingenommen wird, nicht jedoch, woher die dazu notwendigen Informationen stammen (d.h. auf welche Sensoren dabei zurückgegriffen wird), noch wie die Funktion f arbeitet.

Wichtig ist vor und während einer Dienstnutzung die Relevanz einer Kontextinformation. Die Menge der relevanten Entitäten und damit der Kontextinformationen, die diese Entitäten charakterisieren, ändert sich ständig. Die Menge der spezifischen Aspekte, welche die Relevanz einer Entität bestimmt, ist jedoch fix! Somit sind es die Aspekte, die auf der kontextuellen Ebene für jeden Dienst spezifiziert bzw. vor und während der Dienstnutzung referenziert werden müssen. Anhand der spezifizierten Aspekte kann vor der Ausführung eines Dienstes ermittelt werden, ob es relevante Entitäten gibt, und über die so gefundenen assoziierten Kontextinformationen entschieden werden, ob die Ausführung des Dienstes (aus kontextueller Sicht) möglich und ratsam ist, oder nicht.

Zur Laufzeit des Dienstes beeinflußt die Menge der relevanten Entitäten den Dienst in mehrfacher Hinsicht. Zum Beispiel bestimmt sie, ob die Bedingungen, die aus kontextueller Sicht zur Auswahl und Aktivierung des Dienstes führten, noch gegeben sind. Falls nicht, wird die

Ausführung des Dienstes vorübergehend oder permanent abgebrochen. Ein Beispiel hierfür ist die physische Entfernung von einem Point-of-Interest (PoI): Mit zunehmender Distanz macht die Ausführung eines PoI-spezifischen Dienstes immer weniger Sinn. Dieses Beispiel wird im Kapitel 5 noch einmal aufgegriffen und ausführlicher betrachtet.

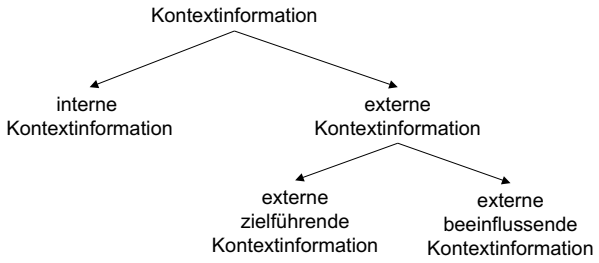

Abbildung 2.8: Klassifikation von Kontextinformationen

In Anlehnung an eine von Ötztürk und Aamodt eingeführte *Context Ontology* [123] kann man diese Art von externen beeinflussenden Kontextinformationen als *environment related context information* bezeichnen, da die kontextuelle „Umwelt“ des Dienstes dessen Ausführung beeinflußt.

Dem gegenüber stehen die aus Sicht eines Dienstes externen Kontextinformationen, die zur Erfüllung des Dienstes selbst zielführend sind. Diese können dementsprechend als *target related context information* klassifiziert werden. Als Beispiel dient hier ein Routing-Dienst von Ort X nach Ort Y, wo als Ort X der momentane Aufenhaltsort (Position) automatisch vorgegeben wird, der aus der aktuellen Situation ermittelt wird.

Die Ansicht von Ötztürk und Aamodt, daß diese externen Kontextinformationen während der Dienstausführung als statisch anzusehen sind, wird allerdings nicht geteilt, wie das o.g. Beispiel der *target related context information* zeigt: Durch die Mobilität des Benutzers ändert sich der aktuelle Standort ständig.

Als Beispiel für eine *internal context information* sei die Anfangszeit eines Meetings am Ort B genannt, die von einem Routen-Dienst bei der Berechnung eines Reiseplans intern zugrunde gelegt wird. Eine Übersicht dieser Klassifikation ist in Abbildung 2.8 dargestellt.

2.3.2 Kontextuelle Service-Interoperabilität

Auf der hier neu eingeführten kontextuellen Ebene der Service Interoperabilität bestimmt die Menge relevanter externer und interner Einflußgrößen auf eine Dienstinteraktion das gemeinsame Verständnis, das zur Überprüfung der Interoperabilität von allen an einer Dienstinteraktion beteiligten Komponenten herangezogen wird. Diese Einflußgrößen zeichnen sich insbesondere dadurch aus, daß ihre Relevanz bezüglich der im Rahmen der Dienstinteraktion zu erfüllenden Aufgabenstellung anhand von Relevanzkriterien beschrieben wird. Diese Relevanzkriterien werden zeitnah zur Dienstinteraktion ausgewertet, um die Menge der Kontextinformationen zu bestimmen, die den Kontext der Dienstinteraktion darstellen.

Zur Definition der Relevanzkriterien wird einerseits eine Sprache benötigt, die es erlaubt, logische Aussagen und optimalerweise auch Relationen auszudrücken, damit Relevanzkriterien wie z.B. „berücksichtige bei *someFunction* alle Entitäten, deren aktueller Status bezüglich des Aspekts *place* ist, und wo diese Informationen bezüglich des Qualitäts-Aspekts *age* kleiner oder gleich *10* auf der *seconds* Skala ist“. Andererseits ist ein Katalog von Bezugsgrößen notwendig, auf die sich Aussagen dieser Art beziehen (place, age, seconds etc.). Dieser Katalog von Bezugsgrößen ist ebenfalls Bestandteil des notwendigen gemeinsamen Verständnisses. In Kapitel 3 wird ein Modell vorgestellt werden, welches als Basis für einen solchen Katalog von Bezugsgrößen dienen wird.

Kompatibilität kann auf der kontextuellen Ebene in dem Sinne definiert werden, daß zwei interagierende Komponenten korrekt zusammenarbeiten, wenn beide Seiten ein identisches Verständnis der relevanten Aspekte haben:

> **Definition 10 (Kontextuelle Kompatibilität)** *Zwei interagierende Komponenten gelten als kontextuell kompatibel, wenn sie ein gemeinsames Verständnis der für die Interaktion relevanten Aspekte haben.*

Dies beinhaltet die Art der Auswahl der relevanten Entitäten sowie die Menge der Zustände, welche die Kontextinformationen in den relevanten Aspekten einnehmen können (einschließlich Verwendung der gleichen Skalen und damit auch der Einheiten). Dies kann wiederum durch den Bezug auf eine gemeinsame Spezifikationen der Relevanzkriterien und eines Katalogs der Aspekte erreicht werden.

Kompatibilität in diesem Sinne kann auch durch Transformation erreicht werden. Dieser Fall setzt voraus, daß Kontextinformationen, die als relevant für eine Dienstinteraktion eingestuft werden, in eine Form transformiert werden können, die der Dienst auswerten kann. Beispiele hierfür werden in Kapitel 4 und 5 gezeigt.

Auf der kontextuellen Ebene setzt die Ersetzbarkeit einer Komponente durch eine andere Komponente die Vollständigkeit des Kontextes bezüglich der Interaktion mit einer dritten Komponente voraus.

> **Definition 11 (Kontextuelle Vollständigkeit)** *Eine Dienstinteraktion ist kontextuell vollständig, wenn für alle der Interaktion zugeordneten relevanten Aspekte, relevante Kontextinformationen vorliegen.*

Vollständigkeit des Kontextes während einer Ersetzung bedeutet, daß die Kontextinformationen bezüglich aller relevanten Aspekte der Interaktion mit der Vorgängerkomponente auf äquivalente Kontextinformationen der Interaktion mit der Nachfolgerkomponente abgebildet werden können. Die Menge der relevanten Aspekte bei der Interaktion mit der Vorgängerkomponente muß nicht notwendigerweise identisch mit der Menge der relevanten Aspekte bei der Interaktion mit der Nachfolgerkomponente sein. In diesem Fall erfordert die Vollständigkeitsbedingung zusätzlich das Wissen über gültige Kontextinformationen der zusätzlichen relevanten Aspekte der Interaktion mit der Nachfolgerkomponente. Ein Beispiel für den Fall

unterschiedlicher relevanter Aspekte ist die Ersetzung eines Fahrplanauskunftdienstes durch einen anderen, wobei für den einen Fahrplanauskunftdienst der Aspekt der Namen von Bushaltestellen und für den anderen der Aspekt der Namen von S-Bahn-Stationen als relevant gilt.

Eine Ersetzung auf kontextueller Ebene setzt also nicht nur äquivalente Kontextinformationen im Falle identischer relevanter Aspekte voraus. Darüber hinaus müssen im Falle nicht identischer relevanter Aspekte auch gültige Kontextinformationen für alle relevanten Aspekte der Interaktion mit der Nachfolgerkomponente bekannt sein.

> **Definition 12 (Kontextuelle Konsistenz)** *Zwei Dienstinteraktionen sind kontextuell konsistent, wenn beide Interaktionen jeweils kontextuell vollständig sind und für alle Elemente der Schnittmenge der der jeweiligen Interaktion zugeordneten relevanten Aspekte gilt, daß die jeweiligen Kontextinformationen aufeinander abbildbar sind.*

Im trivialen Fall der leeren Schnittmenge wird die kontextuelle Konsistenz auf die kontextuelle Vollständigkeit der jeweiligen Interaktion reduziert. Somit ist die leere Schnittmenge auch ein Trivialfall für die kontextuelle Ersetzbarkeit:

> **Definition 13 (Kontextuelle Ersetzbarkeit)** *Eine Komponente gilt als kontextuell ersetzbar durch eine andere Komponente bezüglich einer gegebenen dritten Komponente, wenn die ersetzte Komponente mit der ersetzenden Komponente in jedem relevanten Aspekt bezüglich der Interaktion mit der dritten Komponente kontextuell konsistent ist.*

Wenn also beispielsweise die Verfügbarkeit eines Dienstes (relevanter Aspekt: *ServiceScope*) für eine Aufgabenstellung bei einer bestimmten *environmental related context information* (siehe Abschnitt 2.3.1) gegeben ist, so muß ein auf kontextueller Ebene im Sinne der Ersetzbarkeit interoperabler Dienst bei dieser Kontextinformation ebenfalls verfügbar sein. Als Konkretisierung dieses Beispiels für die Konsistenz seien zwei Dienste A und B genannt, die zu einem gegebenen Ort eine Apotheke mit Notdienst angeben können. Der Dienst A kann dann und nur dann durch Dienst B im Sinne der Interoperabilität ersetzt werden, wenn B für den gleichen Ort, der bei A zu einem Ergebnis führte (Ausgabe einer Apotheke mit Notdienst in der Nähe des eingegebenen Ortes), ein zulässiges Ergebnis ausgibt (gleiche Ortszuständigkeit). Die beiden Ergebnisse müssen jedoch nicht notwendigerweise gleich sein.

Ähnlich verhält es sich mit der *target related context information*: Ersetzbarkeit von A durch B bedeutet hier, daß B im gleichen Kontext wie A zielführend ist. So kann im Routing-Dienst-Beispiel aus Abschnitt 2.3.1 ein Routing-Dienst A durch einen Routing-Dienst B direkt ersetzt werden, sofern A und B Positionsinformationen des gleichen Aspekts als Eingabeparameter akzeptieren.

Die kontextuelle Ersetzbarkeit nimmt eine wichtige Rolle bei dem in Kapitel 5 vorgestellten Verfahren kontextbasierter Service Handover ein. Insbesondere die Bestimmung der Konsistenz wird dort noch einmal aufgegriffen und an einem praktischen Beispiel demonstriert.

2.3.3 Kontextuelles Dienstmodell

Das Munich Network Management (MNM) Team hat in [69] ein generisches Modell (*MNM Dienstmodell*) allgemeiner dienstbezogener Terminolgie, Konzepte und Strukturregeln vorgestellt, mit denen ein Dienst bzw. die Dienstnutzung aus verschiedenen Perspektiven beschrieben werden kann (z.B. *service view* vs. *implementation view*). Bei der Erstellung des Modells wurde beabsichtigt, die bei einer Dienstinteraktion handelnden Aktoren zu identifizieren und zu strukturieren sowie die korrespondierenden inter- und intraorganisatorischen Zusammenhänge zwischen diesen Aktoren festzuhalten. Alle Aktoren werden anhand ihrer Rolle bei einer Dienstinteraktion entweder der *customer domain* oder der *service provider domain* zugeordnet. Elemente, die keiner dieser beiden Domains zugeordnet werden können, werden als *side independent* eingestuft. Im *service view* des Modells bilden diese Elemente einen Dienst auf abstrakte Weise, weshalb die Menge dieser Elemente im Rahmen dieser Arbeit als abstrakter Dienst (*abstract service*) bezeichnet wird (siehe mittlerer markierter Bereich in Abbildung 2.9).

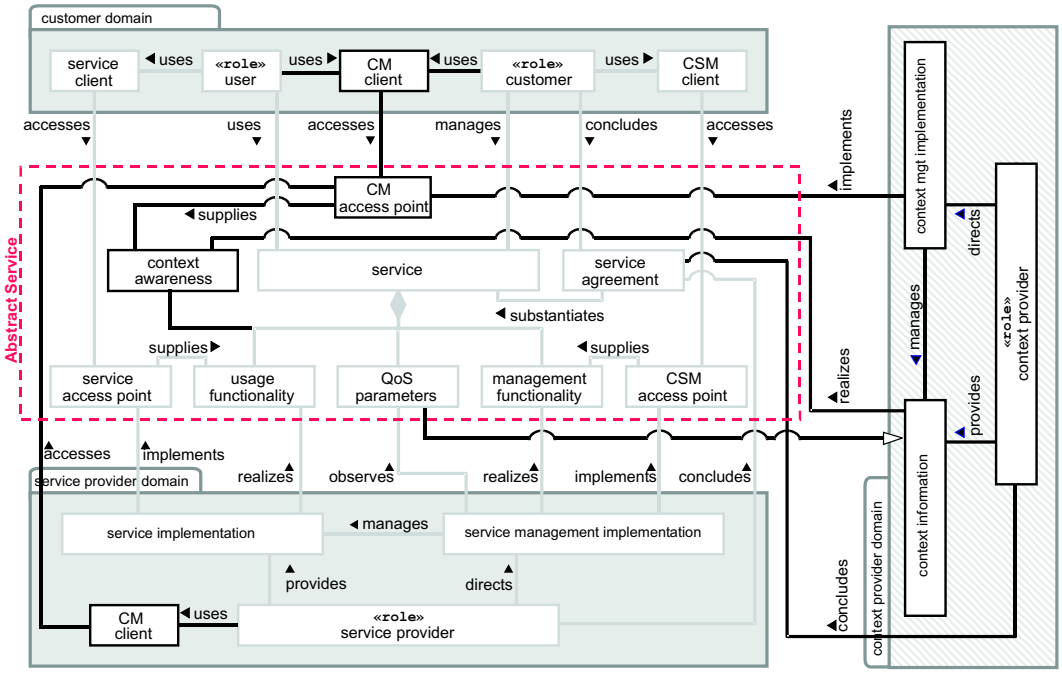

Abbildung 2.9: MNM Dienstmodell mit hervorgehobener Kontexterweiterung (MNMplusCE)

Das Design des MNM Dienstmodells entstand primär vor dem Hintergrund von Netzwerk-Management Aufgaben auf der Basis von Diensten zur Datenübertragung (carrier services). Aber aufgrund der Abstraktionsebene ist dieses Modell auch hervorragend geeignet, auf allgemeine Dienste (non-carrier services, NCS) gemäß Definition 1 auf Seite 6 angewandt zu werden. Darüber hinaus paßt das Modell zur Beschreibung von Ansätzen direkter Dienstnutzung (client ↔ server) wie auch indirekter Dienstnutzung mit Intermediate-Komponente, z.B. einer Middleware (client ↔ middleware ↔ server). [4] Bei der indirekten Dienstnutzung führt

[4] „↔“ drückt eine direkte Interaktionsbeziehung aus

die Middleware einen abstrakten Dienst aus, d.h. sie verhält sich wie ein Client gegenüber der *service provider domain*, und sie verhält sich wie ein Dienstanbieter gebenüber der *customer domain*, indem sie mindestens einen Dienstzugangspunkt (*service access point*) bereitstellt.

Um in der Lage zu sein, kontextuelle Abhängigkeiten von Diensten und Aufgaben der Bereitstellung von Kontext in ähnlicher und vor allem konsistenter Weise zu modellieren, wurde das allgemeine MNM Dienstmodell im Rahmen dieser Arbeit um eine *context provider domain* erweitert (siehe Abbildung 2.9 auf der rechten Seite). Diese Domäne umfaßt die Aktoren, die für die Erfassung, Aufbereitung und Bereitstellung von Kontextinformationen und das Management dieser Aktoren zuständig sind.

Dabei ist die *context provider domain* nicht einfach eine *service provider domain*, die Kontextinformationen liefert. Ihre herausragende Position ist u.a. dadurch motiviert, daß darin gruppierte Aktoren als „third party service provider“ in Dienstinteraktionen zwischen *customer domain* und einer Menge *service provider domains* gleichzeitig involviert sind.

Die *context management implementation* stellt einen *context management access point (CMAP)* bereit, über den beide anderen Domänen des Modells (*customer domain* und *service provider domain*) Zugiff auf den Kontext erhalten, um sowohl eine kontextadaptive Dienstsuche wie auch Dienstnutzung zu ermöglichen. Jeder Dienstnutzer, jeder Dienstanbieter oder auch eine Middleware-Komponente können die für spezifische Aufgaben relevanten Entitäten über ein Interface bestimmen, das vom CMAP bereitgestellt wird, was die zuständige *context management implementation* in die Lage versetzt, adäquate Kontextinformationen zu liefern.

Übertragungsdienste (carrier services) sind im kontextuell erweiterten MNM Dienstmodell als Spezialisierung von allgemeinen Diensten (non-carrier services) gemäß Definition 1 modelliert, und Quality-of-Service (QoS) Parameter, die eine spezifische Dienstinstanz beschreiben, können analog als Spezialisierung allgemeiner Kontextinformationen modelliert werden. Durch diese Vorgehensweise wird eine Betrachtung von Handover-Verfahren für Übertragungsdienste als Spezialisierung von Handover-Verfahren für allgemeine Dienste motiviert [156].

Im Folgenden wird das hier vorgestellte kontextuell erweiterte MNM Dienstmodell zur besseren Unterscheidung vom Original als *MNMplusCE Dienstmodell* bezeichnet.

2.4 Implikationen des Ubiquitous Computing

Das Problem, daß der Kontext einer Anwendung oder eines Dienstes in den Interoperabilitätsbetrachtungen nur unzureichend berücksichtigt wird, kommt insbesondere mit Etablierung der Ubiquitous Computing Systeme [173] zum Tragen. Abbildung 2.10 stellt die Entwicklung des Ubiquitous Computing Paradigmas - inspiriert durch [137] - dar.

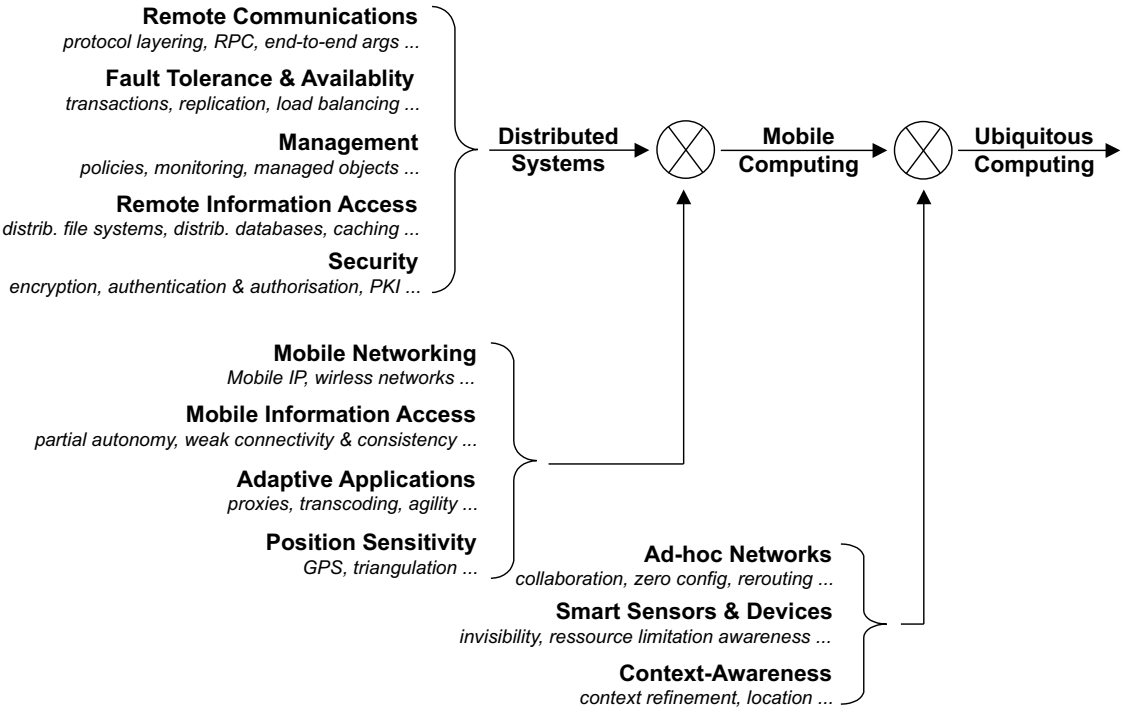

Abbildung 2.10: Entwicklung des Ubiquitous Computing Paradigmas

Ubiquitous Computing Systeme, synonym auch Pervasive Computing Systeme genannt, wurden zu Beginn des neuen Jahrtausends populär. Das Ubiquitous Computing Paradigma steht im wesentlichen für drei neue Facetten der Nutzung von verteilten Systemen:

1. Spontane Vernetzung (Ad-hoc Networking)
2. Intelligente Kleinstgeräte (Smart Devices, Sensors)
3. Kontextadaptive Dienste (Context-Awareness)

Die Berücksichtigung des Kontextes während Dienstsuche und Dienstnutzung erlaubt eine Weiterentwicklung des bis dahin vorherrschenden Paradigmas „any service for any person any time any where (at any cost)“ des *Mobile Computing* hin zum erstrebenswerteren „the right service for the right person at the right time and the right place“ nach Angermann et al. [25] bzw. Angermann [23]. Angermann konzentriert sich dabei auf die Betrachtung der Situation, also der Menge aller Kontextinformationen ohne Berücksichtigung der Relevanz für eine spezifische Aufgabenstellung, insbesondere Dienstinteraktion.

Dabei ist die neu eingeführte kontextuelle Ebene der Interopabilität nicht nur für den letzten Punkt von Bedeutung. Auch die spontane Vernetzung profitiert von der Spezifikation, welche Aspekte der jeweils aktuellen Netzwerksituation relevant und somit im Sinne der Ad-hoc Konfiguration zu berücksichtigen sind [128, 24]. Smart Devices, insbesondere mobile vernetzte Kleinstgeräte wie Mobiltelefone, PDAs oder einfache Sensor-Module (z.B. Thermometer,

Geräuschsensoren etc.) können einerseits wichtige Kontextinformationen über relevante Entitäten (z.B. Personen, Räume etc.) liefern. In Ubiquitous Computing Umgebungen werden Kontextinformationen auf der kontextuellen Interoperabilitätsebene verwendet, die durch Sensoren erfaßt werden und den Zustand relevanter Entitäten charakterisieren. Andererseits profitieren gerade kleine Geräte von Informationen, die aus dem Umfeld der Nutzung gewonnen werden. Geräte dieser Art haben oft nur beschränkte Ein- und Ausgabemöglichkeiten sowie eine limitierte Rechenleistung. Lassen sich Informationen wie z.B. die aktuelle Position aus dem Kontext ermitteln, müssen diese Daten nicht vom Benutzer eingegeben werden.

Eine ausführliche Einführung der meisten der in Abbildung 2.10 genannten Elemente und Evolutionsschritte wird u.a. von Samulowitz in [135] vorgenommen, weshalb an dieser Stelle auf eine Wiederholung weitgehend verzichtet wird. Lediglich auf die Elemente, die über die Darstellungen in [135] hinaus gehen oder die ein besonderes Gewicht im Zusammenhang mit dieser Arbeit haben, werden im Folgenden aufgegriffen. Dazu gehören Implikationen der Mobilität (2.4.1) und der Sensornetze (2.4.2).

2.4.1 Mobilität

Der Mobilität kommt in Ubiquitous Computing Systemen schon deswegen eine besondere Bedeutung zu, weil es sich beim Ubiquitous Computing Paradigma um eine Weiterentwicklung des Mobile Computing Paradigmas handelt. Dabei ist die Mobilität sehr differenziert zu betrachten, da sie je nach Betrachtungswinkel verschiedene Folgerungen impliziert. Die vier wichtigsten Arten der Mobilität und Ihre Implikationen werden im Folgenden vorgestellt:

- Bei der *Mobilität des Benutzers (user mobility)* steht die Agilität und die örtliche Bewegung des menschlichen Benutzers mit einem mobilen Endgerät im Vordergrund. Die Mobilität des Benutzers hat vor allem Auswirkungen auf die Art des Gerätes, welches der mobile Benutzer im Vergleich zum ortsfesten Benutzer unterwegs verwendet: Kleine, leichte und auf Batteriebetrieb ausgelegte Endgeräte. Auch die Interaktion mit dem Gerät selbst unterscheidet sich von der Art der Benutzung ortsfester Geräte. Der Nutzer verwendet unterwegs in der Regel viel weniger Zeit zur Interaktion mit dem Gerät, weil er dafür z.B. im Straßenverkehr keine mentalen Kapazitäten frei hat oder über die Miniatur-Tastatur eines Mobiltelefons keine langen Texte eingeben möchte. Dafür akzeptiert er aber unter Umständen in begrenztem Umfang höhere Kosten, um auch unterwegs elektronischen Zugriff auf aktuelle Daten zu haben.
- Dazu verwandt ist die *Mobilität des Gerätes (device mobility)*. Hier stehen die Implikationen der Mobilität von Geräten im Vordergrund. Die Anforderung mobiler Benutzer, kleine und leichte Endgeräte für die Nutzung unterwegs zu haben, wirkt sich vor allem auf die Leistungsfähigkeit der mobilen Endgeräte im Vergleich zu stationären Endgeräten aus. Einschränkungen sind vor allem beim Speicher (flüchtig und nicht flüchtig), bei der Prozessorleistung, bei der Energieversorgung und bei den Ein- und Ausgabemöglichkeiten gegeben. Auch bei der Netzanbindung dieser Geräte ist mit zum Teil deutlich teureren, qualitativ stark schwankenden Verbindungen zu rechnen. Dafür stehen in der Regel gleich mehrere Netzwerkinterfaces pro Endgerät (z.B. GSM-(HS)CSD + GPRS + Bluetooth bei Mobiltelefonen) zur Verfügung.

- Der Wechsel zwischen verschiedenen Geräten eines Benutzers bei fortgesetzter Nutzung des gleichen Datenbestands wird unter dem Begriff der *Mobilität einer Sitzung (session mobility)* geführt. So ist es z.B. sinnvoll, eine Reiseroute mit einem ortsfesten Endgerät wie einem PC mit großem Bildschirm, zuverlässiger Netzanbindung etc. auszuarbeiten. Anschließend werden die wichtigen Eckpunkte der so gefundenen Route auf ein mobiles Endgerät übertragen, um diese im Verlauf der Reise z.B. auf einem Mobiltelefon abrufen und verändern zu können. Dies ist dann unter Umständen sogar bei fehlendem (z.B. keine Netzabdeckung) oder verbotenem (z.B. Flugzeug) Online-Zugriff möglich. Auswirkungen der Mobilität einer Sitzung sind insbesondere im Bereich der verteilten Datenhaltung zu finden, wo z.B. Aufgaben zur Vermeidung von Dateninkonsistenzen anfallen.

- Wenn sich eine Dienstinstanz durch die Netzwerktopologie bewegt, spricht man von der *Mobilität eines Dienstes (service mobility / code mobility)*. Diese Art der Mobilität findet vornehmlich in Agentensystemen [65] Verwendung. Sie hat besonders hohe Anforderungen an das verteilte Lifecycle-Management. Dieses setzt den Mechanismus der *Migration* voraus, der sich aus folgendem Ablauf zusammensetzt:

  – Unterbrechung der Ausführung eines Dienstes auf einem Gerät
  – Transfer des den Dienst implementierenden Codes zu einem anderen Gerät im Netzwerk
  – anschließende Wiederaufnahme der Ausführung des Dienstes auf dem neuen Gerät an der Stelle, an der er zuvor unterbrochen wurde

  Die Migration und andere Anforderungen sind so schwerwiegend, daß viele von ihnen auf mobilen Endgeräten aufgrund der Ressourcen-Beschränkungen gar nicht umgesetzt werden können, wie in [157] gezeigt wird. Auch die Sicherheit ist bei der Mobilität von Diensten ein Problem. Wenn fremder Code auf dem eigenen Gerät ausgeführt werden soll, muß es einen funktionierenden Sicherheitsmechanismus geben, um sowohl das Gerät vor dem mobilen Code als auch den mobilen Code vor dem unberechtigten Zugriff aus dem Gerät, auf dem er läuft, zu schützen. Vor allem das Sicherheitsproblem führt auch heute noch dazu, daß sich Systeme mit mobilen Agenten nur deutlich gehemmt außerhalb des universitären Bereichs verbreiten.

Betrachtet man die Dienstinteraktion im Sinne des Kontextes, bedeutet die Mobilität nichts anderes als eine *Veränderung von Kontextinformationen* (in der Regel über die Zeit). Die Mobilität ist eine der Ursachen für die besondere Anforderung in Ubiquitous Computing Systemen, sich ständig auf veränderte Umgebungsbedingungen einstellen zu müssen. Aber sie ist auch Quelle von Kontextinformationen, die aus der Umgebung gewonnenen wurden. Insbesondere aus der Mobilität der Geräte in Ubiquitous Computing Systemen und der daraus resultierenden örtlichen Veränderung des Gerätes kann eine interessante Kontextinformation gewonnen werden: Die örtliche Position in Bezug auf ein Referenzsystem (z.B. WGS84), modelliert als *Location*. Typische Fragestellungen, die u.a. durch Anwendung eines Location Modells beantwortet werden sollen, sind „Was ist der momentane Ort X von Objekt Y?“ bzw. „Welche Objekte Z_i sind in der Nähe von Ort X?“. Ortsinformationen werden dabei üblicherweise entweder *symbolisch*, d.h. durch Beschreibung eines Ortes mit abstrakten Symbolen und Namen wie „Terminal A“, oder *geometrisch* durch Angabe von Koordinaten in einem

Referenzsystem angegeben. Das zugrundeliegende Modell wird dementsprechend entweder als symbolisches Modell oder geometrisches Modell bezeichnet. Ein Location Modell, das beide Formen integriert betrachtet, wurde von Leonhardt [102] vorgeschlagen. Aus Gründen der Skalierbarkeit und zur Auswertung der Relation der relativen Nähe werden in beiden Modelltypen Ortsangaben normalerweise hierarchisch organisiert. Je nach Aufgabenstellung, z.B. der Verwaltung statischer Datenbestände mit hoher Komplexität der einzelnen Ortsbeschreibungen [145] (Polygonzüge u.ä.) gegenüber der Verwaltung sehr dynamischer Datenbestände mit relativ einfachen Ortsbeschreibungen [176] (XY-Koordinaten u.ä.), können sowohl die Organisationsform als auch das Ortsmodell für die jeweilige Aufgabenstellung optimiert sein.

Die Gewinnung von Kontextinformationen und ihre automatische Verarbeitung im Service Framework wird nicht uneingeschränkt positiv gesehen. Ein kritischer Artikel über die Probleme des Datenschutzes ist z.B. in [18] zu finden, während ein kritischer Bericht über allgemeine Auswirkungen des Pervasive Computing auf Gesundheit und Umwelt in [85] zu finden ist. Dahingegen stellt ihre Verwendung aus rein technischer Sicht eine deutliche Bereicherung dar.

2.4.2 Sensornetze

Sensornetze wurden bereits von Samulowitz in [135] betrachtet, wobei Sensoren als Elemente von *Smart Nodes*, ihre föderative Organisation auf der Netzwerkebene und ihre Eignung zur Gewinnung von Kontextinformationen eine Rolle spielten. Darüber hinaus werden in diesem Abschnitt noch einige Beispiele für interessante Sensortypen und die von ihnen direkt gelieferten Daten (siehe Abbildung 2.11) sowie ihre Verarbeitung vorgestellt. Weitere Beispiele sind bei Chtcherbina und Franz [56], Schmidt et al. [140] oder Schmidt und Laerhoven [141] zu finden. Eine umfangreiche Bestandsaufnahme der Einsatzgebiete von Sensoren zur Unterstützung der Context-Awareness wird u.a. von Chen und Kotz in [51] vorgenommen.

Abbildung 2.11: Vielfältige Sensordaten

Die Art der Informationen, die eine Dienstinteraktion beeinflussen können, ist sehr vielseitig. Allgemein kann jede von einem Sensor erfaßte Größe als Kontextinformation dienen und ist damit Teil der Situation einer Dienstinteraktion nach Definition 8. Die Sensoren selbst können in Form einer eigenständigen Hardware-Komponente in das Netzwerk integriert sein, oder auch nur als reine Software-Komponente vorliegen. Letztere werden häufig als Agenten realisiert, da ein hoher Grad an Übereinstimmung zwischen den Eigenschaften von Agenten

(spezialisierte Aufgabenstellung, partielle Autonomie etc.) und denen von Software-Sensoren besteht.

Wie in Abschnitt 2.3.1 erläutert, werden sekundäre Kontextinformationen im Rahmen der Erfassung primärer Kontextinformationen nachgeschalteter Verarbeitungsschritte gewonnen, was *Kontextveredelung (context refinement)* genannt wird. Werden bei diesem Veredelungsprozeß Kontextinformationen mehrerer Quellen gleichzeitig zu einer neuen Kontextinformation zusammengeführt, spricht man auch von der *kontextuellen Datenfusion (context fusion)*. Da das Ergebnis eines kontextuellen Datenfusionsprozesses wieder eine Kontextinformation ist, kann der Prozeß als (Software-)Sensor modelliert und implementiert werden, um die Details des Vorgangs zu kapseln bzw. zu abstrahieren und damit alle Vorteile eines offenen, verteilten Systems zu nutzen (Erweiterbarkeit, Redundanz, Management etc.)

Eine Architektur, die diesen Ansatz der Kontextveredelung und Kontextfusion verfolgt, wurde im Rahmen des Projekts TEA [13, 141] entworfen. Abbildung 2.12 zeigt deren Verarbeitungsprozeß auf verschiedenen Ebenen.

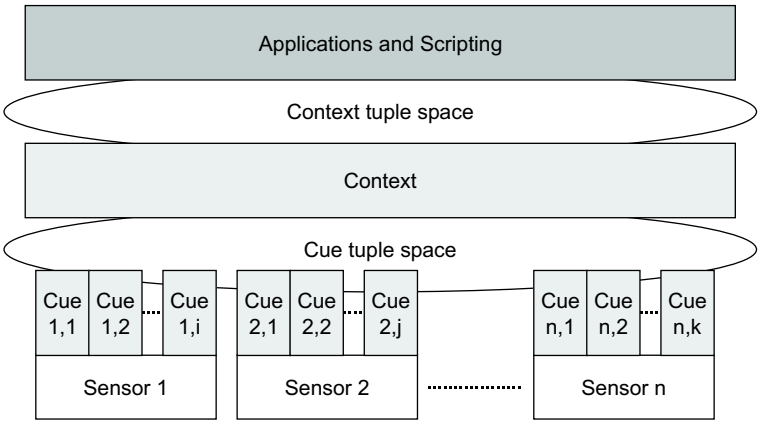

Abbildung 2.12: Kontextveredelung in TEA

Cues stellen in TEA Informationen dar, die unmittelbar durch das Auslesen von Sensoren gewonnen werden, und sind damit in etwa äquivalent zu primären Kontextinformationen im Sinne dieser Arbeit. Jeder Cue erhält seine Informationen von genau einem Sensor, allerdings können auch andere Cues ihre Informationen vom gleichen Sensor erhalten. In TEA werden Cues vorwiegend benutzt, um Änderungen der Sensorik transparent für die Ebene der eigentlichen Kontextverarbeitung zu machen und das von den Sensoren ausgehende Datenvolumen zu reduzieren oder zu extrapolieren, insbesondere unter Verwendung von Statistiken. Der Fusionsprozeß, also die Zusammenführung mehrerer Cues zu komplexeren Kontextinformationen höherer Ordnung, vergleichbar mit sekundären Kontextinformationen, findet dann auf einer eigenen Kontextebene statt. Zur temporären Zwischenspeicherung von Kontextinformationen einfacher Ordnung (Cues) und höherer Ordnung stehen in TEA Datenpuffer bereit, die Tupelraum *tuple space* genannt werden. Um die Anzahl von Kontextinformationen im jeweiligen Tupelraum zu begrenzen, erhält jede Kontextinformation eine Gültigkeitsdauer, nach deren Ablauf sie automatisch wieder entfernt wird. Jede Kontextinformation höherer Ord-

nung erhält neben einer Gültigkeitsdauer auch noch einen weiteren Qualitätsparameter, der das Maß der Sicherheit der erzeugenden Instanz angibt. Auf der Anwendungsebene werden Mechanismen bereitgestellt, im Sinne der Context-Awareness auf die drei Ereignisse *entering a context*, *leaving a context* und *while in context* auf der Basis von Schwellwertentscheidungen zu reagieren.

Einen ganz anderen Ansatz verfolgt Samulowitz in [135]. Hier wird ein datenzentrisches Protokoll namens *Smart Stack Routing (SSR)* dazu verwendet, Daten von Sensoren beim Durchwandern von Nachrichten durch Sensornetzwerke zu sammeln. Steueranweisungen in den Nachrichten legen fest, wie beim Durchlaufen der einzelnen Sensorknoten der Fusionsprozeß ablaufen soll. Dieses Verfahren skaliert möglicherweise schlecht in großen Sensornetzen aufgrund der Speicherung von Pfadinformationen in der Nachricht. Allerdings konnten Ghose et al. in [70] zeigen, daß gerade eine datenzentrische Vorgehensweise in Kombination mit Datenreplikation eine optimale Ausnutzung von Sensornetzwerken erlauben. Diese Kombination wirkt sich besonders positiv auf zwei der größten Problembereiche von Sensornetzwerken aus: die Anfälligkeit auf Störungen und geringe Energie-Kapazitäten [128].

2.4.3 Adaptive Dienstnutzung in mobilen Endgeräten

Eine Herausforderung in ubiquitären Systemen besteht insbesondere in der Lokalisierung von Diensten (*service discovery*) in einer neu besuchten oder sich ständig verändernden Umgebung als Hauptaufgabe der Dienstvermittlung, die ihrerseits eine der zentralen Aufgaben jedes Service Frameworks ist. Die Lokalisierung ist kein einmaliger Vorgang pro Dienstnutzung, sondern erstreckt sich aufgrund der sich ständig verändernden Umgebungsbedingungen auch auf die Ausführungszeit eines Dienstes. *Adaptive Dienstnutzung* umfaßt im Folgenden sowohl die Anpassung des Service Frameworks im Sinne der Dienstvermittlung an die ständig variierende Verfügbarkeit unterschiedlicher Dienste, als auch die Berücksichtigung sich ändernder externer und interner Einflußgrößen durch die einzelnen Dienste.

Erfolgt eine Dienstnutzung auf einem mobilen Endgerät, so sind vor und während der Dienstnutzung die besonderen Eigenschaften mobiler Endgeräte zu berücksichtigen. Mobile Endgeräte zeichnen sich im Vergleich zu Endgeräten im Festnetz in der Regel neben unterschiedlicher Hard- und Software insbesondere durch Beschränkungen der Prozessor- und Speicherkapazität (flüchtige wie nicht flüchtige), limitierte Ein- und Ausgabemöglichkeiten und stark variable Netzwerkverbindungen aus, was typische Merkmale von ubiquitären Geräte sind. Durch die Berücksichtigung dieser Merkmale in ihrer individuellen Ausprägung auf dem jeweiligen Gerät können einerseits Überlastungen des Geräts vermieden werden. Solche Überlastungen entstehen z.B. bei der Verarbeitung von Dokumenten im Endgerät, die größer als der Hauptspeicher des Geräts sind. Andererseits kann die Dienstnutzung durch Kenntnis der Merkmale im Sinne z.B. der Nutzerpräferenzen optimiert werden. Ein Beispiel für den letztgenannten Fall wäre der Download einer größeren Datenmenge auf einem Endgerät mit mehreren Netzwerkinterfaces (z.B. Bluetooth und GPRS) nur zu den Zeiten, zu denen eine kostengünstige Onlineverbindung besteht [24]. Ein anderes Beispiel wäre die serverseitige Konvertierung von Grafiken bezüglich der unterstützten Grafikformate eines Endgeräts sowie die Skalierung der Grafiken in eine Auflösung, die auf dem Endgerät mit der Displaygröße z.B. eines Mobiltelefons überhaupt anzeigbar ist [146]. Dieses Beispiel zeigt, daß die Informationen über die

beschränkenden Parameter bzw. Präferenzen an vielen Stellen eines verteilten Systems (hier des Service Frameworks) und nicht nur im Endgerät selbst bekannt sein müssen, um eine Optimierung zu ermöglichen.

Zur Beschreibung der beschränkenden Parameter bzw. Preferenzen wurden von unterschiedlichen Gremien verschiedene Standards entworfen. Sie alle ermöglichen die *profilbasierte Adaption* der Dienstnutzung, da die Informationen, auf die die jeweiligen Algorithmen zur Adaption der Dienstsuche oder Dienstnutzung zurückgreifen, aus statischen Profilen stammen. Werden dabei insbesondere individuelle Nutzerpräferenzen berücksichtigt, spricht man auch von *personalisierter Adaption*.

Einer dieser Standards ist der vom W3C entworfene *Composite Capabilities / Preferences Profile (CC/PP)* [168], der ein Modell und darauf aufbauend ein Basisvokabular zum Beschreiben von Geräteeigenschaften und Nutzerpräferenzen spezifiziert. Es handelt sich im wesentlichen um einen Katalog von Beispieldaten. Interessierte Dritte wie z.B. Gerätehersteller und Anwendungsentwickler sind aufgefordert, eigene Attribute zu definieren, um ihre Geräte oder Applikationen zu beschreiben. Entwurfsgrundlage für CC/PP war ein Web-Browsing-Scenario, bei dem die ausgetauschten Profildaten der Aushandlung von Inhaltsformaten zwischen Web-Browser und Web-Server (*content negotiation*) dienen. Die Nutzung eines CC/PP Profils erfolgt dementsprechend nach folgendem Schema: Wenn ein Endgerät eine Anfrage per HTTP an einen Web-Server schickt, übermittelt es im HTTP-Header das CC/PP Profil in der Anfrage mit. Der Web-Server kann daraufhin die zu verarbeitenden Daten filtern, transformieren und entsprechend den Anforderungen des Clients adaptieren. CC/PP Profile werden in Ressource Description Framework (RDF) Dokumenten mit Tripeln der Form Subjekt-Prädikat-Objekt repräsentiert (siehe auch Abschnitt 3.1.2). RDF kann in verschiedenen Formaten serialisiert werden (z.B. als Graph), jedoch legt die CC/PP Spezifikation als Serialisierungsformat XML fest.

Ein anderer Standard zur Beschreibung von Profilen ist das vom WAP Forum entwickelte *User Agent Profile (UAProf)* [171]. UAProf baut auf CC/PP auf, und damit auch auf RDF und XML. Die UAProf Spezifikation definiert die Kategorien `HardwarePlatform`, `SoftwarePlatform` und `BrowserUI` als Spezialisierung (subclass) der generischen CC/PP `Component` Klasse sowie einige Attribute zu diesen spezialisierten Klassen. CC/PP und UAProf finden u.a. auch im *Mobile Execution Environment (MExE)* [19] Entwurf des 3GPP Verwendung. Beide Standards sehen ebenfalls bereits standardmäßig vor, aus Datenreduktionsgründen nicht immer vollständige Profile zwischen den beteiligten Kommunikationspartnern auszutauschen, sondern nur die Teile eines Profils, die von definierten Referenzprofilen abweichen. Der umgekehrte Weg, das Zusammenführen der Daten des Referenzprofils und der inkrementellen Daten, wird *Profile Resolution* genannt und ist in Abbildung 2.13 dargestellt. Bei der Resolution ist insbesondere zu beachten, daß zumindest im Vorfeld, in der Regel aber zum Zeitpunkt der Resolution, eine Verbindung des Resolvers zum Profile Repository bestehen muß. Sollte das Referenzprofil zum Zeitpunkt der Resolution nicht verfügbar sein, dann können auch die bei der Interaktion ausgetauschten Profildaten nicht verwendet werden.

Werden bei der adaptiven Dienstnutzung nicht nur Daten aus statischen Profilen bei den Adaptionsalgorithmen verwendet, sondern auch dynamische Daten externer und interner Einflußgrößen berücksichtigt, so ermöglicht dies die *kontextbasierte Adaption* der Dienstnutzung.

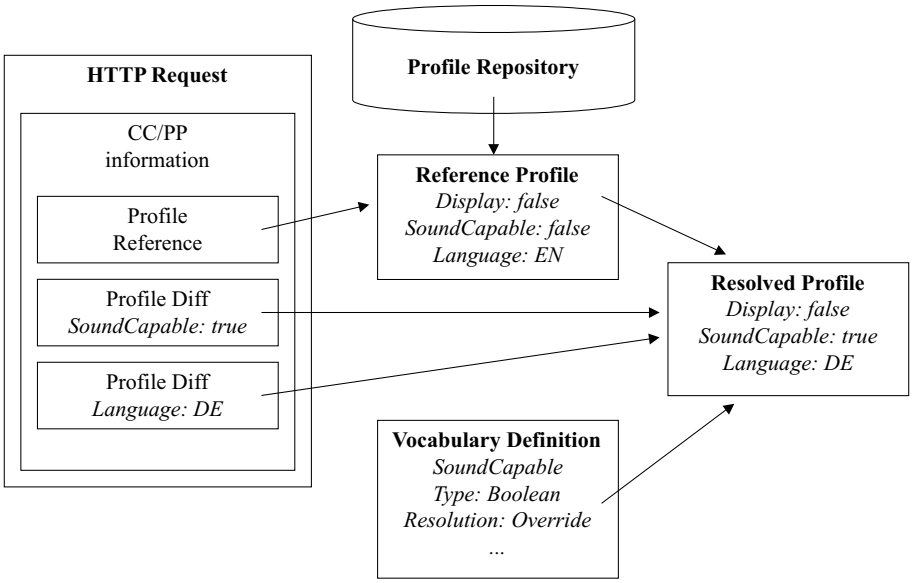

Abbildung 2.13: CC/PP und UAProf Profile Resolution

Erste Versuche, diese Art von Daten mit den vorhandenen Standards zur Beschreibung von Profildaten wie CC/PP und UAProf auszutauschen, waren nicht sehr vielversprechend. Indulska et al. führen dies in [90] insbesondere auf Schwächen in der XML-Serialisierung bzw. generell in der Repräsentation in RDF zurück. Auch Butler sieht in [46] insbesondere Änderungsbedarf am Serialisierungsformat von CC/PP und UAProf. Aber auch Mängel im Design von CC/PP selbst, z.B. fehlende Vorgaben zu Updates einzelner Attribut-Werte und die allgemein höhere Komplexität von Kontextmodellen werden in [90] als Mangel angeführt.

Ein anderer Ansatz, der in begrenztem Maße auch die Beschreibung kontextueller Bedingungen in eine Profildefinition mit einfließen läßt, ist die *Pervasive Profile Description Language (PPDL)* [56]). Der Umfang dieser Sprache und der Umfang der auswertbaren kontextuellen Größen scheint aber recht statisch zu sein, und die Designprinzipien sind nicht veröffentlicht und daher in hohem Maße proprietär.

Die sich ändernden Umgebungsparameter lassen sich hervorragend durch die in dieser Arbeit vorgestellten Modelle und Protokolle abbilden, da sie in ureigenster Weise den Kontext einer Dienstinteraktion darstellen.

Da eine der wichtigsten Aufgaben jedes Service Frameworks die Dienstvermittlung ist und im Verlauf dieser Arbeit häufiger auf die verschiedenen Varianten eingegangen wird, werden im Abschnitt 2.4.3 einige allgemeine Protokolle zur Dienstvermittlung behandelt. Im Anschluß daran werden einige konkrete Service Frameworks mit Schwerpunkt auf Architekturen für mobile Dienste und Geräte vorgestellt, die verschiedene Ansätze zur Lösung der jeweiligen Aufgabenstellungen bei der (kontext-)adaptiven Dienstnutzung darstellen. Bei diesen kon-

kreten Service Frameworks finden zum Beispiel teilweise eigene Dienstvermittlungsprotokolle Verwendung.

Allgemeine Dienstvermittlungsprotokolle

Die Vermittlung von Diensten ist eine der wichtigsten Aufgabe in jedem Service Framework. Die grundsätzliche Aufgabenstellung für Dienstvermittlungsprotokolle ist identisch:

- Suche nach Dienstvermittlern und Diensten in spontan vernetzten Umgebungen
- Automatische Anpassung an die sich ständig ändernde Netzwerksituation
- Einschränkung des Suchraums durch Angabe von Parametern, die den Dienst charakterisieren

Trotzdem unterscheiden sich die einzelnen Varianten erheblich bezüglich folgender Merkmale:

- Die Suche nach Dienstvermittlern ist netzwerkspezifisch
- Art und Umfang der spezifizierbaren Parameter, die einen Dienst beschreiben (*service template*)

Im Folgenden wird eine Übersicht über die gängigen Dienstvermittlungsprotokolle gegeben, die unabhängig von einem bestimmten Service Framework in vielen verteilten Systemen Verwendung finden. Dabei wird insbesondere auf die Gemeinsamkeiten, aber mehr noch auf die Unterschiede und die sich daraus ergebenden Vor- und Nachteile für die verschiedenen Einsatzzwecke eingegangen.

SLP Das IETF *Service Location Protocol (SLP)* nach RFC 2608 [78] ist ein agentenbasiertes Protokoll zur Dienstvermittlung in IP-Netzen. Es gibt drei Arten von Agenten mit unterschiedlichen Aufgaben: Ein *User Agent* übernimmt im Auftrag der suchenden Entität die Suche nach einem Dienst beim *Directory Agent*, der wiederum eine Liste von Diensten führt, über die er von *Service Agents* erfährt. Mehrere Directory Agents können dabei zwecks Lastverteilung und Fehlertoleranz zu Föderationen zusammengeschlossen werden. Die Suche nach Directory Agents seitens der beiden anderen Typen von Agenten erfolgt per IP Multicast.

Dienstangebote werden beim gewünschten Directory Agent per Unicast vom Service Agent unter Angabe einer URL, Angaben zu Diensttyp und optional weiteren beschreibenden Parametern angemeldet (siehe Beispiel in Abbildung 2.14). Hierzu kann auf Service Templates nach RFC 2609 [77] zurückgegriffen werden. Wenn diese Anmeldung nicht innerhalb einer festgelegten Zeitspanne wiederholt wird, dann wird der Eintrag für diesen Dienst wieder automatisch vom Directory Agent entfernt, was eine zeitverzögerte „selbstreinigende“ Wirkung hat (vgl. Jini Leases).

Der User Agent schickt zur Suche eines Dienstes ein Template mit den gewünschten, den Dienst beschreibenden Parametern an den Directory Agent, der im Falle eines positiven

```
URL = service:printer:tcppostscript
Attributes = (&(queuelength<=3)(papersize=A4)(!(status=paper_out)))
```

Abbildung 2.14: SLP Service Filter

Matchings mit einer Liste erfolgreicher Dienstendpunkte antwortet, was in diesem Fall Uniform Resource Locators (URLs) sind. SLP verwendet bei den Templates *LDAPv3 Suchfilter* [169, 88], die in eingeschränktem Maß auch ein weiches Matching auf der Basis von Prädikaten wie *greaterOrEqual*, *lessOrEqual*, *present* oder auch *substrings* erlauben (vgl. `Attributes=..` in o.g. Beispiel).

In SLP kann der Verfügungsbereich eines Dienstes (*Scope*) innerhalb einer Anmeldung durch eine Liste von Scopenamen angegeben werden. In diesem Fall ist es Aufgabe des User Agents, zu ermitteln, ob sich der Dienstsuchende in einem dieser Scopes befindet. Der Scope-Mechanismus von SLP ist sehr einfach und beschränkt sich auf die Überprüfung des Scopenamens. Für komplexere Aufgaben, wie sie z.B. im Rahmen von Handover-Verfahren benötigt werden, ist der Scope-Mechanismus von SLP unzureichend.

Jini *Jini* [29] ist eine Entwicklung von Sun Microsystems. Es basiert auf Java, und setzt eine *Java Virtual Machine (JVM)* sowie ein bereits konfiguriertes (meistens TCP/IP-)Netzwerk auf allen Geräten voraus, die Jini einsetzen. In Jini erfolgt eine konsequente Dienstorientierung, die als Paradigma hat, daß alles (Hardware, Software, Benutzer etc.) ein Dienst ist. Zentrale Komponente des Jini-Systems ist der Lookup-Service (LUS), der gemäß dem Dienstparadigma wie alle übrigen Dienste als Dienst über ein spezifiziertes Interface ansprechbar ist, um nach anderen Diensten zu suchen.

Die Suche nach dem LUS wird in Jini *Discovery* genannt, die Suche nach einem Dienst innerhalb des LUS *Lookup*. Jeder Server, der einen Dienst im System bekannt machen möchte, muß seinen Dienst bei einem LUS seiner Wahl registrieren (*Join*). Dazu erhält er vom LUS eine Referenz des Objekts *ServiceRegistrar*, mit deren Hilfe der Server beim LUS ein sogenanntes *Proxy-Objekt* in serialisierter Form und eine Menge von Dienstattributen (*Entries*) hinterlegt, die den Dienst näher beschreiben. Jeder Eintrag im LUS verliert nach Ablauf einer angebbaren Zeit (*Lease*) seine Gültigkeit und wird dann aus dem Verzeichnis entfernt.

```
public interface ServiceTemplate implements Serializable {
  public ServiceTemplate(ServiceID serviceID,
                         Class[] serviceTypes,
                         Entry[] attributes);
}
```

Abbildung 2.15: Jini ServiceTemplate

Ist ein Client auf der Suche nach einem neuen Dienst, dann schickt er ein sogenanntes

ServiceTemplate an den LUS. Dieses ServiceTemplate beinhaltet wahlweise eine eindeutige Service-ID, den Typ eines Services in Form eines Interfaces, eine Menge von Attributen oder eine Kombination dieser drei Informationstypen, siehe Abbildung 2.15.

Von jedem Eintrag im LUS, der dem angegebenen ServiceTemplate genügt, wird dann ein *ServiceItem* an den anfragenden Client geschickt. Der dabei verwendete Algorithmus erfordert ähnlich wie bei SLP ein exaktes Matching der hinterlegten Parameter mit den angefragten Parametern, bei der Standard-Implementierung des LUS sogar in serialisierter Form. Der Client hat dann die Möglichkeit, über das ServiceItem seiner Wahl auf das dazugehörige, beim LUS hinterlegte Proxy-Objekt zuzugreifen. Über den Funktionsumfang des Proxy-Objekts macht Jini keine einschränkenden Aussagen. Im einfachsten Fall beinhaltet der Proxy bereits den gesamten Service, im anderen Extremfall ist der Proxy nur für die Umsetzung des Kommunikationsprotokolls mit dem Server zuständig.

UPnP Im Gegensatz zu z.B. Jini setzt *Universal Plug and Play (UPnP)* [148, 9] bereits an den unteren Ebenen des Netzwerk-Protokollstacks an und ermöglicht damit die Kooperation von Komponenten ohne Konfigurationsvoraussetzungen. So bietet UPnP mit der Unterstützung des *Dynamic Host Configuration Protocols (DHCP)*, *AutoIP* und *Multicast DNS (MDNS)* drei wichtige Basisprotokolle, um die Konfiguration eines IP-Netzes auch bei spontan vernetzten Geräten auf ein Minimum zu reduzieren.

```
HTTP Request:     M-SEARCH * HTTP/1.1
                  S: uuid:ijklmnop-7dec-11d0-a765-00a0c91e6bf6
                  Host: 239.255.255.250:reservedSSDPport
                  Man: "ssdp:discover"
                  ST: ge:fridge
                  MX: 3

HTTP Response:    HTTP/1.1 200 OK
                  S: uuid:ijklmnop-7dec-11d0-a765-00a0c91e6bf6
                  Cache-Control: no-cache="Ext", max-age = 5000
                  ST: ge:fridge
                  USN: uuid:abcdefgh-7dec-11d0-a765-00a0c91e6bf6
                  AL: <blender:ixl><http://foo/bar>
```

Abbildung 2.16: SSDP Request/Response

Nach der automatischen Konfiguration eines neuen Geräts im Netzwerk (UPnP Addressing, Ebene 0) benachrichtigt dieses die erreichbaren Knoten im Netzwerk per HTTP Multicast UDP (HTTPMU) über seine Existenz, die wiederum dem neuen Gerät mit einer HTTP Unicast UDP (HTTPU) antworten (UPnP Discovery, Ebene 1). Die so ausgetauschten Nachrichten beinhalten URLs, die einerseits den zur Verfügung gestellten Dienst identifizieren und andererseits auf den zuständigen *Description Server* des jeweiligen Geräts zeigen, von dem weitere Beschreibungen des Dienstes im XML-Format abgerufen werden können. Die dabei zu verwendende Syntax und das Protokoll wer-

den im *Simple Service Discovery Protocol (SSDP)* [72] definiert, welches seinerseits die HTTP-Erweiterungen MAN [119] und DASL [131] aufgreift und verwendet.

Service Namen (USN) und Service Typen werden in SSDP als UUIDs [12] ausgedrückt (siehe Beispiel in Abbildung 2.16, entnommen aus [72]), und auch SSDP erfordert exaktes Matching.

Bluetooth SDP Das *Bluetooth Service Discovery Protocol (Bluetooth SDP)* ist Teil des Bluetooth Protocolstacks [8]. Es setzt direkt auf dem *Logical Link Control and Adaptation Protocol (L2CAP)* auf, welches Verbindungen zu Geräten in der Reichweite von Bluetooth aufbauen kann. SDP erlaubt das Auffinden von Diensten, die auf anderen Bluetooth-Endgeräten in Reichweite angeboten werden. Jedes Bluetooth-Gerät, welches Dienste für andere Bluetooth-Geräte anbietet, unterhält als SDP Server eine Liste von Datensätzen (*Service Records*), welche die angebotenen Dienste beschreiben und die über ein im *SDP Profile* spezifiziertes Interface abgefragt werden können. Die Bluetooth-Spezifikation legt weder fest, wie diese Datensätze intern vorgehalten werden (z.B. in einer Datenbank oder als XML-File), noch wie auf die jeweiligen Dienste zugegriffen wird, sondern lediglich wie die Informationen über die Dienste in den Service Records codiert sind, und wie auf die Service Records zugegriffen werden kann. Es gibt kein zentrales Dienste-Register, und es erfolgt auch keine periodische Ausstrahlung von Dienstbeschreibungen ähnlich den *Announcements* bei Jini oder SLP - der Austausch von Dienstbeschreibungen erfolgt rein bilateral zwischen einem SDP Client und einen SDP Server auf der Basis eines einfachen Request/Response-Protokolls.

Die Signalisierung über das Vorhandensein eines neuen Bluetooth-Geräts in Reichweite ist nicht Bestandteil von SDP und erfolgt auf anderen Ebenen des Bluetooth-Protokollstacks. Im Fall einer Veränderung der Menge der erreichbaren Bluetooth-Geräte kann der SDP Client dazu verwendet werden, Dienstbeschreibungen von neuen SDP Servern auf zwei verschiedene Arten zu erhalten:

1. Service Browsing
2. Service Searching

Im ersten Fall erfolgt eine Suche nach Diensten ohne a-priori Wissen über die Eigenschaft der angebotenen Dienste. Diese Art der Suche dient in der Regel dazu, einem Menschen die Liste der auf dem fernen Gerät angebotenen Dienste (z.B. Liste der unterstützten Zugangsprofile wie Serial Profile, Dialup Networking Profile oder LAN Access Profile) anzuzeigen und wird selten direkt von einer Anwendung verwendet. Letztere verwenden in der Regel die zweite Art (Service Searching), bei der nach Dienstbeschreibungen mit bestimmten charakterisierenden Merkmalen gesucht wird. Diese Merkmale werden als *Service Attribute* der Service Records ausgedrückt. Die Namen der Attribute sind dabei nicht frei wählbar, sondern müssen als UUID [12] angegeben werden. Eine besondere Bedeutung hat das Attribut `ServiceClassIDList` (welches natürlich auch als UUID repräsentiert ist), die darin aufgelisteten Werte (ebenfalls UUIDs) kennzeichnen, zu welchen Dienstklassen der durch den aktuellen Service Record beschriebene Dienst als Instanz anzusehen ist. Das bedeutet, durch Angabe einer bestimmten `ServiceClassIDList` im *ServiceSearchPattern* kann nach Diensttypen ähnlich den `serviceTypes` in Jini gesucht werden, siehe Beispiel in Abbildung 2.17 (UUIDs wo möglich durch offizielle Bezeichnung ersetzt).

```
SDP_ServiceSearchRequest {
    ServiceSearchPattern =
        DataElementSequence[1] {
            (ServiceClassIDList_uuid, PrinterServiceClassID_uuid)
        },
    MaximumServiceRecordCount = 5,
    ContinuationState = 0
}
```

Abbildung 2.17: Bluetooth SDP ServiceSearchPattern

Die Bluetooth-Spezifikation definiert folgende verschiedene Typen von Werten, die den Attributen in den Service Records zugewiesen werden können: `Nil`, `Unsigned Integer`, `Signed Twos-Complement Integer`, `UUID`, `String`, `Boolean`, `DataElementSequence`, `DataElementAlternative` und `URL`. `DataElementSequence` und `DataElementAlternative` sind Listen von Elementen der o.g. Typen, der Unterschied zwischen den beiden Typen liegt in der Bewertung der Elemente der Liste: Während die Elemente von `DataElementSequence` im „AND“-Sinne gewertet werden, erfolgt eine Bewertung der Elemente einer `DataElementAlternative` im „OR“-Sinne. Bei einer Suchanfrage erfolgt im SDP Server immer ein exakter Vergleich der Attribute der Anfrage mit Attributen der Service Records im SDP Server. Relative Angaben, die serverseitig ausgewertet werden, können nicht gemacht werden. So gibt es zwar ein vordefiniertes Attribut `ServiceAvailability`, mit dem die verfügbare Kapazität einer durch einen Service Record beschriebenen Dienstinstanz bezüglich der Annahme weiterer Clients angegeben werden kann. Es macht jedoch keinen Sinn, einen Wert für dieses Attribut als Matching-Kriterium in einem ServiceSearchPattern zu verwenden, da in diesem Fall nur Service Records übermittelt würden, deren aktuelle Auslastung *exakt* dem angegebenen Wert entspicht. Die Auswertung relativer Angaben kann hier nur auf der Seite des SDP Clients erfolgen, nachdem alle verfügbaren Service Records an den SDP Clients übermittelt wurden, z.B. durch Angabe ausschließlich einer bestimmten ServiceClassID. Auf einem SDP Client können dann beispielsweise in der weiteren Verarbeitung nur die Service Records berücksichtigung finden, deren `ServiceAvailability` mindestens 20% beträgt.

UDDI Die Dienstsuche mittels *Universal Description, Discovery and Integration (UDDI)* [14, 17] ist eine der Säulen der Web Service Architektur und beschreibt die Verwendung des UDDI Verzeichnisdienstes zum Publizieren von Informationen auf der einen Seite, und zum Auffinden und Abrufen von Informationen über Web Services und deren Anbieter auf der anderen Seite. Das UDDI zugrundeliegende Informationsmodell definiert im wesentlichen Attribute, die dazu verwendet werden können, einzelne Dienstanbieter, ihre Beziehungen zu anderen Dienstanbietern, ihre angebotenen Dienste und die Zugriffsmöglichkeiten auf diese Dienste zu beschreiben. Das Informationsmodell wird in XML repräsentiert.

Eine der Kern-Datenstrukturen von UDDI ist das `tModel`. Mit `tModels` kann die Kom-

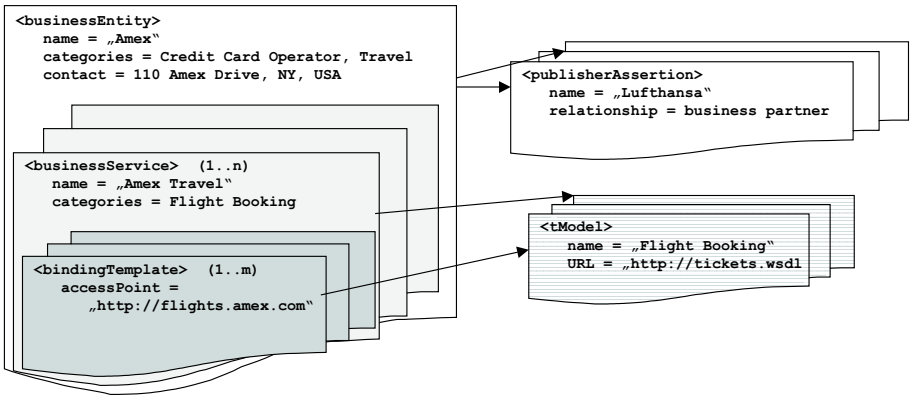

Abbildung 2.18: Struktur der UDDI Einträge

patibilität zu einer Spezifikation, einem Konzept bzw. einem gemeinsamen Entwurfsmuster ausgedrückt werden [15]. Wenn eine bestimmte Spezifikation wie z.B. eine in WSDL [55] formulierte Interface Definition eines Dienstes oder die WSDL Spezifikation als zweite Säule der Web Service Architektur selbst als `tModel` bei einer UDDI Registry registriert wird, erhält sie von dieser eine UUID [12] zugewiesen. Diese UUID kann im Folgenden bei der Beschreibung von Dienstinstanzen dazu verwendet werden, die Kompatibilität der Dienstinstanz mit der durch die UUID referenzierten Spezifikation anzuzeigen. Ein Dienst wird in UDDI durch die Datenstruktur `businessService` repräsentiert, und die Details, wie und wo auf den Dienst zugegriffen werden kann, werden durch ein oder mehrere ineinander verschachtelte `bindingTemplates` angegeben. Ein `bindingTemplate` spezifiziert mittels des Elements `accessPoint` den Zugangspunkt für einen Dienst, sowie mit einer Liste von `tModels` eine Beschreibung des Dienstes. Eine Übersicht über diese Abhängigkeiten ist in Abbildung 2.18 dargestellt.

Um einen Dienst bei einer UDDI Registry anzumelden verwendet ein Dienstanbieter die *Publishing API* der UDDI Registry, die entsprechende Funktionen zum Hinterlegen der `businessServices`, `tModels` etc. als Web Service bereitstellt. Für die Client-Seite bietet die *Inquiry API* der UDDI Registry nach [16] drei verschiedene Arten von Anfragemustern an, von denen eigentlich nur die erste einer Dienstsuche im eigentlichen Sinn entspricht:

1. Browse Pattern
2. Drill-Down Pattern
3. Invocation Pattern

Bei einer Dienstsuche mit dem Browse Pattern können Anfragen gestellt werden, wo davon auszugehen ist, daß das Ergebnis nicht eindeutig ist (z.B. „Suche alle Provider, die im Reisemarkt tätig sind“). Antwort der UDDI Registry ist eine Liste mit einer Zusammenfassung von Informationen der gefundenen Einträge. Die zweite Art von Anfrage (drill-down) kann dann dazu verwendet werden, den kompletten Informationsbestand zu einer der per browse pattern gefundenen Dienstbeschreibungen abzufragen Das letztgenannte Suchmuster (invocation pattern) dient der Integritätsüberwachung und kann

z.B. dazu verwendet werden, aktualisierte Informationen über Dienstzugangspunkte zu erhalten, ist aber für die eigentliche Dienstsuche eher uninteressant da zu speziell.

```
<find_service generic="2.0" xmlns="urn:uddi-org:api_v2" businessKey="">
  <findQualifiers>
    <findQualifier>orAllKeys</findQualifier>
    <findQualifier>soundex</findQualifier>
  </findQualifiers>
  <categoryBag>
    <keyedReference
        tModelKey="uuid:a035a07c-f362-44dd-8f95-e2b134bf43b4"
        keyName="RSS" keyValue="movie" />
    <keyedReference
        tModelKey="uuid:c0b9fe13-179f-413d-8a5b-5004db8e5bb2"
        keyName="Motion Picture and Video Industries"
        keyValue="5121" />
  </categoryBag>
  <tModelBag>
    <tModelKey>uuid:8a056b70-bfe8-4fac-90cd-820c26dc2e48</tModelKey>
  </tModelBag>
</find_service>
```

Abbildung 2.19: Beispiel einer UDDI Dienstsuche

Die Dienstsuche nach dem browsing pattern erfolgt mit einem der fünf Anfragekategorien `find_binding`, `find_business`, `find_relatedBusiness`, `find_service` und `find_tModel`. Abbildung 2.19 zeigt beispielhaft, wie `find_service` verwendet wird, um News-Dienste gemäß der Spezifikation *RDF Site Summary (RSS) 0.9x - Feeds* [36] zu suchen, die entweder in der allgemeinen Beschreibung das Schlüsselwort „Movie“ haben oder nach dem NAICS Klassifikationsschema [7] als Filmbeschreibung klassifiziert wurden [91].

Auch in der UDDI Registry erfolgt der Vergleich der Parameter einer Anfrage mit den in der Registry bekannten Dienstbeschreibungen auf der Basis von exaktem Matching. Lediglich für String-basierte Parameterwerte kann durch Angabe eines zusätzlichen `findQualifiers` mit dem Namen `soundex` das lexikalische Matching „aufgeweicht“ und in eine Ähnlichkeitssuche umgewandelt werden. Hierbei werden auch Strings als Treffer erkannt, die sich in wenigen Buchstaben vom Suchkriterium unterscheiden. Ansonsten wird eine stringbasierte Dienstbeschreibung nur als Treffer erkannt, wenn sie dem Parameterwert aus einer Anfrage entweder vollständig oder zumindest dem Anfang des entsprechenden Parameterwertes entspricht. Ein vergleichbarer Mechanismus ist für Zahlenwerte (z.B. ≤ 5) oder URLs (z.B. alle Dienste vom Server somesite.com) etc. bisher leider nicht vorgesehen.

Ein Vergleich der hier vorgestellten allgemeinen Dienstvermittlungsprotokolle verdeutlicht die Gemeinsamkeiten und Unterschiede der Verfahren. Während UPnP und SLP sowohl mit

als auch ohne zentralen Verzeichnisdienst arbeiten, benötigen Jini und UDDI zwingend einen solchen, wohingegen Bluetooth SDP einen solchen gar nicht kennt. In mobilen Dienstszenarien mit den stark schwankenden und unzuverlässigen Netzwerkverbindungen ist eine Abhängigkeit zu einem zentralen Verzeichnisdienst nachteilig, was ohne weitere Maßnahmen (Caching etc.) gegen eine Verwendung dieser Protokolle auf dem mobilen Endgerät spricht.

Andererseits fallen die Vorteile eines zentralen Verzeichnisdienstes wie z.B. eine gleichzeitige Berücksichtigung und Bewertung möglichst vieler Dienstangebote bereits während des Matchings weg, wenn das Dienstvermittlungsprotokoll wie im Fall von Bluetooth SDP ausschließlich eine bilaterale Dienstsuche zwischen Komponenten des Ad-hoc Netzwerks vorsieht. Hinzu kommt bei Bluetooth SDP, daß das Protokoll direkt auf dem L2CAP Layer des Bluetooth Protokollstacks aufsetzt und damit spezifisch für Bluetooth-Verbindungen geeignet ist. Wünschenswert wäre jedoch die Möglichkeit, mit Bluetooth SDP auch Dienste lokalisieren zu können, die sich u.U. im Netzwerk hinter dem bilateralen Bluetooth-Knoten befinden, bzw. dort auch per zentralem Verzeichnisdienst (z.B. LDAP) vermittelt werden. Ein solches Verfahren wurde unter dem Namen *SDPoverIP* zum Patent angemeldet [149], spielt aber für diese Arbeit keine wesentliche Rolle.

Gegen Jini spricht in mobilen Endgeräten weiterhin die starke Bindung des Protokolls an Java als Ausführungsplattform. Es gibt zwar auch *Java Virtual Machines (JVM)*, die auf kleinen Endgeräten mit beschränkter Leistungsfähigkeit (daher oft unter dem Begiff *ressource limited devices* zusammengefaßt) laufen. Darunter fallen z.B. die KVM für Mobiltelefone und Pager oder die CVM für Set-Top-Boxen, beide aus der *Java2 Micro Edition (J2ME)* [132]. Aber Jini und insbesondere der darin verwendete RPC-Mechanismus namens *Remote Method Invocation (RMI)* stellen Anforderungen an die JVM, die über die Leistungsfähigkeiten der JVMs auf dem mobilen Endgerät hinausgehen [150, 42, 94].

Ein Nachteil aller hier vorgestellten allgemeinen Verfahren ist die Tatsache, daß mehr oder weniger ausschließlich exaktes Matching unterstützt wird. So ist es mit allen Protokollen möglich, z.B. einen Drucker zu finden, der Farbausdrucke anbietet. Aber einen Drucker in der unmittelbaren örtlichen Umgebung mit der kürzesten Warteschlange zu suchen, ist mit diesen Protokollen schwer möglich, wenn nicht sogar gänzlich unmöglich. Hier können das im Rahmen dieser Arbeit vorgestellte Modell und die darauf aufbauende Sprache weiterhelfen, wie später gezeigt wird.

Andere Arbeiten, die sich mit der Analyse der Dienstvermittlungsprotokolle beschäftigen, sind z.B. von Bettstetter und Renner [39] oder Chakraborty und Chen [49]. Von Samulowitz und Michahelles werden in [135, 115] einige der hier vorgestellten Dienstvermittlungsprotokolle gegenübergestellt und insbesondere auf ihre Eignung aus der Perspektive des Dienstsuchenden überprüft.

Beispiele konkreter Service Frameworks

Service Frameworks sind für die Überwachung und Steuerung der unterschiedlichen Stationen des Lebenscyclus eines Dienstes zuständig, ähnlich den Aufgaben einer Agenten-Plattform [65, 157]. Darüber hinaus hat ein Service Framework in der Regel noch weitreichendere Aufga-

ben wie die Bereitstellung von Basisdiensten, persistentem Speicher, Sicherheitsmechanismen, Einrichtungen zur Kommunikation der Dienste mit dem Framework und anderen Diensten etc.

Da ein Schwerpunkt dieser Arbeit die Nutzung von Diensten auf mobilen Endgeräten ist, wird im Folgenden eine Auswahl von Service Frameworks vorgestellt, die eben diese mobile Dienstnutzung auf die eine oder andere Art ermöglichen. Die dabei eingesetzten Verfahren und Architekturen unterscheiden sich zum Teil ganz erheblich und zeigen die Breite möglicher Lösungsansätze. Die hier vorgestellte Auswahl ist jedoch weder vollständig noch repräsentativ.

Centaurus Service Framework. Das Centaurus [92] Framework wurde an der Universität von Baltimore County, MD, USA, entwickelt. Dem System liegt eine konsequente Dienstorientierung zugrunde, d.h. alle Hardware- und Softwarekomponenten werden im System durch Dienste repräsentiert. Mobile Endgeräte haben lediglich die Funktion eines „Portals“ im Sinne eines „thin clients“ zum Framework, alle Dienste werden im verteilten, dezentral und hierarchich organisierten Backbone in sogenannten *Service Managern* ausgeführt. Architekturen dieser Art werden auch *Surrogate* Architekturen genannt, da die eigentliche Dienstausführung stellvertretend auf einem anderen Rechner (*surrogate host*) stattfindet. Der eigentliche Client bringt nur das User Interface zur Anzeige. Bei diesen Architekturen muß zur Ausführung eines Dienstes immer eine Online-Verbindung zwischen mobilem Endgerät und *surrogate host* bestehen, eine lokale Ausführung von Diensten ist in der Regel nicht möglich. Die Dienstsuche erfolgt in Centaurus ebenfalls im Backbone, basierend auf Dienstbeschreibungen, die in XML mit proprietärer *Document Type Definition (DTD)* namens *CCML* codiert sind. Als Kontextinformationen werden lediglich Positionsinformationen aus der Infrastruktur bereitgestellt und im *surrogate host* verarbeitet, weshalb Centaurus als *location-aware*, aber nicht als *context-aware* bezeichnet werden kann.

Ninja Service Framework. Ninja [75, 86] wurde an der Universität von Berkeley, USA, entworfen. Die zugrundeliegende Architektur besteht aus den folgenden vier Basiselementen:

1. *bases*, d.h. Dienste, die im Backend auf Clustern von Workstations laufen
2. *active proxies*, d.h. Intermediatoren, die typischerweise Transformations-Operationen ausführen
3. *units*, d.h Dienste, die Stellvertreter für die unterschiedlichen Arten von Geräten darstellen
4. *paths*, d.h. Kontrollstrukturen zur Verwaltung von typisierten Datenflüssen durch eine Kette von units, proxies und anderen Diensten

Das Design der Architektur ist primär auf die Bereitstellung und Verarbeitung von Internet-Diensten für mobile Endgeräte ausgelegt. Die Dienste an sich, und damit insbesondere auch das State-Management, laufen im Sinne einer Surrogate Architekur im Backend-Netzwerk ab. Zur Dienstvermittlung wird in Ninja ein proprietärer, XML-basierter *Service Discovery Service (SDS)* [59] eingesetzt, welcher in den *bases* und damit ausschließlich im Backend Verwendung findet. Ninja berücksichtigt im Rahmen interner Kommunikation über Sensoren erfaßte Statusinformationen des Netzwerks in der Umgebung des Benutzers und ist daher in begrenztem Umfang kontextadaptiv.

Moca Service Framework. Das Moca [35, 133] System wurde am IBM Watson Research Center, NY, USA, entwickelt und nutzt die Leistungsfähigkeiten moderner mobiler Endgeräte dahingehend, daß ein Teil der Dienste direkt auf den Endgeräten ausgeführt werden. Als zentrale Komponente auf dem Endgerät wird durch das Framework eine Service Registry bereitgestellt, welche ähnlich einem Jini Lookup Service (LUS) aufgebaut ist und insbesondere per Multicast Announcement Informationen über die verfügbaren Dienste erhält. Die so verbreiteten Dienstbeschreibungen beschränken sich allerdings lediglich auf den Namen eines Interfaces sowie den Dienstendpunkt (*port*) der zugehörigen Implementation, also die Signatur eines Dienstes. Eine Dienstsuche mit Attributen, die den Dienst eingehender charakterisieren oder gar die Semantik oder den Kontext beschreiben, ist nicht vorgesehen. Weil Dienste auf dem Endgerät selbst ausgeführt werden, müssen Komponenten des Frameworks auch Aufgaben des Life-Cycle-Managements, der Überwachung von Security Policies etc. wahrnehmen. Im Gegensatz zu Centaurus und Ninja ist Moca also keine „thin client" Architektur. In Moca finden Kontextinformationen keine besondere Berücksichtigung, weshalb dieses Service Framework zwar viel stärker das Endgerät bei der Dienstnutzung involviert, aber diese Dienstnutzung lediglich profilbasiert adaptiert.

Capeus Service Framework. Das Capeus [115, 136] Framework verfolgt einen Dokumenten-zentrischen Ansatz, bei dem ein datenzentrisches Protokoll zur Weiterleitung von Dienstanfragen und Anwendungsdaten insbesondere unter Verwendung kontextueller Bedingungen an entsprechende Serverkomponenten eingesetzt wird. Obwohl die prototypische Implementierung von Capeus bisher nur auf mobilen Endgeräten der Kategorie Notebook läuf, sieht das Konzept auch den Einsatz kleinerer, weniger leistungsstarker Endgeräte wie Mobiltelefone vor, was bei der Art der Dienstnutzung auch schlüssig ist: Zentrales Element des Frameworks sind sogenannte *Context Aware Packets (CAPs)*, die dazu dienen, gewünschte Services auf einem hohen Abstraktionsniveau zu beschreiben. Diese CAPs werden aufgrund einschränkender Kriterien (*Context Constraints* bzw. *Meta Constraints*) nach dem Store-And-Forward Prinzip zu den gewünschten Knoten im Netzwerk geroutet, wo der entsprechende Dienst Operationen basierend auf dem Inhalt des CAPs ausführt. Bei der Dienstausführung im Capeus System handelt es sich um eine rollenbasierte Reaktion auf eine im CAP beschriebene Bedingung, dargestellt durch sogenannte *Interaction Pattern*. Zur Suche nach geeigneten Diensten wird innerhalb des Systems auf das Service Location Protokoll (SLP, siehe Abschnitt 2.4.3) zurückgegriffen. Da die Verarbeitung und Weiterleitung der CAPs kontextgetrieben ist, handelt es sich beim Capeus Framework natürlich um einen in hohem Maße kontextadaptiven Ansatz.

2.5 Zusammenfassung

Dieses Kapitel hat Grundzüge des Themengebiets der Interoperabilität vermittelt, und damit eine Einordnung der neu vorgestellten kontextuellen Ebene der Service-Interoperabilität in die verschiedenen Ebenen der Interoperabilität ermöglicht. Hierfür wurde zunächst der Bezug zu den verteilten Systemen hergestellt. Dabei wurden die einzelnen Ebenen vorgestellt, auf denen Interoperabilität durch Anwendung verschiedener Techniken erreicht wird. Dazu zählt insbesondere der Bezug aller an einer Interaktion beteiligten Komponenten auf ein gemeinsa-

mes Verständnis. Die wichtigste Ebene ist dabei die Ebene der Service-Interoperabilität. Diese Ebene beschäftigt sich mit der Interoperabilität zwischen zwei oder mehr an einer Dienstinteraktion beteiligten Komponenten aus den beiden Perspektiven Kompatibilität und Ersetzbarkeit. Sie wird in der wissenschaftlichen Literatur differenziert nach den Modulen Signatur, Protokoll und Semantik betrachtet. Es konnte vermittelt werden, daß diese Dreiteilung der Service-Interoperabilität nicht ausreicht, da insbesondere der Kontext einer Dienstinteraktion zu wenig berücksichtigt wird. Das Ergebnis der Analyse der bisherigen Arbeiten auf dem Gebiet der Service Interoperabilität war daher die Einführung eines vierten Moduls, der *Kontextebene*. Die thematische Fokussierung der Untersuchung kontextueller Abhängigkeiten und Möglichkeiten von Diensten auf einer eigenen Ebene ermöglicht insbesondere die Trennung von Diensten, die zwar bezüglich der drei anderen Ebenen interoperabel sind, auf kontextueller Ebene jedoch nicht.

Zu Beginn der Einführung dieses vierten Moduls stand eine detaillierte Analyse des Kontextbegriffs anhand der zahlreichen Definitionen dieses Begriffs in der wissenschaftlichen Literatur. Daraus resultierte die Notwendigkeit, die in dieser Arbeit verwendete Interpretation der Terminologie zu spezifizieren (vgl. Definitionen 7 bis 9). Diese Definitionen integrieren insbesondere eine Erweiterung um den Begriff des *Aspekts* sowie eine Verfeinerung des Begriffs der *Relevanz*.

Bei der kontextuellen Service-Interoperabilität bestimmt die Menge relevanter externer und interner Einflußgrößen auf eine Dienstinteraktion das gemeinsame Verständnis, das zur Überprüfung der Interoperabilität von allen an einer Dienstinteraktion beteiligten Komponenten herangezogen wird. Eine Teilmenge des gemeinsamen Verständnisses sind Relevanzkriterien, welche die Relevanz der Einflußgrößen in Bezug auf die im Rahmen einer Dienstinteraktion zu erfüllende Aufgabenstellung beschreiben. Es wurde erläutert, warum Relevanz deutlich mehr als örtliche und zeitliche Nähe ist. In diesem Zusammenhang wurde die Erweiterung eines allgemeinen Dienstmodells um Aktoren zur Erfassung, Aufbereitung, Bereitstellung und Anwendung von Kontextinformationen, zur Entgegennahme und Bearbeitung von Relevanzbedingungen und deren Management vorgestellt.

Im letzen Teil dieses Kapitels wurden die Implikationen auf die kontextuelle Service-Interoperabilität vorgestellt, die sich aus der Etablierung des Ubiquitous Computings ergeben. Neben den Voraussetzungen der verschiedenen Arten der Mobilität und der Verwendung von Sensornetzen als Quelle von Kontextinformationen hat vor allem die adaptive Dienstnutzung in mobilen Endgeräten einen besonderen Stellenwert im Zusammenhang mit dieser Arbeit. Typische Merkmale von mobilen Endgeräten sind z.B. starke Beschränkungen der verfügbaren Ressourcen wie Speicher und Ein- bzw. Ausgabemöglichkeiten. Gerade diese mobilen Endgeräte, typische Komponenten in Ubiquitous Computing Systemen, profitieren enorm von der Auswertung kontextueller Zusammenhänge vor und während der Nutzung von Diensten im Sinne kontextadaptiver Dienstinteraktionen auf dem mobilen Endgerät. Andererseits sind sie selbst auch sehr gute Quellen für Kontextinformationen, da die enge Bindung von mobilem Endgerät und Nutzer zur Gewinnung von Informationen basierend auf verschiedenen Aspekten wie dem momentanen Aufenthaltsort oder diverser Nutzerpräferenzen verwendet wird.

Im folgenden Kapitel wird ein Modell zur Abbildung von kontextuellem Wissen vorgestellt,

welches als Grundlage der Spezifikation des gemeinsamen Verständnisses auf der Ebene der kontextuellen Service-Interoperabilität dient.

Kapitel 3

Eine Sprache zur Modellierung von kontextuellem Wissen

3.1 Wissensmodelle und Shared Understanding

Wie im Verlauf des letzten Kapitels deutlich wurde, sind alle Interoperabilitätsbetrachtungen auf den gemeinsamen Bezug aller an einer Dienstinteraktion beteiligten Komponenten auf ein gemeinsames Verständnis zurückzuführen. Dieses gemeinsame Verständnis, *shared understanding* oder *shared knowledge* genannt, repräsentiert das Wissen über die Umstände, unter denen die Interoperabilität von zwei oder mehr Komponenten gegeben ist. Während der Mensch auch mit relativ informellen Beschreibungen dieser Umstände noch ganz gut klar kommt, ist der Computer auf eine formelle und damit ihn auswertbare Form der Beschreibung angewiesen.

Terminologische Wissensrepräsentationssprachen können dazu verwendet werden, das taxonomische Wissen eines Anwendungsbereichs in einer strukturierten Weise darzustellen. Anwendung für terminologische Repräsentationssysteme finden sich zum Beispiel im Bereich der Sprachverarbeitung, beim automatischen Planen und bei der Lösung von Konfigurationsaufgaben. Interessant sind terminologische Sprachen vor allem deshalb, weil sie eine klare logische Formalisierung besitzen.

Es gibt in der Forschung viele verschiedene Ansätze, sich der Problematik um die Repräsentation von Wissen und deren automatische Auswertung durch den Computer zu nähern. Insbesondere der Forschungsbereich der künstlichen Intelligenz (KI) hat zu zahlreichen Lösungsvorschlägen in diesem Themenfeld beigetragen. Dies hat zu einer Vielzahl von Wissensmodellen und Wissensrepräsentationsformen geführt, die unter zum Teil sehr unterschiedlichen Gesichtspunkten entworfen wurden und dementsprechend für die eine oder andere Aufgabenstellung mehr oder weniger geeignet sind. Eine Auswahl von unterschiedlichen Wissensmodellen sind:

- Taxonomien, Identifikationssysteme
- Thesauri
- TopicMaps
- Semantische Netze
- Entity-Relationship-Modell (ER-Modell)
- ...
- Ontologien

Obwohl diese Liste nicht vollständig ist, unterscheiden sich die einzelnen Modelle bereits so deutlich in ihren Eigenschaften, daß im folgenden Abschnitt nur kurz auf die wichtigsten Unterscheidungsmerkmale inspiriert durch [111] vor dem Hintergrund einer Eignung zur Spezifikation und der automatischen Auswertbarkeit von kontextuellem Wissen eingegangen wird.

3.1.1 Wissensrepräsentationsformen

Eine *Taxonomie* ist eine Menge von hierarchisch geordneten Kategoriebezeichnungen, die vornehmlich zur Klassifikation großer Datenbestände verwendet werden. Sehr eng damit verwandt sind die *Identifikationssysteme*, die keine Kategorien sondern individuelle Identifikatoren beinhalten. Beispiele für Taxonomien sind Branchenkennungen wie NAICS [7], Produkt- und Dienstleistungskennzahlen wie UNSPSC [10] oder geografische Regionenbezeichnungen wie die nach ISO 3166. Beispiele für Identifikationssysteme sind eindeutige Bezeichner nach Dun & Bradstreet D-U-N-S [3] oder auch Telefonnummern.

Während Taxonomien und Identifikationssystem mehr oder weniger dazu geeignet sind, das Wissen über Primitive (Kategorie- oder Identitätsbezeichnungen) festzuhalten, kann mit Hilfe von ursprünglich in der Bibliographie entwickelten *Thesauri* auch in eingeschränktem Umfang das Wissen über Beziehungen zwischen Primitiven ausgedrückt werden. Eine Menge von vorgegebenen Relationen („ähnlich“, „Unterbegriff“, „Synonym“, ...) erlaubt die Anordnung der Primitive als gerichteter Graph, es können jedoch keine selbstdefinierten Relationen hinzugefügt und verwendet werden.

TopicMaps [40] bestehen aus abstrakte Dinge beschreibende Knoten (Topics) einschließlich Beschreibungen ihrer Gültigkeitsbereiche (Scopes), sowie einer erweiterbaren Menge von Relationen und zugeordneten Dokumenten (Occurences). Sie werden vorzugsweise zur Darstellung des Inhalts der Wissensbasis mit Hilfe verschiedener Sichten in für den Menschen optimierter Form und damit für die Navigation eines Menschen durch den Inhalt verwendet. Zur Zeit fehlt den noch recht jungen TopicMaps ein wohldefiniertes Datenmodell. An verschiedenen Abfragesprachen wird zur Zeit gearbeitet.

Einen ähnlichen Ansatz verfolgen auch die *Semantischen Netze* [21] (die übrigens nicht mit dem Semantic Web nach [37] zu verwechseln sind), aus denen später die Description Logics (DL) [160, 87] entwickelt wurden. Auch sie können als gerichteter, kantenbeschrifteter Graph von Knoten (Objekte) und Verzweigungen (Relationen) betrachtet werden. Semantische Netze unterstützen darüber hinaus das Konzept der Vererbung von Relationen.

Die *ER-Modelle* [53] haben ihren Hauptverwendungszweck in Datenbanken, wo auch zahlreiche Erweiterungen des Grundmodells wie das *Relationenmodell* [167] und das *Netzwerkmodell* [167] oder auch die *temporalen ER-Modelle* [74] entworfen wurden. Wohlunterscheidbare Dinge der realen Welt werden im ER-Modell als *Entität* bezeichnet, und gleichartige Entitäten werden in *Entitätsmengen* (Tabellen) gruppiert. Entitätsmengen werden über *Attribute* (Spaltennamen) Werte aus einem definierten Wertebereich zugeordnet. Unterschiedliche Entitätsmengen stehen im ER-Modell über *Relationen* in Beziehung, die zusätzlich zu Verknüpfungsinformationen der in Beziehung stehenden Entitätsmengen auch eigene Attribute haben können. Wichtig im Zusammenhang mit dieser Arbeit ist, daß der Name einer Entitätsmenge oder Relation, sowie deren Attribute als semi-zeitinvariant, der Inhalt der Entitätsmenge oder Relationsmenge (Zeilen einer DB-Tabelle) jedoch als zeitveränderlich anzusehen ist. Das ER-Modell kennt das Konzept der Vererbung (spezielle *is-a*-Relation) und Kardinalitätsangaben (1:1, 1:n, m:n). Als Anfragesprache für Datenbanksysteme, die das ER-Modell implementieren, hat sich weitgehend SQL durchgesetzt.

Ein Wissensmodell bietet sich jedoch aufgrund seiner speziellen Eigenschaften im Rahmen dieser Arbeit zur Modellierung von kontextuellem Wissen besonders an: die *Ontologien*. Daher wird im Folgenden ausführlicher auf diese Wissensrepräsentationsform eingegangen.

Der Begriff der „Ontologie“ entstammt der Philosophie, wo er eine lange Historie als Oberbegriff für die Bedeutung der Existenz hat. Eine der bekanntesten Definitionen der Bedeutung von Ontologien in der Informatik ist die von Gruber [76] aus dem Jahr 1993:

> *Eine Ontologie ist eine explizite Spezifikation einer gemeinsamen Konzeptualisierung. (Eine abstrakte, vereinfachte Sicht auf die Welt).*

Swartout et al. [158] weisen mit ihrer Definition bereits auf die Verwendung von Ontologien als Basis von Wissensbasen hin:

> *Eine Ontologie ist eine hierarchisch strukturierte Menge von Ausdrücken zur Beschreibung einer Domäne, die als Grundgerüst für eine Wissensbasis dienen kann.*

Uschold und Grüninger [163] legen hingegen Wert auf die Eigenschaften von Ontologien zur Vermeidung von Fehlinterpretationen:

> *Eine Ontologie kann verschiedene Formen haben, aber notwendigerweise beinhaltet sie ein Vokabular von Ausdrücken und eine Spezifikation ihrer Bedeutung. Dies beinhaltet ihre Definition und Hinweise, wie die Konzepte untereinander in Beziehung stehen, was zusammen der Domäne eine Struktur auferlegt und die möglichen Interpretationen der Ausdrücke begrenzt.*

Uschold und Grüninger nannten dabei drei verschiedene Haupteinsatzgebiete von Ontologien:

- Kommunikation
- Automatisches Schließen (Reasoning)
- Repräsentation und Wiederverwendung von Wissen

Mit dem ersten Aufgabenfeld (Kommunikation) wird bereits explizit die Kompatibilität als Perspektive der Interoperabilität (siehe Abschnitt 2.2.1) angesprochen. Das in einer Ontologie festgelegte Vokabular und das Netzwerk der Relationen (Beziehungen) haben einen stark normativen Charakter für die bei jeder Dienstinteraktion ablaufende Kommunikation. Mit anderen Worten definiert eine Ontologie einen Katalog von Konzepten und ihre Beziehungen untereinander, weshalb eine Ontologie in weniger wissenschaftlichen Publikationen auch gelegentlich als „Vokabular“ bezeichnet wird. Eine Ontologie wird zum gemeinsamen Verständnis (*shared understanding*) durch die Tatsache, daß die Bedeutung des Vokabulars von allen Komponenten identisch interpretiert und angewandt wird. Ist das *shared understanding* auf der Basis einer oder mehrerer Ontologien definiert, dann können Mehrdeutigkeiten trotz Integration unterschiedlicher Anwender- oder Komponentenperspektiven weitgehend vermieden werden. Darüber hinaus wird eine toolgestützte Überprüfung auf Konsistenz ermöglicht.

Diese Konsistenzprüfung ist nach dem automatischen Schließen eine der Hauptaufgaben eines *Reasoners*. Reasoning bezeichnet einen Prozeß, bei dem aus *vorhandenem* Wissen, Annahmen oder Vermutungen *neues* Wissen, Annahmen oder Vermutungen gewonnen werden. Ein Reasoner macht also Wissen, das zu einem Zeitpunkt implizit in der Wissensbasis steckt, durch Auswertung von Regeln (*Axiome*) explizit. Die Art der Auswertung von Regeln (z.B. in einem neuronalen Netz) hängt stark von der zugrundeliegenden Logik ab und bestimmt dabei die Güte eines Reasoners. Wenn das Wissen aus symbolisch codierten Axiomen abgeleitet wird, spricht man von einer *Inference Engine* als Spezialisierung des Reasoners. Der Begriff der *Inferenz* beschreibt dabei das aufbereitete Wissen, das aufgrund von logischen Schlussfolgerungen gewonnen wurde. Inferenzmaschinen generieren durch automatische Ableitung aus symbolisch codierten Axiomen neues Wissen mittels eines formalen Logikkalküls. Der Vorgang wird *inferencing* genannt.

Wichtiges Mittel zur Gewinnung von Wissen sind hierbei die *deduktiven* Inferenzen, die im Sinne „harter“ logischer Schlußfolgerungen Aussagen sind, die dann der Wissensbasis hinzugefügt werden. Diesen stehen die *abduktiven* und *induktiven* Inferenzen gegenüber, die durch „weiche“ Schlußfolgerungen als Wahrscheinlichkeiten oder Indizien ein Lernen der Wissensbasis ermöglichen. Zur Erweiterung kontextuellen Wissens mit Hilfe einer Inferenzmaschine bieten sich allerdings vorwiegend deduktive Inferenzen an, da es zu viele verschiedene Fälle des Kontextes gibt, um in absehbarer Zeit gute Lerneffekte zu erziehlen. Deshalb wird im weiteren Verlauf dieser Arbeit auf abduktive und induktive Inferenzen verzichtet.

Die Modellierung von Wissen in Ontologien erfolgt auf zwei Ebenen: Einerseits auf der Ebene von *Konzepten* und andererseits auf der Ebene der *Instanzen*. Letztere werden auch *Fakten (facts)* genannt oder *Artefakte (artifacts)*, was insbesondere auf den künstlichen Charakter des Wissens hinweisen soll. Ontologische Konzepte sind in etwa vergleichbar mit Klassen in einer objektorientierten Programmiersprache oder Entitätsmengen im ER-Modell (siehe auch Abbildung 3.1). Alle drei dienen u.a. der Gruppierung von gleichartigen Entitäten, alle unterstützen das Konstrukt der Ableitung und eignen sich somit zur Erstellung einer Konzept- bzw. Klassen-Hierarchie, und alle dienen der Typisierung von Instanzen. Hauptunterschied ist,

daß ontologische Konzepte nicht objektbasiert, sondern eigenschaftenbasiert (*property centric*) sind, d.h. während Klassen durch ihre Attribute und Methoden definiert sind, werden Konzepte durch ihren *Argumentrahmen* (Definitionsbereich und Wertebereich der Eigenschaften) definiert. Dies ist insbesondere dann nützlich, wenn auf der Ebene der ontologischen Instanzen nur unvollständiges Wissen vorliegt, und daraus trotzdem auf das zugrundeliegende Konzept geschlossen werden soll. Das heißt zum Beispiel, daß ein Reasoner in der Lage ist, aus dem Faktum „hat 4 Reifen“ zu schließen, daß es sich um das Konzept „Auto“ oder um eine „große Spielplatzschaukel“ handelt. Ontologische Konzepte und Instanzen können Eigenschaften von mehreren Oberkonzepten gleichzeitig erben (Mehrfachvererbung), was nichteinmal für Klassen und Instanzen in jeder objektorientierten Programmiersprache gilt (z.B. Java).

Ontologie	Datenbank	Programmiersprache
Concept	Table	Class
Instance	Data Record	Instance
Relation	Table w/ FK	Method
Attribute	Attribute	Attribute
Axiom	?	Algorithmus?

Abbildung 3.1: Vergleich der Elemente von Ontologien

Der „Erfolg“ einer Ontologie bemißt sich nach der Zahl ihres Gebrauchs. Denn sinnvolle Entscheidungen und große Interoperabilität sind erst dann gegeben, wenn die Ontologie ein gewisses Maß an „Autorität“ hat, die durch weite Verbreitung und Nutzung erreicht wird. Für diese verbreitete Nutzung ist auch ein gewisses Maß an Vollständigkeit nötig, damit auch alle Entscheidungen im relevanten Szenario getroffenen werden können. Nützlich ist hier die Eigenschaft der Ontologien, immer nur domänenspezifisch zu sein und damit nur einen Ausschnitt der realen Welt zu modellieren. Selbst eine einzelne domänenspezifische Ontologie ist in der Regel nicht komplett und abschließend von einem einzelnen Autor geschrieben, wenn sie auch nur annähernd komplett sein soll. Dazu ist das darzustellende Wissen, sogar über relativ kleine Ausschnitte der Realität, viel zu komplex. Stattdessen muß man davon ausgehen, daß über ein Gebiet mehrere Autoren schreiben, deren Ansätze sich im besten Fall genau ergänzen, aber nicht widersprechen. Um dennoch ein möglichst vollständiges Wissen zu berücksichtigen, müssen die Ontologie-Fragmente unterschiedlicher Autoren, die zu unterschiedlichen Zeiten in unterschiedlichen Dokumenten festgelegt sind, kombiniert werden (verteilte Komposition). Die Zusammenführung ist eine der Aufgaben eines Reasoners, wobei die Überprüfung der Konsistenz der Daten eine offensichtliche Teilaufgabe darstellt. Die Möglichkeit der Erweiterung und Kombination mit vorhandenen Fragmenten einer Ontologie durch Einbindung neuer oder Entfernen alter Ontologie-Fragmente zur Laufzeit stellt genau das große Maß an Flexibilität dar, welches in verteilten Systemen notwendig ist. Dies ist im übrigen auch ein Vorteil gegenüber dem ER-Modell, welches wie erwähnt auf der Strukturebene semi-zeitinvariant[1] ist. Außerdem ist die Anzahl der modellierten Konzepte in Ontologien

[1] Änderungen an der Struktur einer Datenbank (z.B. Hinzufügen einer Spalte in einer Tabelle) werden viel seltener vorgenommen als Änderungen am Datenbestand selbst (z.B. Anlegen eines neuen Datensatzes)

in der Regel sehr viel größer als in ER-Modellen, was aber auch ein performantes System zur Verwaltung der Konzepte erfordert.

Sowohl auf der Ebene der Konzepte als auch auf der Ebene der Instanzen kann in Ontologien Wissen über *Relationen* ausgedrückt werden, d.h. wie zwei Konzepte oder zwei Instanzen miteinander in Beziehung stehen. Ein Reasoning ist bei Ontologien, im Gegensatz zu z.B. den Semantischen Netzen, auf beiden Ebenen möglich. Eigene Relationen *zwischen* diesen Ebenen über die `instanceOf`-Beziehung der übrigen Wissensrepräsentationsformen hinaus sind in Ontologien ebenfalls modellierbar. Sie ermöglichen die zunächst gewöhnungsbedürftige Konstellation, bei der dasselbe Objekt in einer Aussage als Konzept und in einer anderen Aussage als Instanz behandelt wird, was auch als *Konzept-Instanz-Dualismus* bekannt ist.

Im weiteren Verlauf dieser Arbeit wird an den Stellen, bei der die besonderen Eigenschaften von Ontologien Verwendung finden, auf diesen Tatbestand gesondert hingewiesen.

3.1.2 Ontologiesprachen

Wie bereits erwähnt, hängt die Leistungsfähigkeit eines Reasoners von der zugrundeliegenden Logik [41, 143], beziehungsweise von den verwendeten Elementen einer Logik, ab. Ohne allzusehr auf die Details einzugehen, stellt die folgende Auflistung die Mächtigkeit bezüglich der Ausdrucksstärke einer Auswahl von Logiken in aufsteigender Reihenfolge dar:

- Aussagenlogik (AL)
- Description Logic (DL)
- Frame-Logic (FL)
- Prädikatenlogik I, II (PL1/FOL, PL2/HOL)

Zu der Ausdrucksstärke sind der Aufwand der Entscheidbarkeit einer Aussage in der entsprechenden Logik und in dem Zusammenhang die Anforderungen an einen Reasoner umgekehrt proportional zueinander. Allerdings können unter gewissen einschränkenden Annahmen die Anforderungen an einen Reasoner deutlich reduziert werden. So führt beispielsweise Borgida in [43] an, daß Aussagen in Prädikatenlogik erster Ordnung, die zwei oder drei Individuenvariablen haben, die gleiche Ausdruckskraft haben wie Aussagen in Description Logic (womit die DL eine echte Teilmenge der PL ist). Und Frame-Logic hat eine Syntax der Mächtigkeit von Prädikatenlogik II, während ihre Semantik die Mächtigkeit der Prädikatenlogik I hat, womit die Frame-Logic bezüglich der Ausdrucksstärke der Description Logic überlegen ist [5]. Weitere Betrachtungen zum Vergleich der Frame-Logic mit der Description Logic sind z.B. bei Balaban [32] zu finden.

Interessant sind terminologische Wissensrepräsentationsprachen auch deshalb, weil sie eine klare logische Formalisierung besitzen. Zum einen zeigt diese Formalisierung, daß diese Sprachen als Teilklassen der Prädikatenlogik erster Stufe aufgefaßt werden können. Aus algorithmischer Sicht ist hier wichtig, daß viele der terminologischen Sprachen entscheidbare Teilklassen der Prädikatenlogik liefern, die nicht mit bekannten entscheidbaren Fragmenten zusammenfallen. Hier konnten z.B. von Horrocks et al. [87] Entscheidungsalgorithmen für Description

Logics entwickelt werden, die auf einem modifizierten Tableau-Verfahren beruhen. Diese Verfahren liefern nach Baader et al. [30] im allgemeinen Algorithmen, die optimal bezüglich der Worst-Case-Komplexität des betrachteten Inferenzproblems sind.

Zum anderen ermöglicht diese Formalisierung die Implementierung von Wissensrepräsentationssystemen, deren Verhalten vorhersagbar und unabhängig von einer speziellen Implementierung ist. Dadurch wird sowohl die Transparenz des Systems für den Benutzer erhöht als auch ein Vergleich verschiedener Systeme ermöglicht. Einen wichtigen Beitrag dazu liefert u.a. der Anspruch an Logik-Systeme, daß die Auswertung des Regelsatzes unabhängig von der Reihenfolge der Auswertung der einzelnen Regeln sein muß.

Im folgenden werden einige Ontologiesprachen vorgestellt, die auf unterschiedlichen Arten von Logiken basieren und im weiteren Verlauf dieser Arbeit eine Rolle spielen. Da ein Teil dieser Sprachen auf RDF und RDF Schema aufbauen, erfolgt der Vollständigkeit und des besseren Verständnisses wegen zunächst jedoch eine kurze Einführung in RDF und RDF Schema.

RDF und RDF Schema

Eine vor allem durch die Aktivitäten im „Semantic Web“ [37] relativ hohe Verbreitung haben Ontologiesprachen erlangt, die auf dem *Ressource Description Framework (RDF)* [100, 66] basieren. Entwurfsziele der seit 1997 vom W3C für die Annotation von Dokumenten mit Meta-Informationen konzipierten Sprache waren u.a. eine kompakte Syntax, effiziente Verarbeitung, möglichst große Ausdruckskraft, Unterstützung von großen Knowledge Bases, sowie syntaktische und semantische Interoperabilität. Leider sind diese Entwurfsziele nur zum Teil erreicht worden.

Die Grundlage von RDF ist sein Datenmodell, das unabhängig von der konkreten Syntax beschreibt, wie ein RDF Ausdruck aufgebaut ist. Durch die Verwendung eines klar definierten Datenmodells ist es leicht möglich, unterschiedliche Serialisierungen für RDF anzubieten. Besonders interessant ist im Rahmen dieser Arbeit die Serialisierung in XML als generelles Datenaustauschformat. Das Datenmodell von RDF ist an die grundlegende Form von Aussagen angelehnt: „*Subjekt S* hat die Eigenschaft *Prädikat P* mit Wert *Objekt O*“ (SPO-Modell). Aufgrund dieser Dreiteilung werden Aussagen dann auch *Tripel* genannt. Sie werden auf die drei Grundtypen des Datenmodells abgebildet:

- Ressourcen: Alle Entitäten, zu denen Aussagen gemacht werden, werden in RDF als Ressourcen bezeichnet. Alle übrigen Datentypen leiten sich von Ressourcen ab. Ressourcen müssen dabei immer durch eine URI eindeutig identifizierbar sein.
- Eigenschaften: Eine Eigenschaft ist eine bestimmte Charakteristik zur Beschreibung einer Ressource, z.B. „Autor“ oder „Titel“. Jede Eigenschaft hat eine spezifische Bedeutung, deren Definition jedoch nicht Teil des Grundmodells von RDF ist.
- Aussagen: Aussagen bestehen aus der Kombination einer Ressource, einer Eigenschaft und eines Wertes für die Eigenschaft, worin sich wiederum der Tripel-Charakter einer Aussage zeigt. Der Wert der Eigenschaft kann wieder eine Ressource oder auch ein Literal sein. Die drei Teile werden, wie oben schon erläutert, als Subjekt, Prädikat und

Objekt der Aussage bezeichnet. Da auch Aussagen Ressourcen sind, ist es möglich, Aussagen über Aussagen zu machen, was als *Reifikation* bezeichnet wird. Reifikation erweitert entscheidend die Ausdrucksmacht der Sprache, man verlässt damit aber die Prädikatenlogik erster Ordnung. Dies wurde jedoch bewußt in Kauf genommen, da nach Überzeugung der Autoren von RDF auf diese Ausdrucksmöglichkeit im Semantic Web nicht verzichtet werden kann, speziell auch mit Hinsicht auf Relativierung und Vertrauenswürdigkeit von Aussagen.

Das SPO-Modell stellt einen gerichteten, kantenbeschrifteten Graphen dar. Knoten repräsentieren dabei Ressourcen bzw. Entitäten, die Kanten sind binäre Relationen (Eigenschaften der Quell-Ressource mit Wert Ziel-Ressource oder Literal) zwischen Ressourcen/Entitäten, analog zum ER-Modell. Die graphische Darstellung des SPO-Modells hebt noch besonders die *Literale* hervor, also atomare Werte im Gegensatz zu „echten“ Objekten. Solche Literale sind auch Ressourcen, sie werden speziell als *Wert*, als Ziel-Ressource, in Aussagen verwendet. Eine solche Unterscheidung wird im ER-Modell nicht vorgenommen. Andererseits ist es im ER-Modell möglich, zu einer Entität abstrakt ihre Attribute anzugeben, d.h. ohne ihnen einen Wert zuzuordnen. Dies ist in RDF nicht möglich, da im SPO-Modell von RDF auch diese Attribute als Ressourcen über eine Eigenschaft/Relation (z.B. „hasAttribute“) definiert sein müssten, oder bei der Anwendung auf konkrete Instanzen müssten die Attribute selbst Eigenschaften der Entität mit konkret zugewiesenen Werten sein.

Der Nachteil von RDF ist, daß selbst einfache Aussagen ohne zusätzliche Informationen über die Bedeutung der verwendeten Vokabeln (insbesondere von Eigenschaften und Werten) nicht möglich sind. Diesem Nachteil tritt *RDF Schema (RDF-S)* [45] durch die Einführung von Klassen und Einschränkungen von Eigenschaften entgegen:

- Typisierung: In RDF ist es nicht möglich, über eine Ressource auszusagen, von welchem Typ sie ist. Insbesondere kann man Ressourcen, die Eigenschaften repräsentieren, nicht von anderen Ressourcen unterscheiden. Dies ermöglich RDF-S nun durch die Definition von Klassen und Unterklassen. Dadurch kann zuerst ein Modell der Struktur des zu repräsentierenden Wissens geschaffen werden, das dann mit den Ressourcen instanziiert wird. Dieses Klassenmodell ist aber - anders als z.B. in Java - nicht objektbasiert, sondern eigenschaftenbasiert („*property-centric*“), d.h. daß nicht eine Klasse mit ihren Eigenschaften/Attributen definiert wird, sondern es werden die *Argumentrahmen* der Eigenschaften beschrieben, also welche Ressourcen Subjekt und Objekt für die Eigenschaft sein können.

- Einschränkungen von Eigenschaften: Durch die Typisierung können mit RDF-S Ressourcen als Eigenschaften ausgewiesen werden. Diese können noch detaillierter beschrieben werden durch die Angabe, auf welche Klassen von Ressourcen sie anwendbar sind („domain“) und welche Ressourcen sie als Wert haben können („range“).

Mit RDF-S ist es dann möglich, neue Typen von Ressourcen und neue Eigenschaften zu definieren. Deshalb ist RDF-S eine *schema specification language*. Im Gegensatz zu XML Schema werden aber nicht nur rein syntaktische Bedingungen über die Struktur eines XML Dokumen-

tes festgelegt, sondern durch die Semantik der vordefinierten Schlüsselbegriffe Informationen über die Interpretation einer Aussage, die ein Reasoner dann umsetzt [66, 98].

Für die Sprache von RDF und RDF-S (im Folgenden in der Kombination mit *RDF/S* abgekürzt) wurden mehrere Notationsformen festgelegt, die geeignet sind, die zugrundeliegende Graphstruktur umzusetzen. Die intuitivste davon ist die sogenannte *Tripelsyntax*, die sich genau an der Dreiteilung jeder Aussage in Subjekt, Prädikat und Objekt orientiert. Die bedeutendste wohl aber ist die Serialisierung in XML, da dadurch eben alle Vorteile von XML genutzt werden können, nicht zuletzt die schon überall existierenden Parser. Dies ist deswegen auch die kanonische Syntax für RDF. Sie existiert auch in einer verkürzten Form, die aber trotzdem immer noch die XML-typische „Geschwätzigkeit“ aufweist.

Die Arbeit an RDF/S ist bis heute nicht endgültig abgeschlossen: Erst im September 2001 hat das W3C zwei weitere Working Drafts zu RDF/S veröffentlicht, um Kritikpunkte – vor allem an der Syntax und der fehlenden (modell-)theoretischen Fundierung von RDF – zu entschärfen.

DAML+OIL und OWL

Wie bereits im vergangenen Abschnitt erwähnt, sind Ontologien ein wichtiger Bestandteil des Semantic Web. Ontologiesprachen des Semantic Web verwenden RDF/S als Grundgerüst, welches wiederum in XML serialisiert wird, siehe Abbildung 3.2.

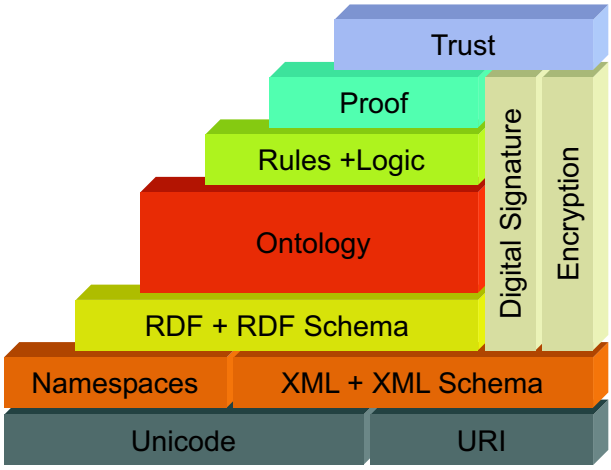

Abbildung 3.2: Semantic Web Stack

Eine der Ontologiesprachen des Semantic Web ist *DAML+OIL*, eine Fusion der beiden Sprachen *DAML-Ont* und *OIL*.

DAML-Ont ist das Ergebnis des *DARPA Agent Markup Language (DAML)*-Programms, das 1999 vom Department Of Defense (DoD) der US-Regierung unter Leitung der *Defense Advanced Research Projects Agency (DARPA)* mit der Zielsetzung gestartet wurde, Grundlagen

für das Semantic Web zu erarbeiten. Als Ausgangspunkt wurde das vom W3C vorgeschlagene RDF/S gewählt, das um Sprachkonstrukte aus dem Bereich der objektorientierten und frame-basierten Sprachen erweitert werden sollte, um eine größere Mächtigkeit zu erreichen.

Ontology Interference Layer (OIL) ist eine Ontologie-Austauschsprache, die ebenfalls seit 1999 in Europa vom *Information Society Technology (IST)*-Programm innerhalb der Unterprojekte *On-To-Knowledge* und *IBROW* entwickelt wurde. Auch hier war Ziel, das Semantic Web weiter voranzutreiben. OIL besitzt eine klar definierte, auf XML und RDF/S basierende Syntax.

Die Aktivitäten um DAML-Ont und OIL hatten also dasselbe Ziel und ähnliche Ansatzpunkte, weshalb dann 2000 im Rahmen der *Web-Ontology (WebOnt) Working Group* des W3C mit der Zusammenlegung zu DAML+OIL begonnen wurde, als man erkannte, daß man dadurch die jeweiligen Stärken herausarbeiten und die jeweiligen Schwächen minimieren konnte. Insbesondere versuchte man in DAML+OIL das Vokabular zur Definition der Semantik auszubauen. Dazu zählen z.B.:

- Erweiterbarkeit: Durch die Kombination mehrer Ontologie-Fragmente kann es zu Überschneidungen zwischen den verschiedenen Dokumenten kommen, die durch Konstrukte wie z.B. <daml:equivalentTo> und ähnliche in DAML+OIL gekennzeichnet werden können

- Auszeichnung besonderer Beziehungsformen: Es kann angegeben werden, ob es sich bei einer Relation um eine inverse, eine symmetrische oder transitive Eigenschaft handelt.

- Zwei Arten von Eigenschaften: Insgesamt wurde auf dem Gebiet der Eigenschaften viel gearbeitet. In der erweiterten Fassung von DAML+OIL vom März 2001 wurde die Unterstützung von *simpleType*s aus XML-Schema eingeführt. Dies bringt dem Entwickler von Ontologien zunächst einiges an Erleichterung, da all diese Typen nicht mehr neu angelegt werden müssen und schon ein einigermaßen breites Feld abdecken. Die Einführung brachte aber eine entscheidende Änderung zu RDF/S mit sich: Ziel-Ressourcen von Eigenschaften können nun entweder „echte“ Ressourcen, also in RDF-S oder DAML+OIL definierte Klassen (bzw. deren Instanzen) sein, oder XML-Schema-Typen (*datatypes*). Entsprechend dieser beiden Arten von Eigenschaften muß zu einer korrekten Behandlung durch eine Inferenzmaschine auch anhand der Eigenschaften differenziert werden. Es gibt in DAML+OIL nun also nicht mehr wie in RDF nur <rdf:Property>, sondern zwei verschiedene Typen von Eigenschaft: <daml:ObjectProperty> und <daml:DatatypeProperty>.

- Einschränkungen: Zusätzlich zu den Einschränkungen von Eigenschaften auf „Definitions-“ und „Wertebereich“ werden die Möglichkeiten zu Einschränkungen auf Klassen und Eigenschaften erhöht. Innerhalb von <rdfs:Restriction> kann auf jede Ressource zugegriffen werden und diese mit beliebigen anderen Eigenschaften näher spezifiziert werden. Weiterhin wurden Kardinalitäten von Eigenschaften eingeführt.

- Aufzählungen: Abgesehen davon wurden noch die Möglichkeiten der Aufzählungen zu RDF/S erweitert, indem zusätzlich zu „Sequence“, „Alternative“ und „Bag“ aus RDF-S in DAML+OIL der *parseType* „daml:collection“ für geschlossene Listen zur Verfügung gestellt wird.

Modelltheoretische Grundlage für DAML+OIL ist die Description Logic, deren Eignung zur Definition von Ontologien z.B. von Baader et al. in [30] motiviert wird. Baader et al. verweisen darauf, daß die Semantik von DAML+OIL in SHIQ [87] überführt werden kann, die als ausdrucksstarke Description Language klassifiziert wird und einen „guten Kompromiß zwischen der gewünschten Ausdrucksstärke und der Komplexität des Reasonings“ darstelle. Unter dieser Voraussetzung kann beispielsweise die *FaCT* Inferenzmaschine, die auf SHIQ aufbaut, als Reasoner für von DAML+OIL in SHIQ überführte Konzepte und Fakten verwendet werden. Andere Reasoner wie *Loom*, *Classical* oder *Racer* können unter ähnlichen Voraussetzungen zur Deduktion von Inferenzen angewandt werden.

DAML+OIL fand große Anwendung in der Praxis, und viele Tools wurden dafür entwickelt, ebenso wie schon eine Menge an Ontologien in DAML+OIL geschrieben wurden (siehe z.B. [1]).

Im November 2001 wurde bei der WebOnt Working Group damit begonnen, eine neue, standardisierte Sprache für das Semantic Web zu entwickeln, die von Anfang an von einer zentralen, unabhängigen Stelle entworfen sein sollte. Schließlich spielen bei einer Entwicklung durch ein Konsortium aus der Wirtschaft - wie bei DAML+OIL - immer die besonderen Interessen der jeweiligen Firmen eine wichtige Rolle. Mit dieser neuen *Web Ontology Language (OWL)* [126] sollten einige „geerbte“ Designfehler von DAML+OIL ausgeräumt werden, obwohl DAML+OIL weiterhin einen wichtigen Einfluß auf das Design hatte, und als Ausgangspunkt für OWL diente [57].

OWL ist als Schema in RDF/S definiert, jede OWL-Ontologie ist somit also auch ein wohlgeformtes und gültiges RDF-Dokument. Dadurch sind für OWL natürlich auch alle Serialisierungsarten, die in der Lage sind, RDF-Graphen umzusetzen, korrekte Serialisierungen.

Die Ausdruckskraft von OWL gliedert sich in drei Stufen, die sich in drei Facetten von OWL widerspiegeln:

- *OWL Lite* wurde mit dem Zweck definiert, eine einfache Sprache zu erzeugen, welche hauptsächlich die Bedürfnisse des Nutzers nach einer Klassifikationshierarchie und einfachen Beschränkungseigenschaften befriedigt. Es ist damit speziell geeignet für Nutzer oder Entwickler von Tools, die mit einem einfachen Basissatz an Sprachmitteln beginnen möchten.
- *OWL Description Logic* (kurz OWL-DL) beinhaltet den vollständigen OWL Wortschatz, welcher unter einer Anzahl einfacher Beschränkungen interpretiert wird. So kann hier z.B. eine Klasse nicht gleichzeitig Instanz einer anderen Klasse sein. Der Funktionsumfang entspricht damit in etwa dem von DAML+OIL. Er wurde so gewählt, um die bisherigen, auf *Description Logic* ausgerichteten Ansätze zu unterstützen, die damit leicht auf OWL-DL umzustellen sein sollen.
- *OWL Full* beinhaltet den gleichen Wortschatz wie OWL-DL, dieser wird aber umfassender interpretiert wird als in OWL-DL. Es entfallen alle Beschränkungen. Die Sprache OWL Full gilt in diesem Umfang aber als nicht mehr entscheidbar [126, 5].

Eine OWL-Ontologie kann unabhängig von der konkreten Unterart ebenso wie DAML+OIL

taxonomische Beziehungen zwischen Klassen, Eigenschaften von Datentypen, Beschreibungen der Attribute von Elementen von Klassen, Objekt-Eigenschaften, Beschreibungen der Beziehungen zwischen Elementen von Klassen und Instanzen von Klassen oder Eigenschaften enthalten. Eine Gegenüberstellung der Leistungsmerkmale von RDF/S, DAML+OIL und OWL wird z.B. von Gil und Ratnakar in [71] sowie in [6] vorgenommen.

F-Logic

Einen etwas anderen Ansatz verfolgt die Ontologiesprache *F-Logic* [60, 110], die ihre Wurzeln in der Frame-Logik hat [97, 21]. Die Repräsentation von F-Logic ist weder RDF/S- noch XML-basiert. Stattdessen wird eine Syntax verwendet, die sich stark an der Prädikatenlogik anlehnt. Sie verfügt aber auch über einige sehr angenehme Erweiterungen, die z.B. den objektorientierten Charakter der Frames ausdrücken oder die Ebene, auf die sich eine Aussage bezieht (Konzept- oder Instanz-Ebene). Ein einfaches Beispiel einer in F-Logic formulierten Ontologie kann Abbildung 3.3 entnommen werden.

```
/* concepts */
person.
man::person.
woman::person
person[father=>man; mother=>woman].

/* facts (instances) */
abraham:man.
sarah:woman.
isaac:man[father->abraham; mother->sarah].
ishmael:man[father->abraham; mother->hagar:woman].
abraham[marriedWith->sarah].

/* rules (axioms) consisting of a rule head and a rule body */
FORALL X,Y X[son->>Y] <- Y:man[father->X].
FORALL X,Y X[son->>Y] <- Y:man[mother->X].
FORALL X,Y X[daughter->>Y] <- Y:woman[father->X].
FORALL X,Y X[daughter->>Y] <- Y:woman[mother->X].
FORALL X,Y X[married->Y] <- Y[marriedWith->X].

/* query is a special rule without head */
FORALL X <- EXISTS Y X:woman[son->>Y[father->abraham]].
```

Abbildung 3.3: F-Logic Beispiel

Wie das Beispiel zeigt, ist die Syntax von F-Logic sehr eingängig. Konzepte und Instanzen werden durch erstmalige Nennung deklariert. Relationen auf der Konzeptebene (`=>` für einwertige, `=>>` für mehrwertige) beziehungsweise auf der Instanzebene (`->` für einwertige, `->>`

für mehrwertige) sind dabei klar unterscheidbar. Der Übergang von der Konzeptebene auf die Instanzebene über die `instanceOf`-Relation (Syntax `I:C`) hat ebenso eine besondere, bereits in der Sprache verankerte Darstellungsform, wie die `subconceptOf`-Relation (Syntax `C1::C2`) zwischen zwei Konzepten. Eine beliebig große Anzahl weiterer Relationen können in der Ontologie definiert werden. Die Nähe zur Prädikatenlogik wird vor allem durch die Quantoren `FORALL` und `EXISTS` deutlich.

Typ- bzw. Wertzuweisungen erfolgen analog zum Subjekt-Prädikat-Objekt-Modell in der Art `S[P->>O]`. Zuweisungen können jedoch auch in komplexerer Form vorgenommen werden, z.B. durch die Angabe eines spezifischen Konzepts C wie in `S:C[P->>O]`, von dem S in diesem Beispiel eine Instanz ist. Oder es werden mehreren Attributen gleichzeitig Werte zugewiesen wie in `S[P1->>O1; P2->>O2; P2->>O3]`. In diesen Beispielen ist die Nähe zur Objektorientierung durch den mit `[..]` aufgespannten Frame zu erkennen. In Anlehnung an objektorientierte Programmiersprachen wird auf Konzeptebene auch gerne die folgende Sichtweise verwendet: Aussagen der Form `S[P@(I1,I2)=>O]` deklarieren eine *Methode P* einer *Klasse S*, die den Definitionsbereich der Methode auf Objekte oder Daten des *Typs I1* für den ersten bzw. des *Typs I2* für den zweiten Inputparameter sowie den Wertebereich der Methode auf Objekte oder Daten des *Typs O* beschränken. Über die Methode P wird auf Instanzebene auf den Inhalt der Instanz der Klasse S zugegriffen, wodurch Objekte der Klasse O oder Daten des Typs O geliefert werden.

Das Reasoning findet bei F-Logic sowohl auf der Konzept-Ebene wie auch auf der Instanz-Ebene statt. Denn im Unterschied zur Description Logic kann ein Reasoner aus in Frame Logic formulierten Ontologien nicht nur Instanzen und ihre Werte als Antwort einer Anfrage ableiten, sondern auch die Konzepte und ihre Attributnamen. Zur Anfrage der Inferenzmaschine wird keine eigene Anfragesprache z.B. SQL bei das ER-Modell implementierenden Datenbanken verwendet, sondern stattdessen F-Logic selbst: Eine F-Logic *Query* ist eine eingeschränkte Form einer F-Logic Regel (*Axiom*), bei welcher der sogenannte *rule head* leer ist. Dabei kann das bei einer Query zu ermittelnde Wissen eine Belegung für jede Stelle einer F-Logic Formel sein. So ist zum Beispiel eine Anfrage `FORALL X <- X[mother->sarah]`, was bei einer Wissensbasis, die mit den Daten aus Abbildung 3.3 gefüllt ist, `X = isaac` liefern würde, ebenso gültig wie eine Anfrage `FORALL X <- isaac[mother->X]` (Ergebnis `X = sarah`) oder `FORALL X <- isaac[X->sarah]` (Ergebnis `X = mother`).

Eine besondere Stärke von F-Logic sind die Axiome. Neben der Formulierung einfacherer Regeln, die die Symmetrie (siehe letzte Regel im o.g. Beispiel), Inversität oder Transitivität einer Beziehung ausdrücken, sowie der schon erwähnten Sonderform als Query, ermöglichen sie die Definition komplexer Sachverhalte.

Dies kann am Beispiel der Kardinalitäten verdeutlicht werden: F-Logic enthält als Sprache nur eine eingeschränkte Form von Kardinalitätsangaben. So kann mit `C[A=>R]` ausgedrückt werden, daß ein Attribut A des Konzepts C einwertig ist und den Wertebereich R hat, bzw. mit `C[A=>>R]`, daß ein Attribut A des Konzepts C mehrwertig ist und den Wertebereich R hat. Andere Kardinalitäten müssen durch selbstdefinierte Erweiterungen repräsentiert werden, z.B. mit einstelligen Prädikaten `human::Cardinalized[hasParent=>>human; mincard@(hasParent)=>2; maxcard@(hasParent)=>2]`. Zur Laufzeit kann mit der Query `FORALL I,R <- cardinalityError(I,R)` basierend auf dem in Abbildung 3.4 dargestellten

Axiom festgestellt werden, welche Instanzen I bezüglich ihrer Relation R den auf Konzeptebene festgelegten Kardinalitätsangaben widersprechen. So würde z.B. `peter:human[hasParent ->>{paul, maria, josef}]` zu einer Fehlermeldung führen. Insbesondere von den Axiomen wird daher im weiteren Verlauf dieser Arbeit häufiger Gebrauch gemacht.

```
rule cardinalityError:
FORALL obj,method cardinalityError(obj,method) <- EXISTS concept,Y1,Y2,Z1,Z2,V1,V2
  concept::Cardinalized AND
  (concept[mincard@(method)=>V1] AND ((obj:concept[method->Y1] OR
    obj:concept[method->>Y1]) AND count(obj,Y1,Z1)) AND less(Z1,V1)) OR
  (concept[maxcard@(method)=>V2] AND ((obj:concept[method->Y2] OR
    obj:concept[method->>Y2]) AND count(obj,Y2,Z2)) AND greater(Z2,V2)).
```

Abbildung 3.4: Kardinalitäten in F-Logic

Reasoner für in F-Logic formulierte Ontologien sind z.B. die Inferenzmaschinen *SiLRI* oder deren kommerzieller Nachfolger *OntoBroker* [60]. OntoBroker bietet neben deduktivem Inferenzing eine Reihe sehr nützlicher Fähigkeiten, die bei der Umsetzung des im weiteren Verlauf dieses Kapitels vorgestellten Modells und dessen Nutzung in verteilten Dienstumgebungen von Vorteil sind. Dazu zählen beispielsweise eine Reihe mitgelieferter sogenannter „Builtins“[2], die u.a. die Notwendigkeit doppelter Datenhaltung durch den direkten Zugriff auf performante Datenbanken per `dbaccess` Builtin (siehe Beispiel in Abbildung 3.5) reduzieren. Es können darüber hinaus auch eigene Builtins integriert werden, die dann im Deduktionsprozeß verwendet werden.

```
FORALL Hotel <-
  dbaccess( location_tbl,
            F(hotelName_col, Hotel, locatedNear_col, "CeBIT"), // cf. SQL WHERE
            "mysql","db","mysqldb.context-aware.org" ).
```

Abbildung 3.5: Integration von Datenbanken zur Verwaltung großer Datenmengen

OntoBroker hat sich mit seinen Eigenschaften positiv aus der Menge der getesteten Inferenzmaschinen hervorgehoben und wurde daher - wo nicht etwas ausdrücklich anderes erwähnt wird - zur Evaluierung der in dieser Arbeit vorgestellten Konzepte verwendet.

[2] eigenständige, auch von Dritten programmierbare und in den Reasoningprozeß integrierbare Programm-Module

3.2 Entwurf eines Kontextmodells

Im Folgenden wird ein neuer Ansatz zur Modellierung kontextueller Konzepte, Fakten und Abhängigkeiten vorgestellt. Grundlegend für diesen Ansatz ist die Verwendung von Ontologien zur Beschreibung des Modells und darauf basierender Ausprägungen. Die Verwendung von Ontologien ist dabei insbesondere gewinnbringend für die Spezifikation des kontextuellen Wissens als *shared understanding*, welches Grundlage von kontextueller Service Interoperabilität ist.

Das Modell, welches im Folgenden zunächst motiviert und dann im Detail vorgestellt wird, ist die Basis für eine Sprache zur Spezifikation kontextueller Interoperabilität, der *Context Ontology Language (CoOL)*. Mit den Elementen dieser Sprache können kontextuelle Fakten und kontextuelle Beziehungen spezifiziert werden. Kontextuelle Interoperabilität zwischen den an einer Dienstinteraktion beteiligten Komponenten kann dann durch Anwendung dieser erstellten Spezifikation ermöglicht werden. Da die Sprache auf Ontologien aufsetzt, kann neben einer Taxonomie von kontextuellen Fakten und Relationen mit der Hilfe von Inferenzregeln auch implizites Wissen über den Kontext explizit angegeben werden.

Die Sprache selbst besitzt keine monolithische Struktur, sondern ist aus verschiedenen Fragmenten aufgebaut, die in zwei Untermengen gruppiert werden. Der Teil, der das in diesem Kapitel vorgestellte Modell umsetzt, wird einer Kerngruppe, *CoOL Core*, zugeordnet. Alle übrigen Fragmente, die in den darauf folgenden Kapiteln vorgestellt werden, haben vornehmlich Referenz-, Adaptions- und Integrationsaufgaben, und sind deshalb in einer Integrationsgruppe, *CoOL Integration*, zusammengefaßt.

3.2.1 Stand der Technik

In den letzten Jahren wurden bereits einige Ansätze zur Modellierung von Kontext vorgestellt. Sie alle sind stark beeinflußt von der jeweiligen Definition des Kontextbegriffs. Die wichtigsten Varianten der Definitionen, auf denen sich die unterschiedlichen Modelle stützen, wurden bereits in Abschnitt 2.3.1 im Rahmen der Analyse des Kontextbegriffs vorgestellt.

Der Stand der Technik der Modellierung von Kontext läßt sich klassifizieren nach der Art der Datenstrukturen, die zum Austausch von Kontextinformationen verwendet werden und üblicherweise in die folgenden Kategorien fallen:

Key-Value Modell Das Modell der Key-Value Paare ist die einfachste verwendete Datenstruktur zur Modellierung von Kontextinformationen. Bereits Schilit et al. [138] haben den Kontext mit Hilfe von Key-Value Paaren modelliert, indem sie den Wert der Kontextinformationen (z.B. Ortsangaben) per Environment-Variable einer Anwendung zur Verfügung stellen. Häufig findet das Key-Value Modell insbesondere in verteilten Dienstumgebungen (z.B. Capeus, siehe Abschnitt 2.4.3) Anwendung, in denen auch die Dienste selbst durch einfache Attribute beschrieben sind, bzw. das eingesetzte Dienstvermittlungsverfahren (SLP, Jini etc.) exaktes Matching auf Schlüsselwortbasis durchführt (siehe Abschnitt 2.4.3). Key-Value Paare sind besonders einfach zu verwalten, sind aber

nicht besonders gut zur Strukturierung geeignet.

Markup-Tag Modell Im Markup-Tag Modell wird der Kontext in Form von Markup-Tags mit Attributen und Inhalt modelliert, die rekursiv weitere Markup-Tags beinhalten können. Typische Vertreter dieses Modells sind Profile, die auf Serialisierungen in *Standard Generic Markup Language (SGML)*, dem „Urvater" aller Markup-Sprachen wie XML, und dessen verschiedenen Derivaten basieren. Hierzu zählen z.B. die *Comprehensive Structured Context Profiles (CSCP)* [81] und die *CC/PP Context Extension* [90]. Beide orientieren sich an den in Abschnitt 2.4.3 beschriebenen Standards CC/PP [168] und UAProf [171] mit RDF/S Aussagenmodell und XML Serialisierung. Sie erweitern bzw. ergänzen diese auf verschiedene Weise, um die höhere Komplexität und die im Vergleich zu statischen Profilen in der Regel viel höhere Dynamik von Kontextinformationen abzudecken. Andere, nicht auf CC/PP aufbauende Ansätze wie die *Pervasive Profile Description Language (PPDL)* [56] oder *ConteXtML* [134] modellieren den Kontext in wenigen, festgelegten Dimensionen und haben daher recht eingeschränkte Einsatzfelder.

Grafisches Modell Grafisch orientierte Verfahren wie die *Unified Modeling Language (UML)* [121] oder das *Object Role Model (ORM)* [79] können natürlich ebenfalls zur Modellierung des Kontextes verwendet werden. Henricksen et al. erweitern beispielsweise das ORM um kontextuelle Klassifikations- und Beschreibungsmerkmale [83], die anschließend auf ein ER-Modell einer relationalen Datenbank abgebildet werden [84].

Objekt-orientiertes Modell TEA's *Cues* [140] (siehe Abschnitt 2.12) oder das *GUIDE Active Object Model* [54] können als typischer Vertreter eines objektorientierten Ansatzes zur Modellierung des Kontextes angesehen werden. In Ansätzen dieser Art sind Kontextinformationen in einem Objekt als Statusinformationen gekapselt, und ein Zugriff auf diese Informationen ist nur über die Interfaces des Objekts möglich.

Logik-basiertes Modell In Logik-basierten Modellen wird der Kontext durch Fakten in einem Regel-basierten System modelliert. Üblicherweise werden Kontextinformationen durch Hinzufügen neuer Logik-Regeln in ein System eingebracht, bzw. wird mit Hilfe von logischen Regeln das kontextuelle Wissen abgeleitet. Vertreter dieser Modellierungs-Variante sind das Location-orientierte Multimediasystem von Bacon et al. [31], das *Sensed Context Model* von Gray und Salber's [73] sowie der als *Erweiterte Situationstheorie* bezeichnete Ansatz von Akman und Surav [20].

Ontologie-basiertes Modell Einer der ersten Ansätze der Modellierung des Kontextes auf der Basis von Ontologien war der von Ötztürk und Aamodt [123]. Aus der Auswertung psychologischer Studien zum Unterschied zwischen dem *Erinnern (recall)* und dem *Erkennen (recognize)* von verschiedenen Sachverhalten in Verbindung mit Kontextinformationen haben sie die Notwendigkeit abgeleitet, das Wissen verschiedener Domänen zu normieren und zu kombinieren, und haben sich dabei aufgrund der dafür optimalen Fähigkeiten (siehe Abschnitt 3.1.1) der Ontologien bedient. Ein anderer, noch sehr junger Ansatz der Modellierung von Kontextinformationen auf der Basis einer Ontology ist das CoBrA System [52].

Diese Kategorisierung gibt nur eine Primärzuordnung der genannten Modelle an, da einige der genannten Modelle in andere Darstellungsformen überführt werden können, bzw. als hybride Modelle verschiedene Klassifikationsmerkmale erfüllen.

Das neue, im weiteren Verlauf dieser Arbeit vorgestellte Modell ist bezüglich der oben vorgenommenen Aufteilung der Kategorien ein hybrides Modell: Während die eigentliche Modellierung des kontextuellen Wissens als Ontologie erfolgt, werden im Modell logikbasierte Erweiterungen dazu verwendet, das Wissen zu komplettieren und vor allem individuell und kontextuell optimal anzuwenden.

3.2.2 Anforderungen an das neue Modell

Aufgrund des geplanten primären Verwendungszwecks des neuen Modells zur Spezifikation und Überprüfung des *shared understanding* als Grundlage kontextueller Interoperabilität in verteilten Dienstumgebungen standen beim Design des neuen Modells vor allem folgende Anforderungskriterien im Vordergrund:

- Hoher Formalisierungsgrad
- Verteilte Komposition
- Partielle Validierung
- Vollständigkeit und Qualität der Informationen
- Vergleich nicht skalarer Typen
- Integrationsfähigkeit in bestehende Dienstumgebungen

Diese Anforderungen an das neue Modell zur besonderen Eignung im Rahmen kontextadaptiver Dienstnutzung werden im Folgenden näher erläutert.

Hoher Formalisierungsgrad

Ein Nachteil vieler bisher veröffentlichter Verfahren zur Modellierung von Kontext ist ein zu geringer Grad an Formalität. Je genauer die Spezifikation des *shared understanding* durch das zugrundeliegende Modell ermöglicht wird, desto besser lassen sich auch diese (Kontext-) Informationen zur Überprüfung von Interoperabilitätsbedingungen durch den Computer nutzen.

Oft wird viel Wert auf Klassifikationshierarchien (z.B. [83, 123]) und umfangreiche Attributsammlungen (z.B. [90, 138]) im Sinne der Spezifikation des Definitionsbereichs von Kontextbeschreibungen gelegt. Andere Modelle wie z.B. das *Sensed Context Model* von Gray und Salber [73] verzichten gleich ganz auf die Angabe einer formalen, abstrakten Definition des Modells oder zeigen wie bei Akman und Surav [20] nur an vereinzelten Beispielbetrachtungen, wie ein Modell konstruiert werden kann, ohne dies allgemein zu tun. Die Spezifikation des Wertebereichs der einzelnen Attribute beschränkt sich allzuoft auf die Angabe des Datentyps (String, Integer etc.). Dies ist jedoch bei weitem nicht ausreichend, da beispielsweise in verteilten Dienstumgebungen bei der Dienstvermittlung mit exaktem Matching die angefragten Attributwerte mit den jeweiligen, den Dienst beschreibenden (Kontext-)Informationen exakt übereinstimmen müssen, um als positives Matching erkannt zu werden. Daher ist eine genaue und in sich abgeschlossene Spezifikation des Wertebereichs der einzelnen Attribute

wünschenswert, und damit ein höherer Formalisierungsgrad der auf dem Kontextmodell basierenden Daten.

Mit der abgeschlossenen Spezifikation des Wertebereichs einher geht die Forderung nach einer Unterstützung durch das neue Modell von unendlich großen Wertebereichen, z.B. der Menge der Natürlichen Zahlen $\mathbb{N}$. Außerdem müssen auch solche Wertebereiche spezifiziert werden können, deren Inhalt nicht konstant ist, sondern sich als Ergebnis einer parametrisierten Funktion wie z.B.

$$\mathcal{W}_{POInearMe} = \{PointOfInterest \;\; poi \mid$$

$$poi \in near(myPosition), myPosition = GaussKrueger('367032','533074')\}$$

ergibt. Die relative Ordnung der Elemente des Wertebereichs, auf deren Grundlage Vergleichsoperationen durchgeführt oder Wertebereiche definiert werden, soll im neuen Modell individuell für die jeweilige Aufgabenstellung angegeben werden können.

Die Verknüpfung des Kontextmodells mit den Modellen der Signatur-, Protokoll- und Semantikebene der Service-Interoperabilität erhöht ebenfalls den Grad der Formalisierung. So sollte jede Kontextinformation nicht nur durch die Syntax des Modells erfaßbar sein, sondern auch einer bestimmten Dimension des Situationsraums semantisch zugeordnet werden können. Dies ermöglicht ein breites Anwendungsfeld semantischer Verfahren zur Beschreibung und Auswertung von Kontextinformationen über die semantische Zugehörigkeit zur Dimension des Situationsraums.

In Abschnitt 2.3.1 wurde bereits die wichtige Rolle der *Relevanz* herausgearbeitet. Sie bestimmt den Unterraum des Situationsraums, der den Kontext einer spezifischen Aufgabenstellung bestimmt. Die Rolle der Relevanz wird in vielen Modellen vernachlässigt. Das neue Modell soll jedoch ein Anker für die Relevanz darstellen, d.h. die Angabe individueller Relevanzkriterien nicht zuletzt durch einen hohen Grad an Formalität und normierter Semantik des Modells selbst und damit vererbter semantischer Typisierung erlauben.

Verteilte Komposition

In einem verteilten System gibt es keine zentrale Instanz, die für die Erstellung und Pflege von Daten und Diensten, insbesondere von Kontextbeschreibungen, zuständig ist. Verteilte Systeme zeichnen sich gerade dadurch aus, daß die Erstellung und Pflege von Daten und Diensten zeitlich und netztopologisch von einer Vielzahl von Autoren verteilt vorgenommen wird, was in der Ausprägung der verteilten Systeme als Ubiquitous Computing Systeme wie in Abschnitt 2.4 beschrieben einer besonders hohen Dynamik unterliegt.

Andere Modelle berücksichtigen diese Dynamik in der Regel nur auf der Instanzebene und sind auf der Konzeptebene relativ statisch, da sie wie beispielsweise bei Henricksen et al. [84] auf der Konzeptebene lediglich semi-zeitinvariante Abbildungen auf ein ER-Modell vornehmen (vgl. auch Abschnitt 3.1.1).

Das neue Modell soll der hohen Dynamik u.a. durch die Möglichkeit der verteilten Komposition auf Konzept- und Instanzebene Rechnung tragen. Das Problem nicht eindeutiger

Namensgebung wird dabei wie üblich durch die Verwendung von eindeutigen Namensräumen minimiert.

Partielle Validierung

Aus der verteilten Komposition ergibt sich, daß das kontextuelle Wissen an keiner Stelle und zu keiner Zeit vollständig an einem Knoten des verteilten Systems vorliegt. Trotzdem ist es sehr wünschenswert, wenn die Spezifikation des kontextuellen Wissens auf Konzept- und Instanzebene zumindest partiell, d.h. soweit bekannt und im Zugriff, automatisiert vom Ersteller der Spezifikation auf Korrektheit geprüft werden kann. Gerade die Modellierung komplexer kontextueller Zusammenhänge ist sehr fehlerträchtig. Daher wirkt sich eine möglichst frühzeitige, toolgestützte Validierung der erstellten Spezifikation des kontextuellen Wissens stark fehlerminimierend aus. Außerdem soll das neue Modell nach Möglichkeit bereits durch inhärente Konstrukte Fehlangaben vermeiden helfen bzw. durch fehlersichernde Maßnahmen Fehler besser erkennbar machen.

Vollständigkeit und Qualität der Informationen

Die Menge der zu einem Zeitpunkt verfügbaren Kontextinformationen bezüglich der relevanten Entitäten ist in Ubiquitous Computing Umgebungen insbesondere durch die spezielle Charakteristik von Sensornetzen (siehe Abschnitt 2.4.2) oft nicht vollständig oder auch mehrdeutig.

Daher soll das neue Modell bereits in den Basiskonzepten Meta-Informationen verankert haben, welche die Qualität einer Kontextinformation beschreiben können. Diese Beschreibungen soll nach Möglichkeit auf dem gleichen Modell basieren wie die Kontextinformationen selber, damit auch deren Auswertung mit den gleichen Verfahren möglich ist.

Die auf dem neuen Modell aufbauenden Verfahren sollen darüber hinaus in der Lage sein, auch mit fehlenden, unvollständigen oder mehrdeutigen Kontextinformationen korrekt umzugehen. Dazu zählt auch der korrekte Umgang mit neuartigen Kontextinformationen, deren Typ im Sinne der verteilten Systeme zu einem späteren Zeitpunkt, insbesondere zeitlich nach Einführung des Auswerteverfahrens, angelegt und im System bekannt gemacht wurden. Wenn aufgrund irgendwelcher Umstände eine eineindeutige Zuordnung von Kontextinformationen zu der Klasse, aus der sie erzeugt wurden, nicht möglich ist, soll auch die Mehrdeutigkeit bei der Auswertung entsprechend berücksichtigt werden.

Vergleich nicht skalarer Typen

Der Vergleich nicht skalarer Typen wird bei der Nutzung der meisten der bisher bekannten Kontextmodelle durch die Anwendung einer hochspezialisierten Vergleichsfunktion (siehe z.B. [67]) als inhärenter Sonderfall der Modellanwendung realisiert oder wird allgemein auf einen Vergleich einer String-Repräsentation der Daten zurückgeführt. Beide Varianten haben den

Nachteil, daß sie entweder zu speziell sind (da sie nicht auf andere Typen von Kontextinformationen basierend auf dem gleichen Modell anwendbar sind) oder zu allgemein (da sie die speziellen Eigenschaften der jeweiligen Typen von Kontextinformationen nicht berücksichtigen). Keines der bekannten Modelle sieht jedoch - soweit erkennbar - bereits allgemein im Modell den Vergleich nicht skalarer Typen von Kontextinformationen vor.

Eine Anforderung an das neue Modell ist daher, daß nicht skalare Typen von Kontextinformationen nicht als Sonderfall behandelt werden, sondern alle Typen, ob skalar oder nicht skalar, mit dem gleichen Modell und dessen Semantik erfaßt werden.

Integrationsfähigkeit in bestehende Dienstumgebungen

Da das neue Modell zur besonderen Eignung im Rahmen kontextadaptiver Dienstnutzung eingesetzt werden soll, ist natürlich die Integrationsfähigkeit des Modells in bestehende Dienstumgebungen ein wichtiges Entwurfskriterium. Dies betrifft einerseits die Repräsentation des Wissens und andererseits die Möglichkeiten zur Abfrage des Wissens. Optimalerweise werden für beide die gleichen Verfahren eingesetzt.

Wo eine direkte Integration des Modells z.B. aufgrund unterschiedlicher Serialisierungsdarstellungen (z.B. XML-basierte Dienstnutzungsumgebung á la Web Services vs. nicht-XML-basierter Serialisierung von Kontextinformationen) nicht möglich ist, muß die Integrationsfähigkeit durch zusätzliche Module sichergestellt werden.

3.2.3 ASC-Modell

Im Folgenden wird nun das neue Modell aus zwei verschiedenen Sichten vorgestellt. Zunächst erfolgt vorab eine grundsätzliche Einführung in die Semantik der verwendeten Basiskonzepte und Relationen. Die Sichtweise wird als horizontale Perspektive bezeichnet, weil neben einer kurzen Einführung der Konzepte vor allem die wichtigsten horizontalen Beziehungen zwischen den Konzepten im Vordergrund stehen. Gleichzeitig fungiert die Kurzeinführung der Konzepte als eine Art Vorwärtsreferenz auf die Konzepte, davon abgeleitete Unterkonzepte und deren Instanzen, die anschließend detailliert und framebezogen vorgestellt werden. Durch diese Zweiteilung mit verschiedenen Sichten erfolgt eine Beschreibung des Modells, die einerseits für den Leser nachvollziehbar und andererseits vollständig ist.

Horizontale Sicht des Modells

In diesem Abschnitt werden die Grundidee des im Rahmen dieser Arbeit neu entwickelten Modells durch eine horizontale Sicht auf die drei namensgebenden Basiskonzepte *Aspekt (aspect)*, *Skala (scale)* und *Kontextinformation (context information)* und ihre Beziehungen untereinander anhand eines einfachen Beispiels geometrischer Ortsinformationen vorgestellt. Hierbei ist unter *Konzept* der – in Abschnitt 3.1.1 eingeführte – ontologische Begriff des Konzepts zu verstehen. Deshalb erfolgt eine erste Untergliederung des in Abbildung 3.6 dargestellten

Beispiels in die Ebene der Konzepte (*concepts*, Rechtecke) und die Ebene der Instanzen (*facts*, Ellipsen). Die in dieser Abbildung verwendete Symbolik der Relationen orientiert sich an der in Abschnitt 3.1.2 vorgestellten Notation zur Kennzeichnung von Relationen auf der Konzeptebene (`=>` für einwertige, `=>>` für mehrwertige) beziehungsweise auf der Instanzebene (`->` für einwertige, `->>` für mehrwertige). Die Beschriftung von Konzepten, Relationen und Instanzen gibt ihren Namen an, wobei der Übergang von der Konzeptebene auf die Instanzebene über die `instanceOf`-Relation durch die Syntax `I:C` in der Beschriftung verdeutlicht wird. Ebenso wird die `subconceptOf`-Relation zwischen zwei Konzepten durch die Syntax `C1::C2` besonders gekennzeichnet. Abstrakte Konzepte, die nicht direkt instantiiert werden können, sind kursiv gesetzt.

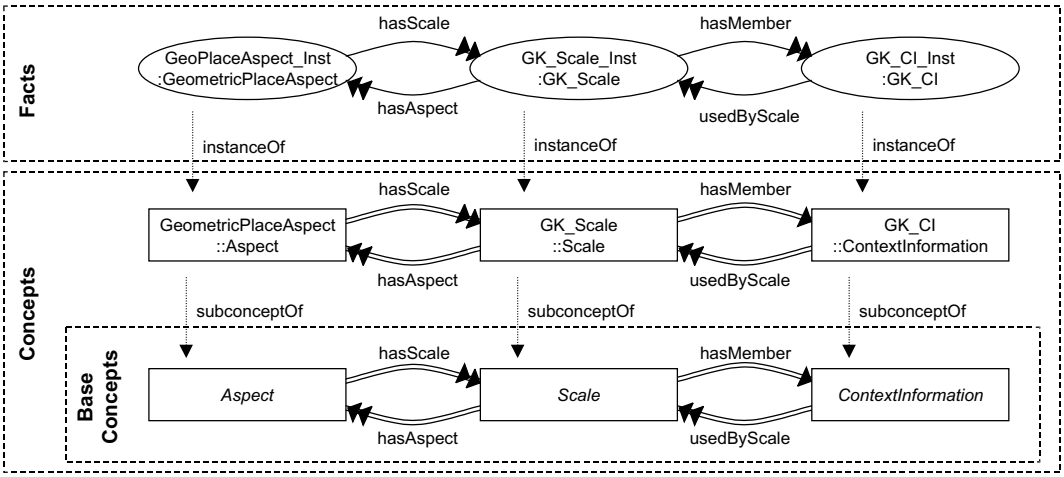

Abbildung 3.6: ASC Basiskonzepte und ihre Relationen, horizontale Sicht

Das ASC-Modell leitet sich stark aus der Analyse des Kontextbegriffs in Abschnitt 2.3.1 ab. Insbesondere der im Vergleich zu früheren Definitionen neu eingeführte Begriff des *Aspekts* (siehe Definition 4) hat eine zentrale Rolle im Modell. Einerseits ist jede Instanz des Konzepts *Aspekt* eine Aggregation von semantisch äquivalenten *Skalen*. Diese werden über die Relation `hasScale` einem Aspekt zugeordnet. Andererseits ist jede Instanz des Konzepts *Aspekt* eine Dimension des Situationsraums, d.h. ein Identifikator für eine Gruppe von Werten oder Symbolen mit eigener, identifizierbarer Semantik. Daher ist der *Aspekt* auch ein primärer Anknüpfungspunkt für Beschreibungen der semantischen Interoperabilitätsebene.

Die Diversität unterschiedlicher *Aspekte* ist nach dem Modell bei den unterschiedlichen Unterkonzepten des Aspekts und nicht bei der Instanz des Aspekts verankert. Damit Aspekte bezüglich der Instanzen der im Folgenden vorgestellten Konzepte *Skala* und *Kontextinformation* nicht auf einer Metaebene befinden, gibt es zu jedem Konzept *Aspekt* eine korrespondierende Instanz des *Aspekts*. Dieser Sachverhalt ist in Abbildung 3.6 durch die `instanceOf`-Relation zwischen `GeographicPlaceAspect` auf Konzeptebene und `GeographicPlaceAspect_Inst` auf Instanzebene dargestellt. Obwohl diese Konstellation die Verwendung des *Singleton Patterns* [68] nahelegt, wird von seiner Verwendung an dieser Stelle abgesehen, da der Mehrwert dieses Entwurfsmusters in einem sehr schlechten Verhältnis zu seiner praktischen Umsetzung in

verteilten Systemen steht.

Schon mehrmals wurde im Verlauf dieser Arbeit auf die fehlende formale Grundlage des Wertebereichs der meisten anderen Modelle zur Beschreibung des Kontextes in seiner jeweiligen Definition hingewiesen. Im hier vorgestellten ASC-Modell wird diese Lücke durch das Konzept der *Skala* geschlossen.

Eine Instanz des Konzepts *Skala* bzw. eines entsprechenden Unterkonzepts (siehe `GK_Scale` in Abbildung 3.6) ist eine zunächst ungeordnete Menge von über die Relation `hasMember` zugeordneten *Kontextinformationen* nach Definition 6, die den Wertebereich der Kontextinformationen dieser Skala darstellen. Mit anderen Worten: Eine gültige Kontextinformation bezüglich eines Aspekts ist immer ein Element einer der Skalen des Aspekts. In Abbildung 3.6 wird mit `GK_Scale_Inst` eine solche konkrete Instanz des Unterkonzepts `GK_Scale` bezeichnet.

Zum Beispiel können dem Aspekt `GeographicPlaceAspect_Inst` zwei Skalen zugeordnet sein, `WGS84_Scale_Inst` und `GK_Scale_Inst`. Eine zunächst *wohlgeformte* Kontextinformation könnte dann eine Objektinstanz sein, welche in einer objektorientierten Programmiersprache wie Java mit `new GK_CI("367032", "533074")` erzeugt wurde. Ist die dadurch repräsentierte Instanz auch in der durch `GK_Scale_Inst` repräsentierten Menge enthalten, so handelt es sich nicht nur um eine *wohlgeformte* Kontextinformation, sondern um eine *gültige* Kontextinformation bezüglich des Aspekts `GeographicPlaceAspect_Inst`.

In Abbildung 3.6 wird eine Unterscheidung der Wertigkeit der `hasMember`-Relation auf Konzept- und Instanzebene vorgenommen. Hierdurch werden die individuellen Spezifikationsmöglichkeiten auf beiden Ebenen u.a. genutzt, um auszudrücken, daß alle von einer Skala aggregierten Instanzen von Kontextinformationen vom gleichen Typ sein müssen. Auf diesen Sachverhalt wird bei der nun folgenden vertikalen Sicht auf das Modell noch einmal detaillierter eingegangen.

Vertikale Sicht des Modells

Die Konzepte des Modells werden im Folgenden in einer frameorientierten Sicht im Detail vorgestellt. Dabei wird auf die zum Teil bereits in der horizontalen Sicht des Modells vorab eingeführten Konzepte und ihre Beziehungen zurückgegriffen. In diesem Abschnitt erfolgt eine tiefgehendere Betrachtung der einzelnen Elemente des Modells.

Die verwendeten Konzepte, Relationen etc. werden im Folgenden zur Veranschaulichung teilweise graphisch dargestellt. Bei diesen Grafiken wird die in Abbildung 3.7 gezeigte Symbolik verwendet. Rechtecke stellen darin ontologische Konzepte dar, Ellipsen ontologische Instanzen (Fakten). Alle horizontalen Linien führen zu einem Prädikat des oberen Konzepts und sind links durch ihren Typ (ObjectProperty oder DatatypeProperty), und rechts durch Kardinalitätsangaben bezeichnet. Vertikale Linien bündeln alle zu einem Frame des Konzepts gehörenden Prädikate. Neben jedem Prädikat steht ein Konzept, das den Wertebereich des Prädikats angibt.

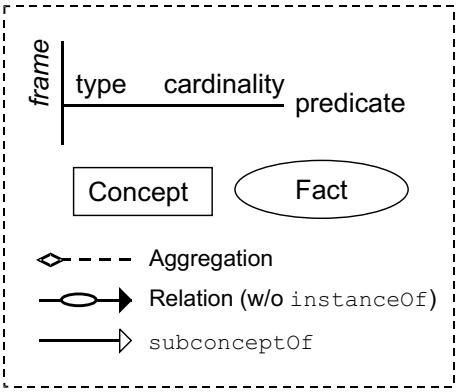

Abbildung 3.7: Legende der verwendeten Symbolik

Operationen und Parameter

Aus mathematischer Sicht versteht man unter einer n-stelligen Operation auf einer Menge M eine eindeutige Abbildung von M^n in M [63]. Definitions- und Wertebereich einer Operation können sich auch unterscheiden und müssen getrennt angegeben werden. Aus Sicht der Informatik ist eine Operation eine Funktion mit n Eingabeparametern aus dem jeweiligen Definitionsbereich und einem Ausgabeparameter aus dem Wertebereich.

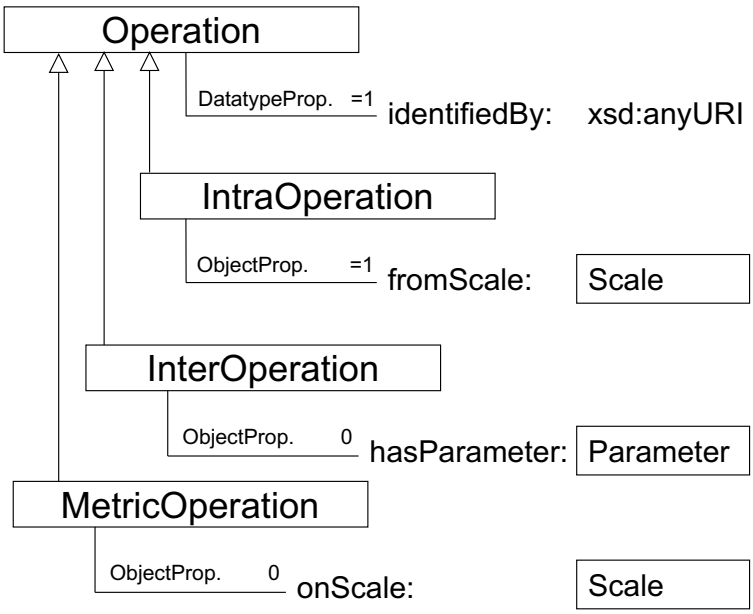

Abbildung 3.8: Das Konzept der Operation

Das ontologische Konzept der *Operation* (siehe Abbildung 3.8) ist ein Oberkonzept für ver-

schiedene Arten von Operationskonzepten im ASC-Modell. Über Instanzen dieses Konzepts erhält eine dem Reasoner nachgeschaltete Architekturkomponente Zusatzinformationen über einzelne Operationen eines Dienstes, die dadurch für andere Instanzen des ASC-Modells für bestimmte Aufgabenstellungen anwendbar werden.

Das Oberkonzept der *Operation* verweist über die Relation `identifiedBy` zunächst lediglich auf eine Beschreibung der Signatur einer Operation, beispielsweise einen `portType`-Identifikator in einer WSDL-Datei, wie er in Abschnitt 2.2.2 vorgestellt wurde. Durch Zuordnung einer Operation zu einer Ziel-Skala kann die Skala als Wertebereich einer Operation definiert werden. Gleiches gilt für jeden einzelnen Eingabeparameter einer Operation: Durch Zuordnung eines Eingabeparameters zu einer Quell-Skala kann die Skala als Definitionsbereich dieses Parameters einer Operation definiert werden. Wird im Reasoningprozeß aufgrund einer entsprechend formulierten Anfrage eine Instanz des Konzepts Operation identifiziert, hat eine nachgeschaltete Komponente des Service Frameworks dadurch bereits auf der Signaturebene ausreichend Wissen, um die Operation syntaktisch korrekt initiieren zu können.

Eines der im ASC-Modell differenzierten Unterkonzepte der Operation ist die *IntraOperation*. Das Konzept der IntraOperation wird dazu verwendet, Operationen zu beschreiben, die als Abbildungsfunktion von einer Skala in eine andere Skala des *gleichen* Aspekts verwendet werden können. Das bedeutet, die Elemente der einen Skala sind der Definitionsbereich, die Elemente der anderen Skala der Wertebereich der IntraOperation. Eine IntraOperation ist immer der Ziel-Skala zugeordnet. Wie im letzten Absatz beschrieben entfällt hierduch die Notwendigkeit der expliziten Definition eines Wertebereichs, da dieser implizit durch die Ziel-Skala gegeben ist. Für die automatisierte Verarbeitung im Service Framework ist es notwendig, daß dieses weiß, woher die Werte einer identifizierten IntraOperation stammen. Zu diesem Zweck verfügt jede IntraOperation über die Relation `fromScale`, die auf Instanzebene Auskunft darüber gibt, aus welcher Skala die Elemente genommen und als Eingabewerte für die IntraOperation verwendet werden sollen. Da IntraOperationen immer Werte genau einer Quell-Skala auf Werte einer Ziel-Skala abbildet, handelt es sich um eine 1-stellige Operation, deren Definitionsbereich die Quell-Skala ist.

Wie beschrieben dienen IntraOperationen dazu, die Abbildungsfunktion von einer Skala in eine andere Skala des *gleichen* Aspekts zu charakterisieren. In ähnlicher Weise wird das Konzept der *InterOperation* dazu verwendet, Operationen zu beschreiben, die als Abbildungsfunktion von einer Skala in eine Skala eines *anderen* Aspekts dienen können. Auch eine InterOperation ist immer der Ziel-Skala zugeordnet, welche den Wertebereich der InterOperation definiert. Entscheidender Unterschied in der Modellierung ist, daß InterOperationen auch komplexe Operationen mit mehreren Eingabewerten beschreiben können. Aus diesem Grund muß für jeden Eingabeparameter einer InterOperation mittels der Relation `hasParameter` auf eine Instanz des Konzepts *Parameter* (siehe Abbildung 3.9) verwiesen werden. Die Summe aller einer InterOperation zugeordneten Parameter bestimmt den Definitionsbereich der InterOperation.

In jeder Instanz des Konzepts *Parameter* wird über die Relation `partName` der Parametername innerhalb der Signaturbeschreibung der InterOperation spezifiziert, auf den sich die über die Relation `contentFromScale` angegebene Skala bezieht. Somit bindet eine Instanz von *Parameter* die Werte einer Quellskala an einen spezifischen Eingabeparameter der Operation. Da über die so referenzierte Quellskala über deren Relation `hasAspect` der zugehörige Aspekt

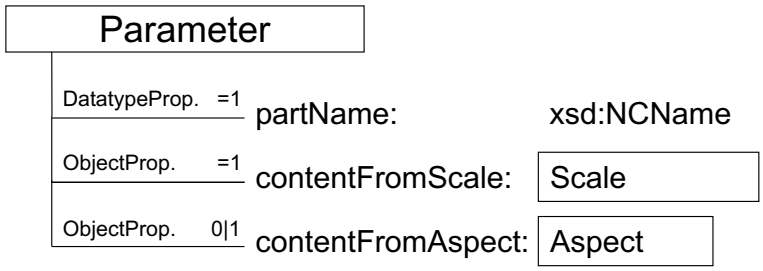

Abbildung 3.9: Das Konzept des Parameters

ermittelt werden kann, ist die Angabe eines Aspekts über die Relation `contentFromAspect` in der Parameterbeschreibung optional.

Für beide Formen der Abbildung von Skalen existiert *keine* a-priori Aussage, ob die Abbildung injektiv ist (in dem Fall würde jedes Element der Ziel-Skala genau einem Element der ursprünglichen Skala zugeordnet) oder surjektiv (in diesem Fall würde jedes Element der Ziel-Skala durch eine Abbildung erreicht), oder gar beides (bijektiv). So ist beispielsweise eine InterOperation, welche die Elemente einer „ComfortClassScale“ = {economy, business, first} auf die Elemente einer „BookingClassScale“ = {S,W,V,Q,H,M,B,Y,Z,D,C} abbildet, weder surjektiv noch injektiv, die umgekehrte Abbildung von Elementen der „BookingClassScale“ auf Elemente der „ComfortClassScale“ jedoch wenigstens surjektiv. Abbildungen, die nicht mindestens injektiv sind, haben in der Regel Auswirkungen auf die Qualität der Information, die durch abgebildete Kontextinformationen ausgedrückt wird.

Ein weiteres der im ASC-Modell differenzierten Unterkonzepte der Operation ist die *MetricOperation*. Eine MetricOperation verweist auf eine Operation, die in der Lage ist, die Elemente einer Skala zu ordnen. Die Notwendigkeit dieses Konzepts ergibt sich aus der Tatsache, daß Skalen als ungeordnete Menge modelliert sind. Wenn man bei skalaren Kontextinformationen noch geneigt ist, von einer natürlichen, inhärenten Ordnung auszugehen, so gibt es beispielsweise bei einer Skala von Farbwerten wie „rot“, „grün“ oder „blau“ auch inhärent keine natürliche Ordnung. Durch Existenz und Anwendung einer MetricOperation auf eine Skala entsteht aus der Skala im Modell eine Metrik.

Die mit einer MetricOperation spezifizierte Methode gibt von je zwei Elementen der gleichen Skala dasjenige Element zurück, welches im Sinne der gewünschten Anordnung kleiner/größer/besser/billiger/etc. ist. Die Aufgabe einer MetricOperation ist damit ähnlich der von *LDAPv3 MatchingRules* [169] oder der einer `java.lang.Comparable` Interface-Implementierung des Java API. Da die MetricOperation nicht zwingend einer einzigen Skala zugeordnet ist, kann über die Relation `onScale` spezifiziert werden, für welche Skala die MetricOperation gilt.

Durch die Anwendung einer MetricOperation ist insbesondere auch der Vergleich bzw. eine spezifische Anordnung von *nicht skalaren* Kontextinformationstypen möglich. Auf diese Weise können z.B. Kontextinformationen, die Entitäten in Form von Gauss-Krüger-Koordinaten der geografischen Position beschreiben, im Sinne des ansteigenden Abstands zu einer gegebenen

Referenzposition geordnet werden. Die Möglichkeit der Relativierung nicht-skalarer Kontextinformationen zu anderen Kontextinformationen des gleichen Typs zeichnet das ASC-Modell gegenüber anderen Kontextmodellen besonders aus.

Aspekte

Hauptaufgabe eines Aspekts (siehe Abbildung 3.10) ist die Aggregation semantisch äquivalenter Skalen. Semantisch äquivalent bedeutet, daß alle Werte oder Symbole der in einem Aspekt zusammengefaßten Skalen den Zustand einer Entität bezüglich der gleichen inhaltlichen Fragestellung charakterisieren. So sind beispielsweise die dem Aspekt `GeographicPlaceAspect` zugeordneten `WGS84_Scale` und `GK_Scale` des vorangegangenen Beispiels deswegen als semantisch äquivalent anzusehen, weil die in ihnen enthaltenen Werte trotz unterschiedlicher zugrundeliegender Koordinatensysteme den Zustand einer Entität in Form einer geografischen Positionsinformation beschreiben. Weitere Beispiele für das Konzept Aspekt sind neben dem *geometrischen Ort (GeometricPlaceAspect)* auch die *Geschwindigkeit (SpeedAspect)* oder der *Abstand (SpatialDistanceAspect)*.

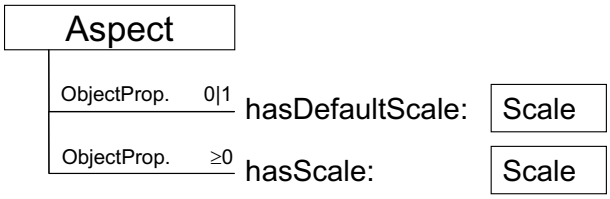

Abbildung 3.10: Das Konzept des Aspekts

Für je zwei dem gleichen Aspekt zugeordnete Skalen S_1 und S_2 gilt im ASC-Modell die Nebenbedingung, daß jeder Wert bzw. jedes Symbol der Skala S_1 auf einen Wert bzw. Symbol der Skala S_2 abgebildet werden kann. Aus mathematischer Sicht wird weder Surjektivität noch Injektivität der Abbildung von S_1 auf S_2 gefordert, lediglich Totalität, d.h. jedem Element aus S_1 ist ein Wert aus S_2 zugeordnet. Die Abbildung wird im Modell über die an anderer Stelle in diesem Abschnitt eingeführten *IntraOperationen* modelliert. Diese Nebenbedingung ermöglicht den Zugriff auf die Werte einer Skala von jeder der anderen $n-1$ Skalen eines Aspekts mit insgesamt n Skalen durch die Ausführung einer Sequenz von IntraOperationen. Durch den gerichteten Charakter der Operationen macht diese Nebenbedingung die Definition von mindestens n IntraOperationen (Ring-Konstellation, siehe Abbildung 3.11) erforderlich. Sofern eine entsprechend höhere Anzahl von verfügbaren IntraOperationen existiert, können weitere Übergänge zwischen den Skalen eines Aspekts bis zur Vollvermaschung mit $n*(n-1)$ IntraOperationen spezifiziert werden, siehe Abbildung 3.12. Letzteres kann Vorteile zur Laufzeit haben, denn bei Ring-Konstellation mit n spezifizierten IntraOperationen ist im schlechtesten Fall eine Sequenz von $n-1$ IntraOperationen auszuführen, bei Vollvermaschung mit $n*(n-1)$ spezifizierten IntraOperationen immer nur *eine* IntraOperation. Beim Systemdesign ist daher der erhöhte Programmieraufwand zur Bereitstellung der IntraOperationen gegen den Zeitgewinn von einer im Mittel geringeren Anzahl von Berechnungsschritten zur Laufzeit individuell abzuschätzen.

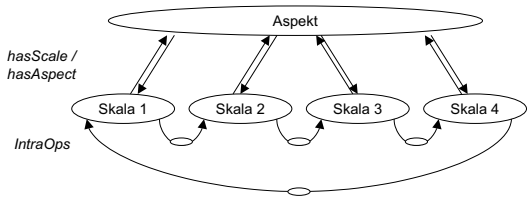

Abbildung 3.11: Ring-Netz

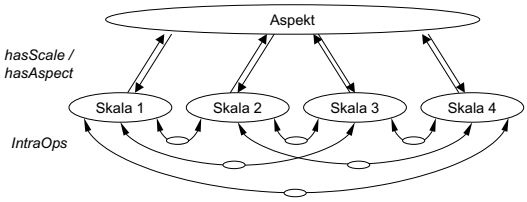

Abbildung 3.12: Vollvermaschtes Netz

In verteilten Systemen, insbesondere in denen zur kontextadaptiven Dienstnutzung, unterliegt nicht nur der Kontext einer Dienstinteraktion gemäß Definition 7 (d.h. die Menge der relevanten Kontextinformationen) einer gewissen Dynamik, sondern auch die Menge der zu berücksichtigenden *Typen* von Kontextinformationen. Dem wird im ASC-Modell u.a. dadurch Rechnung getragen, daß die Menge der zugehörigen Skalen eines Aspekt zunächst auch leer sein darf (Kardinalität=0, siehe Abbildung 3.10). Diese sogenannten *abstrakten Aspekte* erhalten erst zu einem späteren Zeitpunkt, möglicherweise durch einen anderen Autor, eine oder mehrere Skalen zugewiesen. Hier findet das besondere Merkmal der verteilten Komposition von Ontologien Verwendung.

Aus der Menge der Skalen eines Aspekts kann eine als Default-Skala ausgezeichnet sein, so daß bei der späteren Verwendung eines Aspekts nicht immer eine Skala angegeben werden muß.

Skalen

Das Konzept der Skala (siehe Abbildung 3.13) dient vornehmlich der Formalisierung des Wertebereichs von Kontexinformationen. Jede Skala aggregiert Kontextinformationen genau eines Typs. Dabei wird die Aggregationsbeziehung zwischen einer Skala und einer Kontextinformation durch die Relation `hasMember` der Skala ausgedrückt.

Eine Nebenbedingung des ASC-Modells ist, daß alle Elemente einer Skala auf Instanzebene vom gleichen „Typ“ sind. Dies wird in Abbildung 3.6 auf Seite 71 durch eine einwertige Relation zwischen Skala und Kontextinformation auf Konzeptebene angezeigt, im Unterschied zu einer mehrwertigen Relation auf Instanzebene. Formal ist diese Nebenbedingung gleichbedeutend mit der Forderung, daß alle Instanzen von Kontextinformationen, die der gleichen Instanz einer Skala zugeordnet werden, von einem gemeinsamen Unterkonzept 1. Ordnung des Basiskonzepts `ContextInformation` subsummiert werden. In diesem Zusammenhang werden Kontextinformationen, die gemäß dieser Nebenbedingung einer Instanz einer Skala zugewiesen werden könnten, als *wohlgeformte* Kontextinformationen bezeichnet. Dem gegenüber stehen die wohlgeformten Kontextinformationen, die einer Instanz einer Skala bereits zugewiesen sind, weshalb sie bezüglich dieser Skala als *gültig* bezeichnet werden.

Bei den meisten anderen Modellen zur Beschreibung des Kontextes wird die Einschränkung des Wertebereichs - wenn überhaupt - durch Typisierung vorgenommen, d.h. es erfolgt eine Einschränkung auf einen Datentyp wie Integer oder String. Wird diese Beschreibung mit

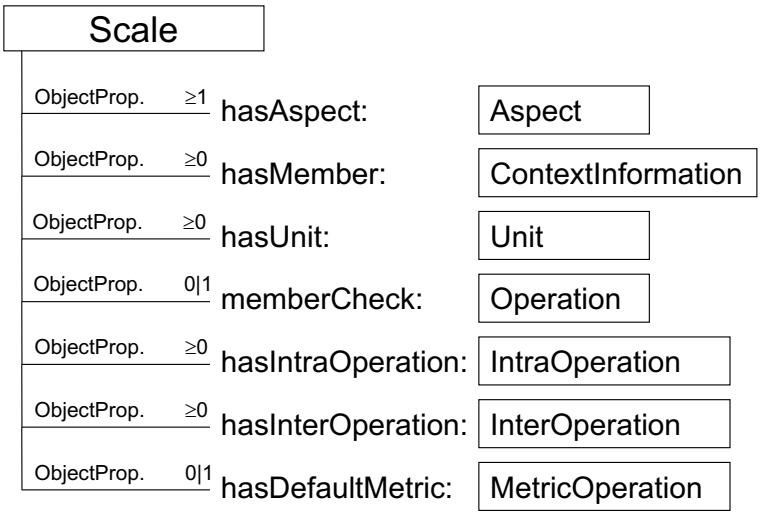

Abbildung 3.13: Das Konzept der Skala

ausschließlich Typenbeschränkungen angewandt, z.B. im Rahmen der Dienstsuche mit den in Abschnitt 2.4.3 vorgestellten Protokollen, kann nicht zwischen Wohlgeformtheit und Gültigkeit wie im Falle des ASC-Modells unterschieden werden. In der Regel ist dann bei textbasierten Beschreibungen nur eine Suche nach Schlüsselwörtern möglich, ohne daß die Menge der zulässigen Schlüsselwörter geschlossen spezifiziert ist.

Auf Instanzebene kann die Menge der aggregierten Kontextinformationen eine beliebige Mächtigkeit aufweisen, d.h. insbesondere auch leer oder sehr groß sein. Im Fall sehr großer Mächtigkeit steigen die Performanceanforderungen an einen Reasoner mindestens linear an, was bei der Auswahl der Ausführungsplattform unbedingt berücksichtigt werden muß.

Im ASC-Modell ist nicht festgelegt, wie die Elemente einer Skalen-Instanz zugeordnet werden. Dies kann durch Aufzählung der Elemente im Sinne einer *Enumeration* geschehen, wenn es sich um eine kleinere, aber vor allem endliche Anzahl von Elementen handelt wie z.B. die möglichen Ausprägungen *first*, *business* und *economy* einer *ComfortClassScale*. Ist die Anzahl der Elemente einer Skala jedoch sehr groß oder gar unendlich, obliegt es der für das zugrundeliegenden Unterkonzept der Skala zuständigen *context management implementation*, wie die Menge „gefüllt“ wird, also z.B. im Rahmen der Instantiierung der Skala. Diese ist gemäß MNMplusCE Dienstmodell (siehe Abschnitt 2.3.3) der *context provider domain* zuzuordnen. Ein Beispiel hierfür ist eine Skala *TemporalDurationScale*, welche alle möglichen Ausprägungen der Dauer einer Dienstnutzung in Sekunden als Wertebereich spezifiziert und damit auf die abzählbar unendliche Menge $\mathbb{N}$ der Natürlichen Zahlen abgebildet werden kann. Zur Verdeutlichung wird im Rahmen dieser Arbeit nach einer *direkten* Skala im ersten Fall und einer *generierten* Skala im zweiten Fall unterschieden. Bei generierten Skalen steht zur Überprüfung, ob eine bestimmte Kontextinformation Element der durch diese Skala ausgedrückten Menge ist, die Operation `memberCheck` zur Verfügung, da eine Überprüfung über die Relation `hasMember` einer linearen Suche entspricht, die sehr lange - bei einer unendlich

großen Menge in der Tendenz unendlich lange - dauert. Die Operation `memberCheck` erwartet immer eine Instanz des Typs der aggregierten Kontextinformationen als einzigen Eingabeparameter und liefert `true` oder `false` entsprechend der Zugehörigkeit der Menge. Da es sich bei der Operation `memberCheck` um eine spezifische Implementierung der Überprüfungsroutine handelt, kann diese auf Zusatzwissen über die Art der Skala zugreifen - beispielsweise ihren skalaren, geordneten Charakter - und die Überprüfung anhand einer Schwellwertentscheidung durchführen. Hierbei kann sowohl auf eine geeignete MetricOperation zur Herstellung einer spezifischen Ordnung auf der Skala, als auch auf eigene Relationen zur Bereichsidentifizierung zurückgegriffen werden.

Eine Skala kann auf vier unterschiedliche Arten konstruiert werden:

1. explizit direkt durch aufzählende Zuweisung
2. explizit generiert bei der Instantiierung
3. implizit generiert durch Angabe einer IntraOperation
4. implizit generiert durch Angabe einer InterOperation

Die Arten 3 und 4 erlauben die virtuelle Konstruktion einer neuen Skala als Abbildung einer (per IntraOperation) oder auch mehrerer (per InterOperation) anderer Skalen. Dies erlaubt z.B. die Erzeugung verschiedener Skalen eines Aspekts durch Angabe von IntraOperationen, die unterschiedliche Skalierungsfaktoren o.ä. repräsentieren („nautical miles“, „km“ oder „m“ für einen Aspekt „SpatialDistanceAspect“). Implizit erzeugte Skalen werden deswegen als virtuell bezeichnet, da die zugehörige Intra- oder InterOperation nur einzelne Werte des jeweiligen Definitionsbereichs (Quell-Skalen) auf den Wertebereich (Ziel-Skala) abbildet. Die Umwandlung einer impliziten Skala in eine explizite Skala durch vollständige Abbildung, d.h. explizite Anwendung der Operation auf alle Elemente der Quell-Skala, ist nicht möglich, wenn die Quell-Skala unendlich viele Elemente beinhaltet, und oft auch nicht sinnvoll, wenn die Quell-Skala nur endlich viele Elemente enthält. Oft reicht aber auch die explizite Anwendung der Operation auf einzelne *relevante* Elemente aus (z.B. um den räumlichen Abstand weniger Objekte auf einer impliziten Entfernungsskala auszudrücken). In diesem Fall kann dann auch die `hasMember` Relation einer (expliziten) Skala zu einer Kontextinformation ausgewertet werden, was für den Fall von impliziten Skalen nur möglich ist, sofern die generierende Operation injektiv und eine Umkehroperation verfügbar ist.

Durch die Verwendung von InterOperationen können auch neue Skalen erzeugt werden, die auf eine oder mehrere Skalen aus anderen Aspekten zurückgreifen. Ein Beispiel hierfür ist eine Skala „KilometerPerHourScale“, die einem Aspekt „SpeedAspect“ zugeordnet ist. Die Werte einer solchen Skala können mit Hilfe einer InterOperation virtuell erzeugt werden, indem diese die Werte aus zwei Parametern, delta_s aus dem Aspekt „SpatialDistanceAspect“ und delta_t aus dem Aspekt „DurationAspect“, in einfacher Weise berechnet.

Wie durch die Verwendung von Intra- und InterOperationen ein ganzes Netzwerk von ineinander überführbaren Skalen aufgebaut werden kann und welche Informationen noch daraus abgeleitet werden können, wird in Abschnitt 4.5 beschrieben.

Eine Skala ist zwar üblicherweise einem einzelnen Aspekt über die Relation `hasAspect` der

Skala zugeordnet, kann aber im Bedarfsfall auch mehreren Aspekten zugeordnet werden. Die `hasAspect`-Relation der Skala ist somit invers zur `hasScale`-Relation des Aspekts definiert.

Jede Skala sollte eine Angabe über die Einheit (z.B. „Liter“ bei einer „VolumeScale“) der darin aggregierten Werte machen. Diese kann über die Relation `hasUnit` zugewiesen werden. Da gelegentlich alternative Einheiten-Systeme in der Realität Verwendung finden (z.B. zusätzlich „dm^3“ bei der eben genannten „VolumeScale“), können über diese Relation auch mehrere Einheitenbezeichner der gleichen Skala zugewiesen werden.

Jede Skala stellt eine ungeordnete Menge von Kontextinformationen dar. Bei der Anwendung der Skalen, z.B. im Reasoningprozeß, ist jedoch meistens eine bestimmte Anordnung der Elemente der Menge von besonderem Interesse. Diese Anordnung kann durch Anwendung der bereits erwähnten MetricOperation erreicht werden. Um zu kennzeichnen, daß es bezüglich einer Skala bereits eine Standardanordnung gibt, kann über die Relation `hasDefaultMetric` auf eine MetricOperation verwiesen werden, die immer dann Verwendung findet, wenn keine andere MetricOperation explizit angegeben wird.

Kontextinformationen

Der Zustand einer Entität bezüglich eines spezifischen Aspekts wird mit Hilfe des Konzepts der Kontextinformation (siehe Abbildung 3.14) modelliert. Dieses Konzept leitet sich wieder stark aus der Analyse des Kontextbegriffs in Abschnitt 2.3.1, und hier insbesondere Definition 6, ab.

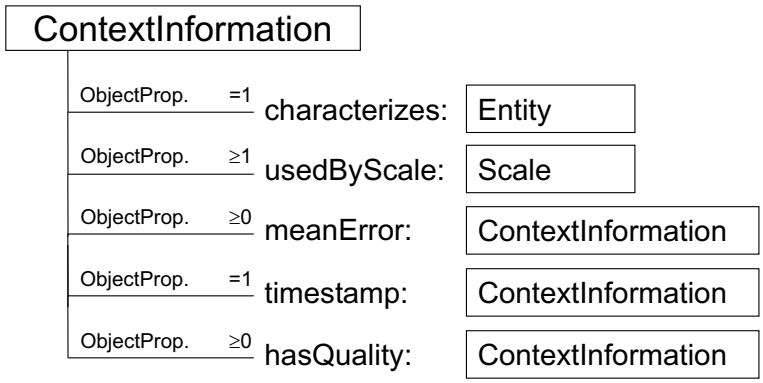

Abbildung 3.14: Das Konzept der Kontextinformation

Der Bezug zur charakterisierten Entität erfolgt über die Relation `characterizes` einer Kontextinformation. Bereits aus der Definition der Kontextinformation ergibt sich, daß jede Kontextinformation genau eine Entität beschreibt.

Eine Kontextinformation ist zwar üblicherweise einer einzelnen Skala über die Relation `usedByScale` zugeordnet, kann aber im Bedarfsfall auch mehreren Skalen zugeordnet werden. Die `usedByScale`-Relation der Kontextinformation ist invers zur `hasMember`-Relation der Skala.

Um alle Arten von Kontextinformationen im ASC-Modell auf die gleiche Weise modellieren zu können, werden auch skalare Datentypen und Zeichenketten durch entsprechende Wrapper-Konzepte ähnlich `Integer::ContextInformation` bzw. `String::ContextInformation` repräsentiert. Eine Objektinstanz, die analog zu `new Integer(10)` erzeugt wurde, kann somit als Kontextinformation beispielsweise einer der Skalen „MeterScale“ oder „KilometerScale“ eines Aspekts „SpatialDistanceAspect“ zugewiesen werden.

Kontextinformationen können selbst auch als Entität gemäß Definition 2 aufgefaßt werden, wodurch sie durch andere Kontextinformationen im Sinne der in Abschnitt 2.3.1 eingeführten Meta-Kontextinformationen charakterisiert sind. Die Meta-Kontextinformationen sind sozusagen Qualitätsangaben über die Kontextinformation, und können direkt über die Relation `hasQuality` an eine Kontextinformation „angeheftet“ werden. So angeheftete Qualitätsinformationen beschreiben die Kontextinformation in freier semantischer Ausrichtung, z.B. bezüglich des Maßes an Sicherheit, mit der die Quelle der Kontextinformation vom Wahrheitsgehalt der Aussage überzeugt ist. Auch ist die Angabe mehrer unterschiedlicher Qualitätsinformationen möglich, welche die Güte der gleichen Kontextinformation bzw. der zugrundeliegenden Informationsquelle beschreiben. Dazu können bespielsweise Kenngrößen zur Beschreibung von Quantisierungsfehlern, Signalstörabstand oder gar komplexe Fehlerverteilungen wie die in [26] vorgestellten herangezogen werden.

Ein wichtiges Kriterium bezüglich der Güte einer Kontextinformation ist in der Regel die *Aktualität* der Information. Diese Größe ist in fast allen Kontextmodellen spezifiziert, insbesondere in solchen, die in Systemen verwendet werden, bei denen die Relevanz einer Kontextinformation inhärent auf die örtliche und zeitliche Nähe abgebildet wird. Die in Abschnitt 2.4.3 vorgestellten Service Frameworks *Centaurus*, *Ninja* und *Capeus* haben beispielsweise ein solches (eingeschränktes) Verständnis des Relevanzbegriffs. Auch im ASC-Modell hat die Aktualität einer Kontextinformation einen besonders hohen Stellenwert, wenn auch ihre Verwendung im Rahmen der Spezifikation der Relevanzkriterien gleichwertig zu der anderer Arten von Kontextinformationen ist. Das Konzept der Kontextinformation verfügt daher im ASC-Modell über die Relation `timestamp`, über die eine absolute Zeitangabe erfolgen kann, zu der die Kontextinformation gültig war.

Sehr häufig wird auch die Qualität einer Kontextinformation bezüglich der *Genauigkeit* benötigt. Als Genauigkeit einer geografischen Positionsinformation wird beispielsweise gerne die Genauigkeit des verwendeten Positionsbestimmungsverfahrens zugrundegelegt. Diese wird z.B. bei GPS als *Dilution of Precision (DOP)* [179] oder als mittlerer Fehler in Metern angegeben. Die Interpretation, was unter Genauigkeit zu verstehen ist, variiert je nach Aspekt, dem eine Kontextinformation über eine Skala zugeordnet ist. Oft ist es so, daß die Genauigkeit auf einen mittleren Fehlerwert abgebildet werden kann. Aus diesem Grund sieht das ASC-Modell im Konzept der Kontextinformation bereits einen Verweis auf entsprechende Qualitätsinformationen über die Relation `meanError` vor. Diese Relation kann, muß aber nicht zur Beschreibung der Qualität der Kontextinformation herangezogen werden. Ihre Verwendung bietet sich aber u.a. deswegen an, weil der mittlere Fehler in der Regel in den gleichen Einheiten wie die Werte der Skala der Kontextinformation selbst angegeben wird.

Da alle Qualitätsinformationen ebenfalls als Kontextinformationen modelliert sind, gelten die gleichen Reasoning-Möglichkeiten wie für alle anderen Kontextinformationen erster Ord-

nung, insbesondere der Bezug zu einer Skala und deren Bezug zu einem Aspekt. Auf die Qualitätsinformationen kann dadurch unmittelbar bei der Formulierung der Relevanzkriterien zurückgegriffen werden, die im weiteren Verlauf dieser Arbeit noch detaillierter vorgestellt werden. Sie erlauben Aussagen der Art "Berücksichtige nur die Kontextinformationen, deren mittlerer Fehler unter dem Schwellwert $E_{Schwelle}$ liegt".

3.3 Umsetzung des Modells als Context Ontology Language

Zur Spezifikation von kontextuellem Wissen – und damit insbesondere zur Beschreibung und Feststellung kontextueller Service-Interoperabilität – muß das im vorhergehenden Abschnitt vorgestellte ASC-Modell in eine Beschreibungssprache umgesetzt werden. Eine Möglichkeit der Umsetzung ist die im Folgenden vorgestellte *Context Ontology Language (CoOL)*.

Da das Modell auf der Basis ontologischer Konzepte aufgebaut ist, liegt es nahe, das Modell in einer Ontologiesprache zu definieren. Unterschiedliche Ontologiesprachen haben, wie in Abschnitt 3.1.2 erläutert, unterschiedliche Mächtigkeiten bezüglich ihrer Ausdrucksstärke aufgrund der verschiedenen zugrundeliegenden Logiken. Sie ermöglichen die automatisierte Überprüfung der von einer bestehenden Ontologie abgeleiteten Unterkonzepte und den darauf basierenden Fakten. Diese Überprüfung erfolgt mittels eines sogenannten *Validators*, der die Korrektheit bezüglich einer Definition der Sprache feststellen kann. Darüber hinaus ist ein *Reasoner* in der Lage, in der Spezifikation ontologischer Konzepte und Fakten implizit enthaltenes Wissen durch Auswertung von Regeln explizit zu machen. So gesehen stellt die Wahl der Ontologiesprache die Weichen für die Mächtigkeit und Anwendbarkeit des unter Verwendung des ASC-Modells spezifizierten Wissens.

Das ASC-Modell und das darauf aufbauend spezifizierte Wissen ist jedoch nicht isoliert zu betrachten. So erfordert z.B. die Anwendung des so modellierten Wissens in einem Service Framework die Integration des Modells in vorhandene Strukturen und Protokolle, um beispielsweise auszudrücken, daß ein Parameter p eines Methodenaufrufs einem bestimmten Aspekt A zuzuordnen ist. Darüber hinaus können auch von den Basiskonzepten abgeleitete Unterkonzepte als Teil der Sprache gesehen werden, die gerade dadurch einen hohen Grad an Wiederverwertbarkeit erlangen.

Aufgrund dieser Zusammenhänge wird unter der in diesem Kapitel vorgestellten Context Ontology Language nicht eine einzelne Sprache verstanden, sondern eine Art „Baukasten“ mit verschiedenen „Werkzeugen“ zur Beschreibung von kontextuellem Wissen und verschiedenen „Plug-Ins“ zur Integration in verschiedene Anwendungsumgebungen.

Zur besseren Unterscheidung wird der Teil der Sprache, in dem das ASC-Modell selbst definiert wird, als *CoOL Core* bezeichnet. Die übrigen Fragmente (z.B. ASC-spezifische Erweiterungen der Web Service Protokolle SOAP und WSDL, häufig genutze Unterkonzepte von Aspekt, Skala und Kontextinformation etc.) werden dem Teil der Sprache zugeordnet, der als *CoOL Integration* bezeichnet wird. Diese Trennung dient allerdings nur der Übersichtlichkeit und hat keine weitere Bedeutung in der Interpretation des Modells.

CoOL Core wurde im Rahmen dieser Arbeit in drei verschiedenen Ontologiesprachen umgesetzt, um die Vor- und Nachteile der jeweiligen Sprache zu analysieren. Bei diesen Ontologiesprachen handelt es sich um DAML+OIL, OWL-DL und F-Logic. Der Rest dieses Abschnitts gliedert sich wie folgt: In Abschnitt 3.3.1 wird zunächst die Umsetzung in DAML+OIL bzw. OWL-DL vorgestellt, gefolgt von der Umsetzung des ASC-Modells in F-Logic in Abschnitt 3.3.2. Beide Varianten werden anschließend in Abschnitt 3.3.3 gegenübergestellt und bezüglich ihrer jeweiligen Vor- und Nachteile miteinander verglichen.

3.3.1 Umsetzung in OWL-DL und DAML+OIL

Da die Differenz zwischen DAML+OIL und OWL Description Logic (OWL-DL) aufgrund der in Abschnitt 3.1.2 geschilderten Zusammenhänge sehr gering ist, wird in diesem Abschnitt primär die Umsetzung in OWL-DL vorgestellt, und nur dort auf die Unterschiede zur Umsetzung in DAML+OIL hingewiesen, wo diese auch Gewicht haben.

Beide Ontologiesprachen sind dem Umfeld des Semantic Web zuzuordnen und basieren auf dem Ressource Description Framework (RDF) und dessen Erweiterung RDF Schema (RDF-S). Abbildung 3.15 zeigt die daraus resultierende Einordnung der Context Ontology Language in den Semantic Web Stack.

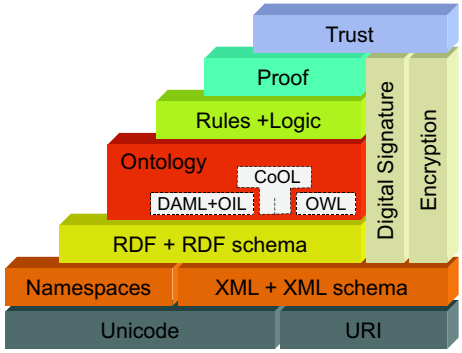

Abbildung 3.15: Einordnung von CoOL im Semantic Web Stack

Für RDF/S sind unterschiedliche Darstellungsformen definiert, z.B. die Darstellung als Graph. Die gebräuchlichste Darstellungsform ist jedoch die Serialisierung in XML, die auch im Rahmen dieser Arbeit verwendet wird. Daraus ergibt sich, daß zur eigentlichen Umsetzung des ASC-Modells ein „XML- und RDF/S-Rahmen“ mit Headerinformationen wie Angaben zu verschiedenen Namensräumen und Versionsinformationen gehört, auf den an dieser Stelle nicht näher eingegangen wird.

Die Repräsentation ontologischer Konzepte erfolgt in OWL als Klasse (`owl:Class`). Daher werden alle Basiskonzepte wie Aspekt, Skala etc. in OWL als Klasse definiert. Abbildung 3.16 zeigt dies am Beispiel der Definition der Klasse `Aspect`. Die Umsetzung der Relationen des Konzepts erfolgt wie in OWL üblich als Eigenschaft (`owl:ObjectProperty`) der Klasse, für die als Definitionsbereich die zugehörige Klasse und als Wertebereich das Zielkonzept angegeben wird.

Mit Hilfe von Kardinalitätsangaben können obere und untere Schranken für die Anzahl erlaubter Vorkommen bestimmter Relationen auf Instanzebene angegeben werden. Zum Beispiel macht es keinen Sinn, wenn mehr als eine Skala pro Aspekt als Default-Skala ausgezeichnet wird. Die Festlegung, daß es sich bei `hasDefaultScale` um ein `rdfs:subPropertyOf` von `hasScale` handelt, erlaubt die Angabe einer Default-Skala, ohne daß diese zusätzlich noch einmal mit `hasScale` als Skala angegeben werden muß, da alle Instanzen der Relation `hasDefaultScale` automatisch im Reasoner auch als Instanzen der Relation `hasScale` er-

```
<owl:Class rdf:ID="Aspect">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasDefaultScale"/>
      <owl:maxCardinality
        rdf:datatype="&xsd;nonNegativeInteger">1</owl:maxCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<owl:ObjectProperty rdf:ID="hasScale">
  <owl:inverseOf rdf:resource="#hasAspect"/>
  <rdfs:domain rdf:resource="#Aspect"/>
  <rdfs:range rdf:resource="#Scale"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="hasDefaultScale">
  <rdfs:subPropertyOf rdf:resource="#hasScale"/>
</owl:ObjectProperty>
```

Abbildung 3.16: Spezifikation des Aspekts in OWL-DL

kannt werden. Auf ähnliche Weise wie im Beispiel der Aspekte erfolgt auch die Definition der übrigen Klassen wie `Scale`, `ContextInformation` etc. Bereits an diesem einfachen Beispiel wird die für XML Serialisierungen typische „Geschwätzigkeit“ deutlich, weshalb in diesem Abschnitt auf eine vollständige Darstellung der Umsetzung des Modells verzichtet wird.

```
<owl:ObjectProperty rdf:ID="hasOperation">
  <rdfs:range rdf:resource="#Operation"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="memberCheck">
  <rdfs:subPropertyOf rdf:resource="#hasOperation"/>
  <rdfs:domain rdf:resource="#Scale"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="hasIntraOperation">
  <rdfs:subPropertyOf rdf:resource="#hasOperation"/>
  <rdfs:domain rdf:resource="#Scale"/>
  <rdfs:range rdf:resource="#IntraOperation"/>
</owl:ObjectProperty>     <!-- usw. -->
```

Abbildung 3.17: Vererbungsbeziehung der Operationen in OWL-DL

Wie hierbei nicht nur die Vererbungsbeziehung bei Konzepten, sondern auch von Relationen genutzt wird, zeigt das Beispiel der Relationen `hasIntraOperation`, `hasDefaultMetric` etc. in Abbildung 3.17. Ausgedrückt wird diese Vererbungsbeziehung von Relationen ebenfalls durch das Sprachelement `rdfs:subPropertyOf`, welches somit die Entsprechung zu

`rdfs:subClassOf` bei Klassen ist.

```
<owl:Class rdf:ID="Integer">
  <rdfs:subClassOf rdf:resource="#ContextInformation"/>
</owl:Class>

<owl:DatatypeProperty rdf:ID="hasIntegerValue">
  <rdfs:domain rdf:resource="#Integer"/>
  <rdf:range rdf:resource="xsd:integer"/>
</owl:DatatypeProperty>

<Integer> <!-- anonymous instance, assigns itself to a scale -->
  <hasIntegerValue rdf:datatype="xsd:integer">41</hasIntegerValue>
  <characterizes rdf:resource="#SomePersonEntity"/>
  <usedByScale rdf:resource="#MeterScaleOfDistanceAspect"/>   <!-- ... -->
<Integer/>
```

Abbildung 3.18: Spezifikation und Anwendung von Wrapperklassen in OWL-DL

Um auch skalare Datentypen und Zeichenketten im Reasoner wie Kontextinformationen behandeln zu können, wurden Wrapperklassen für diese Datentypen definiert. Abbildung 3.18 skizziert diese Vorgehensweise anhand eines einfachen Beispiels.

```
<owl:Class rdf:ID="WGS84Scale">
  <rdfs:subClassOf rdf:resource="http://foo.org#someOtherConcept"/>
  <rdfs:subClassOf rdf:resource="#Scale"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasMember"/>
      <owl:allValuesFrom rdf:resource="#WGS84ContextInformation"/>
      <!-- daml:toClass would be used here in DAML+OIL  -->
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="WGS84ContextInformation">
  <rdfs:subClassOf rdf:resource="#ContextInformation"/>
</owl:Class>
```

Abbildung 3.19: Spezifikation von Unterkonzepten in OWL-DL

Weiterhin wurden, wie erwähnt, zu Evaluierungszwecken bereits einige individuelle Aspekte, Skalen usw. als Unterkonzepte definiert, die als Bestandteil von *CoOL Integration* die Menge der Basiskonzepte ergänzen und darüber hinaus ihre Anwendungmöglichkeiten beispielhaft zeigen. Da die Basiskonzepte ihre Eigenschaften und insbesondere auch ihre semantische Bedeutung an die Unterkonzepte vererben, haben auch alle Unterkonzepte die entsprechenden Bedeutungen aus dem ASC-Modell während des Reasoningprozesses. Durch Mehrfachverer-

bung können den Unterkonzepten jedoch weitere Eigenschaften und Bedeutungen hinzugefügt, bzw. die vorhandenen eingeschränkt werden, um das Wissen, das damit ausgedrückt werden soll, auf formaler Ebene zu beschreiben und für einen Reasoner zugänglich zu machen.

Abbildung 3.19 stellt dies am Beispiel des Unterkonzepts `WGS84Scale` ausschnittsweise dar, wo diese Klasse einerseits als Unterklasse der Basisklasse `Scale` und andererseits einer weiteren Klasse definiert wird, deren Definition außerhalb des aktuellen Namespaces liegt. Außerdem wird in diesem Beispiel der Wertebereich der Eigenschaft `hasMember` auf Instanzen festgelegt, die von der Klasse `WGS84ContextInformation` instantiiert wurden. Um in DAML+OIL eine dem Element `owl:allValuesFrom` vergleichbare Einschränkung des Wertebereichs der `hasMember` Relation auf Instanzen einer spezifischen Unterklasse von `ContextInformation` vorzunehmen, ist dort das Element `daml:toClass` zu verwenden.

Durch die Definitionen auf Konzeptebene wird festgelegt, über welche Eigenschaften eine Klasse verfügt. Dies beinhaltet natürlich auch die Beziehungen zwischen den einzelnen Instanzen. Durch die Verwendung von Klassifikationselementen der zugrundeliegenden Sprache OWL-DL bzw. DAML+OIL, die einer Klasse oder einer Eigenschaft eine bestimmte semantische Bedeutung zuweist (z.B. `inverseOf`, `subClassOf`, `subPropertyOf` etc.), wird das implizite Wissen explizit und kann von einem Reasoner entsprechend ausgewertet werden. So ist ein Reasoner beispielsweise in der Lage, die in Abbildung 3.20 skizzierten Fakten der Instanzebene dahingehend auszuwerten, daß eine Anfrage über die Property `hasMember` der Skala `WGS84ScaleInst` die angegebene anonyme `WGS84ContextInformation` liefert, obwohl diese nicht über `hasMember` der Skala hinzugefügt wurde. Dies wird durch die Definition der Property `hasMember` der Skala als `inverseOf` der Property `usedByScale` einer Kontextinformation erreicht.

```
<WGS84Scale rdf:ID="WGS84ScaleInst">
 <hasAspect rdf:resource="#GeometricPlaceAspectInst"/>
 <hasUnit rdf:resource="#WGS84UnitInst"/>   <!-- ... -->
</WGS84Scale>

<Unit rdf:ID="WGS84UnitInst">
  <rdfs:label>Lat Lon Alt</rdfs:label>
  <rdfs:comment>Latitude and Longitude in Degree, Altitude in m</rdfs:comment>
</Unit>

<WGS84ContextInformation> <!-- anonymous instance -->
  <characterizes rdf:resource="#SomePersonEntity"/>
  <usedByScale rdf:resource="#WGS84ScaleInst"/>
  <timestamp rdf:resource="#SomeTimestampContextInformation"/>   <!-- ... -->
</WGS84ContextInformation>
```

Abbildung 3.20: Spezifikation von Fakten in CoOL-OWL

Um die verteilte Komposition, ein besonderes Merkmal von Ontologien, von auf dem ASC-Modell basierendem Wissen zu erleichtern, wurde – sofern möglich und sinnvoll – auf die Angabe einer unteren Schranke für Kardinalitäten (`owl:minCardinality`) verzichtet. Dies

ermöglicht z.B. die zeitlich und netztopologisch versetzte Spezifikation eines Aspekts und der zugehörigen Skalen.

3.3.2 Umsetzung in F-Logic

Eine alternative Umsetzung des ASC-Modells erfolgte in der ebenfalls in Abschnitt 3.1.2 vorgestellten Ontologiesprache F-Logic. Es sei insbesondere noch einmal auf die Einführung der Notation von F-Logic im genannten Abschnitt hingewiesen, die an dieser Stelle nicht noch einmal wiederholt werden soll.

Die Serialisierung von Konzepten und Relationen in F-Logic ist wesentlich kompakter im Vergleich zu XML. So kompakt, daß die im Rahmen dieses Abschnitts vorgestellten Fragmente die Umsetzung des ASC-Modells in F-Logic fast vollständig beschreiben.

Zur besseren Strukturierung und zur Gewährleistung von Überschneidungsfreiheit der Namen der Unterkonzepte und Instanzen können die jeweiligen Elemente in verschiedenen Namensräumen gruppiert werden. Obwohl eine solche Aufteilung in verschiedene Namensräume bei der Umsetzung des ASC-Modells in F-Logic vorgenommen wurde, ist bei den Beispielen in diesem Abschnitt bewußt auf diese Unterscheidung verzichtet worden, um die Beispiele auf die wesentlichen Syntax-Elemente zu beschränken.

```
#Aspect[ #hasDefaultScale=>#Scale; #hasScale=>>#Scale ].

#Scale[ #hasAspect=>>#Aspect;
        #hasMember=>#ContextInformation;
        #hasUnit=>>#Unit;
        #memberCheck=>#Operation;
        #hasIntraOperation=>>#IntraOperation;
        #hasInterOperation=>>#InterOperation ].

#ContextInformation[  #characterizes=>#Entity;
                      #usedByScale=>>#Scale;
                      #meanError=>>#ContextInformation;
                      #timestamp=>#ContextInformation;
                      #hasQuality=>>#ContextInformation ].

#Operation[ #identifiedBy=>xsd#anyURI ].
#IntraOperation::#Operation[ #fromScale=>#Scale ].
#InterOperation::#Operation[ #hasParameter=>>#Parameter ].
#MetricOperation::#Operation[ #onScale=>>#Scale ].
```

Abbildung 3.21: Spezifikation der Basiskonzepte des ASC-Modells in F-Logic

Insbesondere der objektorientierte Charakter von F-Logic kommt der Kompaktheit der Definition von Konzepten zugute, ohne daß die Übersichtlichkeit für den Ersteller dadurch verloren geht. So zeigt beispielsweise Abbildung 3.21 die Definition der Konzepte Aspekt, Skala und

Kontextinformation sowie deren Relationen in wenigen Zeilen. Wie das Beispiel der Operationen zeigt, können dabei gleichzeitig die `subconceptOf`-Relation selbst wie weitere Relationen eines Unterkonzepts angegeben werden.

Die eigentliche Mächtigkeit von F-Logic liegt aber in der Verarbeitung von Axiomen. Axiome erweitern die Möglichkeiten des automatischen Schließens – eine der charakteristischen Fähigkeiten von Ontologiesystemen – enorm. Wäre die F-Logic eine Sprache des Semantic Web, dann würde man die Axiome bereits der Ebene der *Regeln und Logik* zuordnen, die im Semantic Web Stack auf der Ebene der Ontologien aufsetzt (siehe Abbildung 3.15 auf Seite 84). Abbildung 3.22 zeigt den Einsatz von Axiomen am Beispiel der Äquivalenzen zu `rdfs:subPropertyOf` und `owl:inverseOf`, die als solche nicht Bestandteil von F-Logic sind, aber auf einfache Weise als Axiom formuliert werden können.

```
// subPropertyOf-equivalent relations
FORALL S,A (A[#hasScale->>S]) <- (A:#Aspect[#hasDefaultScale->S:#Scale]).

// inverseOf-equivalent relations
FORALL S,A (A[#hasScale->>S]) <- (S:#Scale[#hasAspect->>A:#Aspect]).
FORALL S,A (S[#hasAspect->>A]) <- (A:#Aspect[#hasScale->>S:#Scale]).
FORALL S,K (K[#usedByScale->>S]) <- (S:#Scale[#hasMember->>K:#ContextInformation]).
FORALL S,K (S[#hasMember->>K]) <- (K:#ContextInformation[#usedByScale->>S:#Scale]).
```

Abbildung 3.22: Mächtigkeit von F-Logic Axiomen

Die Spezifikation von Unterkonzepten und Fakten ist ebenso einfach, wie das Beispiel in Abbildung 3.23 zeigt. Zunächst wird durch Überladen der `hasMember` Relation des Unterkonzepts `WGS84_Scale` der Skala eine Beschränkung des Wertebereichs auf Instanzen des Unterkonzepts `WGS84_CI` der Kontextinformationen erreicht. Anschließend wird in Ausschnitten gezeigt, wie die Instantiierung und Verknüpfung von Instanzen angegeben wird.

```
// restrict the range of hasMember by overloading on concept level
#WGS84_Scale::#Scale[ #hasMember=>>#WGS84_CI::#ContextInformation ].
#GK_Scale::#Scale[ #hasMember=>>#GK_CI::#ContextInformation ].

// create aspect and scale on instance level
#GeometricPlace_AspectInst:#GeometricPlace_Aspect[
  #hasDefaultScale->#WGS84_ScaleInst;
  #hasScale->>#GK_ScaleInst
].
#WGS84_ScaleInst:#WGS84_Scale[
  #hasAspect->>#GeometricPlace_AspectInst;
  #hasIntraOperation->>#getWGS84fromGK_IntraOpInst;
  #hasInterOperation->>#getWGS84fromStreet_InterOpInst   // ...
].
```

Abbildung 3.23: Ontologische Fakten basierend auf der Umsetzung in F-Logic

Die Nebenbedingung des ASC-Modells, daß ein Wert einer beliebigen Skala eines Aspekts auf jede andere Skala des gleichen Aspekts durch eine Sequenz von IntraOperationen abgebildet werden kann, ist in F-Logic durch komplexe Axiome darstellbar, wie Abbildung 3.24 zeigt. Das erste Axiom (`reachabilityError`) aus dieser Abbildung benutzt für alle Kombinationen von je zwei verschiedenen Skalen des gleichen Aspekts die beiden anderen Axiome (`intraSeqN1` und `intraSeqNpp`) um festzustellen, ob es eine Sequenz aus IntraOperationen gibt, die eine Abbildung der Werte der einen Skala auf die andere Skala ermöglicht. Die beiden letztgenannten Axiome sind dabei induktiv aufgebaut. Beim Aufruf einer Query mit festem Aspekt (z.B. `FORALL S <- reachabilityError(GeometricPlaceAspectInst, S).`) erhält man alle Skalen des als Parameter übergebenen Aspekts, die von irgendeiner anderen Skala des gleichen Aspekts nicht durch eine Sequenz von IntraOperationen erreicht werden.

```
// check reachability of any scale from within any other scale of the same aspect
rule reachabilityError: FORALL A,S,S1 #reachabilityError(A:#Aspect, S:#Scale) <-
(
  A:#Aspect[#hasScale->>S:#Scale] AND A:#Aspect[#hasScale->>S1:#Scale] AND
  NOT equal(S, S1) AND NOT #existsIntraSequence(S, S1)
).
// existsIntraSequence (step n=1) from one Scale (Spre) to another (S) ..
rule intraSeqN1: FORALL S,Spre (#existsIntraSequence(S, Spre)) <- EXISTS Op
(
  equal(S:#Scale,Spre:#Scale) OR
  (S:#Scale[#hasIntraOperation->>Op] AND Op[#fromScale->>Spre:#Scale])
).
// ..and (step n->n+1) from one Scale (Spre) to another (S) by intermediate (Smid)
rule intraSeqNpp: FORALL S,Spre (#existsIntraSequence(S, Spre)) <- EXISTS Smid,Op
(
  #existsIntraSequence(S:#Scale, Smid:#Scale) AND
  Smid[#hasIntraOperation->>Op] AND Op[#fromScale->Spre]
).
```

Abbildung 3.24: Nebenbedingung der Skalen eines Aspekts in F-Logic

3.3.3 Vergleich der Umsetzungen

In den vorhergehenden beiden Abschnitten wurde die Umsetzung des ASC-Modells mit zwei (bzw. drei, wenn OWL-DL und DAML+OIL getrennt betrachtet werden) unterschiedlichen Ontologiesprachen vorgestellt: OWL-DL/DAML+OIL und F-Logic. Beide Ansätze der Umsetzung eigenen sich prinzipiell gleichermaßen gut zur verteilten Komposition einer Ontologie, haben aber ihre Stärken und Schwächen in unterschiedlichen Bereichen. Aus den Grundlagenbetrachtungen in Abschnitt 3.1.1 ist klar, daß die Ausdruckskraft von F-Logic gegenüber einer Description Logic als prinzipiell größer zu werten ist. In diesem Abschnitt werden jedoch nicht die zugrundeliegenden Ontologiesprachen verglichen, sondern die Umsetzbarkeit des ASC-Modells in die jeweilige Sprache bzw. die Rahmenbedingungen, die sich daraus jeweils ergeben. Daher werden insbesondere auch die zum Zeitpunkt der Erstellung dieser Arbeit verfügbaren Implementierungen von Reasonern und anderen Tools bei der dem Vergleich zugrundeliegenden Bewertung berücksichtigt, was Probleme der praktischen Implementierung adressiert.

Als Vorteil der XML- und RDF/S-basierten Umsetzung in OWL/DAML+OIL ist zum einen die Verfügbarkeit einer Vielzahl von Validatoren, Parser-Implementierungen und anderen Tools wie grafischen Editoren zu werten. Gerade die Validatoren erlauben die Überprüfung der Konsistenz der die Ontologie spezifizierenden Dokumente auf den verschiedenen Ebenen. Dies ist bei der verteilten Komposition der Ontologien enorm wichtig, weil dadurch bereits viele Inkonsistenzen auf der syntaktischen Ebene verhindert werden. Zum anderen verfügen beide Ontologiesprachen bereits „von Haus aus“ über einige sehr nützliche und vor allem wohldefinierte Sprachmittel, um die Konzepte und ihre Relationen untereinander näher zu charakterisieren wie z.B. Kardinalitäten oder der Typ der Relation (symmetrisch, invers, transitiv). Diese Sprachmittel konnten bei der Umsetzung des ASC-Modells gewinnbringend eingesetzt werden. Durch die Beschränkung auf die Verwendung der Elemente von OWL im Sinne der OWL Description Logic (OWL-DL) ist auch formal die Entscheidbarkeit der darin formulierten Aussagen sichergestellt [87].

Ein Nachteil der Umsetzung in OWL/DAML+OIL ist die ausladende „Geschwätzigkeit“ der XML-Serialisierung. Sie steht zu einem der Hauptanliegen der Erstellung einer Ontologie – der Abbildung komplexer Zusammenhänge der Realität – etwas im Widerspruch, da sie vor allem Ursache für hohe Fehlerquoten im manuellen Erstellungsprozeß ist. Dies wird durch die getrennte Betrachtung von Konzepten (Klassen) und Relationen (Properties) auf dokumentglobaler Ebene noch verstärkt. Vermutlich ist gerade deswegen die Zahl der verfügbaren Validatoren und grafischen Editoren, die der Häufigkeit von Fehleingaben entgegen steuern, so hoch.

Ein weiterer Nachteil der Ontologiesprachen OWL und DAML+OIL ist, daß es bis jetzt keine Reasoner-Implementierungen gibt, die zur Repräsentation des Wissens (Ontologie) und zur Abfrage des Wissens (Query) die gleiche Ontologiesprache verwenden. Das bedeutet, daß das ontologische Wissen zwar in OWL oder DAML+OIL formuliert werden kann, zur Abfrage jedoch eine andere Sprache bzw. eine API oder gar GUI der Reasoner-Implementierung zu verwenden ist. Außerdem ist in den bekannten Reasonern für DAML+OIL und OWL keine Schnittstelle zur Einbindung eigener Operationen vorgesehen, womit einige Konstrukte des Modells, z.B. das der `MetricOperationen`, nur schwer oder überhaupt nicht umsetzbar sind.

Ein Vorteil der Umsetzung des ASC-Modells in F-Logic ist zunächst die objektorientierte und kompakte Syntax der Sprache selbst. Sie ermöglicht die Erstellung einer Ontologie auf übersichtlichere und auch für den Menschen sehr viel nachvollziehbarere Art, da hier die üblichen Vorteile der Objektorientierung (Kapselung, Vererbung, Wiederverwendbarkeit etc.) Anwendung finden. Dadurch ist auch das Modell für den Menschen subjektiv ein wenig greifbarer. Positiv ist bei dieser Umsetzung auch, daß die Queries in F-Logic eine Untermenge der Axiome sind, und somit die Modellierung des Wissens in einheitlicher Weise wie die Abfrage des Wissens geschieht. Ein ganz klarer Vorteil der Umsetzung in F-Logic ist die Verfügbarkeit der OntoBroker Inferenzmaschine mit den in Abschnitt 3.1.2 beschriebenen nützlichen Zusatzfähigkeiten.

Die Verfügbarkeit des OntoBroker ist gleichzeitig ein Nachteil dahingehend, daß er außer den weniger umfangreichen Vorgängern die einzige bekannte Implementierung einer auf F-Logic basierenden Inferenzmaschine ist. So ist die Anzahl der verfügbaren Validatoren bzw. weiteren Tools sehr gering. Der neben OntoBroker erhältliche Editor zur - teilweise grafi-

schen - Erstellungen von Ontologien in F-Logic namens OntoEdit schließt zwar bereits einige Lücken in dem Bereich. Konkurrierende Implementierungen der Inferenzmaschine selbst sowie der dazugehörigen Tools wären jedoch sehr hilfreich, z.B. für Performanzanalysen. So ist die maschinelle Validierung einer Ontologie in F-Logic ohne OntoBroker nahezu unmöglich. Die fehlende Möglichkeit der Serialisierung von F-Logic in XML ist dann ein Nachteil, wenn andere Komponenten eines Systems, in dem F-Logic zum Einsatz kommt, sehr stark auf XML aufbauen. Dies ist z.B. bei den auch im Rahmen dieser Arbeit im Fokus stehenden Web Services der Fall und erfordert definierte Schnittstellen für die Interoperabilität beider Serialisierungsformen.

Fazit: Beide Arten der Umsetzung des ASC-Modells haben, wie geschildert, sowohl in OWL/DAML+OIL als auch in F-Logic für die Erstellung einer (Kontext-)Ontologie gewisse Vor- und Nachteile. Während für die OWL/DAML+OIL-Variante vornehmlich die recht große Unterstützung verschiedenster Tools spricht, bietet sich die Nutzung von F-Logic vor allem durch die kompakte Syntax und die größere Ausdruckskraft an. Aus der Perspektive der Anwendung der erstellten Ontologien, d.h. bei der Abfrage des ontologischen Wissens, überwiegen zur Zeit ganz klar die Vorteile von F-Logic im Zusammenhang mit der *OntoBroker* Inferenzmaschine.

Daher wird im Rahmen dieser Arbeit folgende Vorgehensweise gewählt: Die Erstellung und Erweiterung des ontologischen Wissens erfolgt wahlweise mit DAML+OIL, OWL oder F-Logic auf der Basis der jeweiligen Umsetzung des ASC-Modells (*CoOL Core*). Hierbei kann insbesondere auf alle verfügbaren Tools der jeweiligen Ontologiesprache zurückgegriffen werden, also beispielsweise *OilEd* oder *OntoEdit*. Anschließend werden alle so erstellten Unterkonzepte und Fakten der *OntoBroker* Inferenzmaschine zugeführt. Dazu ist es notwendig, daß alle Ontologiefragmente, die in DAML+OIL oder OWL formuliert wurden, vorher in F-Logic überführt werden müssen. Volz et al. [166] haben gestützt auf die Arbeit von Borgida [43] gezeigt, daß dies prinzipiell möglich ist, solange man sich bei OWL auf den Umfang von OWL-DL beschränkt. Im Fall von DAML+OIL ist die Umwandlung mit Hilfe von *OntoEdit* sogar toolgestützt möglich. Die Abfrage des zusammengeführten ontologischen Wissens im *OntoBroker* erfolgt dann durch geeignet formulierte F-Logic Queries. Dabei dienen die verschiedenen noch vorzustellenden Elemente von *CoOL Integration* u.a. als Brücke zu den XML-basierten Protokollen des Systems.

3.4 Bewertung des Modellentwurfs

In Abschnitt 3.2.2 wurden die Anforderungskriterien an das neue Modell eingehend vorgestellt. Im Folgenden wird eine Bewertung des Modellentwurfs anhand dieser Kriterien vorgenommen.

Hoher Formalisierungsgrad

Ein hoher Formalisierungsgrad wird bereits durch Verwendung von Ontologien erreicht. Wie in Abschnitt 3.1.1 erläutert, haben das in einer Ontologie festgelegte Vokabular und das Netzwerk der Relationen einen stark normativen Charakter für die bei jeder Dienstinteraktion ablaufende Kommunikation.

Darüber hinaus ist die Granularität an verschiedenen Stellen des Modells feiner als in anderen Modellen (vgl. Abschnitt 3.2.1). So erlaubt die Art der Modellierung von Skalen als Menge nicht nur eine Überprüfung des Datentyps, sondern auch spezifischer Instanzen des gleichen Datentyps. Dies erleichtert insbesondere auch das in verteilten Dienstumgebungen häufig im Rahmen der Dienstvermittlung verwendete exakte Matching (vgl. Abschnitt 2.4.3). Die für das Matching in Frage kommenden Attributwerte werden als Teil des „Ontologie-Vokabulars" spezifiziert. Sie bilden einen wichtigen Teil des *shared knowledge* auf kontextueller Ebene der Service-Interoperabilität.

Trotz des hohen Formalisierungsgrades bietet das allgemeine Modell eine hinreichende Flexibilität, um anwendungsspezifische Merkmale wie die individuelle Initialisierung von Skalen oder die individuelle Sortierung der Elemente einer Skala über `MetricOperationen` zu erfassen. So legt das Modell z.B. nicht fest, auf welche Art und Weise eine Skala instantiiert wird. Dies kann durch vollständige Aufzählung aller Elemente im Sinne einer *Enumeration* geschehen, oder durch Anwendung eines Konstruktors, der die Skala instantiiert und initialisiert. Letzteres ermöglicht beispielsweise Skalen, die eine unendliche Anzahl von Elementen beinhalten. Eine Überprüfung über die Relation `hasMember`, ob eine bestimmte Kontextinformation Element der durch diese Skala ausgedrückten Menge ist, entspricht einer linearen Suche. Diese dauert sehr lange – bei einer unendlich großen Menge in der Tendenz unendlich lange. Bei generierten Skalen steht daher zur Überprüfung die Operation `memberCheck` zur Verfügung. Dies ist eine skalenspezifische Operation und kann somit für die jeweilige Skala optimiert werden.

Die Verankerung von Skalen und den darin enthaltenen Kontextinformationen in einem Aspekt erlaubt auch eine eindeutige Zuordnung der Kontextinformationen zu einer spezifischen Dimension des Situationsraumes, was eine Formalisierungslücke anderer Kontextmodelle schließt. Die Menge aller Aspekte mit eigener, identifizierbarer Semantik spannt somit den Situationsraum auf, in dem die als relevant erkannten Entitäten charakterisiert werden können. Ein Aspekt ist daher immer auch ein primärer Anknüpfungspunkt für Spezifikationen der semantischen Interoperabilitätsebene.

Verteilte Komposition

Das Modell sollte die verteilte Komposition der Spezifikation von kontextuellem Wissen unterstützen. Durch den Einsatz von Ontologien wird diese Anforderung erfüllt.

Ein charakteristisches Merkmal von Ontologien ist, daß das Hinzufügen, Ändern und Entfernen von Wissensfragmenten auf der Konzeptebene ähnlich dynamisch erfolgt wie auf der Instanzebene. Somit ist man durch die Verwendung von Ontologien zur Modellierung der Basiskonzepte wie Aspekt, Skala, Kontextinformation etc. nicht den Einschränkungen in verteilten Systemen hinsichtlich der Dynamik unterworfen, die beispielsweise unter Verwendung von ER-Modellen auf der Strukturebene bestehen (vgl. Abschnitt 3.1.1). Zur zeitlich und netztopologisch verteilten Komposition der Spezifikation von kontextuellem Wissen können also Konzept- und Instanzebene gleichermaßen gut eingesetzt werden.

Weiterhin wird die verteilte Komposition im Modellentwurf u.a. auch dadurch unterstützt,

daß nach Möglichkeit keine minimalen Kardinalitäten angegeben sind. Diese würden die gleichzeitige Spezifikation unterschiedlicher Konzepte (z.B. daß zu einem Aspekt auch immer mindestens eine Skala gehört) erfordern, um keine Fehlermeldung bei der Anmeldung der Spezifikation am Reasoner zu provozieren.

Die Menge der jeweils am Reasoner bekannten Konzepte und Instanzen bestimmt den Umfang des Wissens, das dem Reasoning zugrunde liegt. Insbesondere kann auch aus Teilwissen Informationen gewonnen werden (vgl. Unterpunkt „Vollständigkeit“).

Partielle Validierung

Die Überprüfbarkeit von Teilspezifikationen insbesondere mit Hilfe von Validatoren hängt stark von der verwendeten Ontologiesprache ab, in der das neue Modell umgesetzt wurde (vgl. Abschnitt 3.3.3).

Bei der XML- und RDF/S-basierten Umsetzung in OWL/DAML+OIL ist die Verfügbarkeit einer Vielzahl von Validatoren, Parser-Implementierungen und anderen Tools wie grafischen Editoren als vorteilhaft zu werten. Gerade die Validatoren erlauben die Überprüfung der Konsistenz der Teilspezifikationen. Dies ist enorm wichtig, weil durch den frühzeitigen Einsatz der Validatoren bei der Erstellung der Teilspezifikationen bereits viele Inkonsistenzen auf der syntaktischen Ebene verhindert werden.

Dem gegenüber steht bei der Umsetzung in F-Logic (heute) nur eine sehr eingeschränkte Auswahl an Tools zur partiellen Validierung. Hier ist die ansonsten im Rahmen dieser Arbeit im Vordergrund stehende Umsetzung des Modells in F-Logic etwas benachteiligt.

Vollständigkeit und Qualität der Informationen

Der potentiellen Unvollständigkeit von kontextuellem Wissen wird im Modellentwurf im wesentlichen durch zwei Vorgehensweisen entgegengewirkt.

Einerseits ist die Art des Rückschlußes einer ontologischen Instanz auf das zugrundeliegende ontologische Konzept relativ tolerant gegenüber unvollständigen Spezifikationen. Dieser im Rahmen des Reasonings vorgenommene Rückschluß ist – wie in Abschnitt 3.1.1 erläutert – eigenschaftenbasiert (*property centric*). Er unterscheidet sich damit wesentlich von der objektbasierten Vorgehensweise bei objektorientierten Programmiersprachen . So kann der Reasoner beispielsweise durch geeignet formulierte Axiome dazu gebracht werden, beliebige Konzepte als Skala zu betrachten, wenn das Konzept über die Relation `hasMember` verfügt, unabhängig vom tatsächlichen Typ des jeweiligen Konzepts.

Andererseits erlaubt die Gliederung in Konzepte und Unterkonzepte eine Rückführung von unvollständig definierten Unterkonzepten auf die jeweiligen Basiskonzepte im Sinne einer Betrachtung von Basisklassen in einer Programmiersprache. So kann beispielsweise auch aus unvollständig instantiierten Unterkonzepten des Basiskonzepts `ContextInformation` die zugehörige Entität ermittelt werden, die durch diese Kontextinformation charakterisiert wird.

Die Erkennung der Korrektheit bzw. die Behandlung der Mehrdeutigkeit von Kontextinformationen wird durch die Verknüpfung von Meta-Kontextinformationen erleichtert. Diese Meta-Kontextinformationen stellen Qualitätsindikatoren für die eigentlichen Kontextinformationen dar. So kann beispielsweise durch geeignet formulierte Axiome zum Ausdruck gebracht werden, daß ein Reasoner nur Kontextinformationen berücksichtigen soll, deren mittlerer Fehler unterhalb eines bestimmten Schwellwerts liegt. Der Indikator für den mittleren Fehler einer Kontextinformation ist – wie einige andere – bereits fest als `meanError` Relation einer Kontextinformation im Modell verankert.

Vergleich nicht skalarer Typen

Der Vergleich und damit auch die anwendungsspezifische Ordnung nicht skalarer Kontextinformationstypen wird durch zwei Maßnahmen erreicht: Erstens durch Modellierung der Skalen als ungeordnete Mengen, bei der die oft vom Menschen implizierte lexikalische oder numerische Ordnung der Elemente der Skalen keine Berücksichtigung findet. Und zweitens durch Anwendung einer `MetricOperation`, die aufgabenspezifisch die Ordnung der Elemente der Skala festlegt. Beides zusammen erlaubt die dienstunabhängige Spezifikation von Skalen und ihre anwendungsspezifische Nutzung während einer Dienstinteraktion.

Damit erfordert der Vergleich und die Ordnung nicht skalarer Typen keine Sonderbehandlung im System. Stattdessen werden alle Typen – ob skalar oder nicht skalar – mit dem gleichen Modell erfaßt. Ein ähnlich allgemeingültiger Ansatz zum Vergleich nicht skalarer Kontextinformationstypen ist von keinem anderen Kontextmodell bekannt.

Integrationsfähigkeit in bestehende Dienstumgebungen

Das Modell ist zunächst allgemeingültig und dienstunabhängig definiert. Die Integrationsfähigkeit des Modells und von darauf basierendem kontextuellen Wissen in bestehende Dienstumgebungen hängt daher stark von der jeweiligen Dienstumgebung ab. Der Anpassungsaufwand ist proportional zum „Verwandtheitsgrad“ der betrachteten Dienstumgebung mit den Elementen des Modells und den verschiedenen Umsetzungen. So erleichtern beispielsweise die XML-basierte Umsetzung des Modells in OWL und DAML+OIL natürlich die Integration in XML-basierte Dienstumgebungen wie bei einer Web Service Umgebung. Klar ist auch, daß die Architektur in der Regel um eine Reasoner-Komponente erweitert werden muß, da diese normalerweise in bestehenden Dienstumgebungen nicht vorhanden ist.

Die umgebungsspezifische Integration erfordert häufig die Definition von zuätzlichen Schemata. Auch ist die Definition allgemein häufig gebräuchlicher Unterkonzepte und Instanzen als Katalog von Nutzem. Beides ist im Basismodell nicht enthalten. Allerdings werden Lösungen für beide Punkte in Kapitel 4 vorgestellt.

3.5 Zusammenfassung

Im diesem Kapitel wurde ein Modell zur Abbildung kontextuellen Wissens vorgestellt, welches als Grundlage der Spezifikation des gemeinsamen Verständnisses (*shared knowledge*) auf der Ebene der kontextuellen Service-Interoperabilität dient.

Das neue Modell stützt sich auf eine Reihe von Anforderungen, die aus einer Analyse des Stands der Technik der Kontextmodellierung abgeleitet wurden. Diese Anforderungen motivieren insbesondere die Verwendung von Ontologien, die sich im Rahmen einer Analyse verschiedener Repräsentationsformen als besonders geeignet herausgestellt haben. Hierbei finden als Grundlage Ontologiesprachen Verwendung, die sowohl zur Spezifikation von auf dem Modell basierenden kontextuellem Wissen, als auch zur Abfrage und Auswertung des spezifizierten kontextuellen Wissen im Rahmen von ontologischem Reasoning eingesetzt werden können.

Es wurden eine Reihe von Basiskonzepten des Modells wie *Aspekt*, *Skala* etc. einschließlich ihrer Semantik vorgestellt. Das Modell beschreibt die Bedeutung der Konzepte und ihrer Beziehungen zueinander sowohl auf der ontologischen Konzept- wie auch auf der ontologischen Instanzebene. Auf beiden Ebenen können sie als Bausteine für die Formulierung von Relevanzbedingungen dienen (worauf im weiteren Verlauf noch einmal ausführlich eingegangen werden wird).

Das neue Modell wurde exemplarisch in zwei Arten von Ontologiesprachen umgesetzt. Diese Umsetzungen bilden den Kern einer Context Ontology Language. Es wurden die jeweiligen Stärken und Schwächen der Spracharten in Bezug zur Umsetzbarkeit des vorher eingeführten Modells erörtert und miteinander verglichen.

Das Kapitel schloß mit einer Bewertung, in wie weit die zu Beginn des Kapitels erläuterten Anforderungen an das neue Modell im Modellentwurf berücksichtigt wurden. Es konnte gezeigt werden, daß ein Großteil der Anforderungen bereits durch die Auswahl von Ontologien als Mittel der Repräsentation von kontextuellem Wissen erfüllt werden konnten. Bis auf wenige Ausnahmen im Bereich der Integration wurden die verbleibenden Anforderungen durch spezielle Konstrukte des Modells wie die `MetricOperation` oder umsetzungsspezifische Erweiterungen in der jeweiligen Ontologiesprache abgedeckt.

Im nachfolgenden Kapitel werden verschiedene Anwendungsmöglichkeiten des Modells und der Sprache zur Spezifikation und Auswertung von kontextueller Service-Interoperabilität vorgestellt.

Kapitel 4

Anwendungen des Modells und der Sprache

Die Anwendungsmöglichkeiten des ASC-Modells zur Erstellung und Abfrage von Kontextontologien sind sehr vielseitig. Nach der Vorstellung einer zur Anwendung der Context Ontology Language geeigneten Systemarchitektur werden im weiteren Verlauf dieses Kapitels vier Anwendungsgebiete exemplarisch vorgestellt. Diese Anwendungsgebiete sind stellvertretend für viele andere Fragestellungen zu sehen und dienen daher der Darstellung der Breite der Anwendungsszenarien. Ein weiteres spezifisches Anwendungsgebiet wird mit größerer Detailtiefe im anschließenden Kapitel behandelt.

4.1 Systemarchitektur

Die Architektur des im Rahmen dieser Arbeit verwendeten Systems ist schematisch in Abbildung 4.1 dargestellt, wobei ein besonderer Fokus auf der *context provider domain* liegt.

Das Design der Architektur hat sich insbesondere aus den Vorüberlegungen zum MNMplusCE Dienstmodell (Abschnitt 2.3.3), den Service Frameworks (Abschnitt 2.4.3), der Dienstvermittlung (Abschnitt 2.4.3) sowie den Ontologien und deren Reasoning (Abschnitt 3.1.1) entwickelt. Die prototypische Umsetzung dieser Architektur hatte u.a. den Zweck, die Anwendbarkeit des ASC-Modells zu verifizieren.

Zentrale Komponente der *context provider domain* ist der Reasoner, der das auf dem ASC-Modell basierende kontextuelle Wissen sammelt, zusammenführt, auf Konsistenz prüft, durch Auswertung von Regeln das Wissen erweitert und Anfragen auswertet. Im Falle der prototypischen Umsetzung dieser Architektur kommt dafür die *OntoBroker* Inferenzmaschine zum Einsatz. Diese Inferenzmaschine stellt die Wissensbasis (die *Ontologie*) dar. Ihr steht einerseits mit der Umsetzung des ASC-Modells (CoOL Core in der F-Logic Variante) das Basiswissen zur Verfügung. Andererseits wird sie darauf aufbauend mit Kontextinformationen, neuen Unterkonzepten etc. von verschiedenen Informationsquellen versorgt. Die Inferenzmaschine ist

dadurch in der Lage, neues kontextuelles Wissen auf Konzept- und Instanzebene über Entitäten, Aspekte, Skalen, Kontextinformationen, Operationen usw. abzuleiten, was ein Hauptziel der im Rahmen dieser Arbeit untersuchten Fragestellungen ist.

Das kontextuelle Wissen kann von der Inferenzmaschine abgefragt werden, und zwar durch Angabe von Relevanzkriterien. Zur Entgegennahme der Relevanzkriterien stellt die *context management implementation* u.a. das *context management access point (CMAP)* Interface bereit.

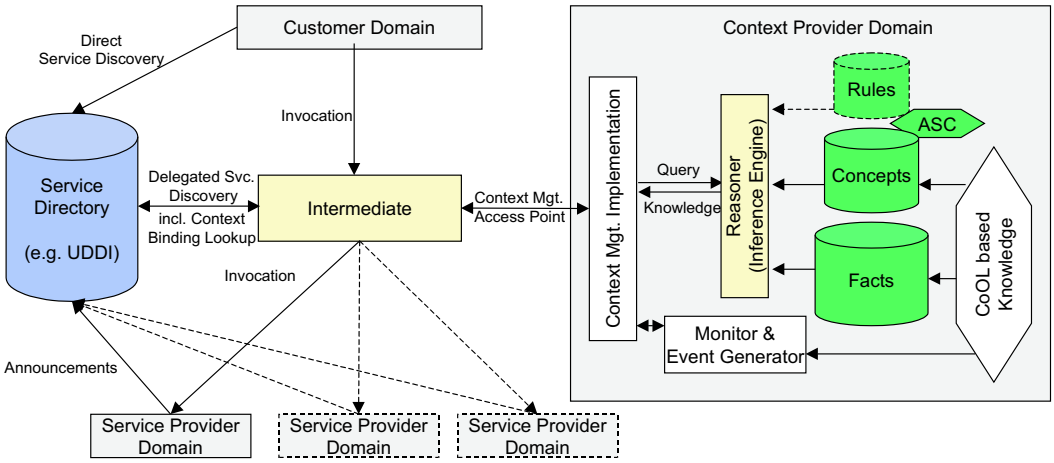

Abbildung 4.1: Systemarchitektur

Leider kann die eingesetzte Inferenzmaschine nur synchron auf Anfragen reagieren, jedoch nicht interessierte Komponenten asynchron über Änderungen des Wissensbestands im Sinne des *Observer Patterns* [68] informieren. Daher wird dieses Verhalten in der Architektur über eine Monitor- und Ereignisgenerator-Komponente nachgebildet, um sich über asynchrone Ereignisse der Art „Ich bin in der Nähe eines Faxgerätes“ benachrichtigen zu lassen. Dazu muß sich die an diesem Ereignis interessierte Komponente über den CMAP beim Kontext Provider mit einer entsprechenden Bedingung „Benachrichtige mich, wenn ich in der Nähe eines Faxgerätes bin“ registrieren. Die Monitor- und Ereignisgenerator-Komponente ist dafür verantwortlich, die hinterlegten Bedingungen jedesmal zu überprüfen, sobald Teile der Bedingungen von Änderungen an der Wissensdatenbank betroffen sind. Zu diesem Zweck werden Änderungen des Wissens nicht nur der Inferenzmaschine zugeführt, sondern parallel auch der Monitor-Komponente.

Die vorgestellte Architektur erlaubt einer *Intermediate* genannten Komponente der Middleware (siehe Abbildung 4.1 in der Mitte) einen abstrakten Dienst – wie in 2.3.3 beschrieben – zu konkretisieren. Dabei verhält sich der Intermediate wie ein Service Client gegenüber der *service provider domain* und gleichzeitig wie ein Service Provider gegenüber der *customer domain*. Von dieser Stellvertreter-Eigenschaft wird im Verlauf dieser Arbeit sehr häufig Gebrauch gemacht. Das ist z.B. immer dann der Fall, wenn die Adaption der Dienstinteraktion auf kontextuelle Einflüsse transparent gegenüber mindestens einer der beteiligten Domänen ablaufen soll. Ähnlich verhält es sich, wenn aufgrund der Kontextadaptivität allgemeine Zusatzauf-

gaben für das Service Framework anfallen, z.B. die Initiierung eines Handoververfahrens zu einem anderen Service Provider. Welche Zusatzaufgaben das noch sind, wird im Detail in den jeweiligen Abschnitten beschrieben.

Dienstvermittlung und Kontext

Wie bei den allgemeinen Dienstvermittlungsverfahren in Abschnitt 2.4.3 erläutert und anhand einiger Beispiele belegt, werden bei den meisten Verfahren zur Dienstvermittlung (Jini, SLP, ...) im wesentlichen drei Arten von Informationen als Matchingkriterien herangezogen:

- Eindeutiger Name bzw. Identifikator (ID) eines Dienstes
- Eindeutiger Name bzw. Identifikator (ID) eines Interfaces, welches von einem Dienst implementiert wird
- Eine Menge von vorgegebenen und frei definierbaren Attributen, die einen Dienst beschreiben

Basierend auf den o.g. drei Arten von Informationen erfolgt bei einer Dienstanfrage ein Vergleich mit den einzelnen Dienstangeboten. Dabei wird bei den untersuchten Verfahren fast ausschließlich exaktes Matching angewendet, d.h. daß z.B. der Inhalt der einzelnen Attribute einer Dienstanfrage mit den Werten der gleichen Attribute des jeweiligen Dienstangebots exakt übereinstimmen muß, um als Match erkannt zu werden.

Diese Art der Beschreibung von Dienstangeboten und Dienstanfragen ist ausreichend zur Beschreibung des *shared knowledge* auf der Signaturebene der Interoperabilität (deren Überprüng man als Hauptziel der allgemeinen Dienstvermittlungsverfahren bezeichnen könnte), für andere Ebenen der Interoperabilität jedoch nicht. Verschiedene Erweiterungen versuchen die Mängel dieser allgemeinen Dienstvermittlungsverfahren z.B. im Bereich Semantik [147, 50, 130] durch spezielle Attribute in Verbindung mit Erweiterungen des Matching-Verfahrens zu beheben. Auf eine ähnliche Weise kann auch den Mängeln auf der kontextuellen Ebene der Interoperabilität durch Anwendung des ASC-Modells entgegen gewirkt werden.

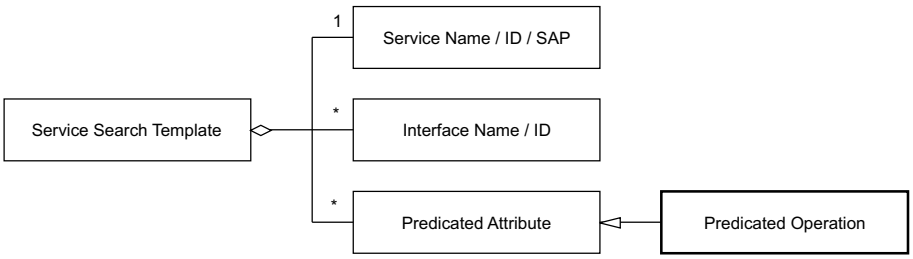

Abbildung 4.2: Strukturdiagramm der Komponenten von Dienstanfragen

Wie Abbildung 4.2 zeigt, wird dabei die Integration des ASC-Modells als Spezialisierung der dienstbeschreibenden Attribute verstanden, wobei die Implementierung vom jeweils einge-

setzten Dienstvermittlungsverfahren abhängt. Wie eine mögliche Integration durch Spezialisierung der Attribute aussieht, kann am Beispiel von SLP gezeigt werden.

SLP verwendet – wie in Abschnitt 2.4.3 beschrieben – Suchfilter auf der Basis von Aussagen mit Prädikaten. Bei der Integration des Kontextes stellt sich das Problem, daß zur Evaluierung von Teilaussagen des Suchfilters im SLP Directory Agent (DA) auf Kontextinformationen zurückgegriffen werden muß, die nicht im DA sondern beim *context provider* vorgehalten werden. Diese Kontextinformationen können entweder vor der Evaluierung vom DA beim *context provider* abgefragt werden, oder Teile der Evaluierung können an den *context provider* delegiert werden. Letzteres ist insbesondere sinnvoll, wenn besondere Relationen oder Operationen verwendet werden sollen, die außerhalb der recht beschränkten Vergleichsmöglichkeiten wie *greaterOrEqual*, *lessOrEqual*, *present* oder *substrings* liegen.

Die Anwendung der Grundidee der von Hughes et al. in [89] vorgeschlagenen Erweiterung der Suchfilter um *Funktionen* innerhalb der Prädikaten-Syntax stellt eine Lösung für die syntaktische Verankerung der Delegation dar. Diese Funktionen nehmen nach Hughes et al. in der Syntax den Platz eines Attributs in den Prädikaten ein. Sie werden zur Laufzeit mit einem mathematischen Ausdruck als Parameter im DA ausgeführt. Das Ergebnis der Funktion wird im weiteren Verlauf wie die Belegung einer Aussagenvariable behandelt. Als Beispiel wird eine Funktion `@rankLow(x)` angegeben, die in einem Suchfilter ähnlich wie in Abbildung 4.3 dazu verwendet werden kann, die Menge der in einem DA bekannten Drucker ohne Papierstau nach einem bestimmten Kriterium (`speed*queueSize`) zu ordnen, um dann den an erster Stelle stehenden Dienst auszuwählen. Auf diese Weise wird jedem Dienstangebot im DA zum Zeitpunkt der Matching-Evaluation ein *dynamisches Attribut* zugewiesen.

```
URL = service:printer
Attributes = (
   & ( @rankLow( speed * queueSize ) = 1 )
     ( !( status = paper_out ) )
)
```

Abbildung 4.3: SLP Service Filter mit Hughes-Funktionen

Die Evaluierung von Kontextinformationen kann gleichermaßen von solchen Funktionen übernommen werden. Beispielsweise kann die Funktion, die im DA während der Evaluierung in der oben beschriebenen Weise ausgeführt wird, als Wrapper-Funktion einer *Operation* des ASC-Modells ausgeprägt sein. Dies ist in Abbildung 4.4 am Beispiel der Operation `memberCheck` dargestellt. Hierbei wird bei Ausführung der Funktion `@memberCheckWrapper` im DA die `memberCheck` Operation beim *context provider* aufgerufen. Diese überprüft, ob die Kontextinformation `CurrentTime_CIInst` in der Skala enthalten ist, die für den Scope-Parameter `openingHoursParam` in der WSDL-Beschreibung analog zu Abbildung 5.5 auf Seite 131 für den `hotelcheckin` Service angegeben ist.

Im Beispiel in Abbildung 4.4 werden zwei weitere Funktionen (`@list`, `@pricePerNight`) des DA verwendet, um eine komplexere Variante der delegierten Evaluation zu zeigen. Das Argument der Funktion `@list("FORALL Hotel...")` ist die in Abbildung 4.14 auf Seite 115

dargestellte F-Logic Relevanzbedingung, die alle Hotels im Umkreis von 5 km um das CeBIT-Gelände als relevant markiert. Durch Weiterleitung der Relevanzbedingung durch die Funktion `@list` an den *context provider* wird die Evaluierung des SLP Suchfilters teilweise an den Reasoner delegiert. Das vom Reasoner daraufhin ermittelte Ergebnis wird von der Funktion `@list` als mehrdimensionales Feld entsprechend dem Vorkommen der Allquantor-gebundenen Variablen an den DA zurückgeliefert. Im obigen Beispiel handelt es sich also um eine zweidimensionale Matrix mit Spalten für `Hotel` und `Position` und einer Zeile pro gefundenem Eintrag. Die Funktion `@list` stellt somit das Bindeglied zu den Kontextinformationen dar. In diesem Beispiel ordnet die Funktion `@pricePerNight` im Sinne der Hughes-Funktionen zusätzlich jedem Element der von `@list` zurückgelieferten Liste von Hotels eine Preisinformation zu, die zu einem positiven Match führt, wenn der Preis $\leq$ 100 EUR ist.

```
URL = service:hotelcheckin
Attributes = (
  & ( @memberCheckWrapper(openingHoursParam,CurrentTime_CIInst) = 1 )
    ( @pricePerNight(@list("FORALL Hotel, Position <- ...")) <= 100 )
)
```

Abbildung 4.4: SLP Service Filter mit Operationen

Die hier dargestellte Vorgehensweise zur Integration des Reasonings von Kontextinformationen in SLP ist ein Beispiel für die verschiedenen Möglichkeiten der Integration in die unterschiedlichen Dienstvermittlungsverfahren. Andere Arten der Integration sind denkbar, wurden im Rahmen dieser Arbeit aber nicht näher untersucht. Mögliche Ansätze sind z.B. die Adoption einer agentenbasierten, mehrstufigen Architektur zur semantischen Dienstsuche, wie sie von Sriharee und Senivongse in [147] vorgeschlagen wurde, ergänzt um Konzepte des ASC-Modells und ihrer Anwendung. Ein alternativer, jedoch nicht näher betrachteter Ansatz wäre auch die Erweiterung der *Context Aware Packets (CAPs)* des Capeus Service Frameworks (siehe Abschnitt 2.4.3) um auf die Konzepte des ASC-Modells abgegestimmte *Context Constraints*.

4.2 Bindungen des Kontextes

Bisher wurde die Anwendbarkeit des ASC-Modells und der Context Ontology Language für die kontextuelle Unterebene der Service-Interoperabilität isoliert betrachtet. Jedoch kann auch die Zusammenführung von Elementen mehrerer Ebenen von Vorteil sein. So kann z.B. auf der Basis des ASC-Modells spezifiziertes kontextuelles Wissen mit Spezifikationen anderer Interoperabilitätsebenen verknüpft werden.

Ein Beispiel für diese Vorgehensweise stellen die Bindungen des Kontextes (*context bindings*) dar, deren Schema-Definitionen der Integrationsgruppe der Context Ongology Language (*CoOL Integration*) zugeordnet sind. Über solche Bindungen werden Aspekte und/oder Skalen des ASC-Modells mit Eingabe- und/oder Ausgabe-Parametern einer Operation eines Dienstes verknüpft („gebunden“). Da jedem individuellen Aspekt eine individuelle Semantik

zugeordnet ist, kann eine Bindung des Kontextes auch als Bindung der Semantik für den jeweiligen Parameter betrachtet werden. Durch das Binden eines Eingabe-Parameters einer Operation an einen Aspekt wird zum Ausdruck gebracht, daß die Implementierung dieser Operation als Eingabewert eine der auf diesem Aspekt definierten Kontextinformations-Instanzen erwartet. Dies kann durch die Angabe einer spezifischen Skala des Aspekts weiter eingeschränkt werden. Ähnlich verhält es sich bei der Bindung eines Ausgabe-Parameters bzw. des Return-Parameters einer Operation an einen Aspekt: Damit wird spezifiziert, daß der Ausgabe- bzw. Return-Parameter eine Kontextinformations-Instanz einer der Skalen des Aspekts sein wird, wenn die Operation ausgeführt wurde. Abbildung 4.5 zeigt dies am Beispiel eines Ausschnitts einer erweiterten WSDL-Beschreibung eines *MapService*.

```
<!-- excerpt from MapService.wsdl -->
<wsdl:message name="showMapRequest"> <!-- input msg -->
  <wsdl:part
    <!-- standard WSDL attributes for wsdl:part  -->
      name="currentPositionParam"
      type="xsd:string"
    <!-- add. attribs expressing context binding -->
      xmlns:cb="http://context-aware.org/cool/wsdl-cb.xsd"
      cb:aspect="urn:#GeometricPlace_Aspect"
      cb:scale="urn:#GaussKrueger_Scale" />
</wsdl:message>
```

Abbildung 4.5: Ausschnitt einer erweiterten WSDL-Beschreibung mit Bindung des Kontextes

Die Operation `showMap` ist mit Elementen von WSDL auf Signaturebene als eine Operation beschrieben, die einen einzigen Eingabe-Parameter vom allgemeinen Typ `String` hat. Mehr kann mit den Mitteln von WSDL nicht zum Ausdruck gebracht werden, insbesondere nicht, was die Bedeutung des Parameters mit dem Namen `currentPositionParam` ist. Während beim menschlichen Betrachter die Wahl *dieses* Namens bestimmte Assoziationen beim Lesen der Teile *current* und *position* weckt, ist diese Assoziationsfähigkeit dem Computer – insbesondere auf Signaturebene – nicht gegeben (siehe auch Abschnitt 2.2.2). Durch die Erweiterung der Parameterbeschreibung mit der Bindung an einen Aspekt und eine seiner Skalen entfällt diese Einschränkung der Ausdruckskraft von WSDL zum Teil: Die Zusatzinformationen sorgen dafür, daß sowohl auf semantischer Ebene (siehe Abschnitt 2.2.4), aber vor allem auf kontextueller Ebene (siehe Abschnitt 2.2.6) das *shared knowledge* ausgebaut und damit die Ausdruckskraft erhöht wird. Die Verankerung von Informationen der Protokoll-, Semantik- und Kontextebene in Beschreibungen der Signaturebene ist dabei nicht ungewöhnlich, wie diverse Beispiele zeigen (u.a. die gelungene semantikbasierte Vermittlung von Web Services nach Sriharee und Senivongse [147]).

Auch wenn das Beispiel in Abbildung 4.5 Bindungen des Kontextes als Erweiterungen einer WSDL-Spezifikation zeigt, ist ihre Anwendungen nicht auf WSDL beschränkt. So ist die Integration von Bindungen des Kontextes z.B. auch in DAML-S Dienstbeschreibungen (siehe Abschnitt 2.2.4) möglich und sinnvoll. Zu diesem Zweck wurde in [154, 155] vorgeschlagen, DAML-S um eine Kontext-Perspektive (*ServiceContext*) der Dienstbeschreibung gemäß Abbildung 4.6 zu erweitern. Diese neue Perspektive bündelt die Spezifikationen des kontextuellen

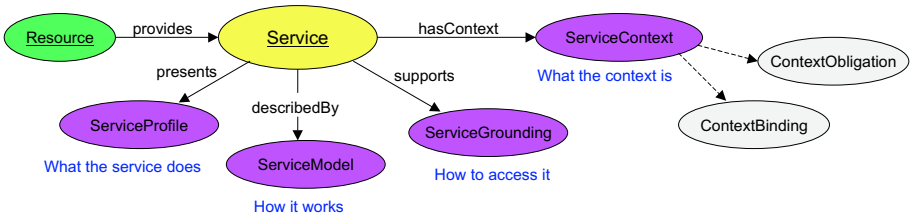

Abbildung 4.6: DAML-S mit Kontexterweiterung

Wissens über einen Dienst und erlaubt insbesondere eine weit detailliertere Spezifikation des kontextuellen Wissens. So sieht DAML-S zwar bereits die Beschreibung des Dienstes mit einigen nicht funktionalen Attributen wie `geographicRadius` und `qualityRating` vor, diese sind jedoch sehr speziell, ihre Definition ist wenig formal, und es wird nur eine sehr geringe Anzahl kontextueller Aspekte berücksichtigt. Das *ContextBinding* Untermodell von *ServiceContext* beinhaltet dabei die notwendigen Definitionen, um Bindungen des Kontextes in DAML-S Dienstbeschreibungen zu integrieren. Dies kann entweder analog zum WSDL-Beispiel direkt innerhalb der Syntax-Spezifikation (bei DAML-S *Grounding* genannt) geschehen, oder – wie in Abbildung 4.7 dargestellt – als externes Dokument mit Referenz auf einen *AtomicProcess* des *ServiceGroundings*.

```
<?xml version="1.0"?>
<!DOCTYPE CoOL [
<!ENTITY xsd  "http://www.w3.org/2000/10/XMLSchema#">
<!ENTITY rdf  "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
<!ENTITY grnd "http://wwww.daml.org/daml-s/Grounding.daml#">
<!ENTITY CoOL "http://context-aware.org/cool/damls-cb.owl#"> ]>

<rdf:RDF xmlns="&CoOL;" xmlns:xsd="&xsd;" xmlns:rdf="&rdf;"
         xmlns:grounding="&grnd;" xmlns:cool="&CoOL;">

  <cool:ServiceParameterBinding>
    <Operation>
      <grounding:damlsProcess
        rdf:resource="urn:PhotoShop.daml#setLocation" />
      <Parameter>
        <PartName rdf:datatype="&xsd;NCName">pickupPlace</PartName>
        <contentFromAspect
          rdf:resource="urn:place.cool#PostalPlaceAspect"/>
        <contentFromScale
          rdf:resource="urn:place.cool#PostalAddrScale"/>
      </Parameter>
    </Operation>
  </cool:ServiceParameterBinding>
</rdf:RDF>
```

Abbildung 4.7: Bindung des Kontextes zum DAML-S Grounding (extern)

Bindungen des Kontextes haben bei Betrachtung der kontextuellen Kompatibilität im Rahmen der kontextadaptiven Dienstvermittlung und Dienstnutzung eine zentrale Rolle, wie in Abschnitt 4.5 vorgestellt wird.

4.3 Zusicherungen des Kontextes

Ein anderes Beispiel aus der Integrationsgruppe der Sprache sind die Schemata zur Spezifikation von Zusicherungen des Kontextes (*context obligations*). Mit einer solchen Zusicherung (Obligation) spricht eine potentiell oder aktuell an einer Dienstinteraktion beteiligte Komponente die Garantie aus, während einer Dienstinteraktion bestimmte Grenzwerte eines im Zusammenhang mit der Dienstinteraktion wichtigen Qualitätsparameters – der Kontextvariablen – nicht zu über- oder unterschreiten. In der Terminologie des ASC-Modells ist die zusichernde Komponente eine Entität und der Qualitätsparameter eine Kontextinformation, die diese Entität in einem spezifischen Aspekt charakterisiert und ggf. einer bestimmten Skala dieses Aspekts zugeordnet ist. Insofern ist eine Zusicherung des Kontextes wieder eine Skala, die den Wertebereich für den Qualitätsparameter darstellt. Diese Betrachtung steht im Einklang mit dem MNMplusCE Dienstmodell (siehe Abschnitt 2.3.3), in dem Quality-of-Service (QoS) Parameter, die einen Dienst beschreiben, als Spezialisierung von Kontextinformationen modelliert werden.

Abbildung 4.8 zeigt die einfache Zusicherung eines PhotoShop Dienstes, eine Lieferzeit von 3 Tagen nicht zu überschreiten. Die dazu verwendeten XML-Elemente sind in einem Schema definiert, welches im Header angegeben wird und der Integrationsgruppe zugeordnet ist.

```
<?xml version='1.0' encoding='UTF-8'?>
<co:ContextObligation    xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
                         xmlns:xsd="http://www.w3.org/1999/XMLSchema"
                         xmlns:co="http://context-aware.org/cool/co.xsd"
                         xmlns="http://context-aware.org/cool/co.xsd">

  <Obligation name="DeliveryEndurance">
    <Entity id="http://provider.com#PhotoShopService"/>
    <Predicate xsi:type="cb:LessOrEqual">
      <Aspect id="#DurationAspect"/>
      <Scale id="#DaysScale"/>
      <MetricOperation id="#StandardInteger"/>
      <ContextInformation xsi:type="xsd:nonNegativeInteger">3</ContextInformation>
    </Predicate>
  </Obligation>

</co:ContextObligation>
```

Abbildung 4.8: Einfache Zusicherung des Kontextes

Zusicherungen des Kontextes haben gewisse Ähnlichkeiten mit den *service level guarantees* in WSLA (siehe Abschnitt 2.3.1). Im direkten Vergleich fällt jedoch auf, daß die Ausdrucksstärke

von *context obligations* größer ist, da eine exakte Spezifikation der Wertebereiche durch die Verweise auf Aspekte und Skalen gegeben ist. Darüber hinaus kann eine Obligation über eine beliebige Anzahl von Meta-Kontextinformationen charakterisiert werden, wohingegen eine weitere Charakterisierung einer Garantie in WSLA auf die Angabe einer zeitlichen Gültigkeit (*validity*) beschränkt ist.

Zusicherungen des Kontextes besitzen bei Betrachtung der kontextuellen Ersetzbarkeit im Rahmen der kontextadaptiven Dienstvermittlung und Dienstnutzung ebenfalls eine zentrale Rolle, wie in Kapitel 5 vorgestellt wird.

4.4 Transfer-Modell

Eine weitere Anwendungsmöglichkeit des ASC-Modells ist seine Verwendung als Transfer-Modell, um die Forschungsergebnisse, die den anderen Modellen zugrunde liegt, zu adoptieren.

Ein gutes Beispeil hierfür ist das bereits erwähnte, grafisch orientierte Kontextmodell von Henricksen et al. [84]. Die Autoren dieses Modells haben den *Object-Role Model (ORM)* Ansatz um kontextuelle Merkmale erweitert. Das zentrale Modellierungskonzept von ORM ist der *fact*, der mit Relationen im ER-Modell vergleichbar ist. Das Modell einer Domäne besteht aus der Identifikation angemessener *fact* Typen und der jeweiligen Rolle, die verschiedene Entitätstypen in der Domäne ausfüllen.

Die Erweiterung des ORM von Henricksen et al. beinhaltet z.B. die Kategorisierung der *fact* Typen bezüglich ihrer Persistenz und ihrer Quelle. So werden *facts* als kontextuell *static* klassifiziert, wenn sie sich niemals verändern, während die Entität, die sie beschreiben, existiert. Alle anderen *facts* werden als *dynamic* klassifiziert. Letztere werden bezüglich ihrer Quelle weiter unterschieden nach *profil-basierten*, *sensierten* oder *abgeleiteten* Typen. Unter Verwendung des ASC-Modells können ORM *facts* als Kontextinformationen und damit auch als Entität modelliert werden. Darauf aufbauend können die o.g. Klassifikationsmerkmale der kontextuellen Erweiterungen des ORM nach Henricksen et al. als Qualitätsaspekt `ExtendedORMFactTypeAspect` modelliert werden, bei dem die genannten Merkmale auf die Elemente {*static, dynamic profiled, dynamic sensed, dynamic derived*} der Skala `ExtendedORMFactTypeScale` abgebildet werden, wie auch in Abbildung 4.9 dargestellt ist. Dadurch kann jeder Kontextinformation unter Verwendung der Relation `hasQuality` eine Meta-Kontextinformation der Skala `ExtendedORMFactTypeScale` zugewiesen werden, welche die Qualität der Kontextinformation im Sinne von Henricksen et al. charakterisiert.

Die Qualitätsindikatoren von Henricksen et al. entsprechen in ihrer Semantik den Meta-Kontextinformationen und sind eine Untermenge der Qualitätsaspekte im ASC-Modell. Ähnlich verhält es sich mit den *history facts*, die sich im ASC-Modell in der Semantik des `TimePeriodAspect` bzw. `AbsoluteTimeAspect` wiederfinden.

Die letzte verbleibende in [84] angesprochene Erweiterung von ORM sind *fact dependencies*, d.h. 2-stellige Relationen, die voneinander abhängige Kontextinformationen besonders kennzeichnen. Im Falle einer so gekennzeichneten Beziehung zwischen zwei Kontextinformationen ändert sich die Charakterisierung einer Entität automatisch, wenn sich eine andere Charakte-

risierung der durch die `dependsOn`-Relation verlinkten Entität ändert. Dieses Verhalten kann im ASC-Modell durch die Existenz einer Intra- oder InterOperation ausgedrückt werden, die zwei Skalen aufeinander abbildet, aus denen die Kontextinformationen stammen. An dieser Stelle ist das ASC-Modell ausdrucksstärker als das kontextuell erweiterte ORM, da es erlaubt, über die Operation exakt die Art der Abhängigkeit zu spezifizieren.

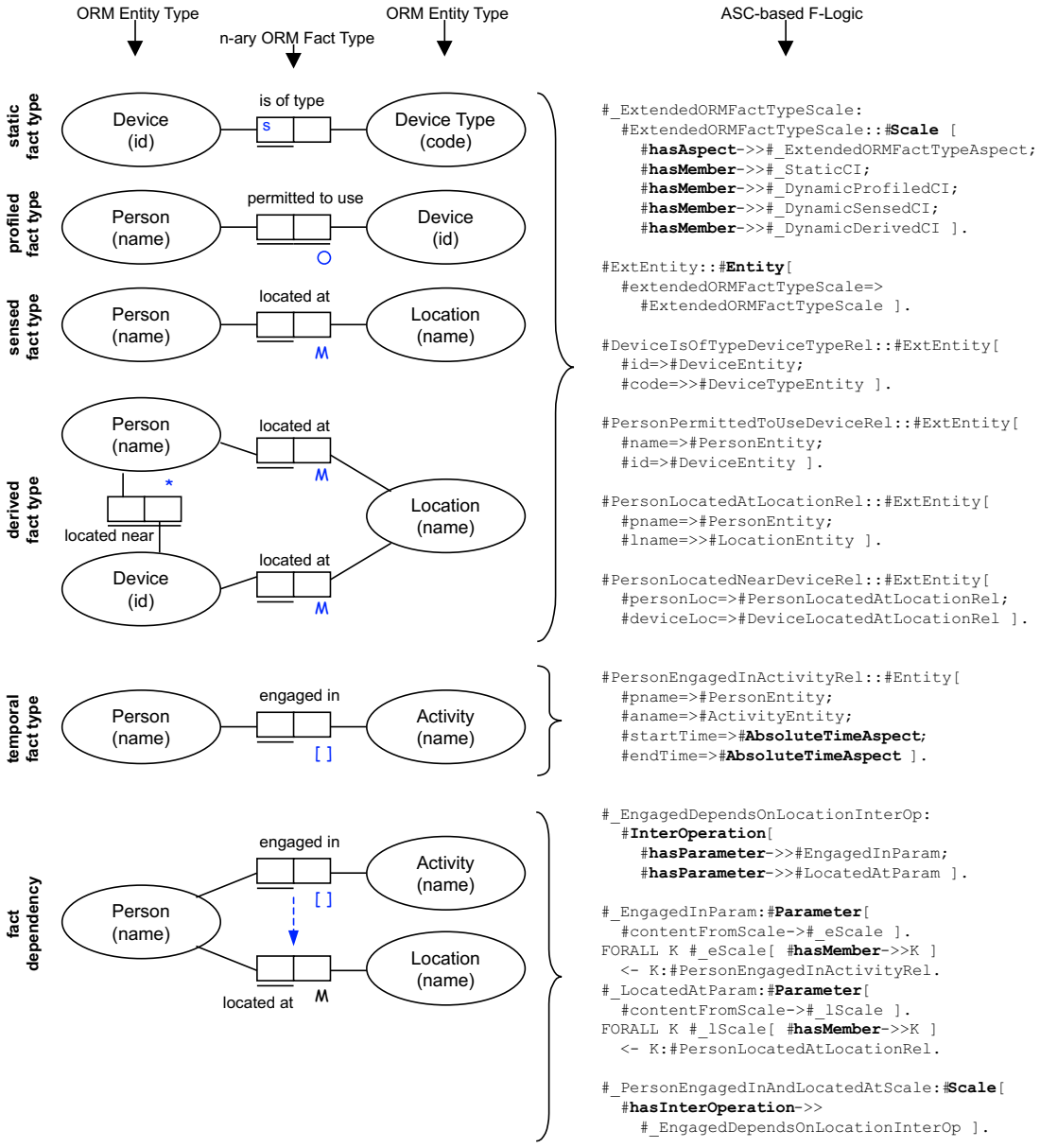

Abbildung 4.9: Transfer der ORM Erweiterung (links) nach ASC (rechts)

Das Beispiel der Adoption des Kontextmodells von Henricksen et al. zeigt das Potential des ASC-Modells, als Transfer-Modell für andere Ansätze der Kontextmodellierung zu dienen.

4.5 Konzepte, Fakten und Reasoning-Netzwerke

Zur Evaluierung der im Rahmen dieser Arbeit vorgestellten Verfahren wurden bereits einige Anwendungen des ASC-Modells teilweise oder vollständig umgesetzt, die als Bestandteil von *CoOL Integration* zur Verfügung stehen [153]. Dabei handelt es sich sowohl um Unterkonzepte der Basiskonzepte, als auch Instantiierungen der Konzepte und Relationen (Fakten). Dem prototypischen Charakter der Testdaten entsprechend sind die Spezifikationen an vielen Stellen nicht vollständig umgesetzt, sondern nur soweit, wie es zur Erprobung des jeweiligen Verfahrens notwendig war. Dennoch gibt der folgende Katalog von Aspekten, Skalen, etc. einen Überblick über die verschiedenen Anwendungsvarianten des Basismodells.

`AbsoluteTimeAspect` erlaubt die Spezifikation eines absoluten Zeitpunkts auf einer `UTCScale` oder einer Skala einer bestimmten Zeitzone, die mit der `UTCScale` jeweils über zwei entsprechende IntraOperationen verlinkt sind. Jeder Kontextinformation ist über die Relation `hasTimestamp` eine Meta-Kontextinformationen aus diesem Aspekt zugeordnet.

`DurationAspect` bildet eine Zeitspanne auf die Anzahl der Millisekunden seit einem spezifischen, aber variablen Zeitpunkt ab. Die zugehörige `DurationScale` korrespondiert mit der Menge der nicht-negativen natürlichen Zahlen. Es existiert aber auch eine `TimeRepresentationScale`, die jedem Wert der `DurationScale` einen für den Menschen besser lesbaren Wert über die `getRepresentationFromDuration` IntraOperation zugeordnet, bzw. in umgekehrter Richtung aus einer speziell formatierten String-Darstellung per `getDurationFromRepresentation` IntraOperation den entsprechenden Wert der `DurationScale` berechnet.

`TimePeriodAspect` ist ein Aspekt bestehend aus zwei Skalen, `AbsAbsTimePeriodScale` und `AbsRelTimePeriodScale`. Beide Skalen werden über eine IntraOperation `getAbsAbsPeriod` bzw. `getAbsRelPeriod` erzeugt, die jeweils zwei Parameter haben. Der erste Parameter ist ein Wert von der `UTCScale` des `AbsoluteTimeAspect`. Während der zweite Parameter von getAbsAbsPeriod ebenfalls ein Wert der `UTCScale` ist, verweist der zweite Parameter von getAbsRelPeriod auf einen Wert der Default-Skala `DurationScale` des `DurationAspect`.

`GeographicPlaceAspect` deckt semantisch alle Arten von geografischen Ortsinformationen ab. Dieser Aspekt hat in der vorgenommenen Umsetzung zwei Skalen, `WGS84Scale` als Default-Skala und `GaussKruegerScale`. Beide Skalen sind gegenseitig verlinkt über die beiden IntraOperationen `getWGS84fromGK` und `getGKfromWGS84`.

`SymbolicPlaceAspect` deckt semantisch alle Arten von symbolischen Ortsinformationen als Text-basierte Beschreibung („Flughafen München, Terminal E“) ab.

`SpatialDistanceAspect` erlaubt die Angabe eines nicht-negativen Wertes auf den Skalen `NauticalMilesScale`, `KilometerScale` und `MeterScale`, die alle untereinander über entsprechende IntraOperationen verlinkt sind. Die `KilometerScale` verfügt über eine zusätzliche `getDistanceBetweenGaussKrueger` InterOperation, mit deren Hilfe die Distanz zwischen zwei absoluten Gauß-Krüger-Koordinaten bestimmt werden kann.

`SpeedAspect` kann zur Angabe der Geschwindigkeit basierend auf den drei Skalen `KnotsScale` (z.B. für die horizontale Geschwindigkeit in der Luftfahrt), `FeetPerMinuteScale` (z.B. für die vertikale Geschwindigkeit in der Luftfahrt), oder `KilometerPerHourScale` (z.B. für die horizontale Geschwindigkeit im terrestrischen Verkehr) verwendet werden. Jede Skala ist (in der Beispielimplementierung) durch eine InterOperation konstruiert, die je einen Parameter aus dem `SpatialDistanceAspect` und dem `DurationAspect` dazu benutzt, die Geschwindigkeit nach der einfachen `delta_s` geteilt durch `delta_t` Methode zu berechnen.

`EventAspect` hat eine einzige Skala, die eine ungeordnete Menge von `EventIds` beinhaltet. Jedes Element der Skala, d.h. jede Instanz der EventId-Klasse, leitet sich außerdem von Kontextinformationen einer Skale des `TimePeriodAspect` im Sinne der Mehrfachvererbung ab.

`PriceAspect` kann dazu benutzt werden, eine Entität bezüglich eines Preises in verschiedenen Währungen zu charakterisieren. Jede Währung wird durch eine eigene Skala repräsentiert, die jeweils gegenseitig über IntraOperationen der Art `getEURfromXXX` und `getXXXfromEUR` mit der Default-Skala `EuroCurrencyScale` verlinkt sind.

`AirlineClassAspect` bestehend aus zwei aufzählungsbasierten Skalen `ComfortClassScale` und `BookingClassScale`. Die Mächtigkeit dieser beiden Skalen ist unterschiedlich, da es mehr Buchungsklassen als Komfortklassen (first, business und economy) gibt, was innerhalb der IntraOperationen zwischen den beiden Skalen berücksichtigt wird. Diese IntraOperationen sind nicht bijektiv, weshalb neben der Abbildung auf die jeweils andere Skala auch die Qualitätsinformationen im Rahmen der Abbildung entsprechend angepaßt werden (siehe dazu insbesondere Abschnitt 3.8).

`WeatherAspect` ist eine abstrakte Basis komplexer Beschreibungen der Zustände des Wetters. Aus ihm leiten sich verschiedene rudimentär implementierte Unteraspekte ab. Die Unteraspekte beschreiben die Semantik verschiedener Aspekte des Wetters wie Luftfeuchtigkeit, Temperatur, Windstärke und -richtung, Bedeckungsgrad und Schichtung der Wolken. Jeder Unteraspekt beinhaltet jeweils eine Skala als Menge von Kontextinformationen. Beim Unteraspekt der Temperatur gibt es zwei Skalen, eine mit Werten in Grad Fahrenheit und eine mit Werten in Grad Celsius.

Darüber hinaus können folgende Aspekte und Skalen insbesondere als Grundlage für Kontextinformationen dienen, welche die Qualität anderer Kontextinformationen charakterisieren (Meta-Kontextinformationen):

`SelfAssuranceProbabilityAspect` gibt semantisch das Maß der Sicherheit einer Informationsquelle an, das die emittierte Information auch den Tatsachen entspricht. Wie jeder `ProbabilityAspect` verfügt dieser Aspekt über eine `ProbabilityScale`, die Wahrscheinlichkeitswerte im Intervall $0 \leq p < 1$ als Instanzen aggregiert. Eine Wahrscheinlichkeit von 100% ist im Modell nicht vorgesehen, da es keine absolute Sicherheit gibt und dies vor allem bei sich widersprechenden Aussagen zur gleichen Entität zu vermeidbaren Problemen führen würde.

`GeometricPlaceAccuracyAspect` bestimmt die Qualität einer geometrischen Ortsinformation anhand verschiedener Skalen. Beispielsweise wird über die `RadialScale` einer Kontextinformation ein bestimmter Radius zugeordnet. Damit wird angegeben, daß die Kontextinformation stellvertretend für alle geometrischen Ortsinformationen innerhalb des Kreises (2D) oder der Kugel (3D) mit diesem Radius steht, was indirekt die Genauigkeit der Kontextinformation beschreibt: je größer der angegebene Radius, umso größer sind mittlere und maximale Abweichung der Kontextinformation von der Realität.

`ExtendedORMFactTypeAspect` zur Charakterisierung von Persistenz und Quelle von kontextuellen Relationen gemäß des Modellierungsansatzes von Henricksen et al. [84] unter Verwendung des ASC-Modells als Transfer-Modell (siehe Abschnitt 4.4).

Diese Liste von Unterkonzepten des ASC-Modells gibt einen ersten Überblick über die vielfältigen Möglichkeiten der Erstellung von Aspekten und Skalen, ist jedoch weder vollständig noch repräsentativ. Eine der wesentlichen Absichten hinter dem Design des ASC-Modells war es ja gerade, eine erweiterbare und dadurch individuell anwendbare Grundlage zur Spezifikation des kontextuellen Wissens zu haben.

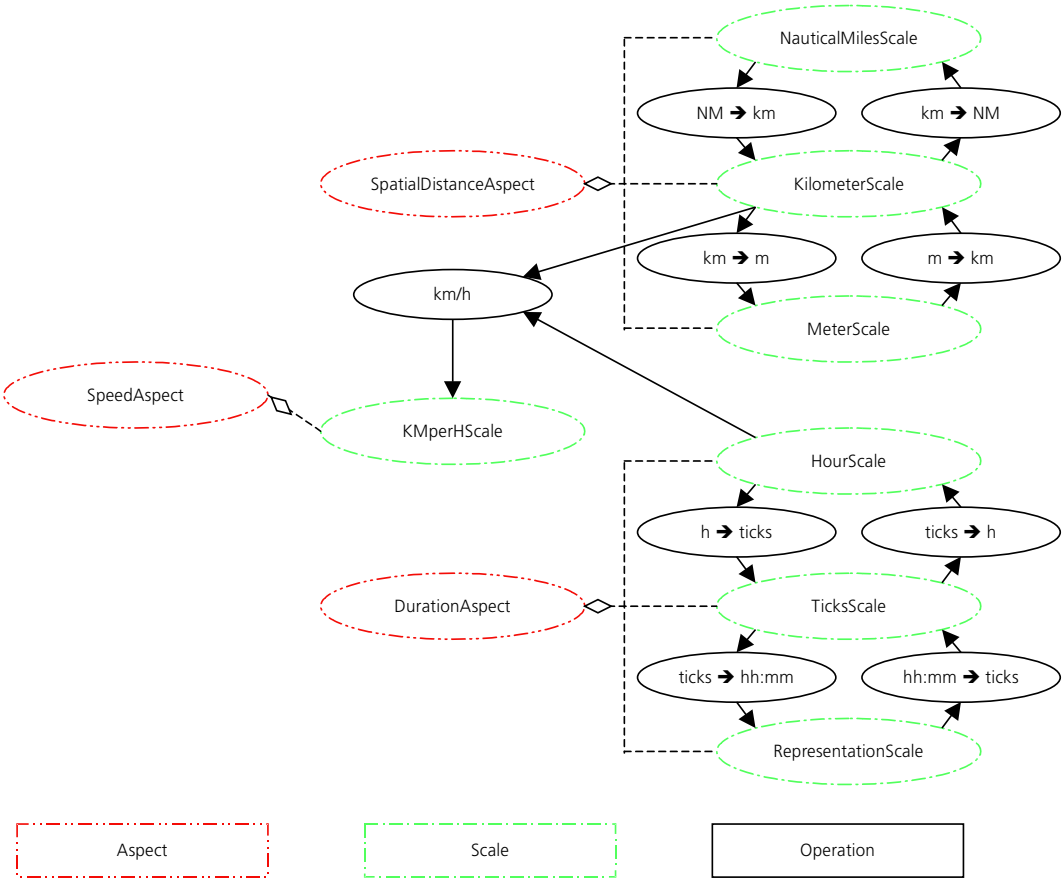

Abbildung 4.10: Netzwerk von Fakten

Abbildung 4.10 zeigt grafisch anhand eines einfachen Beispiels, wie durch die Verwendung von Intra- und InterOperationen ein ganzes Netz von Aspekten samt zugehöriger Skalen entsteht. In diesem Beispiel sind verschiedene Skalen des `SpatialDistanceAspect` über IntraOperationen miteinander verbunden. Analog dazu sind mehrere Skalen des `DurationAspect` über andere IntraOperationen miteinander verbunden. Eine neue `KilometerPerHourScale` eines `SpeedAspect` wird hingegen durch eine InterOperation mit zwei Eingangsparametern, einem aus der `KilometerScale` und einem aus der `HourScale`, erzeugt.

Ein Reasoner kann durch Analyse des Netzwerkes Informationen über die Zusammenhänge und Abhängigkeiten der Knoten ermitteln. So kann der in Abbildung 4.11 dargestellte Ausschnitt aus CoOL dazu genutzt werden, eine Auflistung aller Kompatibilitäts-Pfade durch das per Intra- und InterOperationen aufgespannte Netzwerk zu erhalten. Die dargestellten F-Logic Axiome nutzen dazu über das „Container-Prädikat“ `path` rekursiv aufgebaute Listen, deren Verwendung bereits einen relativ komplexen Inferenzvorgang darstellt. Eine einfache Query der Art `FORALL X <- #path(X)` liefert dann alle Pfade innerhalb des in Abbildung 4.10 skizzierten Netzwerks als Menge von Listen. Das Ergebnis der Query ist in Abbildung 4.12 dargestellt.

```
// Path of Compatibility, induction step n=1
rule pathN1:
FORALL S,Spre,L ( #path([Spre,S]) ) <- EXISTS Op1,Op2,Param
(
  (S:#Scale[#hasIntraOperation->>Op1] AND Op1[#fromScale->Spre:#Scale]) OR
  (S:#Scale[#hasInterOperation->>Op2] AND Op2[#hasParameter->>Param] AND
   Param[#contentFromScale->Spre:#Scale])
).
// Path of Compatibility, induction step n=n+1
rule pathNppIntra:
FORALL S,Spre,L ( #path([Spre|L]) ) <- EXISTS Head,Tail,Op
(
  #path(L) AND unify(L,[Head|Tail]) AND
  Head[#hasIntraOperation->>Op] AND Op[#fromScale->Spre] AND
  NOT inlist(Spre,L) // prevent loops
).
rule pathNppInter:
FORALL S,Spre,L ( #path([Spre|L]) ) <- EXISTS Head,Tail,Op,Param
(
  #path(L) AND unify(L,[Head|Tail]) AND
  Head[#hasInterOperation->>Op] AND Op[#hasParameter->>Param] AND
  Param[#contentFromScale->Spre] AND NOT inlist(Spre,L) // prevent loops
).
```

Abbildung 4.11: Pfadermittlung per Reasoning auf dem Netzwerk von Fakten

Wie in Abschnitt 2.2.6 erläutert, ist eine wesentliche Voraussetzung kontextueller Kompatibilität als Perspektive der kontextuellen Service-Interoperabilität, daß alle an einer Dienstinteraktion beteiligten Komponenten ein identisches Verständnis der relevanten Aspekte haben. Dies beinhaltet die Art der Auswahl der relevanten Entitäten sowie die Menge der Zustände,

welche die Kontextinformationen in den relevanten Aspekten einnehmen können, einschließlich der Verwendung der gleichen Skalen. Dies kann wiederum durch den Bezug auf gemeinsame Spezifikationen der Relevanzkriterien und eines Katalogs der Aspekte erreicht werden.

Jede Bindung des Kontextes stellt – wie in Abschnitt 4.2 beschrieben – eine solche gemeinsame Spezifikation der relevanten Aspekte dar. Durch Berücksichtigung aller an Aspekte und Skalen gebundenen Parameter im Rahmen der Dienstvermittlung wird die Interoperabilität einer Komponente im Sinne der kontextuellen Kompatibilität zu den gefundenen Interaktionspartnern sichergestellt.

Kontextuelle Kompatibilität in diesem Sinne kann auch durch Transformation erreicht werden. Dieser Fall setzt voraus, daß Kontextinformationen, die als relevant für eine Dienstinteraktion eingestuft werden, in eine Form transformiert werden können, die der Dienst auswerten kann. Das oben vorgestellte Verfahren zur Bestimmung aller Pfade durch das per Intra- und Inter-Operationen aufgespannte Netzwerk kann zur Lösung dieser Fragestellung eingesetzt werden.

Im Rahmen der Dienstvermittlung führen Kontextinformationen aus dem für den Dienst benötigten und durch eine Bindung auf Aspekt und Skala spezifizierten Wertebereich für den jeweiligen Parameter zu einem direkten Match. Darüber hinaus können solche als indirekte Matches gewertet werden, von denen ein Pfad nach dem oben beschriebenen Verfahren von anderen Skalen bzw. Aspekten bestimmt werden kann. Am Beispiel aus Abbildung 4.12 bedeutet dies, daß ein Dienst mit mindestens einem an den `SpeedAspect` gebundenen Parameter auch dann als indirekter Match im Rahmen der Dienstvermittlung erkannt wird, wenn zwar keine Kontextinformation auf der Basis der `KMperHScale` verfügbar ist, wohl aber zwei Kontextinformationen jeweils aus `SpatialDistanceAspect` und `DurationAspect`, welche die gleiche Entität charakterisieren. Der Zusammenhang zwischen den letztgenannten Skalen, die beide „gleichzeitig“ Definitionsbereich für jeweils einen Parameter der entsprechenden Inter-Operation sind, und der dritten Skala als Wertebereich der InterOperation, ist artverwandt mit der `dependsOn` Relation aus dem erweiterten ORM, wie sie in Abschnitt 4.4 vorgestellt wurde.

```
// Query: FORALL X <- #path(X).
// Result:
    [#NauticalMilesScale,#KilometerScale]
    [#NauticalMilesScale,#KilometerScale,#MeterScale]
    [#NauticalMilesScale,#KilometerScale,#KMperHScale]
    [#KilometerScale,#MeterScale]
    [#KilometerScale,#NauticalMilesScale]
    [#KilometerScale,#KMperHScale]
    [#MeterScale,#KilometerScale]
    [#MeterScale,#KilometerScale,#KMperHScale] // ... some more ...
```

Abbildung 4.12: Ergebnis der Query als Menge von Listen

Offensichtlich ist auch, daß nur Pfade als Matches gewertet werden dürfen, bei denen für alle Parameter der darin vorkommenden InterOperationen gilt, daß eine Kontextinformation des jeweiligen Definitionsbereichs verfügbar ist (`AND`-Logik). Die Sequenz der Skalen in jedem

Pfad gibt gleichzeitig die Reihenfolge der zu durchlaufenden Operationen einer Kontextinformation an, bevor sie in einem Format vorliegt, das vom Dienst verwertet werden kann. Die Ausführung der Operationen obliegt dem Service Framework, und auch die Verwendung der Inferenzmaschine zur Ermittlung der Pfade ist in das jeweils verwendete Dienstvermittlungsverfahren des Service Frameworks geeignet zu integrieren.

4.6 Relevanz als Anwendung von Reasoning

Aus der Definition 7 des Kontextbegriffs und im Speziellen der Definition 5 des Relevanzbegriffs leitet sich die Notwendigkeit zur Angabe von Relevanzkriterien ab, die zur Laufzeit den Übergang von der Betrachtung der *Situation* zur Betrachtung des *Kontextes* ermöglicht.

Wie dort erwähnt, wird eine Entität als relevant bzgl. einer Aufgabe betrachtet, wenn ihr Zustand mindestens bzgl. eines relevanten Aspekts charakterisiert wird. Eine Entität kann also nur dann als relevant erkannt und bei einer Dienstinteraktion berücksichtigt werden, wenn der Wert einer durch einen Aspekt beschriebenen Zustandsvariablen bekannt ist. So können beispielsweise bei einem Fahrplaninformationsdienst der Bahn zur minutengenauen Auskunft der Abfahrzeiten aus der Menge aller Züge (alle Entitäten) nur diejenigen Züge (relevante Entitäten) berücksichtigt werden, zu denen die aktuelle minutengenaue Verzögerung gegenüber dem Fahrplan (Aspekt) bekannt ist (Relevanzkriterium).

Ein Aspekt ist nach Definition 5 relevant, wenn während der Erfüllung eines Dienstes auf den Zustand bzgl. dieses Aspekts zugegriffen wird, bzw. der Zustand bzgl. dieses Aspekts den Ablauf eines Dienstes beeinflußt. Hierbei wird die Dienstvermittlung selbst als ein Dienst des Service Frameworks angesehen. Die Relevanz eines Aspekts ergibt sich also durch seine Verwendung als Teil eines Relevanzkriteriums.

Dieser Arbeit liegt das *MNMplusCE Dienstmodell* (siehe Abschnitt 2.3.3) zugrunde. Dieses Dienstmodell sieht eine Trennung zwischen *customer domain*, *service provider domain* und *context provider domain* vor. Diese Trennung macht eine Deklaration der dienstinteraktionsspezifischen Relevanz gegenüber der dienstinteraktionsunabhängigen *context provider domain* notwendig. Die Deklaration erfolgt durch Angabe expliziter Relevanzkriterien über den *context management access point (CMAP)*. Mit anderen Worten, durch die Angabe der Relevanzkriterien im Rahmen einer Dienstinteraktion wird ein abstrakter kontextadaptiver Dienst zu einem konkreten kontextadaptiven Dienst.

Bei der Relevanz wird zwischen *externer* und *interner* Relevanz einer Entität in Abhängigkeit der Domäne, welche die Entität identifiziert, unterschieden. Wenn die Relevanz einer Entität bezüglich einer Dienstinteraktion außerhalb der *context provider domain* festgelegt wird, z.B. in der *customer domain*, so wird dies als *externe* Relevanz bezeichnet. Im Falle externer Relevanz weist die andere Domäne die *context provider domain* durch den CMAP auf eine spezifische Entität (z.B. durch einen eindeutigen Bezeichner) hin. Daraufhin liefert der *context provider* alle Kontextinformationen, die ihm zu dieser Entität bekannt sind und diese charakterisieren. Ein Beispiel für eine externe Relevanz ist ein Service Client, der auf eine Entität verweist, der das aktuell genutzte mobile Gerät repräsentiert. Durch Angabe die-

ser Entitäts-Identifikation gegenüber der *context provider domain* ist die *context management implementation* in der Lage, die momentane geografische Position des Endgeräts (z.B. durch Anfrage beim Telefonnetzbetreiber) zu ermitteln. Diese Information kann dann als Kontextinformation an den Service Client übermittelt werden, der daraufhin diese Information als Eingabeparameter eines Dienstaufrufs verwenden kann.

Im Gegensatz dazu zeichnet sich die *interne* Relevanz einer Entität dadurch aus, daß die Entität innerhalb der *context provider domain* identifiziert wird, z.B. anhand vorliegender Kontextinformationen, die als Element einer bestimmten Skala erkannt werden. Auch bei interner Relevanz ist die Angabe von Relevanzkriterien erforderlich, welche die *context provider domain* in die Lage versetzen, entsprechende Entitäten zu ermitteln.

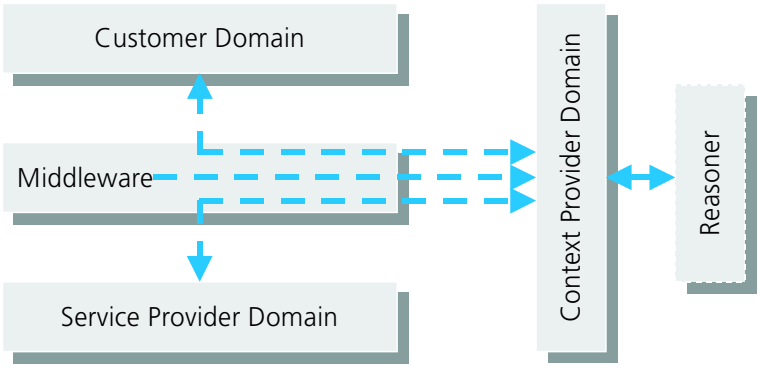

Abbildung 4.13: Relevanzkriterien aus allen Domänen

Abbildung 4.13 verdeutlicht schematisch, daß der Austausch von Relevanzkriterien sowohl von der *customer domain* als auch von der *service provider domain* zur *context provider domain* erfolgt. Da in Service Frameworks in der Regel eine Middleware-Komponente das Bindeglied zwischen *customer domain* und *service provider domain* darstellt, erfolgt auch der Austausch der Relevanzkriterien aus der Sicht der anderen beiden Domänen mit der *context provider domain* über die Middleware-Komponente. Diese kann jedoch auch selbst und unabhängig von den anderen beiden Domänen Relevanzkriterien aufstellen, sofern die Middleware eigenständige Aufgaben im Rahmen einer Dienstinteraktion übernimmt (z.B. Adaptionsaufgaben).

Die in Abbildung 4.1 auf Seite 98 vorgestellte Systemarchitektur ist sehr gut auf dieses Schema abgestimmt. Sie ermöglicht die Auswertung von Relevanzbedingungen als Anwendung des Reasonings. Per CMAP werden Relevanzbedingungen in Form einer Query an die Inferenzmaschine abgesetzt, von dieser ausgewertet, und die entsprechenden Entitäten, Kontextinformationen etc. synchron oder asynchron zurückgeliefert.

Eine externe Relevanzbedingung identifiziert eine Entität eindeutig über ihren Namen (URI). Eine interne Relevanzbedingung ist ein Filter, der auf mehreren verschiedenen Ebenen dazu benutzt werden kann, eine oder mehr relevante Entitäten aus der Menge aller bekannten Entitäten zu identifizieren.

Ein Filter der ersten Ebene spezifiziert einen bestimmten Aspekt, in dem eine Entität charakterisiert ist, um als relevant erkannt zu werden. Ein Beispiel für solch ein Kriterium wäre umgangssprachlich „Berücksichtige alle Entitäten, die bezüglich des Aspekts *ComfortClass* charakterisiert sind“, was wie folgt als korrespondierende F-Logic Query ausgedrückt werden kann:

```
FORALL E,C,S <-
   C:#ContextInformation AND C[#characterizes->E] AND
   C[#hasScale->>S] AND S[#hasAspect->>#ComfortClassAspect].
```

Ein Filter der zweiten Ebene spezifiziert neben dem Aspekt auch eine Bedingung auf der Ebene der Kontextinformationen dieses Aspekts, z.B. „berücksichtige alle Entitäten, die bezüglich des Aspekts *ComfortClass* als *First* charakterisiert sind“, was entsprechend als F-Logic Query ausgedrückt werden kann als:

```
FORALL E,C,S,V <-
   C:#ContextInformation AND C[#characterizes->E] AND
   C[#hasScale->>S] AND S[#hasAspect->>#ComfortClassAspect] AND
   equal(C,#First).
```

Auf einer dritten Ebene spezifiziert ein Filter darüber hinaus auch noch eine Qualitätsbedingung der Art „berücksichtige alle Entitäten, die bezüglich des Aspekts *ComfortClass* als *First* charakterisiert sind, und bei denen diese Information nicht älter als 3 Tage ist“. Eine korrespondierende F-Logic Query würde wie folgt aussehen:

```
FORALL E,C1,C2,S1,V1,V2 <-
   C1:#ContextInformation AND C1[#characterizes->E] AND
   C1[#hasScale->>S] AND S1[#hasAspect->>#ComfortClassAspect] AND
   equal(C1,#First) AND
   C1[#hasQuality->>C2] AND C2[#hasScale->>#AgeInDaysScale] AND
   C2[#hasIntegerValue->V2] AND lessorequal(V2, 3).
```

Abbildung 2.7 auf Seite 23 verdeutlicht die Unterscheidung der Ebenen visuell. Besonders zu beachten ist, daß die Kontextinformation, die eine Entität bezüglich des spezifizierten Aspekts charakterisiert, selbst als Entität betrachtet wird, wenn sie bezüglich eines Qualitätsaspekts durch eine Meta-Kontextinformation charakterisiert wird.

Statt der Identifizierung einer oder mehrerer Entitäten über die Relevanzbedingungen kann die an der Dienstinteraktion beteiligte Komponente, welche die Relevanzbedingung spezifiziert, möglicherweise auch an der charakterisierenden Kontextinformation selbst interessiert sein. Die Modifikationen an den gezeigten F-Logic Anfragen zum Erreichen einer entsprechenden Aussage sind minimal: Im wesentlichen besteht die Änderung im Entfernen der Variable „E“ im Allquantor und im Entfernen aller Teilformeln, die diese Variable enthalten.

Ein weiteres Beispiel, welches die Mächtigkeit der Relevanzbedingungen basierend auf Konzepten des ASC-Modells zeigt, ist in Abbildung 4.14 dargestellt. Die in diesem Beispiel angegebene Relevanzbedingung filtert aus der Menge aller im System bekannten Hotels diejenigen heraus, die von einem Referenzpunkt (hier: CeBIT-Gelände) nicht weiter als eine bestimmte Distanz entfernt sind. Dazu werden gleich mehrere besondere Merkmale des ASC-Modells und von F-Logic als Modellierungs- und Abfragesprache verwendet. Dazu zählt z.B. die Verwendung der implizit generierten Skala `ImpScale`. Diese wird zur Laufzeit über die InterOperation `SpatialDistanceFromSymbolInterOp@(string symbolicPosition)` in Abhängigkeit von einer symbolischen Referenzposition `@("CeBIT")` als Menge aller Entfernungen der bekannten Positionsinformationen von der Referenzposition erzeugt.

Auch hier obliegt die Ausführung der Operation `SpatialDistanceFromSymbolInterOp` wieder dem Service Framework. Das ASC-Konzept der Operationen ist jedoch in der *OntoBroker* Inferenzmaschine nicht nativ bekannt. Daher wird in der prototypischen Umsetzung die bereits erwähnte Erweiterungsfähigkeit dieser Inferenzmaschine mittels *Builtins* dazu verwendet, die durch Operationen modellierten Abläufe über das Builtin `useOperation` in das System zu integrieren.

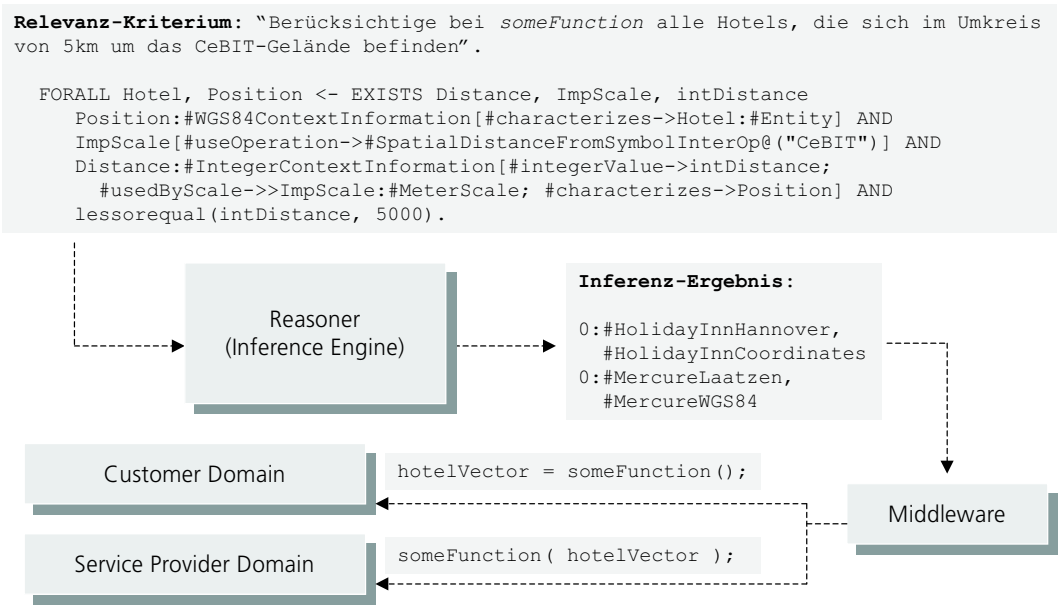

Abbildung 4.14: Implizite Skalen und ihre Verwendung in Relevanzkriterien

Die einzelnen Elemente der implizit generierten Skala, d.h. alle `Distance` Instanzen, sind Meta-Kontextinformationen in Bezug auf die Positionsinformation, da sie deren Distanz zur Referenzposition beschreiben. Da die `Distance`s Instanzen vom Wrapper-Konzept `IntegerContextInformation` sind und keine besondere Metrik-Operation angegeben ist, kann der numerische Wert der Instanz als `intDistance` ausgelesen und innerhalb einer numerischen Relation `lessorequal` verwendet werden.

4.7 Zusammenfassung

In diesem Kapitel wurden verschiedene Anwendungsmöglichkeiten des Modells und der Umsetzungen als Ontologiesprache zur Spezifikation und Auswertung von kontextueller Service-Interoperabilität vorgestellt.

Hierbei stand zunächst die Systemarchitektur im Vordergrund, die eine automatisierte Auswertung des spezifizierten kontextuellen Wissens ermöglicht. Die Architektur setzt das in Kapitel 2 vorgestellte, kontextuell erweiterte Dienstmodell um. Eine Besonderheit dieser Architektur ist die Ausgliederung der Aktoren, die in die Auswertung von Relevanzkriterien sowie die Bereitstellung von Kontextinformationen involviert sind, in eine eigene *context provider domain*. Durch eine Anbindung dieser Domäne an die Dienstvermittlung der Middleware des Service Frameworks konnte exemplarisch für ein Dienstvermittlungsprotokoll gezeigt werden, wie auf der Basis des in Kapitel 3 vorgestellten Modells spezifiziertes Wissen die Dienstauswahl beeinflußt. Somit steuert der Kontext die Dienstinteraktion bereits zeitlich vor der eigentlichen Ausführung eines Dienstes.

Die kontextuelle Ebene der Service-Interoperabilität ist im Anwendungsbereich nicht isoliert zu betrachten. Vielmehr gibt es verschiedene Anknüpfungspunkte zur Signatur-, Protokoll- und Semantik-Ebene. Exemplarisch wurde gezeigt, wie Aspekte und/oder Skalen über *context bindings* mit einem Ein- oder Ausgabeparameter einer Operation eines Dienstes verknüpft werden können. Hierdurch können u.a. Verfahren der einen Interoperabilitätsebene auf Informationen der anderen Interoperabilitätsebene zurückgreifen.

Ähnlich verhält es sich mit der Verwendung des ASC-Modells als Transfer-Modell. Die Ausgewogenheit zwischen festgelegtem Umfang der Modell-Konzepte und der Erweiterbarkeit unter Verwendung von Ontologien ist hinreichend flexibel, um die charakteristischen Merkmale anderer Kontextmodelle größtenteils mit abdecken zu können. Das ASC-Modell bildet somit eine gute Grundlage als Meta- oder Transfer-Modell zur Abbildung zwischen unterschiedlichen Kontextmodellen.

Zur Verdeutlichung der Vielfältigkeit der Anwendung des ASC-Modells wurde ein Katalog von ontologischen Unterkonzepten und deren Instanzen vorgestellt. Basierend auf Informationen dieser Art entsteht ein Netzwerk mit ineinander in Beziehung stehender Fakten. Ein Reasoner kann durch Analyse des Netzwerks Informationen über Zusammenhänge und Abhängigkeiten der Knoten des Netzwerks ermitteln. Auf diese Weise kann ein Reasoner z.B. in Verbindung mit *context bindings* Informationen zur Kompatibilität – eine der Perspektiven der Service Interoperabilität – zu und zwischen Diensten liefern.

Im letzten Teil des Kapitels wurde noch einmal vertiefend auf die Bedeutung von Relevanzkriterien eingegangen. Relevanzkriterien werden unter Verwendung geeigneter Aktoren und Interfaces in der Architektur aus allen Domänen an die *context provider domain* übermittelt. Dort werden sie ausgewertet und erlauben somit die Bereitstellung der relevanten Kontextinformationen. Im vorgestellten Ansatz wird zur Formulierung von Relevanzkriterien auf die Konzepte des ASC-Modells zurückgegriffen, wodurch der eingesetzte Reasoner aus dem mittels der Context Ontology Language spezifizierten Wissen die relevanten Kontextinformationen ableiten kann. Es konnte vermittelt werden, daß Relevanz unter Verwendung einer Vielzahl

verschiedener Aspekte beschrieben werden kann und damit deutlich mehr als örtliche und zeitliche Nähe darstellt.

Ein weiteres spezifisches Anwendungsgebiet wird im anschließenden Kapitel behandelt. Bei den dort betrachteten Handover-Verfahren wird insbesondere der Vorteil kontextadaptiver Dienstinteraktionen deutlich.

Kapitel 5

Kontextbasierte Service Handover

Der Vorteil kontextadaptiver Dienstinteraktionen wird insbesondere auch bei der Betrachtung von Handover-Verfahren deutlich. Voraussetzung für Handover-Verfahren aller Art ist die Interoperabilität – insbesondere aus der Perspektive der Ersetzbarkeit (siehe Abschnitt 2.2.1) – zwischen verschiedenen Komponenten des jeweiligen Systems. Die Perspektive der Ersetzbarkeit steht daher im Fokus dieses Kapitels.

Handover-Verfahren haben eine große Bedeutung in Kommunikationssystemen aller Art, in denen der zu erbringende Dienst ein Kommunikationsdienst ist. Zahlreiche Varianten von Handover-Verfahren wurden für Kommunikationssysteme entworfen und befinden sich auch im produktiven Einsatz. In allgemeinen Dienstumgebungen, in denen der Dienstbegriff etwas weitreichender definiert ist und ein Dienst nicht zwingend ein Kommunikationsdienst sein muß (z.B. Fotoentwicklungsdienst), stellt sich ebenfalls die Frage nach der Anwendbarkeit von Handover-Verfahren.

Dieses Kapitel beginnt daher in Abschnitt 5.1 mit einer Analyse der Prinzipien des Handovers in Kommunikationssystemen und ihrer Klassifikationsmerkmale. Im Anschluß daran wird betrachtet, inwieweit sich die Ergebnisse der Analyse der Handover-Verfahren in Kommunikationssystemen wie GSM [4] auf Handover-Verfahren für allgemeine Dienstumgebungen übertragen und anwenden lassen. Hierzu werden in Abchnitt 5.2 zunächst die wesentlichen und im Zusammenhang mit Handover-Verfahren wichtigen Unterschiede zwischen Kommunikationsdiensten und allgemeinen Diensten herausgearbeitet. In Abschnitt 5.3 wird anschließend gezeigt, wie Handover-Verfahren in allgemeinen Dienstumgebungen insbesondere von der Einbeziehung der kontextuellen Interoperabilitätsebene als Entscheidungsgrundlage für das jeweilige Handover-Verfahren profitieren [156]. Dies wird u.a. am Übergang von physikalischen Verfügungsbereichen eines Kommunikationsdienstes (z.B. die Empfangsreichweite als limitierender Faktor eines zellularen Netzes) auf logische Verfügungsbereiche (z.B. auf der Basis von Öffnungszeiten oder gesetzlichen Vorgaben) bei allgemeinen Diensten gezeigt. Die notwendigen Erweiterungen des Service Frameworks werden vorgestellt und anhand ausgewählter Beispiele demonstriert.

5.1 Das Prinzip der Service Handover

Der Dienst, der von jedem Kommunikationssystem primär bereitgestellt und nach Aktivierung zwischen zwei oder mehr Komponenten erbracht wird, ist der Kommunikationsdienst (*carrier service*), dessen Instanz eine logische Kommunikationsverbindung ist. Als *Handover*-Verfahren bezeichnet man bei Kommunikationsdiensten den Austausch einzelner Komponenten oder Gruppen von Komponenten des Kommunikationssystems während einer Kommunikationsverbindung, sofern während des Austauschs die Kommunikationsverbindung aufrecht erhalten bleibt. Weitere Voraussetzung für die Betrachtung als Handover ist, daß der Kommunikationsdienst bereits aktiviert ist: Sofern keine Kommunikationsverbindung besteht, muß auch keine Vorgänger-Komponente ersetzt werden. D.h. ein einfacher Kommunikationsaufbau zum neuen Partner fällt nicht unter die Handover-Verfahren[1].

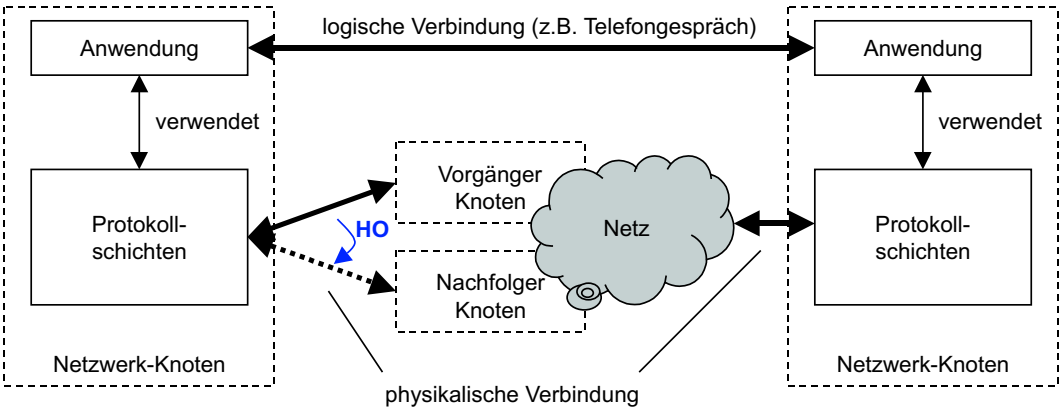

Abbildung 5.1: Handover einer logischen Verbindung

Sowohl der Austausch von Hardware-Komponenten (z.B. Basisstationen eines Mobilfunknetzes) als auch von Software-Komponenten (z.B. das eingesetzte Übertragungsprotokoll) können unter Beibehaltung einer logischen Verbindung als Handover betrachtet werden. Der Ort, an dem sich zu ersetzende und die ersetzende Komponente in der Infrastruktur befinden, ist für die Betrachtung des Handovers zunächst unwichtig. Einerseits kann es sich – wie in Abbildung 5.1 dargestellt – um Komponenten innerhalb der Infrastruktur wie beispielsweise zwei Basisstation handeln. Andererseits ist auch ein Handover zwischen Eck-Komponenten wie zwei Mobiltelefonen möglich, um eine gewisse Unabhängigkeit des Nutzers von einem spezifischen Endgerät zu erreichen (vgl. Implikationen der Mobilität in Abschnitt 2.4.1). Im Folgenden wird diesem Sachverhalt durch Abstraktion in der Art Rechnung getragen, daß die Komponente, die durch einen Handover einen neuen Kommunikationspartner erhält, als statische Entität (static entity) bezeichnet wird, während die ersetzte Komponente (Vorgänger-Entität, predecessor entity) und die ersetzende Komponente (Nachfolger-Entität, successor entity) als dynamische Entitäten bezeichnet werden.

[1] Bewegt man beispielsweise ein Mobiltelefon im ausgeschalteten Zustand in ein Umfeld einer anderen Basisstation, so wird lediglich ein erneuter Anmeldevorgang durchgeführt, nicht jedoch ein Handover initiiert.

Der Austausch einer Komponente im Rahmen eines Handover-Verfahrens setzt voraus, daß die Vorgänger-Komponente mit der Nachfolger-Komponente *interoperabel* bezüglich des aktiven Kommunikationsdienstes ist. Hierbei ist die Interoperabilität der an einem Handover beteiligten Komponenten nicht nur im Sinne der *Kompatibilität* notwendig, sondern insbesondere auch aus der Perspektive der *Ersetzbarkeit*. Letztere wurde in Abschnitt 2.2.1 bereits ausführlich diskutiert und findet in diesem Kapitel häufig Anwendung.

Handover-Verfahren setzen in der Regel eine Infrastruktur voraus, die für die Durchführung und Überwachung aller notwendigen Schritte zur Initiierung und Ausführung eines Handovers zwischen unterschiedlichen Anbieter- oder Nutzer-Domänen zuständig ist[2]. Handover-Verfahren, die gelegentlich zudem als *Handoff*-Verfahren bezeichnet werden, wurden daher in der Vergangenheit für eine Vielzahl von Architekturen entworfen und kommen sehr erfolgreich in produktiven Systemen zum Einsatz, z.B.:

GSM Handover-Verfahren sind eine Schlüsselkomponente in mobilen Telefonnetzen wie GSM, wo ein mobiles Terminal (Endgerät) von einer Basisstation zur nächsten – möglichst unbemerkt für den mobilen Teilnehmer – übergeben wird („handed over"), während sich der mobile Teilnehmer durch die zellulare Struktur des Netzes bewegt [4]. Der zellulare Aufbau des Netzes bestimmt dabei maßgeblich durch die Größe der Zelle in Verbindung mit der Bewegungsgeschwindigkeit des Nutzers, wie oft ein Handover zwischen den Basisstationen der Zellen ausgeführt werden muß. Die Größe der Zelle ist proportional zur maximalen Sende-Reichweite von Basisstation und Endgerät.

Mobile IP Ein breites Anwendungsfeld haben Handover-Verfahren z.B. auch im Bereich *Mobile IP*. Hier macht weniger die Sendeleistung Handover-Verfahren notwendig, als das Adressierungsschema vom IP. Durch das Routing der IP-Pakete über den Netzwerk-Anteil der IP-Adresse kann eine Ortsveränderung eines mobilen Computers nicht unmittelbar auf die jeweilige physikalische Verbindung abgebildet werden: Weder die vorherige IP-Adresse noch eine eventuell neu zugewiesene IP-Adresse des lokalen IP-Netzes lösen das Routing-Problem. Mobile IP begegnet diesem Problem durch verschiedene Infrastruktur-Komponenten. Ein sogenannter *Home Agent* repräsentiert die IP-Adresse, unter der das mobile Endgerät dauerhaft im Heimatnetz ansprechbar ist. Befindet sich das mobile Endgerät mit dem sogenannten *Mobile Agent* in einem fremden Netz, kontaktiert er den dortigen *Foreign Agent*, der daraufhin die von ihm vergebene, temporäre *Care-Of-Adresse* per Tunneling dem Home Agent des Mobile Agent mitteilt. Empfängt der Home Agent nun eine Anfrage von einer Entität, dem sogenannten *Correspondent Host*, so leitet er diese an die bei ihm derzeit registrierte Care-Of-Adresse weiter. Der durch einen Ortswechsel motivierte Austausch der alten Care-Of-Adresse durch eine neue Care-Of-Adresse nach dem oben skizzierten Verfahren ist das Handover-Verfahren von Mobile IP.

Wie das Beispiel Mobile IP zeigt, ist eine wichtige Aufgabe der Infrastruktur neben den Signalisierungs- und Steuerungsfunktionen die Verwaltung und Distribution von Informatio-

[2] Handover-Verfahren innerhalb einer Domäne (Intra-Domain-Handover) können ähnlich wie Handover-Verfahren zwischen unterschiedlichen Domänen (Inter-Domain-Handover) abgewickelt werden, müssen dies aber nicht. Intra-Domain-Handover sind in der Regel proprietär und werden deshalb im Rahmen dieser Arbeit nicht weiter betrachtet.

nen, die mit der aktiven Kommunikationsverbindung assoziiert sind. Dazu zählen Adressierungsdaten, Abrechnungsdaten, QoS-Vorgaben etc. Im Folgenden werden diese Daten als *Sessioninformationen* bezeichnet, da sie jeweils zu einer individuellen, aktiven Datenverbindung (Session) gehören. Die Verwaltung und Distribution von Sessioninformationen ermöglicht insbesondere auch die Mobilität einer Sitzung (session mobility, siehe Abschnitt 2.4.1) in mobilen Systemen, speziell auch in ihrer Ausprägung als Ubiquitous Computing System.

Allgemein gliedert sich jeder Handover-Prozeß in zwei Phasen:

1. Entscheidungsphase (handover decision phase)
2. Ausführungsphase (handover execution phase)

Aufgabe der Entscheidungsphase ist es, eine Entscheidung herbeizuführen, ob und wenn ja wann und zu wem ein Handover durchgeführt werden soll. Dieser Vorgang wird durch den Übergang in die Ausführungsphase initiiert (handover initiation). In beiden Phasen gibt es viele Merkmale, die den Handover auf die eine oder andere Art charakterisieren. Die für diese Arbeit wichtigsten Klassifikationsmerkmale werden im folgenden Abschnitt im Detail vorgestellt.

Klassifikationsmerkmale

Zu Handover-Verfahren in Kommunikationsdiensten wurden bereits zahlreiche wissenschaftliche Untersuchungen gemacht. Die Ergebnisse der zugehörigen Publikationen stützen sich auf eine Reihe von Klassifikationsmerkmalen der jeweils untersuchten Handover-Verfahren. Die folgenden Klassifikationsmerkmale stehen bei dem im weiteren Verlauf dieses Kapitels vorgestellten Ansatz im Vordergrund:

- Grad der Transparenz: Fast, Smooth und Seamless
- Grad der funktionalen Äquivalenz: Horizontal und Vertikal
- Grad der Parallelität: Soft und Hard
- Handlungsspielraum: Reaktiv und Proaktiv
- Rollenbasierte Aufgabenverteilung

Diese Klassifikationsmerkmale werden im Folgenden näher erläutert.

Grad der Transparenz: Fast, Smooth und Seamless

Ein wichtiges Transparenzmerkmal ist die Verzögerung, die sich durch die Durchführung eines Handovers im Vergleich zur Weiterführung der Kommunikationsverbindung mit der Vorgängerentität ergibt. Allgemein läßt sich festhalten, dass ein Handover immer schnellstmöglich durchgeführt werden soll. Einen Handover mit minimaler Verzögerung nennt man einen *Fast*

Handover [113]. Dabei bestimmt sich die Verzögerungszeit des Handovers aus der Zeitspanne, in der die statische Entität keine Kommunikationsmöglichkeit mit einer der dynamischen Entitäten hat.

Eine andere Perspektive der Transparenz ist der Datenverlust, der sich in der o.g. Zeitspanne ereignet. Offensichtlich ist der Datenverlust während des Handovers zu minimieren, um den Grad der Transparenz für die beteiligten Entitäten zu erhöhen. Eine Ersetzung ohne oder mit minimalem Datenverlust nennt man *Smooth Handover* [113].

Während beim Fast Handover die höchste Priorität auf die Minimierung der Verzögerung durch den Handover gelegt wird, liegt beim Smooth Handover die höchste Priorität auf der Minimierung des Datenverlustes. Optimal wäre eine vollständige Transparenz der Ersetzung des Kommunikationspartners für die statische Entität. Erzielt oder am besten approximiert wird dies durch einen *Seamless Handover* [113], der sich aus der Kombination aus Fast und Smooth Handover ergibt.

Nur in seltenen Fällen kann eine vollständige Transparenz erreicht werden. Deshalb muß je nach Art der Kommunikation abgewägt werden, welche Komponente der Transparenz als wichtiger anzusehen ist. Ist, wie bei der menschlichen Sprache, ein kurzzeitiger Datenverlust eher akzeptabel als eine Verzögerung, tendieren entsprechende Kommunikationssysteme wie GSM oder VoiceOverIP eher zum Fast Handover [47], wohingegen Systeme zur reinen Datenübertragung eher zum Smooth Handover mit Datenpufferung tendieren und dabei höhere Verzögerungszeiten in Kauf nehmen.

Grad der funktionalen Äquivalenz: Horizontal und Vertikal

In Abschnitt 2.2.1 ist bereits eine Einführung in die Thematik der funktionalen Äquivalenz erfolgt. Sind die Funktionalitäten der dynamischen Entitäten identisch und variiert die Kommunikationsverbindung vor und nach der Ersetzung höchstens in einigen qualitativen Parametern, so nennt man den Ersetzungsprozeß einen *horizontalen Handover*.

Demgegenüber steht der *vertikale Handover*, bei dem die dynamischen Entitäten eine unterschiedliche Funktionalität bereitstellen. Vertikale Handover erfordern in der Regel einen wesentlich höheren Signalisierungs-, Steuerungs- und Adaptionsaufwand als horizontale Handover. Die Auswirkungen eines vertikalen Handovers sind daher in der Regel wesentlich weniger transparent für den Nutzer als die Auswirkungen eines horizontalen Handovers.

Das Standard-Beispiel für einen horizontalen Handover ist der Zellenwechsel im GSM, bei der die dynamischen Entitäten (Basisstationen) eine äquivalente Funktionalität bereitstellen [4]. Demgegenüber ist der Handover eines mobilen Endgeräts zwischen einer GSM-Basisstation und beispielsweise einer DECT-Basisstation unter Beibehaltung einer aktiven Kommunikationsverbindung den vertikalen Handovern zuzuordnen. Dabei unterscheiden sich die zugrundeliegenden Systemkomponenten teilweise erheblich, wodurch ein erhöhter Adaptionsaufwand z.B. bei den verwendeten Protokollen entsteht. Ein weiterer interessanter Sonderfall eines (doppelten) vertikalen Handovers ist die Überführung der aktiven Telefonverbindung in einen *parked state* und anschließende Reaktivierung. D.h. die aktive Kommunikationsverbindung

wird nicht abgebaut, sondern vorübergehend „geparkt“, um die gleiche Verbindung nach kurzer Unterbrechung (z.B. nach Durchfahren eines Tunnels oder nach dem Umstecken eines Endgerätes an eine andere Telefondose) wieder zu nutzen.

Grad der Parallelität: Soft und Hard

Ist zu jedem Zeitpunkt des Handover-Verfahrens lediglich eine Verbindung zwischen der statischen und jeweils einer der beiden dynamischen Entitäten möglich, so spricht man von einem *Hard Handover*, da es einen harten Einschnitt zwischen der ursprünglichen Kommunikation mit dem Vorgänger und der neuen Kommunikation mit dem Nachfolger gibt.

Können jedoch im Rahmen des Handover-Verfahrens zu einem Zeitpunkt mehrere Verbindungen zu den dynamischen Entitäten parallel genutzt werden, erlaubt dies wesentlich größere Freiheitsgrade beim Design des gesamten Handoverprozesses. Man spricht in diesem Fall von einem *Soft Handover*, da der Ersetzungsprozeß im Vergleich zum Hard Handover wesentlich weicher abläuft: Auch für eine gewisse Zeit nach dem Initiieren des Handovers können noch Daten zwischen der statischen Entität und der Vorgänger-Entität ausgetauscht werden. Die Menge aller Kommunikationspartner, die eine Kommunikation mit der statischen Entität betreiben, nennt man auch *active set*. Die übliche Vorgehensweise beim Soft Handover ist es, präventiv eine Kommunikationsverbindung zur Nachfolger-Entität parallel zur aktiven Verbindung zur Vorgänger-Entität aufzubauen und dabei bereits einen Teil der den Handover begleitenden Signalisierungs- und Steuerungsprotokolle mit dem Nachfolger abzuwickeln. Ist der Zeitpunkt der eigentlichen Ersetzung gekommen, brauchen dann nur noch der „Rest“ der Prozedur durchgeführt und die alte Verbindung zum Vorgänger abgebaut werden.

Beispiele zum Unterschied zwischen Hard und Soft Handover werden u.a. von Wong und Lim in [177] vorgestellt.

Handlungsspielraum: Reaktiv und Proaktiv

Die zeitliche Länge der Entscheidungphase eines Handovers hat großen Einfluß auf den Handlungsspielraum während des gesamten Handover-Verfahrens. Sie wird durch die Reihenfolge der Ereignisse „Problem erkannt“ und „Handeln“ bestimmt.

Lassen sich allgemein Anzeichen für das Eintreten eines Problems erkennen und eine Gegenmaßnahme einleiten, bevor das Problem auftritt, so spricht man von einem proaktiven Ansatz (*make before break*). Dem entsprechend nennt man einen Handover, der aufgrund eines ex ante erkennbaren Problems initiiert wird, einen *proaktiven Handover*.

Ist im Gegensatz dazu das Problem vor dem Eintreten nicht ersichtlich und/oder keine passende Gegenmaßnahme verfügbar, so kann nur noch ein reaktiver Ansatz eingesetzt werden (*break before make*). Der in einem solchen Fall nur noch mögliche *reaktive Handover* schränkt die verfügbaren Optionen für den Handover stark ein. So ist beim reaktiven Handover beispielsweise kein Datentransfer zwischen statischer Entität und Vorgänger-Entität nach Eintreten des Problems mehr möglich.

In der Literatur wird anstatt *proaktiv* auch der Ausdruck *prediktiv* verwendet (vgl. [127]). Das Wort *prediktiv* drückt dabei insbesondere aus, daß man den Problemeintritt vorhersagen kann, z.B. weil sich der Empfangspegel des Signals von der aktuellen dynamischen Entität langsam von oben einem unteren Grenzwert nähert, während der Empfangspegel einer anderen dynamischen Entität bereits über einem bestimmten Schwellwert ist und weiter steigt (siehe Abbildung 5.3 auf Seite 126). Proaktiv bedeutet demzufolge, daß eine Entscheidung zugunsten eines alternativen Kommunikationspartners gefallen ist, und dieser nun für die weitere Nutzung bevorzugt wird.

Ein Beispiel für Proaktivität ist beim Zellenwechsel in GSM durch das terminalseitige, periodische Messen der Empfangsstärke der 16 stärksten Basisstationen am jeweiligen Aufenthaltsort gegeben. Nimmt das Empfangssignal der aktuellen Basisstation stetig ab, so ist in naher Zukunft mit dem Verlassen des Verfügungsbereichs dieser Basisstation zu rechnen. Existieren nun eine oder mehrere Basisstationen, deren Empfangspegel einen höheren Wert aufweisen, kann dem Abbruch der Kommunikation durch einen proaktiven Handover zu einer bezüglich der Empfangsstärke besseren Basisstation vorgebeugt werden.

Rollenbasierte Aufgabenverteilung

Die Verteilung einzelner Aufgaben im Rahmen des Handover-Vorgangs erfolgt meistens rollenbasiert. So gibt es z.B. die Rolle des *controllers*, der den korrekten Ablauf des Handover-Protokolls überwacht und steuert. Oder auch die Rolle des *initiators*, der die Notwendigkeit eines Handovers feststellt und einleitet.

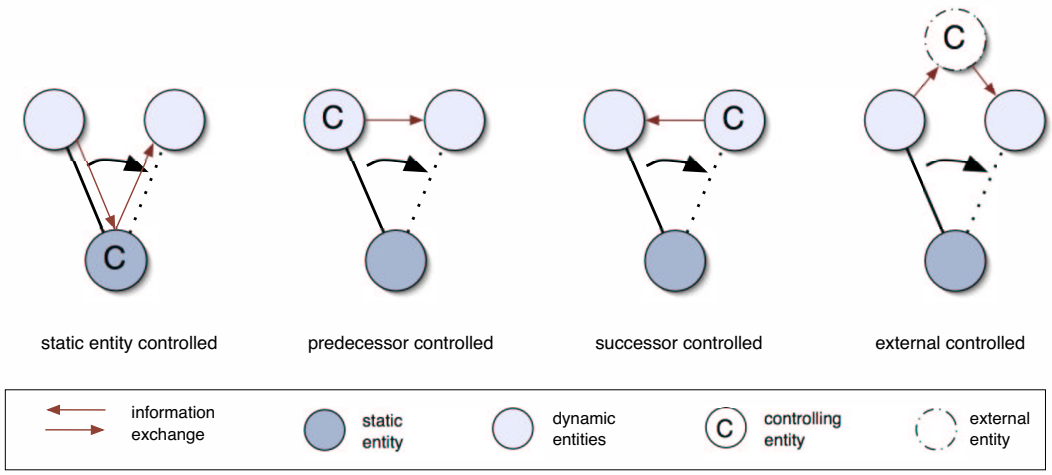

Abbildung 5.2: Position der kontrollierenden Entität

Die Entscheidung, welche der statischen oder dynamischen Entitäten bzw. anderen Komponenten der Infrastruktur eine dieser Rollen einnimmt (siehe Abbildung 5.2), hat unmittelbar Einfluß auf die Art des anwendbaren Handover-Verfahrens. So erfordern beispielsweise alle Handover-Verfahren, bei denen die Rolle des *controllers* bei einer dynamischen Entität

liegt, eine Kommunikationsverbindung *zwischen* den dynamischen Entitäten zumindest für die Zeit der Durchführung des Handovers (z.B. zum Austausch von Sessioninformationen über die aktive Verbindung zur statischen Entität). Diese Kommunikationsverbindung zwischen Vorgänger- und Nachfolger-Entität, der sogenannte *Backlink*, ist aber nicht immer gegeben (z.B. häufig nicht bei vertikalen Handovern). Anders ausgedrückt, wenn es keinen Backlink zwischen den dynamischen Entitäten gibt, sind diese auch nicht direkt kompatibel im Sinne der Perspektive der Interoperabilität *zueinander*, können aber beide jeweils für sich kompatibel zur statischen Entität sein, weshalb eine Ersetzbarkeit der dynamischen Entitäten dennoch gegeben sein kann (siehe Abschnitt 2.2.1).

Die Entität, welche die Rolle des *initiators* hat, trifft die Entscheidung, ob ein Handover stattfinden soll und leitet diesen ggf. ein. Dieser Entität müssen daher alle für den Entscheidungsfindungsprozeß in der Entscheidungsphase wichtigen Parameter bekannt sein, um diese Parameter auch berücksichtigen zu können. Ein vor allem bei drahtlosen Kommunikationsverbindungen als Instanz eines Kommunikationsdienstes wichtiger Parameter ist die Signalstärke eines empfangenen Pilotsignals von den Basisstationen der Umgebung, siehe Abbildung 5.3. Dieser Parameter steht nach Prakash und Veeravalli [129] oft als einzige Grundlage für eine Handover-Entscheidung auf der Seite des mobilen Terminals zur Verfügung. Die Auswertung ist auf ein Pilotsignal beschränkt, sofern keine parallelen Kommunikationsverbindungen zu den anderen Basisstationen als der aktuellen möglich sind (Hard Handover).

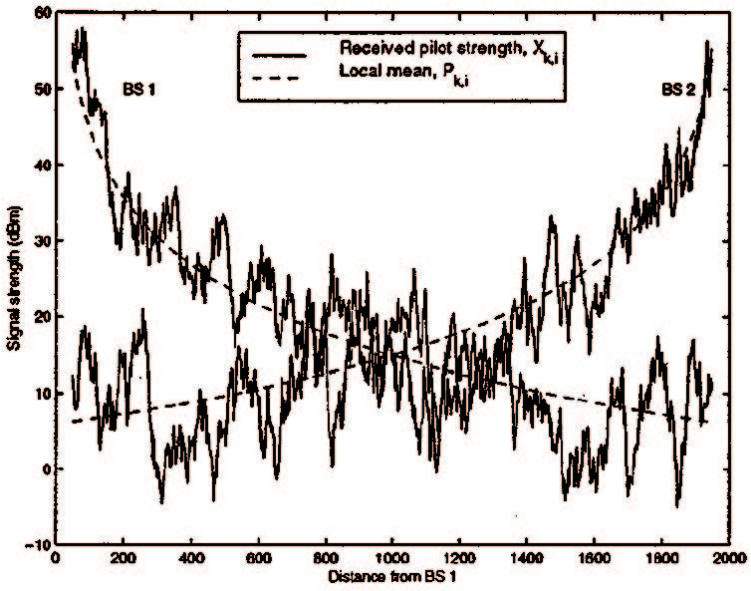

Abbildung 5.3: Feldstärkemessungen als Trigger-Event der Handover-Initiierung

Es kann aber auch sinnvoll sein, daß neben Parametern der aktuellen Verbindung (z.B. Empfangsfeldstärke als Maß des Verfügungsbereichs einer Basisstation) auch auf andere Parameter der Infrastruktur (z.B. aktuelle Auslastung potentieller Nachfolger-Entitäten) bei der Entscheidung für den Handover und für die Auswahl der Nachfolger-Entität zurückgegriffen wird. In Systemen, bei denen diese Informationen beispielsweise auf dem mobilen Terminal

nicht zur Verfügung stehen, wird die Rolle des *initiators* typischerweise von einer Festnetzkomponente erfüllt. Beim Design des GSM Systems hat man sich aus den genannten Gründen z.B. für ein *proactive - hard - predecessor initiated - predecessor controlled* Handover Verfahren entschieden.

5.2 Highlevel Services und Handover

Die in Abschnitt 5.1 beschriebenen Grundlagen und Differenzierungsmerkmale von Handover-Verfahren basieren auf Analysen von Handover-Verfahren in Kommunikationssystemen wie GSM und Mobile IP. In Kommunikationssystemen ist der bereitgestellte Dienst ein Kommunikationsdienst (*carrier service*), dessen Instanzen Kommunikationsverbindungen sind.

Interessante Einsatzgebiete für Handover-Verfahren sind aber auch in Systemen zu finden, die einen allgemeineren Dienstbegriff zugrunde legen. Dazu zählen insbesondere Systeme, welche den Servicebegriff gemäß Definition 1 auf Seite 6 als „eine eindeutig identifizierbare Instanz, die für die Beschaffung von Informationen oder für die Ausführung von Aktionen mit spezifischen Merkmalen verantwortlich ist“ verstehen, den Dienst also nicht auf einen Kommunikationsdienst beschränken. Diese Verallgemeinerung des Dienstbegriffs wird auch durch die Bezeichnungen *non-carrier services (NCS)* oder alternativ *highlevel services* zum Ausdruck gebracht, die im Folgenden häufig Verwendung finden. Beispiele für NCS sind Routingdienste, Ticket-Reservierungsdienste oder auch ein Fotoentwicklungsdienst, wie er in Abschnitt 5.3.3 noch im Detail vorgestellt wird. Gerade Highlevel Services erhalten in der heutigen Zeit immer größere Anerkennung und weite Verbreitung, vor allem durch den ständigen Zuwachs von E-Commerce-, E-Marketing-, E-Learning- etc. Diensten.

Die Instanzen von NCS sind keine Kommunikationsverbindungen, sondern von allgemeinerer Natur. Deswegen wird zur besseren Unterscheidung von Dienstinteraktionen gesprochen, wenn Instanzen von NCS gemeint sind. Ziel einer Dienstinteraktion kann die einmalige (z.B. Preisinformationen), periodische (z.B. News-Service) oder dauerhafte (z.B. Routinginformationen) Bereitstellung von Informationen oder die Ausführung von Aktionen (z.B. Pizza-Bestellung) sein, deren spezifische Merkmale Ein- oder Ausgabeparameter oder die Veränderung des Status des Dienstes sind.

Wie bereits in Abschnitt 2.3.3 erwähnt, ist das MNMplusCE Dienstmodell ebenfalls geeignet, Higlevel Services zu modellieren. Das bedeutet, jede Dienstinteraktion kann als Nutzung einer Entität der *customer domain* eines Dienstes modelliert werden. Dieser Dienst wird von einer Entität der *service provider domain* bereitgestellt (wobei jede Entität mehrere, verschiedene Rollen einnehmen kann). Jede NCS Dienstinteraktion erfordert die Kompatibilität als Perspektive der Interoperabilität zwischen Dienstnutzer und Dienstanbieter. Ein Handover-Verfahren implementiert die Ersetzbarkeit als Perspektive der Interoperabilität (*NCS handover*).

Die Besonderheiten des NCS Handovers im Vergleich zum Handover von Kommunikationsdiensten liegen im Unterschied der Dienstinstantiierung und dessen Implikationen. Beim Handover einer Kommunikationsverbindung als Instanz eines Kommunikationsdienstes wer-

den einzelne Komponenten oder Gruppen von Komponenten des zugrundeliegenden Kommunikationssystems unter der Prämisse der Weiterführung der Kommunikationsverbindung ausgetauscht. Beim Handover einer allgemeinen Dienstinteraktion als Instanz eines NCS werden einzelne Komponenten oder Gruppen von Komponenten des zugrundeliegenden verteilten Systems durch das Service Framework unter der Prämisse der Erfüllung des Dienstinteraktions-Ziels ausgetauscht. Während das Ziel einer Kommunikationsverbindung offensichtlich die Kommunikationsmöglichkeit über die Verbindung ist, muß das Ziel einer NCS Dienstinteraktion als *shared knowledge* spezifiziert sein, um als Parameter der Handover-Entscheidung Berücksichtigung zu finden. Diese Spezifikation des Ziels einer Dienstinteraktion erfolgt sinnvollerweise auf der semantischen Interoperabilitätsebene als *post-condition*.

Prinzipiell ist die in Abschnitt 5.1 vorgestellte Differenzierung von Handover-Verfahren auch auf NCS Handover-Verfahren anwendbar. Dementsprechend kann man NCS Handover-Verfahren nach dem Grad der Transparenz (Fast, Smooth und Seamless), dem Grad der funktionalen Äquivalenz (Horizontal und Vertikal), dem Grad der Parallelität (Hard und Soft), dem Handlungsspielraum (Reaktiv und Proaktiv), sowie der Position der initiierenden und der kontrollierenden Entitäten in der Architektur unterscheiden. Manche dieser Differenzierungsmerkmale können jedoch bei NCS Handover-Verfahren nicht in ihrer originären Form angewendet werden. Insbesondere müssen die darauf basierenden existierenden Protokolle (z.B. zur Behandlung von Sessioninformationen), sowie die Bedeutung von Merkmalen (z.B. die Interpretation der Parallelität von Dienstinteraktionen hinsichtlich Zeitinvarianz[3], Kommutativität[4] und Idempotenz[5]) angepaßt und erweitert werden. Durch Anwendung des in Kapitel 3 vorgestellten ASC-Modells, der darauf basierenden Context Ontology Language und der in diesem Zusammenhang vorgestellten Systemarchitektur können weite Teile der Problemstellungen beim NCS Handover bereits abgedeckt werden, wie im weiteren Verlauf dieses Kapitels deutlich wird.

Wie bei der Einführung der Ersetzbarkeit in Abschnitt 2.2.1 erläutert, gelten Betrachtungen der Ersetzbarkeit nicht nur für Dienste, sondern ganz allgemein für Komponenten verteilter Systeme. Insbesondere können die dynamischen Entitäten, d.h. die Entitäten, zwischen denen ein Handover-Verfahren den Ersetzungsprozeß beschreibt, aus jeder Domäne des MNMplusCE Dienstmodells sein. Dies korrespondiert auch mit den Betrachtungen zur Mobilität in Abschnitt 2.4.1. Ohne Einschränkung der Allgemeinheit werden im weiteren Verlauf dieses Kapitels exemplarisch Handover-Verfahren für die Ersetzung eines Dienstanbieters durch einen anderen Dienstanbieter betrachtet, d.h. die Ersetzung findet auf der Seite der *service provider domains* statt. Alle Aussagen gelten analog für Ersetzungen in den anderen Domänen.

[3] $f(t_1) = f(t_2)$ und $t_1 \neq t_2$
[4] $f(a) + f(b) = f(b) + f(a)$
[5] $f(x) = f^n(x)$

5.3 Service Handover in einem kontextuellen Framework

Ein Unterschied zwischen dem Handover eines Kommunikationsdienstes und dem NCS Handover ist in der Ursache für einen Handover, der Grundlage für die Handover-Entscheidung, zu finden. Bei Kommunikationsdiensten steht als Ursache für einen Handover vor allem der (erwartete) Abbruch einer Kommunikationsverbindung zwischen der statischen und einer dynamischen Entität im Vordergrund. So gibt es nach De Carolis [48] drei Gründe, die einen Handover notwendig machen können:

1. Der erste Grund ist das Eintreten eines Ereignisses, auf welches gezwungenermaßen reagiert werden muss *Forced Handover*. Der Ersetzungszwang basiert auf dem Verlust von QoS, wie z.B. ein unvorhergesehener Abbruch der Kommunikationsverbindung.
2. Der zweite Grund ist die Veränderung der QoS, weshalb in diesem Fall von *QoS-Aware Handover* gesprochen wird. Für die Initiierung des QoS-Aware Handover, d.h. für die Ersetzung der aktuellen dynamischen Entität, spricht eine durch den Wechsel herbeigeführte Leistungssteigerung bei gleichen Kosten oder die gleiche Leistung bei geringeren Kosten [64, 122].
3. Der dritte Grund bezieht sich auf die Veränderung des aktuellen Aufenthaltsortes der statischen Entität (z.B. das Verlassen einer GSM Zelle), weshalb diese Art von Handover unter dem Begriff *Location-Aware Handover* gefaßt wird.

Bei genauer Betrachtung der von De Carolis angeführten drei Gründe für einen Handover kann man feststellen, daß es sich um eine Veränderung des Kontextes handelt. Diese Veränderung des Kontextes, d.h. die Veränderung des durch Kontextinformationen charakterisierten Zustands einer Entität in mindestens einem für die Dienstinteraktion relevanten Aspekt, ist durch Anwendung des ASC-Modells beschreib- und in einer entsprechenden Architektur auswertbar. Verlust (De Carolis' Grund 1) oder Veränderung (De Carolis' Grund 2) von QoS läßt sich als Relation der entsprechenden Kontextinformationen zu einer gegebenen Skala ausdrücken, da Kontextinformationen QoS-Informationen subsummieren (siehe MNMplusCE Dienstmodell in Abschnitt 2.3.3). Gleiches gilt für eine den aktuellen Aufenthaltsort (De Carolis' Grund 3) charakterisierende Kontextinformation, die basierend auf einem Orts-Aspekt eine bestimmte Relation zu einer den Verfügungsbereich eines Dienstes ausdrückenden Skala hat. Besonders letzteres legt die Verwendung von *context bindings* zur Beschreibung des Verfügungsbereichs eines Dienstes nahe, worauf im folgenden Abschnitt im Detail eingegangen wird.

5.3.1 Service Scopes

Über Bindungen des Kontextes (*context bindings*, siehe Abschnitt 4.2) werden Aspekte und/ oder Skalen des ASC-Modells mit einer Operation eines Dienstes verknüpft („gebunden"). Ein Spezialfall der *context bindings* sind Bindungen von Verfügungsbereichen an einen Dienst (*service scopes*). Hiermit spricht ein Dienstanbieter seine Verfügbarkeit für den Fall aus, daß der Zustand eines festgelegten Parameters Element einer bestimmten Skala ist. Dies wird

in Abbildung 5.4 am Beispiel der Öffnungszeiten illustriert, innerhalb derer ein Restaurant (Dienstanbieter) Essen an einen Besteller (Dienstnutzer) liefert. In diesem Beispiel handelt es sich um einen Scope basierend auf einem zeitlichen Aspekt. Außerhalb der als Skala ausgedrückten Öffnungszeiten erklärt sich der Dienstanbieter für nicht verfügbar. Mit anderen Worten ist es nur dann sinnvoll, den Dienst eines Dienstanbieters zu nutzen, wenn sich dieser durch Angabe eines Scopes auch für (kontextuell) zuständig erklärt.

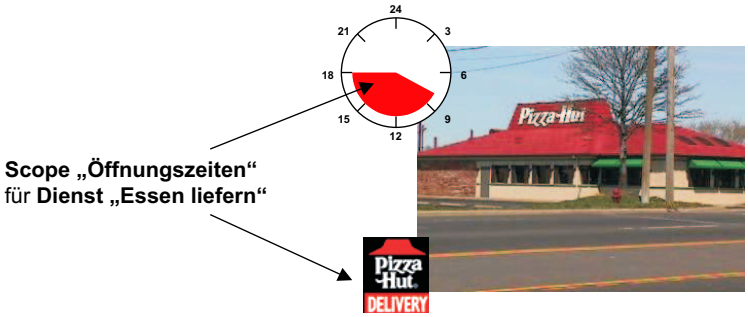

Abbildung 5.4: Beispiel für einen logische Verfügungsbereich

Hauptunterschied zwischen allgemeinen *context bindings* und *scopes* ist, daß erstere Bindungen zu Eingabe- und/oder Ausgabeparametern einer Operation sind, während letztere Bindungen zu einer „impliziten Kontextvariablen“ sind. Diese impliziten Kontextvariablen sind also keine direkten Ein- oder Ausgabeparameter einer Operation, müssen aber vor und während einer Dienstinteraktion mit *gültigen* Kontextinformationen belegt sein, sonst gilt der Dienst als nicht verfügbar. Die Integration von impliziten Kontextvariablen in die WSDL-Beschreibung eines Dienstes wird in Abbildung 5.5 als Erweiterung und Kombination der Beispiele in Abbildung 2.4 (Seite 12) und Abbildung 4.5 (Seite 102) dargestellt.

Über den Mechanismus der Bindungen lassen sich die verschiedensten Parameter bezüglich des Verfügungsbereichs eines Dienstes beschreiben. Die Menge der Parameter geht weit über örtliche und zeitliche Aspekte hinaus, wobei diese beiden Aspekt-Gruppen die gebräuchlichsten sind, um die Verfügbarkeit eines Dienstes zu charakterisieren. In der o.g. WSDL-Beschreibung wird beispielsweise der Eingabeparameter `currentPositionParam` der Operation `showMap` an die Skala `GaussKrueger_ScaleInst` gebunden, was zum Ausdruck bringt, daß der Dienst hier nur die Werte dieser Skala sinnvoll verarbeiten kann. Weiterhin werden die impliziten Kontextvariablen `legalUserParam` und `operationHoursParam` definiert und jeweils an eine Skala gebunden, was zum Ausdruck bringt, daß der Dienst nur als verfügbar gilt, wenn diese impliziten Kontextvariablen mit *gültigen* Werten der jeweiligen Skala belegt sind. Die verwendete Syntax bedient sich dabei wiederum des Elements `wsdl:message` zur Definition der impliziten Kontextvariablen, die dann als `wsdl-cb:scope` der Operation `showMap` zugeordnet werden.

Aufgabe der Scope-Auswertung ist, herauszufinden, ob die aktuelle Belegung der impliziten Kontextvariablen eine gültige Kontextinformation bezüglich der angegebenen Skala ist. Diese Aufgabe wird üblicherweise an das Service Framework delegiert. Dabei wird eine Kontextinformation, die einen Dienstnutzer oder einen anderen für eine Dienstinteraktion relevanten Parameter bezüglich einer Bindung beschreibt, per `memberCheck` auf Zugehörigkeit zur ent-

sprechend zugesicherten Skala geprüft.

Zum Beispiel kann der Name des Benutzers, der den Dienst gerne nutzen möchte, auf Zugehörigkeit zu der in Abbildung 5.5 angegebenen Skala `ServiceSubscriber_661234_ScaleInst` überprüft werden. Bei mehrfachen Bindungen zur gleichen impliziten Kontextvariablen gilt, daß diese aus Sicht des Dienstanbieters als Konjunktion zu interpretieren sind (logisches AND, d.h. der Dienstanbieter sieht sich als zuständig bezüglich aller angegebenen Skalen/Aspekte). Aus Sicht des Dienstnutzers erfolgt die Interpretation als Disjunktion (logisches OR, d.h. der Dienst ist als verfügbar anzusehen, wenn die Kontextinformation in `usedByScale` Relation zu mindestens einer der Skalen des Scopes steht).

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions xmlns:wsdl-cb="http://context-aware.org/cool/wsdl-cb.xsd" ... >
  <!-- ================ MESSAGE section ================= -->
   <wsdl:message name="showMapRequest">
     <wsdl:part name="currentPositionParam" type="xsd:string"
         wsdl-cb:aspect="urn:#GeometricPlace_AspectInst"
         wsdl-cb:scale="urn:#GaussKrueger_ScaleInst" />
   </wsdl:message>
   <wsdl:message name="showMapScope">
     <wsdl:part name="legalUserParam" type="xsd:string"
         wsdl-cb:aspect="urn:#UserId_AspectInst"
         wsdl-cb:scale="urn:#ServiceSubscriber_661234_ScaleInst" />
     <wsdl:part name="openingHoursParam" type="xsd:string"
         wsdl-cb:aspect="urn:#TimePeriod_AspectInst"
         wsdl-cb:scale="urn:#OpeningHoursProviderX_ScaleInst" />
   </wsdl:message>
   ...
   <!-- ================ PORTTYPE section ================ -->
   <wsdl:portType name="ShowMapPortType">
      <wsdl:operation name="showMap" parameterOrder="userId">
         <wsdl:input    message="this:showMapRequest"  name="showMapRequest"/>
         <wsdl:output   message="this:showMapResponse" name="showMapResponse"/>
         <wsdl-cb:scope message="this:showMapScope"    name="showMapScope"/>
      </wsdl:operation>
      ...
   </wsdl:portType>
   ...
</wsdl:definitions>
```

Abbildung 5.5: Beschreibung eines Scopes als WSDL-Erweiterung

Anzumerken bleibt, daß ein Dienstnutzer nicht verpflichtet ist, die Verfügbarkeit eines Dienstes anhand der vom Dienstanbieter veröffentlichten Service Scopes zu überprüfen, bevor eine Dienstinteraktion initiiert wird. Auf der anderen Seite ist die geschilderte Überprüfung der Zuständigkeit vor der Initiierung der Dienstinteraktion sinnvoll, um z.B. unnötige Anfragen an einen Dienstanbieter, möglicherweise über kostenintensive und/oder schmalbandige drahtlose Kommunikationsverbindungen, zu vermeiden.

Service Scopes können aber auch als weiteres Ausschlußkriterium im Rahmen der Dienstvermittlung Verwendung finden, wenn das Ergebnis der Dienstsuche eine Auflistung mehrerer Dienstanbieter ist. Ihr Anwendungsmuster könnte man in dem Fall als *one-shot* bezeichnen, da die Auswertung der Kontextinformation einmalig zum Zeitpunkt der Dienstsuche erfolgt. Wie Service Scopes mit einem *multi-shot* Anwendungsmuster zur Bestimmung *kontextueller Konsistenz* als Grundlage horizontaler NCS Handover zum Einsatz kommen, wird im folgenden Abschnitt erläutert.

Kontextuelle Konsistenz

Wie erläutert, sind Service Scopes Spezialisierungen der Bindungen des Kontextes. Sie drücken die Verfügbarkeit eines Dienstes bezüglich einer Kontextvariablen als Skala gültiger Kontextinformationen aus. Insofern kann die jeweils aktuelle Belegung der Kontextvariablen einen NCS Handover motivieren, wie in Abbildung 5.6 skizziert ist. In dem hier dargestellten Beispiel interagiert ein Dienstnutzer gerade mit dem Dienstanbieter rechts im Bild, z.B. stellt er sich gerade seine Lieblingspizza auf der Webseite des Restaurants zusammen. Sobald das Service Framework feststellt, daß die aktuelle Belegung der zu diesem Dienst gehörenden Kontextvariablen „Aktuelle Uhrzeit“ kurz davor ist, einen durch den Scope vorgegebenen Schwellwert zu überschreiten, kann ein proaktiver, horizontaler NCS Handover zum Dienstanbieter links im Bild eingeleitet werden. Dies setzt voraus, daß ein syntaktisch, protokollarisch und semantisch gleichwertiger neuer Dienst auch kontextuell als verfügbar erkannt wird.

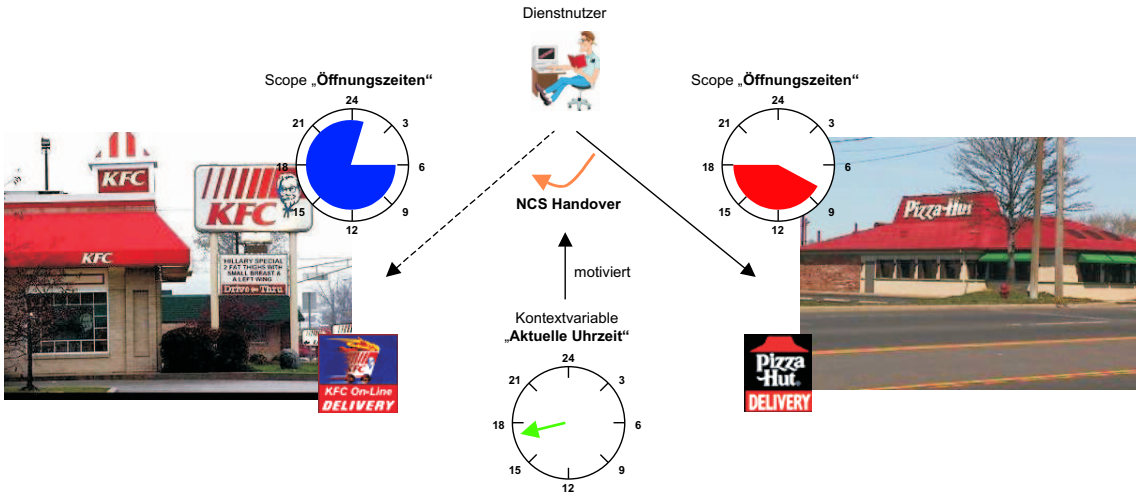

Abbildung 5.6: Beispiel für einen NCS Handover

Im Rahmen dieser Arbeit wird dieser Zustand der beiden Dienstanbieter als *kontextuell konsistent* bezeichnet. Kontextuelle Konsistenz ist also der Zustand, bei dem zwei Dienste bezüglich aller jeweiligen Kontextvariablen für einen Dienstnutzer als kontextuell ersetzbar anzusehen sind. Ist diese kontextuelle Konsistenz zeitlich überlappend oder zumindest zeitlich angrenzend (vgl. Abbildung 5.7), dann ist ein horizontaler NCS Handover zwischen den Diensten

möglich. Im Falle separierter Scopes ist jedoch kein horizontaler, allenfalls ein vertikaler NCS Handover möglich.

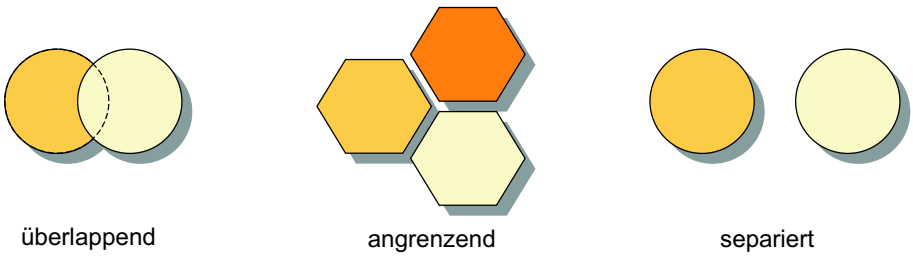

Abbildung 5.7: Scope Konstellationen

Ein *reaktiver* NCS Handover wird notwendig, wenn während einer aktiven Dienstinteraktion eine der für die Dienstinteraktion relevanten Kontextvariablen ab einem bestimmten Zeitpunkt nicht mehr mit einer Kontextinformation belegt ist, die innerhalb des angegebenen Verfügungsbereiches des Dienstanbieters liegt. D.h. die aktuelle Belegung der Kontextvariable ist kein Element der im Scope angegebenen Skala.

Wesentlich mehr Handlungspielraum bleibt dem Service Framework jedoch, wenn das Verlassen des Verfügungsbereichs schon einige Zeit vor dem Verlassen mit einer gewissen Wahrscheinlichkeit indiziert wird. In diesem Fall kann sich das Service Framework auf die Veränderung vorbereiten und ggf. einen *proaktiven* NCS Handover einleiten, was der Transparanz des NCS Handovers gegenüber der statischen Entität zuträglich ist. Proaktivität kann dabei wiederum durch die Anwendung von komplexeren Relevanzregeln erreicht werden, die in Abschnitt 4.6 vorgestellt wurden. Eine solche komplexe Relevanzregel ist in Abbildung 5.8 exemplarisch dargestellt.

```
FORALL openX, remainX, openY, remainY <- EXISTS checkOp
  ((openX is 1) AND
   (#OpeningHoursProviderX_ScaleInst[#hasMember->>#CurrentTime_CIInst] OR
   (#OpeningHoursProviderX_ScaleInst[#memberCheck->checkOp] AND
    #OpeningHoursProviderX_ScaleInst[#useOperation->
      checkOp(#CurrentTime_CIInst)]))) OR
  (remainX::#ContextInformation[#hasScale->>#RemainingSeconds_ScaleInst;
    #characterizes->#CurrentTime_CIInst::#Entity) OR
  ((openY is 1) AND
   (#OpeningHoursProviderY_ScaleInst[#hasMember->>#CurrentTime_CIInst] OR
   (#OpeningHoursProviderY_ScaleInst[#memberCheck->checkOp] AND
    #OpeningHoursProviderY_ScaleInst[#useOperation->
      checkOp(#CurrentTime_CIInst)])))
  (remainY::#ContextInformation[#hasScale->>#RemainingSeconds_ScaleInst;
    #characterizes->#CurrentTime_CIInst::#Entity).
```

Abbildung 5.8: Relevanzbedingung für Service Scope

Diese in F-Logic formulierte Relevanzregel greift das Scope-Beispiel aus Abbildung 5.5 auf Seite 131 auf. Sie ermittelt, ob die Kontextinformation `CurrentTime_CIInst` – die derzeitige Belegung der impliziten Kontextvariable `openingHoursParam` – eine gültige Kontextinformation bezüglich des `showMapScope` für zwei im Rahmen der Dienstvermittlung gefundene `ProviderX` und `ProviderY` ist. Gemäß der Regel wird die Gültigkeit der Belegung für `ProviderX` als `openX=0|1` und analog für `ProviderY` als `openY=0|1` durch die Inferenzmaschine angezeigt. Beide Dienste gelten folglich als kontextuell konsistent bezüglich der Kontextvariable `openingHoursParam`, wenn `openX=1` und `openY=1` sind, was gleichzeitig die Voraussetzung für einen reaktiven oder proaktiven, horizontalen NCS Handover darstellt.

Proaktivität wird durch die Auswertung von Meta-Kontextinformationen ermöglicht. Die in Abbildung 5.8 dargestellte Relevanzregel ermittelt hierzu zusätzlich noch die Werte für `remainX` und `remainY`, die gemäß der Relevanzbedingung die verbleibende Zeit bis zum Ende der „Öffnungszeiten“ des jeweiligen Dienstes in Sekunden ausdrücken. Durch Registrierung bei der *Monitor & Event-Generator* Komponente (siehe Abschnitt 4.1) mit einer Bedingung analog zu

```
CMAP.getEventRegistry().register("FORALL event <- fire(
  equal(openX,1) AND equal(openY,1) AND lessorequal(remainX,3)", event)).
```

kann ein proaktiver NCS Handover durch das Service Framework eingeleitet werden, noch bevor `openX=0` wird. Ob die verbleibenden 3 Sekunden (`lessorequal(remainX,3)`) für das dann zum Einsatz kommende NCS Handover-Protokoll ausreichen und ob dieser auf der anderen Seite bei 3 Sekunden noch als *Fast* NCS Handover gilt, ist individuell für das jeweilige NCS Handover-Protokoll zu spezifizieren.

Allgemein betrachtet läßt sich folgende Aussage aus der Verwendung von Service Scopes als Modellierungs- und Spezifizierungs-Grundlage ableiten:

> *Jede Änderung des Kontextes einer Dienstinteraktion kann einen Service Handover motivieren!*

Insofern ist die Bestimmung der Konsistenz als Grundlage der NCS Handover eine wichtige Aufgabe eines Service Frameworks, welches NCS Handover vorsieht.

5.3.2 Rollenverteilung in der Systemarchitektur

Wie in Abschnitt 2.2 erläutert, ist ein Service Framework für die Überprüfung und Überwachung der verschiedenen Ebenen der Interoperabilität vor und während der Ausführung von NCS Interaktionen zuständig. Eine Auswahl solcher Frameworks wurde bereits in Abschnitt 2.4.3 vorgestellt. Keines dieser Frameworks unterstützt allerdings das Konzept der Handover – falls überhaupt – in dem im Rahmen dieser Arbeit vorgestellten Umfang.

Die Integration von Handover-Verfahren kann dabei prinzipiell auf verschiedene Weisen erfolgen. Einerseits kann versucht werden, nur die notwendigsten Komponenten im Framework

anzusiedeln, während wesentliche Funktionalitäten des Handover-Verfahrens entweder in der *customer domain* oder in den *service provider domains* vorzufinden sind. Diese Vorgehensweise könnte zu der in Abbildung 5.9 skizzierten Architektur führen, bei der die verschiedenen Rollen (siehe Abschnitt 5.1) der Aktoren im Handover-Verfahren von Komponenten der beiden genannten Domänen erfüllt werden. Für diesen Ansatz spricht, daß ein Handover in der Regel sehr dienstspezifische Rahmenbedingungen hat, die möglicherweise individuell und proprietär sehr leicht zu lösen sind.

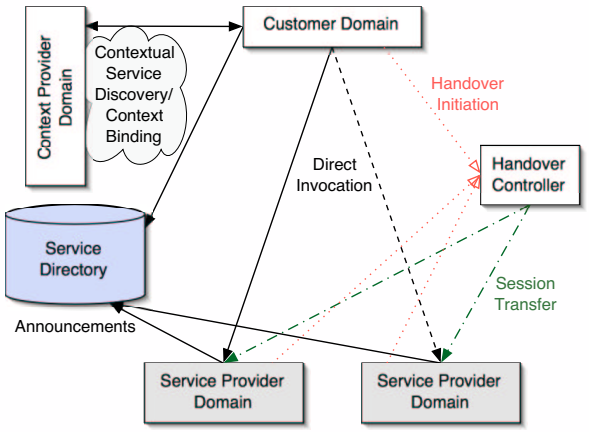

Abbildung 5.9: Direkte NCS Interaktion mit Handover

Alternativ kann versucht werden, möglichst viele Aufgaben im Rahmen eines Handover-Verfahrens an das Framework zu delegieren. Dies schließt sogar die komplette Dienstvermittlung mit ein und führt zu einer Architektur, wie sie in Abbildung 5.10 skizziert ist. Wesentliches Element dieser Architektur ist eine *Intermediate* genannte Komponente des Service Frameworks, die stellvertretend für die Komponenten der *customer domain* und der *service provider domains* verschiedene Rollen der Aktoren im jeweiligen Handover-Verfahren übernimmt. Hierdurch kann u.a. eine weitgehende Transparenz des NCS Handovers gegenüber den Entitäten der anderen beiden Domänen erreicht werden.

Durch die Transparenz hat diese Architektur u.a. den Vorteil, daß auch Dienstnutzer bzw. Dienstanbieter von Handover-Verfahren profitieren können, die selbst keine Vorkehrungen für die Durchführbarkeit von Handover-Verfahren treffen. Die Architektur kann auf eine Weise genutzt werden, bei der ein Dienstnutzer z.B. keine Kenntnis darüber hat, mit welchem Dienstanbieter er gerade eine Dienstinteraktion eingeht: Der Dienstnutzer interagiert ausschließlich mit der Intermediate-Komponente. Wird ein NCS Handover während einer Dienstinteraktion zu einem anderen Dienstanbieter von der Intermediate-Komponente nach Prüfung der Ersetzbarkeit initiiert, sorgt die Intermediate-Komponente für alle ggf. erforderlichen Adaptionszwischenschritte zur Einhaltung der Kompatibilität als Perspektive der Interoperabilität.

Die Intermediate-Komponente in der in Abbildung 5.10 dargestellten Architektur hat die voll-

ständige Kontrolle über die Suche nach einem passenden Dienstanbieter und die anschließende Dienstausführung. Damit fällt ihr insbesondere die Rolle des *handover controllers* zu. Dennoch muß die Dienstnutzer-Komponente ein Mindestmaß an Funktionalität für die Dienstvermittlung unter der Berücksichtigung des Kontextes besitzen, da der Dienstnutzer-Komponente die Angabe von Relevanzkriterien obliegt. Die Art der Integration der Relevanzkriterien ist spezifisch bezüglich des jeweiligen Service Frameworks.

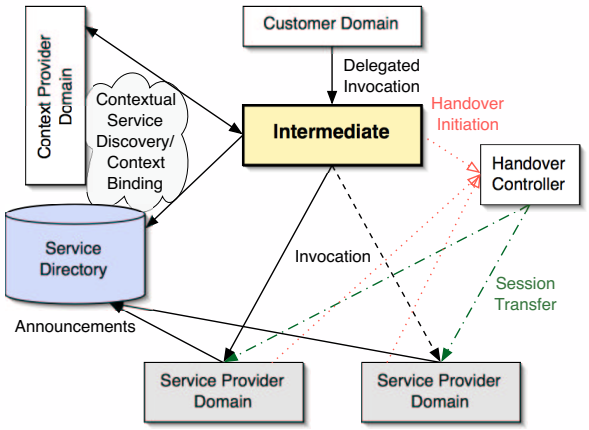

Abbildung 5.10: Delegierte NCS Interaktion mit Handover

Dies kann am Beispiel einer Web Service Umgebung unter Verwendung von SOAP [44] und WSDL [55] verdeutlicht werden. Hierzu sei der *MapService* aus Abbildung 4.5 auf Seite 102 noch einmal aufgegriffen. Nach wie vor kann jede Operation wie `showMap` eines Dienstes unter Angabe aller Parameter direkt gemäß Abbildung 5.9 auf einen Dienst zugreifen oder diese Aufgabe gemäß Abbildung 5.10 an eine Intermediate-Komponente delegieren. Letztere erfüllt in einer Web Service Umgebung sinnvollerweise die in [44] definierte Rolle des *Intermediate SOAP Node* und implementiert dabei das CMAP Interface des MNMplusCE Dienstmodells. Aus der in Abbildung 4.5 gezeigten Bindung des Eingabeparameters `currentPositionParam` an eine Skala kann die Dienstnutzer-Komponente aber auch ableiten, daß die Belegung des Eingabeparameters ebenfalls an eine andere, für die Erfüllung der Kontextadaption zuständige Komponente wie die Intermediate-Komponente delegiert werden kann. Die in diesem Fall notwendige Angabe von Relevanzkriterien kann von der Dienstnutzer-Komponente z.B. als Header-Element des SOAP-Envelopes erfolgen, wie in Abbildung 5.11 skizziert ist.

Mit den Informationen aus dem Header des SOAP-Envelopes und der erweiterten WSDL-Beschreibung des *MapService* ist die Intermediate-Komponente in der Lage, die Dienstausführung an den aktuellen Kontext der Dienstausführung anzupassen. Im obigen Beispiel geschieht dies durch Anfrage beim *context provider* nach einer Kontextinformation, welche die im Header des SOAP-Envelopes angegebene Entität bezüglich der in der WSDL-Beschreibung angegebenen Skala charakterisiert. Dies ermöglicht ebenfalls die Verwendung von Kontextinformationen, auf die der Dienstnutzer z.B. aus rechtlichen oder abrechnungstechnischen Gründen keinen direkten Zugriff hat, wohl aber die Intermediate-Komponente der Middleware (z.B. weil zwischen dem Betreiber der Middleware und einem Mobilfunk-Operator ein entsprechender Vertrag geschlossen wurde).

Das Beispiel in Abbildung 5.11 zeigt weiterhin, wie ein anderes SOAP Sprachelement in diesem Zusammenhang sinnvoll verwendet werden kann: Durch Angabe des Attributs `SOAP-ENV:-mustUnderstand=1` kann der Intermediate-Komponente signalisiert werden, daß diese den Wert für `currentPositionParam` ersetzen *muß*, wohingegen sie bei Angabe von `SOAP-ENV:-mustUnderstand=0` den Wert durch eine Kontextinformation ersetzen *kann*. Die erste Variante ist nützlich, wenn die Dienstnutzer-Komponente keinerlei gültige Informationen für diesen Parameter hat, der Dienst aber ohne Angabe dieses Parameters nicht sinnvoll verwendet werden kann. Die zweite Variante hingegen ermöglicht es der Dienstnutzer-Komponente, einen Wert wie im o.g. Beispiel vorzugeben, der aber von der Intermediate-Komponente ersetzt werden kann, wenn diese über (evtl. aktuellere/bessere/etc.) gültige Kontextinformation verfügt.

```
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
                   xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
                   xmlns:xsd="http://www.w3.org/1999/XMLSchema"
                   xmlns:map="http://context-aware.org/wsdl/MapService.wsdl"
                   xmlns:cb="http://context-aware.org/cool/soap-hdr-cb.xsd">

  <SOAP-ENV:Header>
    <cb:ReleCrit SOAP-ENV:mustUnderstand="0" cb:part="map:currentPositionParam">
      <cb:Entity cb:identifiedBy="urn:PhoneNumber#+4917998765"/>
    </cb:ReleCrit>
  </SOAP-ENV:Header>

  <SOAP-ENV:Body>
    <map:showMapRequest>
      <map:currentPositionParam
        xsi:type="xsd:string">367032 533074</map:currentPositionParam>
    </map:showMapRequest>
  </SOAP-ENV:Body>

</SOAP-ENV:Envelope>
```

Abbildung 5.11: Relevanzkriterium als Header im SOAP Envelope

Im Beispiel oben handelt es sich um ein einfaches Relevanzkriterium unter Angabe einer externen Relevanzbedingung (siehe Abschnitt 4.6). Auf ähnliche Weise können aber auch komplexerer Relevanzkriterien z.B. unter Verwendung von Meta-Kontextinformationen basierend auf Qualitäts-Aspekten angegeben werden.

Die Auswertung von Relevanzkriterien in der Intermediate-Komponente zur Konkretisierung einer Bindung des Kontextes an einen Eingabeparameter einer Operation ist zunächst unabhängig von den in diesem Kapitel schwerpunktmäßig behandelten Handovern. Der folgende Abschnitt zeigt jedoch, daß die Auswertung von Relevanzkriterien in der Intermediate-Komponente essentiell zur Bestimmung der Konsistenz als Anwendung der Relevanz ist. Diese ist ihrerseits eine Voraussetzung für einen NCS Handover und basiert auf der Auswertung einer speziellen Art von Bindungen des Kontextes, den Service Scopes.

5.3.3 Session Handling in einem Beispieldienst

Der konkrete Einsatz eines kontextuellen Frameworks soll im Folgenden am Beispiel einer Interaktion mit einem Fotoentwicklungsdienst, der von mehreren Dienstanbietern angeboten wird und zwischen denen während der Dienstinteraktion ein NCS Handover abläuft, auf der Basis von Web Services dargestellt werden. Dabei wird insbesondere die Frage des Handlings von Sessioninformationen vertiefend aufgegriffen.

Ein Fotoentwicklungsdienst ist semantisch informell beschrieben ein Dienst, an den Bilder in elektronischer Form übermittelt werden („Upload“). Die Aufgabe des Dienstanbieters ist es, aus diesen Daten klassische Bildabzüge auf Papier zu erstellen, die dem Dienstnutzer zur Abholung bereitgestellt oder an eine zu definierende Anschrift versandt werden. Solch ein Fotoentwicklungsdienst hat insofern einen großen Bezug zur Praxis, als daß die Nutzung eines solchen Dienstes von modernen Mobiltelefonen mit integrierter Digitalkamera sehr naheliegend ist, wodurch sich insbesondere die Implikationen der Mobilität (vgl. Abschnitt 2.4.1) auf den Dienst auswirken.

Zu Identifikations- und Abrechnungs-Zwecken gehört zur Nutzung des Dienstes, daß sich ein Dienstnutzer vor der eigentlichen Dienstnutzung gegenüber dem Dienstnutzer authentifiziert. Dies setzt üblicherweise voraus, daß der Dienstnutzer über einen Account beim Dienstanbieter verfügt. Nach erfolgreicher Authentifizierung kann der Dienst in personalisierter Form agieren, d.h. daß persönliche Vorlieben des Anwenders zu Bildformat, Lieferadresse, etc. verwendet werden oder gesondert ausgehandelte Preise zur Berechnung herangezogen und dem Account zugeordnet werden. Somit ist eine erste wichtige Information, welche die Dienstinteraktion charakterisiert, ein Accounting-Token, das im Folgenden als *UserId* bezeichnet wird.

Aus Gründen der Minimierung der Transportkosten von Papierfotos ist es weiterhin üblich, die Produktion von Papierfotos in *Aufträgen* zu bündeln und sowohl die Lieferung einer Menge von einzelnen Fotos wie auch die Abrechnung auf ganze Aufträge anstelle der einzelnen Fotos zu beziehen. Dies bedeutet, daß ein Anbieter eines Fotoentwicklungsdienstes Funktionen zum Anlegen und Verwalten neuer Aufträge bereitstellen muß, die im Folgenden als *OrderId* ebenfalls einer Session zugeordnet sind.

Der gesamte Interaktionsvorgang mit einem Fotoentwicklungsdienst kann allgemein wie folgt skizziert werden:

1. Anmeldung bei einem frei wählbaren Fotoentwicklungsdienst unter Verwendung einer UserId (Login).
2. Anlegen eines neuen Auftrags (liefert eine neue OrderId) oder Aufgreifen eines bereits bestehenden Auftrags unter Verwendung einer bekannten OrderId. Jede OrderId ist eindeutig einer UserId zugeordnet.
3. Upload einer beliebigen Anzahl von Bildern zu einer OrderId sowie Setzen verschiedener Parameter zu Bildqualität, Abholort etc.
4. Auftragsabschluß (Commit). Anschließend weiter bei Schritt 2 oder Abmeldung vom Fotoentwicklungsdienst (Logout).

Die Schritte 2 bis 4 können als Transaktion betrachtet werden. Nach der Erzeugung einer OrderId sind alle Interaktions-Operationen (Upload eines Bildes, Setzen des bevorzugten Abholortes etc.) als atomare Operationen zu betrachten, zwischen denen Checkpunkte liegen. Zu diesen kann jederzeit zurückgekehrt werden (Rollback), z.B. nach dem Verlust einer Datenverbindung zum mobilen Endgerät. Außerdem ist jeder Checkpunkt ein optimaler Zeitpunkt für einen *smooth* NCS Handover, da hier die Wahrscheinlichkeit für einen Datenverlust während des Handovers am geringsten ist.

Die Kontextadaptivität eines Service Frameworks wird bei dem hier vorgestellten Dienstnutzungsszenario derart genutzt, daß das System einen (für den Nutzer nach Möglichkeit transparenten) NCS Handover zwischen mehreren Dienstanbietern vornimmt, die bezüglich Funktionalität und Qualität des Dienstes mindestens gleichwertig, optimalerweise aber ideal bezüglich des jeweiligen Kontextes sind. Konkret kann das Service Framework beispielsweise immer den Dienstanbieter auswählen, dessen Filialen als Abholstation örtlich am nächsten zum aktuellen Aufenthaltsort liegen. Dazu muß das Service Framework eine implizite Kontextvariable auswerten, die den aktuellen Standort des Benutzers als Element einer definierten Skala als Belegung hat. Dies entspricht der Auswertung von Scopes im Rahmen der Handover-Entscheidung.

Die Dienstanbieter unterscheiden sich neben der Funktionalität durch die Eigenschaften ihres Dienstangebots. Zu letzteren gehören beispielsweise die mittlere und die maximale Dauer der Entwicklungszeit oder die Kosten für Entwicklung und/oder Lieferung der Papierfotos. Diese Eigenschaften sind qualitative Größen, die einen Dienst charakterisieren. Als solche lassen sie sich als Zusicherungen des Kontext (context obligation) ausdrücken und im Rahmen der Handover-Entscheidung auswerten.

Ist die Entscheidung für einen NCS Handover gefallen, wird die Durchführung des Handovers initiiert. Zur Durchführung zählen einerseits Aktionen, die den Kommunikationskanal zum ursprünglichen Partner abbauen und einen Kommunikationskanal zum zukünftigen Partner aufbauen. Andererseits zählen dazu auch Maßnahmen, um den Zustand des Dienstes beim neuen Anbieter in der Form wieder herzustellen, wie er vor der Einleitung des Handovers beim alten Anbieter war. Da der Zustand des Dienstes durch Sessioninformationen beschrieben wird, bedeutet dies, die Sessioninformationen in geeignet adaptierter Form an den Nachfolger zu transferieren. Dies kann in zwei Varianten geschehen:

Session Forwarding. Dies bezeichnet den Vorgang, alle zu einer Session gehörenden und für die korrekte Weiterführung des Dienstes erforderlichen Informationen unverändert an den neuen Dienstanbieter zu übergeben. Das bedeutet, daß Vorgänger und Nachfolger ein identisches Verständnis auf allen Interoperabilitätsebenen von den ausgetauschten Sessioninformationen haben. Im Fall des Fotoentwicklungsdienstes bedeutet dies z.B., daß eine gültige OrderId, die beim Vorgänger angelegt wurde, auch beim Nachfolger als solche erkannt und akzeptiert wird, ohne daß eine neue OrderId beim Nachfolger angelegt wurde.

Session Recreation. Dies bezeichnet den Vorgang, eine Session zum Nachfolger neu aufzubauen, und die alten Sessioninformationen auf die neuen Sessioninformationen im Rahmen des Handover-Vorgangs abzubilden. Hierbei spielen auch notwendige, vorberei-

tende Schritte eine Rolle, die auf Protokollebene spezifiziert und zur Laufzeit zusätzlich zum Zeitpunkt des Handovers durchgeführt werden müssen, um den gleichen Zustand beim Nachfolger herbeizuführen (z.B. Login).

Durch Kapselung des Handover-Verfahrens in einer Komponente des Service Frameworks wird der Dienstnutzer von der Komplexität des Handlings von Sessioninformationen entbunden, was gerade bei ressourcenschwachen mobilen Endgeräten von Vorteil ist. Andererseits erfordert die allgemeine Spezifikation von Einflußgrößen für das jeweilige Handover-Verfahren einen hohen Integrations-Aufwand, der aber im Sinne der Wiederverwendbarkeit oft gerechtfertigt erscheint.

Wie schon das einfache Beispiel des Fotoentwicklungsdienstes zeigt, können sehr viele verschiedene Parameter den Handover-Entscheidungsprozeß beeinflussen. Diese müssen in Form von Relevanzkriterien spezifiziert und vom Service Framework ausgewertet werden. Je nach Art des Service Frameworks lassen sich die in dieser Arbeit vorgestellten Konzepte mehr oder weniger gut in das Framework integrieren.

```
<!-- ========== excerpt from MESSAGE section of PhotoShop.wsdl ========= -->
<wsdl:message name="userRegistrationRequest">
   <wsdl:part name="userName" type="xsd:string"/>
   ....
</wsdl:message>
<wsdl:message name="userRegistrationResponse">
   <wsdl:part name="userId" type="xsd:int" cb:handoverSessionInfo="userId"/>
</wsdl:message>
...
<wsdl:message name="createOrderRequest">
   <wsdl:part name="someUserId" type="xsd:int" cb:handoverSessionInfo="userId"/>
</wsdl:message>
<wsdl:message name="createOrderResponse">
   <wsdl:part name="orderId" type="xsd:int" cb:handoverSessionInfo="orderId"/>
</wsdl:message>
...
<wsdl:message name="uploadRequest">
   <wsdl:part name="orderId" type="xsd:int" cb:handoverSessionInfo="orderId"/>
   <wsdl:part name="pictureData" type="xsd:base64Binary"/>
</wsdl:message>
```

Abbildung 5.12: Markierung von Sessioninformationen in WSDL

Beispielsweise erfordert die Integration von Sessioninformationen wie die o.g. UserId oder OrderId in eine Web Service Umgebung eine Erweiterung der standardisierten Web Service Protokolle, da diese Art von Informationen nicht inhärenter Bestandteil der Spezifikationen ist. Das SOAP-Binding an HTTP [44] sieht zwar das Konzept der *SessionId* vor, diese hat jedoch eine andere Semantik und dient vornehmlich der Objekt-Referenzierung. Die Integration von Sessioninformationen in Web Services kann zum Teil durch Markierung von Variablen, die Träger dieser Informationen sind, innerhalb der WSDL Signaturbeschreibung

erfolgen. In Abbildung 5.12 ist dies beispielhaft dargestellt, indem jede Sessioninformation mit `cb:handoverSessionInfo=..` markiert ist. Mit Hilfe dieser Zusatzinformationen in einer WSDL Signaturbeschreibung kann die für den Handover zuständige Komponente des Frameworks, z.B. die Intermediate-Komponente aus Abbildung 5.10 auf Seite 136, viele zum Anlegen und Pflegen von Sessioninformationen notwendige Angaben extrahieren. Dies ist speziell bei Session Recreation nützlich. Im Wesentlichen ist die Vorgehensweise, für jede individuelle in einer Response-Message benannte `cb:handoverSessionInfo=`*Name* einen Eintrag für *Name* in einer Tabelle anzulegen, um dann im weiteren Verlauf den für die jeweilige, an der Interaktion beteiligte Komponente gültigen Wert verwenden zu können.

5.4 Zusammenfassung

In diesem Kapitel wurde ein Anwendungsfall des ASC-Modells und der darauf basierenden Context Ontology Language vertiefend betrachtet. Der Anwendungsfall beschreibt den Handover zwischen verschiedenenen non-carrier services (NCS). Hierzu wird zunächst der Unterschied zwischen Kommunikationsdiensten (carrier services) und allgemeinen Diensten (non-carrier services) erörtert.

Zu Handover-Verfahren in Kommunikationsdiensten wurden bereits zahlreiche wissenschaftliche Untersuchungen gemacht, während diese für allgemeine Dienste eher rar sind. Daher wurden die wichtigsten Merkmale von Handover-Verfahren bei Kommunikationsdiensten analysiert, klassifiziert und in Merkmale für allgemeine Dienste überführt.

Aufbauend auf diesem Klassifikationsschema wurde gezeigt, wie Handover-Verfahren für allgemeine Dienste insbesondere von der Kontextadaptivität der Dienste profitieren. Durch Zuweisung zusätzlicher, Handover-spezifischer Rollen in einer Architektur, wie sie im Kapitel 4 vorgestellt wurde, können viele Aufgaben des Handovers an das kontextuelle Service Framework delegiert werden. Es wurde gezeigt, wie das Konzept der logischen Verfügungsbereiche (*service scopes*) als Spezialisierung von *context bindings* eine Entscheidungsgrundlage für die Initiierung eines Handovers darstellt. Eine wichtige Feststellung in diesem Zusammenhang war, daß jede Veränderung des Kontextes einer Dienstinteraktion einen Handover motivieren kann.

Das Kapitel wurde mit der Vorstellung des PhotoShop Beispieldienstes abgeschlossen, der auf der Basis von Web Services geeignet ist, abhängig vom Kontext einen Handover zwischen verschiedenen Fotoentwicklungsdiensten zu demonstrieren. Hierbei wurde u.a. vertiefend auf das Handling von Sessioninformationen während eines NCS Handovers eingegangen.

Im nachfolgenden Kapitel werden die Ergebnisse der gesamten vorliegenden Arbeit noch einmal zusammengefaßt, bevor in einem Ausblick verschiedene Anknüpfungspunkte für zukünftige Arbeiten identifiziert werden.

Kapitel 6

Abschlußbetrachtungen

In diesem Kapitel erfolgt eine Zusammenfassung der wesentlichen Ergebnisse dieser Arbeit. Ferner wird eine Reihe interessanter Fragestellungen aufgezeigt, die Potential für weitere Forschungsarbeiten geben, jedoch die unmittelbare Thematik der vorliegenden Arbeit überschritten hätten.

6.1 Zusammenfassung

Im Fokus dieser Arbeit steht die kontextuelle Interoperabilität von Diensten in Ubiquitous Computing Umgebungen. Grundlegend ist dabei das Konzept der kontextadaptiven Dienstnutzung, die sowohl die kontextadaptive Dienstauswahl als auch Dienstausführung umfaßt. Obwohl dem Kontext insbesondere mit Etablierung der Ubiquitous Computing Systeme eine zunehmend wichtige Rolle beigemessen wird, hat dieser in bisherigen Forschungsarbeiten zur Interoperabilität nur eine untergeordnete Rolle eingenommen. Vor diesem Hintergrund konnten die klassischen Ansätze zur Betrachtung der Interoperabilität von Diensten um eine kontextuelle Ebene erweitert werden.

Aus der Analyse vorangegangener Forschungsarbeiten wird ersichtlich, daß Interoperabilität auf den gemeinsamen Bezug aller an einer Dienstinteraktion beteiligten Komponenten auf ein gemeinsames Verständnis zurückführbar ist. Dieses gemeinsame Verständnis, *shared understanding* oder *shared knowledge* genannt, repräsentiert auf der kontextuellen Ebene das Wissen über die Abhängigkeit von kontextuellen Fakten, bei denen die Interoperabilität von zwei oder mehr Komponenten gegeben ist. Kern dieser Arbeit war daher die Entwicklung eines Modells zur Projektion von kontextuellen Fakten und Abhängigkeiten auf eine formale Spezifikation des gemeinsamen Verständnisses. Dieses Modell wurde in Kapitel 3 als *ASC-Modell* vorgestellt.

Zur Beschreibung des ASC-Modells werden *Ontologien* verwendet. Dadurch können die Konzepte des Modells, sowie darauf basierende Unterkonzepte und Fakten in einheitlicher Weise als verteiltes, kontextuelles Wissen definiert werden. Weiterhin wird die automatische Aus-

wertung des so spezifizierten Wissens mit Hilfe von ontologischen *Reasonern* im Rahmen der Kontextadaption ermöglicht.

Hierzu wurde das Modell zunächst auf der Basis verschiedener Ontologiesprachen wie DAML+OIL, OWL-DL und F-Logic realisiert und die jeweiligen Umsetzungen miteinander verglichen. Alle drei Umsetzungen bilden den Kern einer nicht monolithischen Sprache zur Modellierung kontextuellen Wissens, auf die fortan mit dem Namen *Context Ontology Language* verwiesen wurde. Das Fazit aus dem Vergleich war keine klare Empfehlung für eine der drei Varianten, sondern der gemischte Einsatz der Umsetzungen je nach Anforderung der Verifikationsmöglichkeiten, der Wahl des Reasoners und anderer Parameter. Für die Verifikation der in Kapitel 4 und Kapitel 5 vorgestellten Anwendungen und Verfahren wurde jedoch das Setup bestehend aus dem *OntoBroker* Reasoner in Verbindung mit der F-Logic-Variante der Umsetzung des ASC-Modells verwendet.

Die Vielseitigkeit der Anwendungsmöglichkeiten der Context Ontology Language zeigen die Kapitel 4 und 5. Allem voran steht natürlich die Verwendung zur Spezifikation bzw. Überprüfung der kontextuellen *Kompatibilität* sowie der kontextuellen *Ersetzbarkeit* von Diensten als jeweilige Perspektive der kontextuellen Service-Interoperabilität. Die kontextuelle Kompatibilität zu einer Operation eines Dienstes wird u.a. durch die Ermittlung eines Pfades durch das Netzwerk ineinander überführbarer Kontextinformationen von unterschiedlichen Skalen oder gar Aspekten mit Hilfe des Reasoners überprüft (siehe Abschnitt 4.5). Dem gegenüber steht die kontextuelle Ersetzbarkeit, bei der ein Dienstanbieter durch einen anderen Dienstanbieter ersetzt wird. Sie wird anhand der logischen Verfügungsbereiche (*service scopes*) für die Dienste der jeweiligen Dienstanbieter ermittelt. Ein Reasoner ist durch Auswertung der Belegung von Kontextvariablen in der Lage, das Verlassen eines Verfügungsbereiches zu erkennen, was einen *Handover* zu einem anderen Dienstanbieter motiviert (siehe Abschnitt 5.3).

Weiterhin konnte auf einen Katalog von Unterkonzepten und Fakten als integraler Bestandteil der Context Ontology Language zur Formulierung von Relevanzkriterien zurückgegriffen werden. Mit Hilfe dieser Relevanzkriterien ist ein Reasoner in der Lage, Kontextinformationen aus dem mittels der Context Ontology Language spezifizierten Wissen abzuleiten.

Es wurden aber auch andere, nicht unmittelbar im Zusammenhang mit der Service-Interoperabilität stehende Anwendungsmöglichkeiten des Modells und der Sprache aufgezeigt. Die Ausgewogenheit zwischen festgelegtem Umfang der Modell-Konzepte und der Erweiterbarkeit unter Verwendung von Ontologien ist z.B. hinreichend flexibel, um die charakteristischen Merkmale anderer Kontextmodelle größtenteils mit abdecken zu können. Das ASC-Modell bildet somit eine gute Grundlage als Meta- oder Transfer-Modell zur Abbildung zwischen unterschiedlichen Kontextmodellen, was am Beispiel der Abbildung eines grafischen Kontextmodells auf das ASC-Modell gezeigt wurde.

6.2 Ausblick

Zukünftige Arbeiten können die Anwendbarkeit des ASC-Modells und der darauf basierenden Context Ontology Language für verschiedene Einsatzzwecke evaluieren. Sehr wahrscheinlich

resultieren aus dieser Evaluierung Vorschläge für Modifikationen und/oder Erweiterungen an Modell und Sprache, die auch für andere Einsatzgebiete gewinnbringend sind. Erste Arbeiten in diese Richtung gibt es bereits. Dazu zählt die Diplomarbeit von Krause [99], der sich nach Analyse verschiedener Kontextmodelle für das ASC-Modell als Grundlage einer Sprache zur dynamischen Komposition von Kontextinformationen entschieden hat. In einer anderen Diplomarbeit untersucht Treu [162] die Eignung des ASC-Modells zur Verwendung in einem speziellen Dienstvermittlungsverfahren in mobilen Ad-Hoc Netzwerken. Weitere Arbeiten können analysieren, inwieweit die Betrachtung von ganzen Wertebereichen einer Skala als *eine* Kontextinformation (z.B. eine Wahrscheinlichkeitsdichtefunktion (PDF) wie in [26] vorgeschlagen) Auswirkung auf den Modellentwurf haben.

Die in Abschnitt 3.2.3 eingeführten Intra- und InterOperationen sind Abbildungen von Kontextinformationen einer oder mehrerer Quellskalen auf eine Zielskala. Die Ausführung dieser Abbildung hat in der Regel Auswirkungen auf die Qualität der Information, welche durch die abgebildete Kontextinformation ausgedrückt wird. Mal kann die Qualität einer Information – z.B. durch Komposition von Kontextinformationen mehrerer Sensoren – verbessert werden, mal wird sie durch Quantisierung der Zielskala reduziert. Zukünftige Arbeiten können dieses Problem aufgreifen und ermitteln, inwiefern eine Änderung der Qualität der Informationen, die das Durchlaufen von Intra- oder InterOperationen verursacht, diesen Operationen zugeordnet werden können. Die Berücksichtigung dieser Qualitäts-Faktoren sollte z.B. eine weitere Optimierung der Suche nach Kompatibilitäts-Pfaden mit Hilfe eines Reasoners analog zu dem in Abschnitt 4.5 vorgestellten Verfahren darstellen.

Die prototypische Implementierung der Architektur beschränkt sich auf die Verwendung einer einzigen Instanz der Inferenzmaschine. Diese Vorgehensweise ist nur unter der voraussetzenden Annahme gültig, daß alle für eine Aufgabenstellung/Dienstinteraktion relevanten Kontextinformationen dieser Instanz bekannt sind. Entfällt diese Voraussetzung, kann das Wissen über den Kontext für eine Dienstinteraktion auf mehreren Inferenzmaschinen verteilt sein, von denen keine vollständiges Wissen hat. Nachfolgende Arbeiten müßten an diesem Punkt ansetzen und Lösungen für dieses Problem erarbeiten, z.B. über Replikationsstrategien in einem föderativen Systemen.

Die Güte des eingesetzten Reasoners hat sicher großen Einfluß auf das Ergebnis des Reasonings. Potential für zukünftige Arbeiten hat beispielsweise eine Analyse, welche Datenmengen (z.B. Instanzen von Kontextinformationen) ein Reasoner vorhalten kann, so daß gewisse Obergrenzen der Performance (z.B. Wartezeit pro Anfrage etc.) nicht überschritten werden. So hat bereits Harrocks in [87] festgestellt, daß das Reasoning auf Instanzebene (A-Box reasoning) „potentiell unendlich lange dauern kann, dies jedoch durch geeignete Optimierung praktikabel wird“. Volz et al. haben in [166] darauf hingewiesen, daß ein Performanzgewinn und die Vermeidung von doppelter Datenhaltung durch Auslagerung der Instanzen in performante Datenbanken mit direkter Anbindung an den Reasoner erreicht wird. Weiterführende Arbeiten sollten sich dieser Thematik annehmen und die Lösungsvorschläge konkretisieren.

Die in Kapitel 5 betrachteten Handover-Verfahren hatten zum Ziel, horizontale Handover zwischen *non-carrier services* durch Auswertung des Kontextes zu ermöglichen. Der Fokus lag hier auf der Ermittlung der kontextuellen Kongruenz, die eine Voraussetzung für einen horizontalen Handover darstellt. Nicht näher betrachtet wurden die Voraussetzungen für vertikale

Handover. Das sind Handover zwischen Diensten, die auf mindestens einer der vier Ebenen der Service-Interoperabilität nicht äquivalent in der Richtung der Ersetzung sind (Ersetzbarkeit wurde als asymmetrische Eigenschaft definiert). Zukünftige Arbeiten können die dadurch offen gebliebenen Fragestellungen aufgreifen und z.B. eine Sprache zur Spezifikation der syntaktischen, protokollarischen, semantischen und kontextuellen Voraussetzungen für vertikale Handover erarbeiten, die im Bereich des Kontextes auf den hier vorgeschlagenen Verfahren aufsetzt.

Literaturverzeichnis

[1] Daml ontology library. http://www.daml.org/ontologies.

[2] Distributed component object model (dcom). http://www.microsoft.com/com/dcom.asp.

[3] Dun & bradstreet D-U-N-S numbers. http://www.dnb.com.

[4] Gsm 03.09: Handover procedures.

[5] Inference engines for the semantic web. http://www.semanticweb.org/inference.html.

[6] Language feature comparison. http://www.daml.org/language/features.html.

[7] North American Industry Classification System (NAICS). http://www.census.gov/epcd/www/naics.html.

[8] Specification of the bluetooth system (core, profiles, assigned numbers). http://www.bluetooth.com.

[9] Universal Plug and Play (UPnP). http://www.upnp.org.

[10] Universal Standard Products and Services Classification (UNSPSC). http://www.unspsc.org.

[11] Web Service Choreography Interface FAQ. http://wwws.sun.com/software/xml/developers/wsci/faq.html.

[12] Iso/iec 11578:1996 standard: Universal unique identifiers. http://www.iso.ch, 1996.

[13] Esprit project 26900: Technology for enabled awareness (tea), 1998.

[14] UDDI technical white paper. http://www.uddi.org, september 2000.

[15] Best Practices: Using WSDL in a UDDI Registry. http://www.uddi.org/pubs/wsdl-bestpractices, June 2001.

[16] UDDI Version 2.0 API Specification. http://www.uddi.org/pubs/ProgrammersAPI, June 2001.

[17] Universal Description, Discovery and Integration (UDDI). http://www.uddi.org, 2001.

[18] No hiding place. *The Economist: Digital dilemmas - A survey of the internet society* (January 2003), 5–8.

[19] 3GPP. Mobile Execution Environment (MExE); Functional Description. Tech. Rep. TS 23.057 V6.1.0, 3rd Generation Partnership Project (3GPP), September 2002.

[20] Akman, V., and Surav, M. The use of situation theory in context modeling. *Computational Intelligence 13*, 3 (1997), 427–438.

[21] Alfonseca, M. Frames, semantic networks, and object-oriented programming in apl2. *IBM Journal of Research and Development 33*, 5 (September 1989), 502–510.

[22] America, P. Designing an object-oriented programming language with behavioural subtyping. In *LNCS 489: Foundations of Object-Oriented Languages, REX School/Workshop* (Noordwijkerhout, The Netherlands, May/June 1990), J. W. de Bakker, W. P. de Roever, , and G. Rozenberg, Eds., vol. 489 of *Lecture Notes in Computer Science (LNCS)*, Springer-Verlag, New York, NY, 1991, pp. 60–90.

[23] Angermann, M. Untersuchungen zu optimiertem prefetching in datennetzen durch verwendung von situationsinformation. Kolloquium Informationstechnik Universitaet Ulm, July 2001.

[24] Angermann, M., and Kammann, J. Cost metrics for decision problems in wireless ad hoc networking. In *Proceedings IEEE CAS 2002* (Pasadena, USA, September 2002).

[25] Angermann, M., Kammann, J., Kuehndel, F., Robertson, P., Steingass, A., and Strang, T. Enjoyable Mobile Commerce. In *Proceedings of 2nd Annual Pervasive Computing 2001* (Gaithersburg, Maryland, USA, May 2001).

[26] Angermann, M., Kammann, J., Robertson, P., Steingass, A., and Strang, T. Software representation for heterogeneous location data sources within a probabilistic framework. In *Proceedings of International Symposium on Location Based Services for Cellular Users (Locellus 2001)* (Munich, Germany, February 2001), pp. 107–118.

[27] Ankolekar, A., Burstein, M., Hobbs, J. R., Lassila, O., Martin, D. L., McIlraith, S. A., Narayanan, S., Paolucci, M., Payne, T., Sycara, K., and Zeng, H. Daml-s: Semantic markup for web services. In *Proceedings of the International Semantic Web Workshop* (2001).

[28] Arkin, A., Askary, S., Fordin, S., Jekeli, W., Kawaguchi, K., Orchard, D., Pogliani, S., Riemer, K., Struble, S., Takacsi-Nagy, P., Trickovic, I., and Zimek, S. Web Service Choreography Interface (WSCI). http://www.w3.org/TR/wsci, 2002.

[29] Arnold, K., O'Sullivan, B., Scheifler, R. W., Waldo, J., and Wollrath, A. *The Jini Specification*. Addison-Wesley, 1999.

[30] Baader, F., Horrocks, I., and Sattler, U. Description logics as ontology languages for the semantic web. In *Festschrift in honor of Jörg Siekmann* (2003), D. Hutter and W. Stephan, Eds., Lecture Notes in Artificial Intelligence, Springer-Verlag.

[31] Bacon, J., Bates, J., and Halls, D. Location-oriented multimedia. *IEEE Personal Communications 4*, 5 (1997).

[32] Balaban, M. The f-logic approach for description languages. *Annals of Mathematics and Artificial Intelligence 15* (1995), 19–60.

[33] Banatre, M., and Couderc, P. Unleashing context-aware application with spatial programming. In *Keynote on IST Mobile Venue 2002: Radio Resource Management and Mobile Location Workshop* (Athens, Greece, May 2002).

[34] Bastide, R., Sy, O., and Palanque, P. Formal specification and prototyping of corba systems. In *LNCS 1628: Proceedings of ECOOP'99* (1999), vol. 1628, Springer-Verlag, pp. 474–494.

[35] Beck, J., Gefflaut, A., and Islam, N. MOCA: A Service Framework for Mobile Computing Devices. In *Proceedings of the ACM International Workshop on Data Engineering for Wireless and Mobile Access* (Seattle, WA, USA, August 1999), ACM, pp. 62–68.

[36] Beged-Dov, G., Brickley, D., Dornfest, R., Davis, I., Dodds, L., Eisenzopf, J., Galbraith, D., Guha, R., MacLeod, K., Miller, E., Swartz, A., and van der Vlist, E. RDF Site Summary (RSS). http://web.resource.org/rss/1.0/spec, May 2001.

[37] Berners-Lee, T., Hendler, J., and Lassila, O. The semantic web. *Scientific American 284*, 5 (May 2001), 34–43.

[38] Bertoa, M. F., and Vallecillo, A. Quality attributes for cots components. In *Proceedings of the 6th ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering (QAOOSE 2002)* (Malaga, Spain, June 2002).

[39] Bettstetter, C., and Renner, C. A comparison of service discovery protocols and implementation of the service location protocol. In *In Proceedings of Sixth EUNICE Open European Summer School - EUNICE 2000* (Twente, Netherlands, September 2000).

[40] Biezunski, M., Bryan, M., and Newcomb, S. R. Iso/iec 13250:2000 topic maps: Information technology – document description and markup languages, December 1999.

[41] Blau, U. *Die dreiwertige Logik der Sprache: Ihre Syntax, Semantik und Anwendung in der Sprachanalyse*. Walter de Gruyter GmbH, 1978.

[42] Bock, C. Zugriff auf Jini-Dienste mit MIDP, August 2002. Diplomarbeit an der Fachhochschule Harz.

[43] Borgida, A. On the relative expressiveness of description logics and predicate logics. *Artificial Intelligence 82*, 1-2 (1996), 353–367.

[44] Box, D., Ehnebuske, D., Kakivaya, G., Layman, A., Mendelsohn, N., Nielsen, H. F., Thatte, S., and Winer, D. Simple Object Access Protocol (SOAP). http://www.w3.org/TR/SOAP/, May 2000.

[45] Brickley, D., and Guha, R. Rdf vocabulary description language 1.0: Rdf schema. http://www.w3.org/TR/rdf-schema/, January 2003.

[46] Butler, M. H. CC/PP and UAProf: Issues, improvements and future directions. In *Proceedings of W3C Delivery Context Workshop (DIWS 2002)* (Sophia-Antipolis/France, March 2002).

[47] Caceres, R., and Padmanabhan, V. N. Fast and scalable wireless handoffs in support of mobile internet audio. vol. 3, pp. 351–363.

[48] Carolis, A. D. Internet Draft - QoS-Aware Handover for Mobile IP: Secondary Home Agent. http://www.alternic.org/drafts/drafts-d-e/draft-decarolis-qoshandover-01.pdf, November 2000.

[49] Chakraborty, D., and Chen, H. Service discovery in the future for mobile commerce. http://www.cs.umbc.edu/∼dchakr1/papers/mcommerce.html, 2000.

[50] Chakraborty, D., and Joshi, A. GSD: A Novel Group-based Service Discovery Protocol for MANETS. In *Proceedings of IEEE Conference on Mobile and Wireless Communications and Networks* (September 2002).

[51] Chen, G., and Kotz, D. A survey of context-aware mobile computing research. Tech. Rep. TR2000-381, Dartmouth, November 2000.

[52] Chen, H., Finin, T., and Joshi, A. Using OWL in a Pervasive Computing Broker. In *Proceedings of Workshop on Ontologies in Open Agent Systems (AAMAS 2003)* (2003).

[53] Chen, P.-S. The entity-relationship model - toward a unified view of data. *ACM Transaction on Database Systems 1*, 1 (March 1976), 9–36.

[54] Cheverst, K., Mitchell, K., and Davies, N. Design of an object model for a context sensitive tourist GUIDE. *Computers and Graphics 23*, 6 (1999), 883–891.

[55] Christensen, E., Curbera, F., Meredith, G., and Weerawarana, S. Web Services Description Language (WSDL). http://www.w3.org/TR/wsdl, 2001.

[56] Chtcherbina, E., and Franz, M. Peer-to-peer coordination framework (p2pc): Enabler of mobile ad-hoc networking for medicine, business, and entertainment. In *Proceedings of International Conference on Advances in Infrastructure for Electronic Business, Education, Science, Medicine, and Mobile Technologies on the Internet (SSGRR2003w)* (L'Aquila/Italy, January 2003).

[57] Connolly, D. Web Ontology (WebONT) Working Group Charter. http://www.w3.org/2001/sw/WebOnt/charter, November 2002.

[58] Couderc, P. *Mobilité contextuelle dans les systèmes d'information.* PhD thesis, Université de Rennes-1, 2001.

[59] Czerwinski, S. E., Zhao, B. Y., Hodes, T. D., Joseph, A. D., and Katz, R. H. An architecture for a secure service discovery service. In *Proceedings of Conference on Mobile Computing and Networking (MobiCom 1999)* (Seattle, WA/USA, August 1999), pp. 24–35.

[60] Decker, S., Erdmann, M., Fensel, D., and Studer, R. Ontobroker: Ontology based access to distributed and semi-structured information. In *Semantic Issues in Multimedia Systems* (Boston/USA, 1999), R. M. et al., Ed., Kluwer Academic Publisher, pp. 351–369.

[61] Dey, A. K. Understanding and using context. *Personal and Ubiquitous Computing, Special issue on Situated Interaction and Ubiquitous Computing 5*, 1 (2001).

[62] Dhara, K. K., and Leavens, G. T. Forcing behavioral subtyping through specification inheritance. In *Proceedings of the 18th International Conference on Software Engineering, Berlin, Germany* (1996), IEEE Computer Society Press, pp. 258–267.

[63] Eisenreich, G. *Lineare Algebra und analytische Geometrie*. Verlag Harri Deutsch, 1989.

[64] Femminella, M., and Pugini, F. Mobile IP Performance in High Mobility Environments. Tech. rep., Universita di Perugia, Italy.

[65] Franklin, S., and Graesser, A. Is it an agent, or just a program?: A taxonomy for autonomous agents. In *Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages* (1996), Springer Verlag.

[66] Furche, T. Rdf: Resource description framework. http://www.pms.informatik.uni-muenchen.de/lehre/seminar/ontology/01ws02/RDF/RDF.pdf, November 2001.

[67] Gaede, V., and Günther, O. Multidimensional access methods. *ACM Computing Surveys 30*, 2 (1998), 170–231.

[68] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. *Design Patterns*. Addison-Wesley, 2002.

[69] Garschhammer, M., Hauck, R., Kempter, B., Radisic, I., Roelle, H., and Schmidt, H. The MNM Service Model — Refined Views on Generic Service Management. *Journal of Communications and Networks 3*, 4 (Dec. 2001), 297–306.

[70] Ghose, A., Grossklags, J., and Chuang, J. Resilient data-centric storage in wireless ad-hoc sensor networks. In *LNCS 2574: Proceedings of the 4th International Conference on Mobile Data Management (MDM2003)* (Melbourne/Australia, January 2003), M.-S. Chen, P. K. Chrysanthis, M. Sloman, and A. Zaslavsky, Eds., Lecture Notes in Computer Science (LNCS), Springer, pp. 45–62.

[71] Gil, Y., and Ratnakar, V. A comparison of (semantic) markup languages. In *Proceedings of the 15th International FLAIRS Conference, Special Track on Semantic Web* (Pensacola, FL, USA, May 2002).

[72] Goland, Y. Y., Cai, T., Leach, P., Gu, Y., and Albright, S. Internet Draft: Simple Service Discovery Protocol 1.0. http://www.upnp.org/download/draft_cai_ssdp_v1_03.txt, October 1999.

[73] Gray, P., and Salber, D. Modelling and Using Sensed Context Information in the design of Interactive Applications. In *LNCS 2254: Proceedings of 8th IFIP International Conference on Engineering for Human-Computer Interaction (EHCI 2001)* (Toronto/Canada, May 2001), M. R. Little and L. Nigay, Eds., Lecture Notes in Computer Science (LNCS), Springer, p. 317 ff.

[74] Gregersen, H., and Jensen, C. S. Temporal entity-relationship models - a survey. Tech. Rep. Technical Report TR-3, TimeCenter, 1997.

[75] Gribble, S. D., Welsh, M., von Behren, R., Brewer, E. A., Culler, D. E., Borisov, N., Czerwinski, S. E., Gummadi, R., Hill, J. R., Joseph, A. D., Katz, R. H., Mao, Z. M., Ross, S., and Zhao, B. Y. The ninja architecture for robust internet-scale systems and services. *Computer Networks, Special Issue on Pervasive Computing 35*, 4 (March 2001), 473–497.

[76] Gruber, T. G. A translation approach to portable ontologies. *Knowledge Acquisition 5*, 2 (1993), 199–220.

[77] Guttmann, E., Perkins, C., and Kempf, J. RFC 2609: Service Templates and Service: Schemes, June 1999.

[78] Guttmann, E., Perkins, C., Veizades, J., and Day, M. RFC 2608: Service Location Protocol, Version 2, June 1999.

[79] Halpin, T. A. *Information Modeling and Relational Databases: From Conceptual Analysis to Logical Design.* Morgan Kaufman Publishers, San Francisco, 2001.

[80] Heiler, S. Semantic Interoperability. *ACM computing surveys*, 27 (1995), 271–273.

[81] Held, A., Buchholz, S., and Schill, A. Modeling of context information for pervasive computing applications. In *Proceedings of SCI 2002/ISAS 2002* (2002).

[82] Hendler, J., and McGuinnes, D. L. Darpa agent markup language. *IEEE Intelligent Systems 15*, 6 (2001), 72–73.

[83] Henricksen, K., Indulska, J., and Rakotonirainy, A. Modeling context information in pervasive computing systems, 2002.

[84] Henricksen, K., Indulska, J., and Rakotonirainy, A. Generating context management infrastructure from high-level context models. In *Industrial Track Proceedings of the 4th International Conference on Mobile Data Management (MDM2003)* (Melbourne/Australia, January 2003), pp. 1–6.

[85] Hilty, L., Behrendt, S., Binswanger, M., Bruinink, A., Erdmann, L., Fröhlich, J., Köhler, A., Kuster, N., Som, C., and Würtenberger, F. Das Vorsorgeprinzip in der Informationsgesellschaft. Auswirkungen des Pervasive Computing auf Gesundheit und Umwelt. Tech. Rep. TA 46/2003, Zentrum für Technologiefolgen-Abschätzung (TA Swiss), August 2003.

[86] Hodes, T. D., Katz, R. H., Servan-Schreiber, E., and Rowe, L. A. Composable ad-hoc mobile services for universal interaction. In *Proceedings of the 3rd ACM International Conference on Mobile Computing and Networking (MobiCom 1997)* (1997), pp. 1–12.

[87] Horrocks, I., Sattler, U., and Tobies, S. Reasoning with individuals for the description logic SHIQ. In *LNCS 1831: Proceedings of the 17th International Conference on Automated Deduction (CADE-17)* (Germany, 2000), D. MacAllester, Ed., no. 1831 in Lecture Notes in Computer Science, Springer Verlag.

[88] Howes, T. RFC 2254: The String Representation of LDAP Search Filters, December 1997.

[89] Hughes, E., McCormack, D., Barbeau, M., and Bordeleau, F. Service recommendation using slp. In *Proceedings of IEEE International Conference on Telecommunications (ICT 2001)* (Bucharest, June 2001).

[90] Indulska, J., Robinsona, R., Rakotonirainy, A., and Henricksen, K. Experiences in using cc/pp in context-aware systems. In *LNCS 2574: Proceedings of the 4th International Conference on Mobile Data Management (MDM2003)* (Melbourne/Australia, January 2003), M.-S. Chen, P. K. Chrysanthis, M. Sloman, and A. Zaslavsky, Eds., vol. 2574 of *Lecture Notes in Computer Science (LNCS)*, Springer Verlag, pp. 247–261.

[91] Januszewski, K. Registering and Discovering RSS Feeds in UDDI. http://www.gotdotnet.com/team/karstenj/docs/rss_in_uddi.aspx.

[92] Kagal, L., Korolev, V., Chen, H., Joshi, A., and Finin, T. Centaurus: A framework for intelligent services in a mobile environment. In *Proceedings of the International Workshop on Smart Appliances and Wearable Computing (IWSAWC 2001)* (April 2001).

[93] Kähkipuro, P., Marttinen, L., and Kutvonen, L. Reaching interoperability through ODP type framework. Tech. Rep. C-1996-96, Department of Computer Science, University of Helsinki, June 1996.

[94] Kaminsky, A. Running Jini Network Technology in Small Places. http://www.cs.rit.edu/∼anhinga/pmrmi.shtml, December 2000. Presentation at the Fifth Jini Community Meeting in Amsterdam/Netherlands.

[95] Kammann, J., and Blachnitzky, T. Split-proxy concept for application layer handover in mobile communication systems. In *Proceedings of the 4th IEEE Conference on Mobile and Wireless Communications Networks (MWCN 2002), Stockholm, Sept.* (2002).

[96] Kammann, J., Strang, T., and Wendlandt, K. Mobile services over short range communication. In *Workshop Commercial Radio Sensors and Communication Techniques - CRSCT 2001* (Linz/Austria, August 2001).

[97] Kifer, M., Lausen, G., and Wu, J. Logical foundations of object-oriented and frame-based languages. *ACM 42*, 4 (1995), 741–834.

[98] Klein, M., Fensel, D., van Harmelen, F., and Horrocks, I. The relation between ontologies and xml schemas. In *Electronic Trans. on Artificial Intelligence. Special Issue on the 1st International Workshop "Semantic Web: Models, Architectures and Management"* (2001).

[99] Krause, M. Eine sprache zur dynamischen komposition von kontextinformationen, August 2003. Diplomarbeit, LMU München.

[100] Lassila, O., and Swick, R. R. Resource description framework (rdf), model and syntax specification. http://www.w3c.org/RDF, February 1999.

[101] Lea, D., and Marlowe, J. Interface-based protocol specification of open systems using PSL. *Lecture Notes in Computer Science 952* (1995), 374 ff.

[102] Leonhardt, U. *Supporting Location-Awareness in Open Distributed Systems*. PhD thesis, Thesis at the Department of Computing, Imperial College of Science, Technology and Medicine, University of London, 1998.

[103] Leymann, F. Web Services Flow Language (WSFL). http://www-3.ibm.com/software/solutions/webservices/pdf/WSFL.pdf, May 2001.

[104] Liebig, V. Galileo. *Ignition* (September 2002), 2–7.

[105] Linnhoff-Popien, C. *Verteilte Systeme*. Augustinus, 1996.

[106] Linnhoff-Popien, C. *CORBA - Kommunikation und Management*. Springer Verlag, September 1998.

[107] Liskov, B., and Wing, J. A behavioural notion of subtyping. *ACM Trans. Prog. Lang. Syst.*, 16 (1994), 1811–1841.

[108] Liskov, B. H., and Wing, J. M. A behavioral notion of subtyping. *ACM Transactions on Programming Languages and Systems 16*, 6 (November 1994), 1811–1841.

[109] Ludwig, H., Dan, A., Franck, R., Keller, A., and King, R. Web Service Level Agreement (WSLA). In *IBM WebService Toolkit Documentation* (2002).

[110] Maier, A. F-logic tutorial. http://www.ontoprise.de.

[111] Maier, A., and Angele, J. Workshop ontology engineering, June 2003.

[112] Mani, A., and Nagarajan, A. Online article: Understanding quality of service for web services, 2002. Available at http://www-106.ibm.com/developerworks/library/ws-quality.html.

[113] Manner, J., Kojo, M., Suihko, T., Eardley, P., Wisely, D., BT, Hancock, R., and Georganopoulos, N. Internet draft: Mobility related terminology. Internet Engineering Task Force.

[114] Meyer, B. Systematic concurrent object-oriented programming. *Communications of the ACM (special issue, Concurrent Object-Oriented Programming, B. Meyer, editor) 36*, 9 (1993), 56–80.

[115] Michahelles, F. Designing an architecture for context-aware service selection and execution. Master's thesis, University of Munich, 2001.

[116] Miller, P. Interoperability: What is it and why should i want it? http://www.ariadne.ac.uk/issue24/interoperability, 2000.

[117] Mohan, C. Dynamic e-business: Trends in web services, 2002.

[118] Murer, T., Scherer, D., and Wuertz, A. Improving component interoperability information. In *Proceedings of Workshop on Component-Oriented Programming (WCOP'96) at 10th European Conference on Object-Oriented Programming (ECOOP'96)* (July 1996), dpunkt, pp. 150–158.

[119] Nielsen, H., Leach, P., and Lawrence, S. Internet Draft: Mandatory Extensions in HTTP (MAN). http://www.w3.org/Protocols/HTTP/ietf-http-ext/draft-frystyk-http-extensions-03.txt, March 1999.

[120] Object Management Group. Corba: Common object request broker specification and architecture. Tech. rep., Object Management Group.

[121] Oestereich, B. *Objektorientierte Softwareentwicklung: Analyse und Design mit der Unified Modeling Language*. R. Oldenburg Verlag, 1999.

[122] Ott, H. J. *Betriebswirtschaftslehre für Ingenieure und Informatiker*. Verlag Vahlen, 1995.

[123] Öztürk, P., and Aamodt, A. Towards a model of context for case-based diagnostic problem solving. In *Context-97; Proceedings of the interdisciplinary conference on modeling and using context* (Rio de Janeiro, February 1997), pp. 198–208.

[124] Paolucci, M., Kawmura, T., Payne, T., and Sycara, K. Semantic matching of web services capabilities. In *First Int. Semantic Web Conf.* (2002).

[125] Pascoe, J. Adding Generic Contextual Capabilities to Wearable Computers. In *2nd International Symposium on Wearable Computers (ISWC 1998)* (1998), pp. 92–99.

[126] Patel-Schneider, P. F., Hayes, P., and Horrocks, I. Owl web ontology language semantics and abstract syntax. http://www.w3.org/TR/owl-semantics/, March 2003.

[127] Perkins, C. E. Internet Draft - Fast Handovers for Mobile IPv6. http://www.alternic.org/drafts/drafts-p-q/draft-perkins-mobileip-handover-00.pdf, November 2000.

[128] Perkins, C. E., Ed. *Ad Hoc Networking*. 2001. ISBN 0-201-30986-9.

[129] Prakash, R., and Veeravalli, V. V. Adaptive hard handoff algorithms. *IEEE Journal on Selected Areas in Communications 18*, 11 (November 2000), 2456–2464.

[130] Ratsimor, O., Chakraborty, D., Tolia, S., Kushraj, D., Kunjithapatham, A., Gupta, G., Joshi, A., , and Finin, T. Allia: Alliance-based service discovery for ad-hoc environments. In *Proceedings of ACM Mobile Commerce Workshop* (September 2002).

[131] Reddy, S., Lowry, D., Reddy, S., Henderson, R., Davis, J., and Babich, A. Internet Draft: DAV Searching and Locating (DASL). http://www.webdav.org/dasl.

[132] Riggs, R., Taivalsaari, A., and VandenBrink, M. *Programming Wireless Devices with the Java 2 Platform, Micro Edition*. Addison Wesley, 2001.

[133] Román, M., Beck, J., and Gefflaut, A. A Device-Independent Representation for Services. In *Proceedings of 3rd Workshop on Mobile Computing Systems and Applications (WMCSA 2000)* (IBM T.J. Watson Research Center, Monterey, CA, December 2000).

[134] Ryan, N. ConteXtML: Exchanging Contextual Information between a Mobile Client and the FieldNote Server. http://www.cs.kent.ac.uk/projects/mobicomp/fnc/ConteXtML.html, August 1999.

[135] Samulowitz, M. *Contextadaptive Service Usage in Ubiquitous Computing Environments.* PhD thesis, LMU Munich, Germany, June 2002.

[136] Samulowitz, M., Michahelles, F., and Linnhoff-Popien, C. Capeus: An architecture for context-aware selection and execution of services. In *New developments in distributed applications and interoperable systems* (Krakow, Poland, September 17-19 2001), Kluwer Academic Publishers, pp. 23–39.

[137] Satyanarayanan, M. Pervasive computing: Vision and challenges. *IEEE Personal Communications* (August 2001), 10–17.

[138] Schilit, B. N., Adams, N. L., and Want, R. Context-aware computing applications. In *IEEE Workshop on Mobile Computing Systems and Applications* (Santa Cruz, CA, US, 1994).

[139] Schilit, W. N. *A System Architecture for Context-Aware Mobile Computing.* PhD thesis, Columbia University, 1995.

[140] Schmidt, A., Beigl, M., and Gellersen, H.-W. There is more to context than location. *Computers and Graphics 23*, 6 (1999), 893–901.

[141] Schmidt, A., and Laerhoven, K. V. How to build smart appliances. *IEEE Personal Communications* (August 2001).

[142] Schneiderman, R. Position - location - whose technology? *Wireless Systems Design* (March 2000), 14–20.

[143] Schöning, U. *Logik für Informatiker. Reihe Informatik, Band 56.* BI Wissenschaftsverlag, 1989.

[144] Schwickert, A. C., and Fischer, K. Der Geschäftsprozeß als formaler Prozeß - Definition, Eigenschaften, Arten. Tech. Rep. Arbeitspapiere WI Nr. 4/1996, Universität Mainz, 1996.

[145] Shekhar, S., Chawla, S., Ravada, S., Fetterer, A., Liu, X., and tien Lu, C. Spatial databases - accomplishments and research needs. *Knowledge and Data Engineering 11*, 1 (1999), 45–55.

[146] Siegl, S. Aufbereitung teilpersonalisierter Daten unterschiedlicher Quellen für den elektronischen Stadtrundgang CityWalk, September 2003. Diplomarbeit, BA Mannheim.

[147] Sriharee, N., and Senivongse, T. Discovering web services using behavioural constraints and ontology. In *LNCS 2893: Proceedings of 4th IFIP WG 6.1 International Conference on Distributed Applications and Interoperable Systems (DAIS2003)* (Paris/France, November 2003), J.-B. Stefani, I. Dameure, and D. Hagimont, Eds., vol. 2893 of *Lecture Notes in Computer Science (LNCS)*, Springer Verlag, pp. 248–259.

[148] Steinfeld, E. F. Devices that play together, work together. *EDN Magazine* (September 2001).

[149] Strang, T. Patentanmeldung DE 102 34 747.6: Verfahren zur Dienstsuche in einem Weitverkehr-Kommunikationsnetz, July 2002. (pending).

[150] Strang, T. Towards Autonomous Context-Aware Services for Smart Mobile Devices. In *LNCS 2574: Proceedings of the 4th International Conference on Mobile Data Management (MDM2003)* (Melbourne/Australia, January 2003), M.-S. Chen, P. K. Chrysanthis, M. Sloman, and A. Zaslavsky, Eds., vol. 2574 of *Lecture Notes in Computer Science (LNCS)*, Springer Verlag, pp. 279–293.

[151] Strang, T., and Linnhoff-Popien, C. Service Interoperabilität auf Kontextebene (Contextual Service Interoperability). In *Proceedings of Workshop XML-Technologien für Middleware / Middleware für XML-Anwendungen (XMIDX2003)* (Berlin/Germany, February 2003), R. Eckstein and R. Tolksdorf, Eds., GI-Edition Lecture Notes in Informatics (LNI), Gesellschaft für Informatik (GI), p. 95ff.

[152] Strang, T., and Linnhoff-Popien, C. Service Interoperability on Context Level in Ubiquitous Computing Environments. In *Proceedings of International Conference on Advances in Infrastructure for Electronic Business, Education, Science, Medicine, and Mobile Technologies on the Internet (SSGRR2003w)* (L'Aquila/Italy, January 2003).

[153] Strang, T., Linnhoff-Popien, C., and Frank, K. Applications of a Context Ontology Language. In *Proceedings of International Conference on Software, Telecommunications and Computer Networks (SoftCom2003)* (Split/Croatia, Venice/Italy, Ancona/Italy, Dubrovnik/Croatia, October 2003), D. Begusic and N. Rozic, Eds., Faculty of Electrical Engineering, Mechanical Engineering and Naval Architecture, University of Split, Croatia, pp. 14–18.

[154] Strang, T., Linnhoff-Popien, C., and Frank, K. CoOL: A Context Ontology Language to enable Contextual Interoperability. In *LNCS 2893: Proceedings of 4th IFIP WG 6.1 International Conference on Distributed Applications and Interoperable Systems (DAIS2003)* (Paris/France, November 2003), J.-B. Stefani, I. Dameure, and D. Hagimont, Eds., vol. 2893 of *Lecture Notes in Computer Science (LNCS)*, Springer Verlag, pp. 236–247.

[155] Strang, T., Linnhoff-Popien, C., and Frank, K. Integration Issues of an Ontology based Context Modelling Approach. In *Proceedings of IADIS International Conference WWW/Internet (ICWI2003)* (Carvoeiro, Algarve, Portugal, November 2003), P. Isaias and N. Karmakar, Eds., vol. 1, IADIS, pp. 361–368.

[156] Strang, T., Linnhoff-Popien, C., and Roeckl, M. Highlevel Service Handover through a Contextual Framework. In *Proceedings of 8th International Workshop on*

Mobile Multimedia Communications (MoMuC2003) (Munich/Germany, October 2003), J. Kaefer and M. Zuendt, Eds., vol. 1, Center for Digital Technology and Management (CDTM), pp. 405–410.

[157] Strang, T., and Meyer, M. Agent-environment for small mobile devices. In *Proceedings of the 9th HP OpenView University Workshop (HPOVUA)* (June 2002), HP.

[158] Swartout, W., Patil, R., Knight, K., and Russ, T. Towards distributed use of large-scale ontologies. In *Proceedings of the 10 th Knowledge Acquisition for Knowledge-Based Systems Workshop (KAW'96)* (1996).

[159] Tanenbaum, A. S. *Distributed Systems.* Prentice Hall, 2002.

[160] Tessaris, S. *Questions and Answers: Reasoning and Querying in Description Logic.* PhD thesis, 2001.

[161] The Distributed Group. Online article: Interoperability. Available at http://www.lsi.us.es/∼tdg/doc/topics/interoperability.html.

[162] Treu, G. Context-aware service discovery in mobile ad-hoc networks, Approx. February 2004. Diplomarbeit, LMU München.

[163] Uschold, M., and Grüninger, M. Ontologies: Principles, methods, and applications. *Knowledge Engineering Review 11*, 2 (1996), 93–155.

[164] Vallecillo, A., Hernández, J., and Troya, J. M. Component interoperability. Tech. Rep. ITI-2000-37, Departmento de Lenguajes y Ciencias de la Computación, University of Málaga, July 2000. Available at http://www.lcc.uma.es/∼av/Publicaciones/00/Interoperability.pdf.

[165] Vallecillo, A., Hernandez, J., and Troya, J. M. Woi'00: New issues in object interoperability. In *LNCS 1964: ECOOP'2000 Workshop Reader* (2000), Springer, pp. 256–269.

[166] Volz, R., Decker, S., and Oberle, D. Bubo - implementing owl in rule-based systems. In *Proceedings of WWW 2003* (Budapest/Hungary, May 2003).

[167] Vossen, G. *Datenmodelle, Datenbanksprachen und Datanbank-Management-Systeme, 2. Auflage.* Addison-Wesley, 1994.

[168] W3C. Composite Capabilities / Preferences Profile (CC/PP). http://www.w3.org/Mobile/CCPP.

[169] Wahl, M., Howes, T., and Kille, S. RFC 2251: Lightwight Directory Access Protocol (v3), December 1997.

[170] Want, R., Hopper, A., Falcao, V., and Gibbons, J. The Active Badge Location System. Tech. Rep. 92.1, Olivetti, ORL, 24a Trumpington Street, Cambridge CB2 1QA, 1992.

[171] WAPForum. User Agent Profile (UAProf). http://www.wapforum.org.

[172] Wegner, P. Interoperability. *ACM computing surveys*, 28 (1996), 285–287.

[173] Weiser, M. The computer for the 21st century. *Scientific American 265*, 30 (September 1991), 94–104.

[174] Weller, S. Online article: Web services qualification, 2002. Available at http://www-106.ibm.com/developerworks/webservices/library/ws-qual/?dwzone=webservices.

[175] Wohed, P., van der Aalst, W. M., Dumas, M., and ter Hofstede, A. H. Pattern based analysis of bpel4ws. Tech. Rep. FIT-TR-2002-04, Queensland University of Technology, Brisbane/Australia, 2002.

[176] Wolfson, O., Xu, B., Chamberlain, S., and Jiang, L. Moving objects databases: Issues and solutions. In *Statistical and Scientific Database Management* (1998), pp. 111–122.

[177] Wong, D., and Lim, T. J. Soft Handoffs in CDMA Mobile Systems. *IEEE Personal Communications Magazine 4*, 6 (December 1997), 6–17.

[178] Yellin, D. M., and Strom, R. Protocol specifications and component adaptors. *ACM Trans. Prog. Lang. Syst.*, 19 (1997), 292–333.

[179] Zarchan, P. Global positioning system: Theory and applications, 1996.

[180] Zaremski, A. M., and Wing, J. M. Specification matching of software components. *ACM Transactions on Software Engineering and Methodology 6*, 4 (1997), 333–369.

Abkürzungs- und Symbolverzeichnis

3GPP	3rd Generation Partnership Project
Abs	Absolute
AL	Aussagenlogik
ASC	Aspect-Scale-ContextInformation
CAP	Context Aware Packet
CC/PP	Composite Capabilities / Preferences Profile
CCML	Centaurus Capability Markup Language
CE	Context Extension
CI	Context Information
CMAP	Context Management Access Point
CoOL	Context Ontology Language
CoBrA	Context Broker Architecture
CORBA	Common Object Request Broker Architecture
CSCP	Comprehensive Structured Context Profiles
CSD	Circuit Switched Data
CVM	Compact Virtual Machine
DA	Directory Agent
DAML	DARPA Agent Markup Language
DAML-S	DAML Services
DAML+OIL	DARPA Agent Markup Language + Ontology Interference Layer
DARPA	Defense Advanced Research Projects Agency
DASL	DAV Searching and Locating
DAV	Distributed Authoring and Versioning
DCOM	Distributed Component Object Model
DHCP	Dynamic Host Configuration Protocol
DII	Dynamic Invocation Interface
DL	Description Logic
dm^3	Kubikdezimeter
DNS	Domain Name Service
DoD	Department of Defense
DOP	Dilution of Precision
DSI	Dynamic Skeleton Interface
DTD	Document Type Definition
EFA	Elektronische Fahrplanauskunft
ER	Entity-Relationship
EUR	Euro
FL	Frame Logic
FOL	First Order Logic

GK	Gauß-Krüger
GNSS	Global Navigation Satellite System
GPRS	General Packet Radio Service
GPS	Global Positioning System
GSM	Global System for Mobile Communications
HO	Handover
HOL	High Order Logic
HSCSD	Highspeed Cirquit Switched Data
HTTP	Hypertext Transfer Protocol
HTTPMU	HTTP Multicast UDP
HTTPU	HTTP Unicast UDP
ID	Identifikator
IDL	Interface Definition Language
IETF	Internet Engineering Task Force
IMEI	International Mobile Equipment Identity
Inst	Instanz
IP	Internet Protocol
ISO	International Organization for Standardization
IST	Information Society Technology
JVM	Java Virtual Machine
KI	Künstliche Intelligenz
km	Kilometer
km/h	Kilometer pro Stunde
KVM	Kilobyte Virtual Machine
L2CAP	Logical Link Control and Adaptation Protocol
LBS	Location Based Service
LDAP	Lightweight Directory Access Protocol
LUS	Lookup Service
m	Meter
MAN	Mandatory Extensions in HTTP
MDNS	Multicast DNS
MExE	Mobile Execution Environment
MNM	Munich Network Management
NAICS	North American Industry Classification System
NCS	Non-Carrier Service
NLP	Natural Language Processing
NM	Nautical Miles
OIL	Ontology Interference Layer
ORB	Object Request Broker

ORM	Object Role Model
OSI	Open Systems Interconnection
OWL	Web Ontology Language
OWL-DL	Web Ontology Language - Description Logic
PC	Personal Computer
PDA	Personal Digital Assistent
PL	Prädikatenlogik
PoI	Point of Interest
PPDL	Pervasive Profile Description Language
QoS	Quality of Service
RDF	Ressource Description Framework
RDF-S	RDF Schema
RDF/S	RDF und RDF Schema
Rel	Relative
RFC	Request for Comment
RMI	Remote Method Invocation
RPC	Remote Procedure Call
RSS	RDF Site Summary
SDP	Service Discovery Protocol
SDS	Service Discovery Service
SdT	Stand der Technik
sec	Sekunde
SGML	Standard Generic Markup Language
SLA	Service Level Agreement
SLP	Service Location Protocol
SLP-DA	SLP Directory Agent
SOAP	Simple Object Access Protocol
SOAP-ENV	SOAP Envelope
SPO	Subjekt Prädikat Objekt
SSDP	Simple Service Discovery Protocol
SSR	Smart Stack Routing
TEA	Technology for Enabled Awareness
UAProf	User Agent Profiles
UbiComp	Ubiquitous Computing
UDP	User Datagram Protocol
UDDI	Universal Description, Discovery and Integration
UML	Unified Modeling Language
UNSPSC	United Nations Standard Products and Services Code
UPnP	Universal Plug and Play
URI	Uniform Ressource Identificator
URL	Uniform Ressource Locator

UTC	Universal Coordinated Time
UUID	Universal Unique IDentifier
W3C	World Wide Web Consortium
WGS84	World Geodetic System 1984
WSCI	Web Service Choreography Interface
WSDL	Web Service Description Language
WSFL	Web Service Flow Language
WSLA	Web Service Level Agreement
XML	Extensible Markup Language
ZV	Zustandsvariable

F-Logic Symbolik

`=>`	einwertige Relation auf Konzeptebene
`=>>`	mehrwertige Relation auf Konzeptebene
`->`	einwertige Relation auf Instanzebene
`->>`	mehrwertige Relation auf Instanzebene
`I:C`	`instanceOf`-Relation, I ist Instanz von C
`C1::C2`	`subconceptOf`-Relation, C1 ist Unterkonzept von C2
`S[P1->>O1; P2->>O2]`	Subjekt-Prädikat-Objekt-Modell, Subjekt S hat mehrwertige Prädikate P1 und P2, denen jeweils die Objekte O1 und O2 zugeordnet werden ([und] beschreiben den Frame)
`S[P@(I1,I2)=>O]`	Objektorientierte Sicht, Methode P einer Klasse S hat zwei Eingabeparameter vom Typ I1 und I2, und liefert Objekte vom Typ O zurück

Index

Lebenslauf

Persönliche Daten

Name:	Thomas Strang
Anschrift:	Untere Läng 5a
	82205 Gilching
Geburtsdatum:	15. Januar 1972
Geburtsort:	Eschweiler
Familienstand:	verheiratet, 1 Kind

Ausbildung

1978 – 1982	Grundschule in Stolberg/Rhld.
1982 – 1991	Goethe-Gymnasium in Stolberg/Rhld.
1991 – 1998	Studium der Informatik mit Nebenfach Elektrotechnik und Vertiefungsgebiet Kommunikationssysteme an der RWTH Aachen
seit 2000	Promotion im Fachbereich Informatik an der Ludwig-Maximilians-Universität München

Berufserfahrung

1988 – 1990	Anstellung als Hilfskraft bei der Firma MICOS in Würselen
1990 – 1998	Studienbegleitende Anstellung als Softwareentwickler und Systemingenieur auf Stundenbasis bei der Firma MICOS in Würselen
1998 – 2000	Vollzeit-Anstellung als Projektingenieur bei der Firma MICOS in Würselen
seit 2000	Anstellung als Wissenschaftlicher Mitarbeiter beim Deutschen Zentrum für Luft- und Raumfahrt (DLR) am Institut für Kommunikation und Navigation in Oberpfaffenhofen