Universität Rostock
Fachbereich Informatik

Situationsgesteuerter mobiler Zugriff auf Digitale Bibliotheken

vorgelegt von Peter Haase
geboren am 31.05.1976 in Neubrandenburg
Erstgutachter: Prof. Dr. Andreas Heuer
Zweitgutachter: Prof. Dr. Bodo Urban
Betreuer: Dr. Holger Meyer, Dr. Thomas Kirste, Astrid Lubinski
Abgabetermin: 14.11.2001

Zusammenfassung

Die Technologie des Mobile Computing ermöglicht es uns bereits heute, auf beliebige Informationen von beinahe jedem Ort und zu jeder Zeit zuzugreifen. Durch die ständig wachsende Menge digitaler Inhalte gewinnt jedoch folgende Frage an Bedeutung: Wie erhalte ich die *richtige* Information am *richtigen* Ort zur *richtigen* Zeit. Insbesondere in mobilen Umgebungen erweist es sich als sinnvoll, Kontextinformationen — Informationen über die Situation des Nutzers und seine Umgebung — zu berücksichtigen, um situationsrelevante Informationen proaktiv bereitstellen zu können.

In dieser Arbeit wird eine Systemarchitektur für einen situationsgesteuerten, mobilen Zugriff auf Digitale Bibliotheken entwickelt und prototypisch implementiert. Als Anwendungsdomäne werden dabei Ausstellungs- und Konferenzbesuche betrachtet.

Abstract

Using state of the art mobile computing technology it is today possible to access any information at almost any place and any time. But considering the rapidly growing amounts of digital content, the following question gains importance: How do I find the *right* information at the *right* place and the *right* time. Specifically in mobile environments it is useful to exploit context information, i.e. information about the situation of the user and its surroundings, to enrich human computer interaction by proactively providing situation relevant information.

In the scope of this master's thesis, a system architecture for situation aware mobile access to Digital Libraries has been developed and prototypically implemented. For demonstration, the domain of conferences and exhibitions has been chosen.

CR-Klassifikation

- **H.3.3** Information Search and Retrieval
- **H.3.7** Digital Libraries
- **H.5.1** Multimedia Information Systems
- **I.2.1** Applications and Expert Systems
- **I.2.4** Knowledge Representation Formalisms and Methods

Keywords

Digital Libraries, Knowledge Representation, Mobile Computing, Intelligent Assistance, Situation Awareness

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

Die Entwicklung der Informations- und Kommunikationstechnologie, insbesondere des Mobile Computing, hat uns der Möglichkeit des Zugriffs auf jede Information von jedem Ort zu jeder Zeit sehr nahe gebracht. Durch die ständig zunehmende Menge an digitalen Inhalten gewinnt jedoch folgende Frage an Bedeutung: Wie erhalte ich die *richtige* Information am *richtigen* Ort zur *richtigen* Zeit. Gerade in mobilen Szenarien wäre es wünschenswert, wenn der Computer "wüsste", welche Informationen in der momentanen Situation die *richtigen* sind, ohne dass der Nutzer in einem langwierigen Prozess nach ihnen suchen muss. Eine neue Qualität in der Interaktion zwischen Mensch und Computer kann hier also erreicht werden, wenn die Situation des Nutzers analysiert wird und situationsrelevante Inhalte proaktiv bereitgestellt werden.

Digitale Bibliotheken haben sich bewährt, große Mengen von digitalen Inhalten zu verwalten. Sie bieten umfangreiche Recherchemöglichkeiten durch Nutzung von Datenbank- und Information-Retrieval-Techniken. Um allerdings einen situationsgesteuerten Zugriff auf diese Inhalte zu realisieren, ohne dass eine explizite Beschreibung des Gesuchten durch den Nutzer notwendig wird, sind neue Konzepte notwendig.

In dieser Arbeit wird eine Systemarchitektur entwickelt, die einen situationsgesteuerten mobilen Zugriff auf Digitale Bibliotheken erlaubt. Eine zentrale Rolle spielt dabei eine geeignete Datenmodellierung, auf deren Basis intelligentes, automatisches Schließen möglich ist. Weitere Schwerpunkte bilden die Situationsanalyse und eine effiziente Zugriffsarchitektur.

Als Anwendungssbeispiel wird ein mobiles Assistenzsystem für Ausstellungs- und Konferenzbesucher entwickelt.

1.2 Vorstellung der Beispieldomäne

Die in dieser Arbeit entwickelte Architektur wird anhand der Domäne des Ausstellungs- und Konferenzbesuches demonstriert werden.

Das folgende Beispiel verdeutlicht, wie ein Assistenzsystem auf einem Konferenzbesuch eingesetzt werden kann. Es wird als Standardbeispiel in der Arbeit Verwendung finden.

Beispiel eines Konferenzbesuches Man stelle sich vor, es findet eine Konferenz statt, bei der die Teilnehmer mit mobilen Endgeräten, z.B. PDAs oder Notebooks, ausgestattet sind. Das mobile Gerät dient dem Teilnehmer als Assistent, der ihm entsprechend seiner Situation relevante Information präsentiert.

So werden dem Teilnehmer zunächst Veranstaltungen zu Themen seines Interesses vorgeschlagen. Er entscheidet sich für den Besuch eines Vortrages. Sobald er den Konferenzraum betritt, erhält er automatisch Informationen zum Vortrag und zum Vortragenden. Er hat die Möglichkeit, die zugehörigen Vortragsunterlagen auf seinem mobilen Gerät zu betrachten und auch zu speichern. Nach dem Vortrag entscheidet sich der Besucher, Projektpräsentationen anzusehen, die an verschiedenen Ständen dargeboten werden. Während er sich über das Gelände bewegt, versorgt ihn der Assistent mit Informationen zu den Präsentationen, sobald er sich einem Stand nähert, der für ihn von Interesse ist. Der Besucher verfügt über ein PDA der neuesten Generation, das auch Videos darstellen kann. Daher werden ihm zu den Präsentationen speziell Videodokumente angeboten. Eine Demonstration beeindruckt ihn besonders und er hat weitergehende Fragen. Leider ist der Projektverantwortliche momentan nicht zugegen. Deshalb notiert sich der Besucher als Aufgabe, seine Fragen an die Person später zu stellen. Zu einem späteren Zeitpunkt teilt der Assistent ihm plötzlich mit, dass die Person, der er eine Frage stellen wollte, in der Nähe ist. Er erhält Informationen zu der Person und ein Foto, anhand dessen er die Person sofort erkennt.

Dieses Beispiel verdeutlicht bereits einige wichtige Aspekte der Domäne des Ausstellungs- und Konferenzbesuches:

- Sie ist hochdynamisch und bietet eine breite Vielfalt von Kontextinformationen.
- Sie bietet viele verschiedene, parallele Aktivitäten. Auf einer Konferenz sind dies z.B. Präsentationen, Meetings, Gespräche, etc.
- Sie erfordert die Bereitstellung von multimedialen Informationen. Dabei kann es sich um vortragsbegleitende Folien, Videos, Publikationen, etc. handeln.

1.3 Digitale Bibliotheken

Die Informations- und Wissensgesellschaft hat sich zum Leitbild unserer Zukunft entwickelt. Wissen ist zum entscheidenden Wirtschaftsfaktor geworden. Der Zugriff auf Wissen hat deshalb eine zentrale Bedeutung für unsere Gesellschaft. Bibliotheken spielen seit jeher eine besondere Rolle bei der Wissensversorgung. Mit Digitalen Bibliotheken bieten sich durch die digitale Form der Dokumente nicht nur neue Speicherungs-, Übertragungs- und Verarbeitungsmöglichkeiten, sondern auch neue Möglichkeiten im Umgang mit Wissen.

Nach [EF00] ist eine digitalen Bibliothek eine Einrichtung,

- die Texte, Bilder, Animationen, Ton- und Videoaufnahmen auf elektronischen Datenträgern vorhält,
- die eine Vielzahl von Bibliotheksdiensten in einem ortsübergreifenden Verbund anbietet und
- deren Bestände und deren Dienste integriert sind, die einen effizienten Zugriff darauf über eine einheitliche Systemoberfäche gestattet und deren "Systemintelligenz" über die der Teile hinausgeht.

Sowohl in Bezug auf Inhalte als auch Dienste bieten Digitale Bibliotheken also eine neue Qualität im Vergleich mit konventionellen Bibliotheken.

Im Kontext der hier gewählten Domäne von Konferenz- und Ausstellungsbesuchen handelt es sich bei den Inhalten um wissenschaftliche Publikationen (z.B. Artikel, Technical Reports), Vortragsunterlagen, Produktinformationen, etc. in Form von Volltexten oder multimedialen Dokumenten. Zu diesen zumeist semi- und unstrukturierten Inhalten existieren weiterhin Metadaten. Metadaten sind Daten, die die digitalen Dokumente der Digitalen Bibliothek beschreiben, also z.B. bibliographische Angaben. In Bezug auf die Dienste der Bibliothek stehen hier vor allem Anfrage- und Retrieval-Dienste im Vordergrund, die das Suchen nach relevanten Inhalten und eine effiziente Bereitstellung dieser Inhalte erlauben. Des weiteren spielen Profildienste, die eine persönliche Sicht auf die Digitale Bibliothek erlauben, sowie die Visualisierung der Inhalte eine Rolle.

Künftig wird sich die Rolle Digitaler Bibliotheken vom klassischen Dokumentenmanagementsystem hin zum Wissensportal bewegen. Dies erfordert natürlich eine neue Form der Repräsentation der Inhalte der Bibliothek: eine wissensbasierte Darstellung unter Ausnutzung semantischer Informationen wird erforderlich. Interessante Ansätze bieten hier Wissensbanken. Sie realisieren eine explizite Verwaltung von Wissen und nutzen dabei Techniken deduktiver Datenbanken und der Wissensrepräsentation.

Allerdings wird kaum möglich sein, den gesamten Inhalt einer Bibliothek formal mit Techniken der Wissensrepräsentation darzustellen. Daher spielen bei der wissensbasierten Darstellung die Metadaten eine herausragende Rolle. Die wissensbasierte Modellierung der Metadaten wird ein zentraler Gegenstand dieser Arbeit sein.

1.3.1 BlueView

Im BlueView Projekt der Universität Rostock [blub] werden Digitale Bibliotheksdienste basierend auf der Architektur von *Virtuellen Dokumentenservern* (VDS) entwickelt. Mit Standardtools wie Volltextdatenbanken, Information-Retrieval- Systemen, objektrelationalen Datenbankmanagementsystemen und Replikations- und Caching-Diensten werden verschiedene heterogene *lokale Dokumentenserver* (LDS) in einem Virtuellen Dokumentenserver vereint. Dies erlaubt den Zugriff auf heterogene Datenquellen über eine einzige Anfrageschnittstelle, die die Möglichkeiten der verwendeten Datenbankanfragesysteme, Information-Retrieval- und Suchdienste integriert. Die verteilte Anfragebearbeitung von BlueView ermöglicht die Kombination der Suche über strukturierten Metadaten mit inhaltsbasierter Suche über Volltexten. Die Architektur ermöglicht eine Replikation sowohl von Metadaten als auch von Volltexten. Außerdem werden lokale Nutzersichten unterstützt.

Abbildung 1.1 zeigt einen Überblick über die BlueView Architektur. Für einen

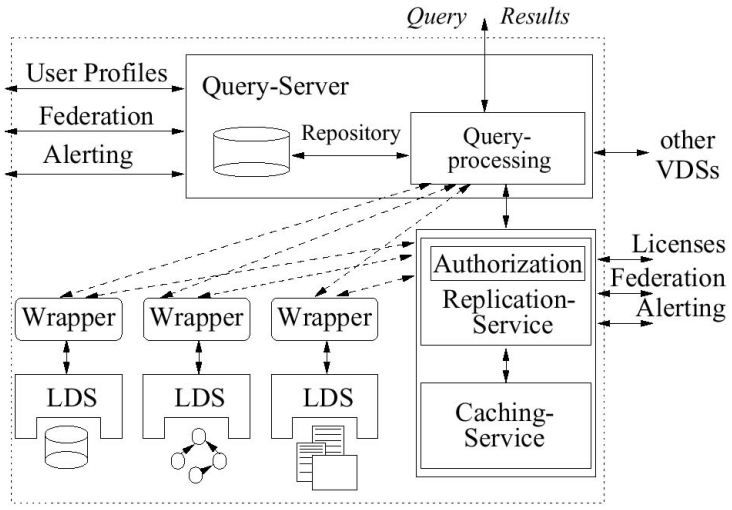

Abbildung 1.1: Der Virtuelle Dokumentenserver der BlueView-Architektur

umfassenden Überblick über das BlueView-Projekt sei auf [HMTP00] verwiesen. In [Tit99] wird eine komplette Architekturbeschreibung für einen Virtuellen Dokumentenserver vorgestellt. In [HD01] wird die Realisierung konkreter Virtueller und lokaler Dokumentenserver beschrieben.

Das BlueView-Projekt ist in Bezug auf folgende Aspekte relevant für die hier vorliegende Arbeit:

Zum einen wird sich zeigen, dass für den Zugriff auf eine Digitale Bibliothek über ein mobiles Endgerät ähnliche Dienste erforderlich sind wie beim transparenten Zugriff auf lokale Dokumentenserver über einen Virtuellen Dokumentenserver. Dies betrifft insbesondere den verteilten Anfrage- und Retrieval-Dienst einschließ-

lich der Konzepte für Caching und Replikation. In der in dieser Arbeit vorgestellten Architektur werden entsprechend ähnliche Komponenten wie in der BlueView-Architektur Verwendung finden.

Zum anderen kann ein Virtueller Dokumentenserver auf der Serverseite direkt in die hier vorgestellte Architektur integriert werden. Damit ließe sich vom mobilen Endgerät aus ein transparenter Zugriff auf verschiedene heterogene Datenquellen realisieren. Dabei können die Dokumentbestände bereits existierender lokaler Dokumentenserver weiterverwendet werden.

1.4 Situationsgesteuerte mobile Assistenz

Die Technologie der portablen und ultraportablen Computer ermöglicht heute eine mobile Assistenz in Umgebungen und Situationen, in denen es bis vor kurzem noch undenkbar war. Die Kombination mit der Technologie der drahtlosen Kommunikation erlaubt den Zugriff auf beliebige Informationen an (beinahe) jedem Ort und zu jeder Zeit. Um eine allgegenwärtige Computerunterstützung ins tägliche Leben sinnvoll zu integrieren, ohne den Menschen von seinen primären Aufgaben abzulenken, bedarf es jedoch weiterer neuer Konzepte.

In der Interaktion zwischen Menschen ist die Berücksichtigung des *Kontextes* eine Selbstverständlichkeit, ein großer Teil der Informationen wird ohne explizite Kommunikation ausgetauscht. Erlaubt man auch dem Computer Zugriff auf Informationen über den Kontext des Nutzers, so kann eine neue Qualität in der Interaktion zwischen Mensch und Computer erreicht werden. Die Kontextinformation kann dann beispielsweise genutzt werden, um dem Menschen proaktiv situationsrelevante Informationen bereitzustellen.

Einen Überblick über die aktuelle Forschung im Bereich Context-Aware Mobile Computing gibt [CK00].

Definitionen von Kontext In [Dey99] wird Kontext definiert als die Information, die genutzt werden kann, um die Situation einer Entität zu beschreiben. Eine Entität ist dabei eine Person, ein Ort oder ein Objekt, das für die Interaktion zwischen dem Nutzer und der Applikation relevant ist, einschließlich Nutzer und Applikation selbst.

Schilit versucht in [Sch95] Kontext zu definieren, indem er eine Einteilung von Kontext in verschiedene Kategorien vornimmt:

- *Computing context* wie Netzwerkverbindungen, zur Verfügung stehende Ressourcen (etwa Drucker, Display), etc.
- *User context* wie Nutzerinteressen und -profil, Ort, soziale Situation (z.B. Teilnahme an einem Meeting), etc.
- *Physical Context* wie Licht-, Geräuschverhältnisse, Temperatur, etc.

Diese Einteilung findet sich in ähnlicher Form in vielen Arbeiten zu kontextsensitiven Systemen wieder, so auch im SAMoA-Framework (siehe 1.4.1). In [AF00] und [Fis01] jedoch ist eine weitere Kategorie von Kontext zu finden, nämlich die Beschreibung der Domäne, in der die Interaktion stattfindet. Zur Beschreibung der Domäne werden *Boundary Objects* eingeführt. Boundary Objects sind Objekte, über die ein gemeinsames Verständnis existiert und die die Interaktion zwischen Menschen sowie Mensch und Computer unterstützen. An dieser Stelle sei schon vorweggenommen, dass auch in dieser Arbeit die Beschreibung der Domäne auf der Basis einer Ontologie eine entscheidende Rolle spielt. Allerdings ist hier die Domänenbeschreibung aufgrund der Festlegung auf eine konkrete Domäne (z.B. Ausstellung oder Konferenz) nicht Bestandteil des Situationsmodells, sehr wohl aber Bestandteil des Inferenzprozesses.

Häufig wird als einzige Kontextinformation nur der Ort verwendet. Dies ist in zahlreichen Arbeiten festgestellt und bemängelt worden, so z.B. in [SBG98]. Andererseits spielt der Ort eine herausragende Rolle unter den Kontextkomponenten, da aus der Ortsangabe oftmals viele weitere Kontextinformationen abgeleitet werden können. So lassen sich zum Beispiel aus dem Aufenthalt einer Person in einem Konferenzraum Schlussfolgerungen ziehen über die soziale Situation.

Nutzung von Kontextinformation Kontextinformationen können auf vielfältige Art und Weise genutzt werden. Schilit [Sch94] unterteilt kontextsensitive Applikationen in Kategorien der Nutzung von Kontextinformation:

1. *Proximate selection*, eine Technologie für User-Interfaces, die Objekte in der Umgebung oder von besonderem Interesse hervorhebt.
2. *Automatic contextual reconfiguration*, der Prozess des dynamischen Hinzufügens oder Entfernens von Komponenten oder Verbindungen zwischen diesen Komponenten
3. *Contextual information and commands*, welche verschiedene Ergebnisse in Abhängigkeit vom Kontext hervorbringen, beispielsweise Informationen über die Umgebung des Nutzers.
4. *Context triggered actions*, einfache IF-THEN Regeln, um das kontextsensitive Verhalten zu spezifizieren.

Während diese Einteilung Klassen von kontextsensitiven Applikationen identifiziert, beschreibt Dey [DA99] drei allgemeine Kategorien von kontextsensitiven Funktionen, die Applikationen unterstützen können:

1. *Presentation*: die Darstellung von Informationen und Diensten
2. *Execution*: die Ausführung eines Dienstes
3. *Tagging*: das Markieren von Informationen für späteres Retrieval

Der Kontextbegriff in dieser Arbeit orientiert sich stark an dem Situationsmodell des SAMoA-Frameworks. Dieses soll im folgenden vorgestellt werden.

1.4.1 Das SAMoA Framework

Das SAMoA Framework (Situation Awareness in Mobile Assistance), entwickelt am Fraunhofer IGD, ist ein Modell für Assistenzsysteme für das Management des persönlichen Tagesgeschäfts.

In [KCI98] werden Anforderungen an ein solches Assistenzsystem genannt: Es sollte

- die verschiedenen Aufgaben des Nutzers verwalten,
- es erlauben, Aufgaben hinzuzufügen oder zu ändern, wenn die Situation es verlangt,
- in Reaktion auf Ereignisse bestimmte Aufgaben in den Vordergrund bringen, z.B. durch die Darstellung relevanter Kontextinformation,
- schnellen Zugriff auf wichtige Informationen bieten,
- bei der Orientierung in neuen Situationen unterstützen.

Das Modell des SAMoA Framework, wie in [KCI98] beschrieben, beruht auf folgenden Konzepten:

- *Aufgaben*: Der Nutzer kann mehrere voneinander unabhängige Aufgaben bzw. Ziele verfolgen, die er parallel bearbeitet. Jeder Aufgabe ist ein Zustand zugeordnet. Zu jeder Aufgabe bedarf es weiterhin einer Beschreibung, die die Struktur der Aufgabe definiert.
- *Kontext*: Oftmals sind mit einer Aufgabe Informationen verbunden, die für ihre Erfüllung notwendig, aber nicht Bestandteil der Aufgabendefinition sind (z.B. Dokumente, Notizen, Skizzen). Diese Sammlung von Informationen heißt Kontext. Dabei kann eine Information Bestandteil mehrerer Kontexte sein, aber jede Aufgabe besitzt nur einen Kontext. [1]
- *Situation*: Die Situation eines Nutzers umfasst folgende Aspekte:
  - Umgebung des Nutzers (z.B. Objekte in seiner Reichweite) und seine Position
  - Aktive Aufgaben des Nutzers mit entsprechenden Zuständen
  - Persönliche Nutzerprofile
- *Dynamische Dialogerzeugung*: Dialogkomponenten sollen sich nahtlos in reale Arbeitsabläufe integrieren, wobei minimale kognitive Voraussetzungen für die Dialogbedienung durch den Benutzer gesichert sein müssen.

[1] Das Konzept Kontext dient also hier nicht zur Beschreibung von Situationen, wie sonst in der Literatur üblich, sondern ist eine Sammlung von Informationen, die mit einer Aufgabe (Task) verbunden ist.

Andererseits sollen Aufgabendefinitionen ad hoc hinzugefügt oder modifiziert werden können. Außerdem ist die Entscheidung, wann eine Teilaufgabe ausgeführt wird, vollständig situationsgesteuert. Dies bedeutet, dass das User Interface dynamisch erzeugt werden muss.

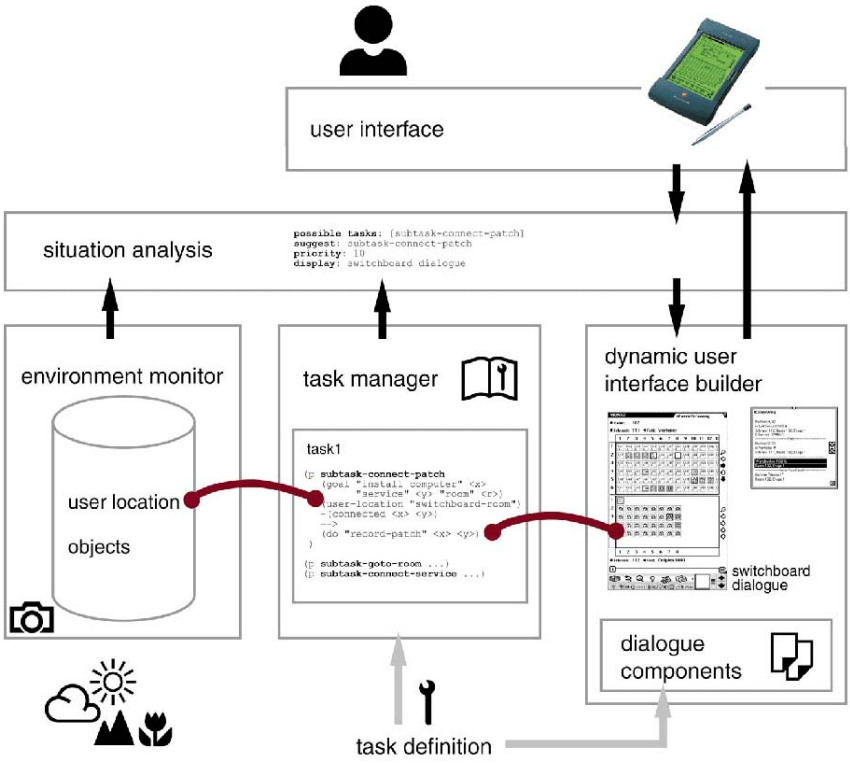

Abbildung 1.2: Die SAMoA Architektur

Abbildung 1.2, ebenfalls aus [KCI98] gibt einen vereinfachten Überblick über die SAMoA Architektur, auf deren Basis diese Konzepte implementiert werden. Zentrale Komponenten sind der Task Manager, der für die Verwaltung der aktiven Aufgaben mit deren Zustand verantwortlich ist, eine Monitoring-Komponente zur Erfassung der Umgebung des Nutzers, eine Komponente zur Situationsanalyse, die Aufgaben zur Ausführung auswählt, sowie die Komponente zur dynamischen Dialogerzeugung.

Eine formalere Beschreibung des SAMoA Modells (in Z-Notation) ist in [Kir00] zu finden.

eGuide: eine Anwendung des SAMoA-Frameworks Konzepte des SAMoA-Frameworks fanden bereits Anwendung in verschiedenen Systemen aus den Bereichen Netzwerkadministration, Medizinische Datenerfassung und Messeführung.

Im eGuide-Projekt [BG01] werden auf der Basis ultraportabler Geräte mobile Ausstellungsführer für Messen entwickelt. Sie wurden bereits auf mehreren Ausstellungen eingesetzt, darunter beispielsweise auf der CeBIT 2000 als offizieller elektronischer Ausstellungsführer.

Die wichtigsten Kontextinformationen in den eGuide-Systemen stellen Informationen über die physische Umgebung des Nutzers dar. Hierzu gehören die aktuelle Position und Objekte in der Umgebung. Zur Positionsbestimmung und Objektidentifikation werden die eigens entwickelten IrDA-Code-Baken [Ide00] verwendet. Ein weiterer Aspekt betrifft die aufgabenbezogene Bereitstellung von Informationen. Zu den typischen Aufgaben im Szenario eines Ausstellungsbesuches gehören dabei das Beziehen von Informationen zu Exponaten, das Erstellen einer persönlichen Tour und das Finden eines bestimmten Ortes. Das eGuide-System bietet hierfür folgende Funktionalitäten:

- Informationen zu Exponaten werden unter Ausnutzung der Positionsinformation automatisch bereitgestellt. Außerdem gibt es die Möglichkeit der Suche nach Ausstellern und Exponaten. Bei der Darstellung von Exponaten und Ausstellern können Themengebiete berücksichtigt werden.
- Exponate können zu einer persönlichen Tour zusammengestellt werden.
- Der kürzeste Weg zu gesuchten Objekten kann dynamisch berechnet werden. Die Wegbeschreibung wird grafisch dargestellt.

Zusätzlich zu den Funktionalitäten bietet das eGuide-System gegenüber herkömmlichen Ausstellungsführen wie Katalogen den Vorteil der erhöhten Aktualität der Daten. Die Aktualität und auch der Umfang der Daten ist allerdings durch die ausschließliche Datenhaltung auf dem mobilen Endgerät beschränkt.

1.4.2 Weitere verwandte Arbeiten

Es werden nun kurz zwei Dissertationen vorgestellt, die Architekturen für kontextsensitive Applikationen zum Gegenstand haben.

“A System Architecture for Context-Aware Mobile Computing” Bei dieser Dissertation von Schilit [Sch95] handelt es sich um eine der ersten Arbeiten, die sich mit generischen Ansätzen für kontextsensitive Applikationen auseinandersetzen. Die wichtigsten Beiträge der Arbeit sind ein Kommunikationsmodell für dynamische Umgebungen und eine Architektur für kontextsensitive Applikationen in dynamischen Umgebungen basierend auf folgenden Komponenten:

- *User Agents* vermitteln über Applikations- und Gerätegrenzen hinweg nutzerspezifische Informationen, wie z.B. Position, Präferenzen etc.

- *Device Agents* sind verantwortlich für die Überwachung von Geräten wie (PDAs, Drucker etc.) und die Bereitstellung von Informationen über Geräteeigenschaften.
- *Active Maps* beschreiben die Position und Eigenschaften von Objekten und bieten einen positionsorientierten Verzeichnisdienst, der Anfragen über Aufenthaltsorte, Routinginformationen u.ä. erlaubt.

Im Unterschied zu der hier vorliegenden Arbeit beschreibt Schilit eine Systemarchitektur, die es verteilten Applikationen in dynamischen Umgebungen erlaubt auf Kontextinformationen zuzugreifen. Von einem konkreten Kontextmodell wird dabei abstrahiert. In dieser Arbeit steht jedoch im Vordergrund, wie Kontextinformation in einer konkreten Anwendung, beispielsweise dem Zugriff auf Digitale Bibliotheken, genutzt werden kann. Zu erwähnen ist auch, dass in Schilits Realisierung der Architektur eine ständige, ununterbrochene Netzwerkverbindung angenommen wird. Dagegen wird die in dieser Arbeit beschriebene Architektur tolerant gegenüber unterbrochenen Netzwerkverbindungen sein.

"Providing Architectural Support for Building Context Aware Applications"
Auch in dieser, etwas jüngeren, Dissertation von Dey [Dey00] wird eine Softwarearchitektur vorgeschlagen, die die Entwicklung von kontextsensitiven Applikationen unterstützen soll. Zunächst wird in der Arbeit erläutert, warum die Nutzung von Kontext sinnvoll, aber auch schwierig ist:
Kontextinformation

- wird durch den Einsatz von speziellen, oft unkonventionellen, Sensoren gewonnen (z.B. GPS),
- muss abstrahiert werden, damit die Applikation sie sinnvoll nutzen kann (z.B. Übersetzung GPS-koordinaten in Straßen- oder Gebäudenamen),
- muss oft aus verteilten, heterogenen Quellen bezogen werden,
- ist hochdynamisch.

Diese Schwierigkeiten werden durch das in der Arbeit vorgestellte **Context Toolkit** adressiert (siehe auch [SDA99]). Zentrale Komponenten sind *Context Widgets*, *Context Interpreter* und *Context Server*. *Context Widgets* agieren als Mediator zwischen Applikation und Umgebung. Sie kapseln die Funktionalität der Sensoren, verstecken ihre Komplexität und dienen als wiederverwendbare und anpassbare Bausteine der Kontexterfassung. *Context Interpreter* dienen der Interpretation von Kontextinformationen, *Context Server* der Aggregation, also der Zusammenfassung mehrerer Kontextinformationen, und Aggregation und Interpretation sind dabei Formen der Abstraktion von Kontext.

Eine konkrete Anwendung der Architektur von Dey ist der **Conference Assistant**, der in [Dey99] vorgestellt wird. Es handelt sich hier um eine prototypische,

mobile, kontextsensitive Applikation, die Konferenzbesucher unterstützen soll. Es werden verschiedene Arten von Kontext genutzt: Zeit, Nutzeridentität, Ort und Aktivität. Die Kontextinformation wird genutzt, um für den Nutzer interessante Präsentationen zu bestimmen und Informationen zu diesen Präsentationen bereitzustellen. Es wird jedoch auch der Kontext anderer Personen berücksichtigt, so ist es z.B. möglich, die Position eines Kollegen zu bestimmen. Der Conference Assistant funktioniert nur unter Laborbedingungen (z.B. setzt er eine ununterbrochene Netzwerkverbindung voraus) und ist aufgrund des prototypischen Charakters noch nicht zum Einsatz gekommen.

In Deys Architektur steht die Schnittstelle zwischen Kontextsensoren und kontextsensitiver Applikation im Vordergrund. In der Systemarchitektur der hier vorliegenden Arbeit wird es ähnliche Komponenten für die Erfassung, die Interpretation und Aggregation sowie das Propagieren von Kontext geben. Jedoch liegt der Schwerpunkt eher auf der Nutzung der Kontextinformationen für die Bereitstellung situationsrelevanter Informationen. Auch in der konkreten Anwendung von Deys Architektur, dem “Conference Assistant”, sind Ähnlichkeiten zur hier behandelten Domäne des Ausstellungs- und Konferenzbesuches vorhanden.

1.5 Zusammenfassung

In diesem Kapitel wurde zunächst motiviert, warum ein situationsgesteuerter mobiler Zugriff auf Digitale Bibliotheken sinnvoll ist: Um Assistenzsysteme nahtlos in das tägliche Leben zu integrieren, ist es wünschenwert, situationsrelevante Inhalte automatisch bereitzustellen.

Es wurde im Anschluss gezeigt, welche Bedeutung Digitale Bibliotheken in der Versorgung mit Wissen haben. Außerdem wurde erläutert, wie Kontextinformationen in intelligenten Assistenzsystemen genutzt werden können.

Kombiniert man die Möglichkeiten Digitaler Bibliotheken mit denen situationsgesteuerter mobiler Assistenz, so wird eine neue Qualität im Zugriff auf digitale Inhalte erreicht. Das Bindeglied zwischen Digitalen Bibliotheken und intelligenter Assistenz werden Techniken der Wissensrepräsentation und Inferenz sein, denen sich Kapitel 3 und 4 widmen. Im nächsten Kapitel werden jedoch zunächst Anforderungen bestimmt, denen eine Systemarchitektur genügen muss, um einen situationsgesteuerten Zugriff auf Digitale Bibliotheken zu ermöglichen.

Kapitel 2

Anforderungen an die Systemarchitektur

2.1 Gesamtüberblick

Ziel ist es, eine Systemarchitektur zu entwickeln, die es ermöglicht, dem Nutzer in einem mobilen Szenario digitale Inhalte zu liefern, die er in seiner momentanen Situation benötigt. Dazu ist zunächst eine Erfassung und Analyse der Situation des Nutzers nötig. Dies erfordert die Überwachung der Umgebung und der Aktivitäten des Nutzers. Des weiteren sind Interessen und Präferenzen des Nutzers zu berücksichtigen. Auf der Basis der Situationsbeschreibung und des Wissens über die Domäne soll es dann möglich sein, relevante Inhalte der Digitalen Bibliothek zu bestimmen und bereitzustellen sowie den Nutzer bei der Erledigung seiner Aufgaben zu unterstützen, indem z.B. Aufgabenschritte vorgeschlagen werden. Die Darstellung von Informationen verlangt dabei die Berücksichtigung der Besonderheiten mobiler Endgeräte. Schließlich soll es auch noch möglich sein, Daten lokal auf dem mobilen Gerät zu verwalten, um auch einen Offline-Betrieb zu ermöglichen.

Die Anforderungen an die eben genannten Teilaspekte werden nun eingehend analysiert.

2.2 Verteilte Architektur

Typischerweise erfolgt in Digitalen Bibliotheken eine zentrale serverseitige Datenverwaltung, so dass sich die Aufgaben des Client auf die Darstellung der Inhalte beschränken. In dem Szenario des mobilen Zugriffs auf Digitale Bibliotheken ist es jedoch sinnvoll, bestimmte Datenanteile lokal auf dem mobilen Gerät zu halten. Dies gilt sowohl für Metadaten als auch für die semi- und unstrukturierten Dokumente. Hauptgründe für eine solche lokale Datenhaltung könnten sein:

- *Effizienz:* Der Zugriff auf Inhalte aus der Digitalen Bibliothek soll möglichst effizient geschehen. Durch eine lokale Datenhaltung kann der Zugriff beschleunigt werden.

- *Offline-Zugriff:* In bestimmten Situationen kann die Online-Verbindung zum Dokumentenserver unterbrochen sein. In diesen Fällen ist ein Arbeiten auf lokal gespeicherten Daten denkbar. Außerdem lassen sich durch Zugriff auf lokale Daten Online-Kosten sparen.

- *Datenschutz:* Handelt es sich bei bestimmten Daten um persönliche oder anderweitig sicherheitsrelevante Daten, sollen diese unter Umständen gar nicht serverseitig gespeichert werden. In diesem Fall ist eine lokale Speicherung unumgänglich.

Nicht nur die Datenhaltung, sondern auch bestimmte Dienste, wie z.B. Anfrage- und Retrieval-Dienste, erfordern eine verteilte Realisierung.

2.3 Situationsanalyse

Ergebnis der Situationsanalyse soll eine möglichst vollständige Beschreibung der aktuellen Situation basierend auf einem Situationsmodell in einem geeigneten Datenformat sein. Auf ihrer Basis sollen kontextsensitive Dienste realisiert werden, wie z.B. die Bereitstellung relevanter Informationen.

Zunächst ist eine Überwachung der einzelnen Kontextanteile z.B. über Sensoren nötig. Änderungen des Kontextes können dabei entweder über einen Subscribe- oder Polling-Mechanismus erfasst werden. In einem nächsten Schritt ist von den spezifischen Eigenschaften der Sensoren, z.B. speziellen Protokollen und Datenformaten zu abstrahieren. Spätestens auf dieser Ebene muss Plattform- und Geräteunabhängigkeit gewährleistet sein. Des weiteren ist eventuell eine Abstraktion der Sensordaten in Form von Interpretation und Aggregation notwendig. Interpretation bedeutet dabei z.B. eine Transformation des Datums “Position=(X,Y,Z)” nach “Der Nutzer betritt den Konferenzsaal”. Aggregation bedeutet das Zusammenfassen von Informationen von verschiedenen Sensoren, wie z.B. GPS-Empfänger und Infrarot-Bake. Außerdem kann es für gewisse Anwendungsfälle sinnvoll sein, eine Historie der Kontextdaten zu führen.

Entsprechend des SAMoA-Referenzmodells soll die Situationsbeschreibung mindestens folgende Teilaspekte umfassen:

- Umgebung des Nutzers

- Aktive Aufgaben des Nutzers mit entsprechenden Zuständen

- Persönliches Nutzerprofil

- Geräteprofil

Das Situationsmodell sollte dabei so angelegt sein, dass es um weitere Teilaspekte erweitert werden kann. Insbesondere ist es zum Beispiel denkbar, auch Kontext zu analysieren, der nicht lokal erfasst werden kann. Dazu wäre eine Kommunikation mit einer externen Komponente, z.B. einem zentralen Kontextserver notwendig.

2.3.1 Überwachung der Umgebung

Die Überwachung der Umgebung des Nutzers beinhaltet die kontinuierliche Erfassung bestimmter Umgebungsmerkmale, dazu zählen insbesondere die Position des Nutzers und Objekte in seiner Reichweite. Die Position spielt dabei eine herausragende Rolle, da aus ihr weitere Informationen über die Umgebung abgeleitet werden können. Denkbar ist aber auch die Erfassung weiterer Größen, wie Geräuschpegel, etc.

Zur Überwachung der Umgebung werden spezielle Sensoren eingesetzt, wie z.B. Infrarotsensoren in Kombination mit Infrarot-Baken.

2.3.2 Aufgabenmanagement

Ziel des Aufgabenmanagement ist die Verwaltung der Aktivitäten des Nutzers. Ein Nutzer kann mehrere, voneinander unabhängige Ziele bzw. Aufgaben verfolgen, die er parallel bearbeitet. Aufgaben können eine komplexe interne Struktur besitzen (z.B. Komposition aus Teilaufgaben) und besitzen einen internen Zustand, der für die Überwachung ihrer Ausführung von Bedeutung ist. Die Beschreibung der Aufgaben als Bestandteil der Situationsbeschreibung ist Basis für das Bereitstellen von aufgabenbezogenen Informationen.

Eine wichtige Rolle dabei spielt die Sprache zur Aufgabenmodellierung. In [KCI99] werden Anforderungen an eine solche Sprache genannt:

- Sie muss einfach und intuitiv genug sein, um dem Endnutzer Aufgabenmodellierung zu ermöglichen.
- Sie muss inkrementelle und “ad hoc”-Modifikation bzw. Erstellung von Aufgaben und Teilaufgaben durch den Nutzer erlauben.

Es werden außerdem Produktionssysteme (eine spezielle Form von regelbasierten Systemen) für die Aufgabenmodellierung vorgeschlagen. Diese bieten interessante Konzepte zur Konfliktlösung (z.B. specificy, recency und rule-ordering), welche eine “ad hoc”-Modifikation von Regelmengen sogar zur Ausführungszeit erlauben. Jedoch sind auch andere Möglichkeiten zur Modellierung von Aufgaben denkbar.

2.3.3 Profildienste

Das Ziel der Verwendung von Nutzerprofilen besteht darin, dem Nutzer Informationen entsprechend seinen Interessen und Präferenzen zu liefern, ihm also seine eigene Sicht auf die Digitale Bibliothek zu geben.

Die Nutzerprofile können sowohl bei der Bestimmung relevanter Informationen, aber auch bei ihrer Darstellung verwendet werden. Bestandteile eines Nutzerprofiles könnten sein:

- Sprache
- Präferenz für bestimmte Medientypen
- Interesse an bestimmten Themen
- Vorbildung (Neuling bis Experte)

Ebenso wie Nutzerprofile können auch Geräteprofile verwendet werden, die die Charakteristika des mobilen Endgerätes beschreiben. Diese können z.B. sein:

- Display-Eigenschaften (Größe, Auflösung, Farbtiefe)
- Multimedia-Fähigkeiten (Audio, Video, Images oder nur Text)

Im Gegensatz zu Nutzerprofilen sind Geräteprofile nicht modifizierbar und sollten möglichst automatisch ausgelesen werden.

2.4 Anfragedienst

Aufgabe des Anfragedienstes ist es, relevante Informationen zu einer Anfrage zu bestimmen.Grundsätzlich sollten, wie bei Digitalen Bibliotheken üblich, folgende Arten von Anfragen unterstützt werden:

- *Suche auf Metadaten:* Hiermit ist die Suche auf beschreibenden Daten gemeint, also etwa bibliographischen Angaben, aber auch z.B. Daten, die eine Ausstellung beschreiben.
- *Information Retrieval:* Dies bezeichnet die Suche auf unstrukturierten Daten, zumeist Volltexten, aber auch Multimedia-Dokumenten wie Audio, Video oder Images.

Zusätzlich zu diesen klassischen Anfragtypen Digitaler Bibliotheken ist ein weiterer Anfragtyp erforderlich, um situationsgesteuert relevante Inhalte bestimmen zu können:

- *Inferenzanfragen:* Hier handelt es sich um spezielle Anfragen, bei denen die Ergebnismenge lediglich in Abhängigkeit von der Situationsbeschreibung des Nutzers durch automatisches Schließen (Inferenz) bestimmt wird. Der Nutzer muss hier also keine explizite Anfrage formulieren.

Als Ergebnis einer Anfrage soll eine Menge von Referenzen (z.B. Objectidentifier) auf die relevanten Dokumente zurückgeliefert werden. Diese Referenzen können dann für das Retrieval der Dokumente verwendet werden. Die Trennung

von Anfrage- und Retrieval-Dienst ermöglicht eine bessere Effizienz bei der Bereitstellung von Informationen.

Die Anfragen sollten über geeignete Interoperabilitätsstandards des Internet an den Anfragedienst gestellt werden. Ebenso sollten die Anfrageergebnisse entsprechend eines Standards zurückgeliefert werden. Dies ist notwendig, um Interoperabilität zu ermöglichen und um die Digitale Bibliothek öffentlich nutzbar zu machen. Daher bietet sich insbesondere die Verwendung von Internet-Standards an.

2.5 Retrieval-Dienst

Die Referenzen als Ergebnis einer Anfrage können zum Retrieval der eigentlichen Daten genutzt werden. Dabei wird zwischen den folgenden zwei Formen unterschieden:

- *Retrieval von Metadaten* liefert beschreibende Daten zu Dokumenten, aber auch beliebigen anderen Ressourcen oder Entitäten, die nicht elektronisch übertragbar sind.
- *Retrieval von Dokumenten* liefert die eigentlichen Inhalte der Bibliothek, wie Volltexte, Videos, Images, etc.

Für die Umsetzung des Retrieval sind die Besonderheiten der verteilten Datenhaltung zu beachten. Grundsätzlich sind zwei verschiedene Verfahren zur lokalen Datenspeicherung zu unterscheiden, Replikation und Caching:

- *Replikation* erfolgt auf expliziten Wunsch des Nutzers. Die replizierten Daten werden nicht automatisch verdrängt, sondern nur durch Nutzer initiiert gelöscht.
- *Caching* geschieht automatisch und ist für den Nutzer transparent. Da der Cache zumeist in seiner Größe beschränkt ist, müssen die Daten mittels einer geeigneten Strategie verdrängt werden.

Weiterhin ist zu berücksichtigen, ob zwischen den Datenbeständen, die lokal gespeichert sind, und denen des Dokumentenservers eine Teilmengenbeziehung besteht, oder ob es sich um partiell disjunkte Mengen handelt. Im Normalfall wird nur ein Teil des Datenbestandes der Digitalen Bibliothek lokal auf dem mobilen Gerät gespeichert werden. Das bedeutet, die lokal gespeicherten Daten bilden eine Teilmenge des Datenbestandes des Dokumentenservers. Einfach dagegen sind die beiden Fälle, in denen sämtliche oder aber gar keine Daten lokal auf dem mobilen Gerät gespeichert werden.

Auch muss beachtet werden, wo auf den Daten auch Änderungen (schreibender Zugriff) erlaubt sind. Für ein konkretes Szenario sind entsprechende Replikations- bzw. Caching-Strategien auszuwählen. Sind nur Änderungen auf dem zentralen Datenbestand des Dokumentenservers möglich, genügt eine “Single-Update-Site”

Strategie. Sind auch Änderungen durch den mobilen Nutzer erlaubt, so werden Synchronisationsstrategien für "Update Anywhere" mit Konfliktlösung notwendig. Einen Überblick über gängige Replikationsstrategien, insbesondere im heterogenen Umfeld, gibt [Haa00].

Der Retrieval-Dienst sollte das Level-Of-Detail-Konzept unterstützen. Das bedeutet, dass Informationen zu einem Objekt in verschiedenen Granularitätsstufen bereitgestellt werden können.

Ebenso wie der Anfragedienst soll der Retrieval-Dienst entsprechende Interoperabilitätsstandards verwenden.

2.6 Visualisierung und Interaktion

Bei der Visualisierung und Interaktion sind die Besonderheiten mobiler Endgeräte zu berücksichtigen. Das Ziel dabei ist es, den kognitiven Overhead zu minimieren, das heißt, die Nutzung des mobilen Assistenten muss sich möglichst nahtlos in den jeweiligen Arbeitsprozess des Nutzers integrieren.

Die Ressourcen zur Visualisierung und Interaktion variieren dabei über die Bandbreite mobiler Endgeräte beträchtlich. Auch hier ist deshalb, ebenso wie bei den Anfragediensten, die Nutzung von Geräteprofilen sinnvoll.

Die Darstellung von bestimmten Objekttypen und Strukturen wie z.B. Listen, Tabellen, hierarchischen Strukturen oder Karten sollte dabei generisch sein. Wie bei Digitalen Bibliotheken üblich, sollten Konzepte wie Navigation und Browsing unterstützt werden.

Dynamische Dialogerzeugung auf der Basis einer Dialogbeschreibungssprache soll nicht Gegenstand der Arbeit sein. Sehr wohl sollten jedoch Kontextinformationen bei der Darstellung berücksichtigt werden. Auch Techniken wie Level of Detail können berücksichtigt werden.

2.7 Zusammenfassung

In diesem Kapitel wurden die Anforderungen an eine Systemarchitektur für den situationsgesteuerten mobilen Zugriff auf Digitale Bibliotheken analysiert.

Zentral sind dabei zum einen die Anforderungen an die Situationsanalyse. Um aus einer Situationsbeschreibung relevante Inhalte ableiten zu können, ist ein komplexes Situationsmodell erforderlich. Bestandteile dieses Situationsmodells sind die Umgebung des Nutzers, seine Aufgaben sowie Geräte- und Nutzerprofile.

Zum anderen betreffen die Anforderungen die Anfrage- und Retrieval-Dienste. Zusätzlich zu den bekannten Anfragetypen Digitaler Bibliotheken ist ein neuer Anfragetyp erforderlich, der als Parameter lediglich die Situationsbeschreibung verwendet. Außerdem sind Anfrage- und Retrieval-Dienste verteilt zu realisieren, um einen effizienten Zugriff auch bei Netzwerkausfällen zu erlauben.

Die Systemarchitektur wird in Kapitel 5 beschrieben. Das nächste Kapitel beschäftigt sich zunächst mit der Datenmodellierung.

Kapitel 3

Datenmodellierung

In diesem Kapitel werden zunächst Besonderheiten und Anforderungen an die Datenmodellierung analysiert. Aus diesen Anforderungen wird ersichtlich werden, dass eine wissensbasierte, ontologiegestützte Datenmodellierung sinnvoll ist. Nach einer kurzen Einführung in Konzepte wissensbasierter Systeme wird gezeigt, dass das RDF (Resource Description Framework) mit seinen Erweiterungen geeignete Mittel zur Datenmodellierung bietet. Schließlich wird eine Ontologie in DAML, einer speziellen RDF-Erweiterung, für die Domäne Ausstellungs- und Konferenzbesuch erarbeitet.

3.1 Anforderungen an die Datenmodellierung

Bei der Datenmodellierung sind verschiedene Klassen von Daten zu berücksichtigen:

- Unstrukturierte und semistrukturierte Multimedia-Daten, wie Volltexte, Bilder, Videos, Audiodaten etc.
- Daten, die diese multimedialen Inhalte der Bibliothek beschreiben
- Daten, die das konkrete Anwendungszenario beschreiben (z.B. ein Ausstellungsplan)
- Daten, die die momentane Situation des Bibliotheksbenutzers beschreiben

Während die erste Klasse von Daten, die eigentlichen Inhalte der Bibliothek, für die Datenmodellierung an dieser Stelle relativ uninteressant ist, liegt die Herausforderung der Datenmodellierung bei den drei letzten Klassen. Bei diesen Daten handelt es sich um Metadaten [1]. Die drei Klassen stehen dabei eng miteinander in Beziehung, auch ist die Einordnung in eine bestimmte Klasse nicht immer eindeutig: So kann ein Vortragender auf einer Konferenz auch gleichzeitig Autor eines

[1] Bei diesen Metadaten handelt es sich nicht nur um Daten beschreibende Daten, sondern auch um Beschreibungen von anderen — einschließlich physischen — Objekten

Artikels in der Digitalen Bibliothek sein. Die Beziehungen sollen später genutzt werden, um automatisch schließen zu können, welche Inhalte in einer bestimmten Situation relevant sind.

Die Metadaten sind Gegenstand der folgenden Betrachtungen. Dabei sind folgende Aspekte bei ihrer Modellierung besonders wichtig:

- *Interoperabilität:* Sowohl auf syntaktischer als auch auf semantischer Ebene ist Interoperabilität wichtig. Um einen einfachen Datenaustausch, z.B. zwischen Herstellern, Verlagen und Providern, aber auch zwischen Digitaler Bibliothek und mobilem Endgerät zu ermöglichen, muss es möglich sein, sowohl Schemata als auch Instanzen des Datenmodells in einem standardisierten Format zu serialisieren. Die Verwendung von Standards ist hier insbesondere aufgrund der Heterogenität der beteiligten Kommunikationspartner wichtig.

- *Ontologiebasierte Wissensrepräsentation:* Um ein intelligentes Schließen (Inferenz), wie man es von Expertensystemen kennt, zu ermöglichen, bietet es sich an, auf semantische Datenmodelle und Beschreibungssprachen der Wissensrepräsentation zurückzugreifen. Es ist zudem sinnvoll, das gesamte Wissen über die Domäne, die Inhalte der Bibliothek und Situationen auf eine einheitliche, möglichst standardisierte, Ontologie abzubilden. Ontologien spielen eine entscheidende Rolle in der Darstellung, der Verarbeitung und im Austausch von Wissen. Sie ermöglichen ein gemeinsames Verständnis einer Domäne, so dass eine neue Qualität in der Kommunikation zwischen Menschen und Maschinen erreicht werden kann.

- *Erweiterbarkeit:* Im Hinblick auf die Wiederverwendbarkeit bestehender Schemata muss das Datenmodell Konzepte für Erweiterbarkeit bieten. Dies ist zum Beispiel wichtig für die Übertragung auf geänderte oder völlig neue Domänen.

Bevor aufgezeigt wird, wie RDF mit seinen Erweiterungen den eben genannten Anforderungen gerecht wird, sollen einige Erläuterungen zu wissenbasierten Systemen und Ontologien folgen.

Wissensbasierte Systeme Wissensbasierte Systeme, oft Expertensysteme genannt, sind Systeme, die explizit dargestelltes Wissen verarbeiten. Wissen ist dabei eine spezielle Form von Information, aus der neue Informationen abgeleitet werden kann. So kann z.B. ein intelligentes Assistenzsystem aus dem Wissen über die Domäne und die Situation des Nutzers ableiten, welche Dokumente für den Nutzer relevant sind. Es existieren verschiedene Verfahren zur Repräsentation von Wissen. Als wichtigste Vertreter seien Semantische Netze, Description Logic und Frame-basierte Modelle genannt. Insbesondere Frame-basierte Modelle werden im folgenden noch mehrmals eine Rolle spielen, so z.B. in 3.3 (Datenmodell des RDF) und

4.1.1 (Frame-Logic). Einen Überblick über verschiedene Verfahren der Wissensrepräsentation gibt [Fen00b] unter besonderer Berücksichtigung der Verwendung von Web-Standards.

Ontologien Ontologien spielen eine entscheidende Rolle in der Darstellung und im Austausch von Wissen. Sie ermöglichen ein gemeinsames Verständnis einer Domäne, das auch zwischen Menschen und Maschinen kommuniziert werden kann. Sie sind Gegenstand der Forschung in den Bereichen Knowledge Management, Sprachverarbeitung, Kooperative Informationssysteme sowie Intelligente Informationsintegration.

Eine Ontologie ist eine formale Beschreibung von wichtigen Konzepten einer bestimmten Domäne. Typischerweise definiert eine Ontologie Klassen von Objekten und organisiert diese Klassen in einer Hierarchie. Jede Klasse besitzt Eigenschaften, die kennzeichnend für diese Klasse sind. Des weiteren werden Relationen zwischen Klassen bzw. Objekten dieser Klassen definiert. Ontologien spielen in dieser Arbeit vor allem aus folgenden Gründen eine Rolle:

- Sie sind die Basis für ein wissensbasiertes Inferenzsystem (siehe 4.1).
- Sie ermöglichen eine standardisierte Darstellung von Wissen und damit auch den Datenaustausch zwischen Ausstellern, Konferenzorganisatoren, Firmen, Applikationsentwicklern, etc. Durch die Integration von Semantik bieten sie mehr als ein einfaches Datenaustauschformat.

3.2 Die Modellierungssprachen des Semantic Web

Das erklärte Ziel der *Semantic Web Community* [sem01] ist es, das Web um formale Semantik anzureichern und somit den Weg zum so genannten “Semantic Web” zu ebnen [Fen00c]. Die Modellierungssprachen des Semantic Web bauen entsprechend des Schichtenmodells, das in Abbildung 3.1 dargestellt ist, aufeinander auf.

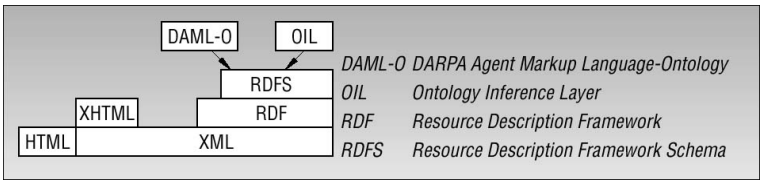

Abbildung 3.1: Schichtenmodell der Sprachen des Semantic Web

Es wird nun aufgezeigt, wie die Sprachen des Semantic Web für die Datenmodellierung geeignet sind. Es werden dazu insbesondere die Anforderungen des vorherigen Abschnitts (Interoperabilität, ontologiebasierte Wissensrepräsentation und Erweiterbarkeit) berücksichtigt.

Interoperabilität XML hat sich zum vielversprechenden Standard für den Datenaustausch entwickelt und leistet damit einen entscheidenden Beitrag in Bezug auf Interoperabilität. XML definiert eine Standardsyntax für den Datenaustausch, bietet mit DOM ein generisches Dokumentmodell und erlaubt, die Struktur von Dokumenten mit DTDs oder XML Schemata zu beschreiben. XML bietet allerdings keine Formalismen, die Daten in diesen Dokumenten zu interpretieren. Die Interoperabilität auf syntaktischer Ebene allein reicht nicht aus. RDF und die Sprachen des Semantic Web dagegen ermöglichen eine semantische Interoperabilität. Sie nutzen XML lediglich als Syntax und stehen damit nicht in Konkurrenz zu XML.

In Bezug auf die Interoperabilität lassen sich nach [DM00] die folgenden drei Ebenen unterscheiden:

- *Syntax Layer:* Die Interoperabilität auf syntaktischer Ebene erfordert die Serialisierung der Information und ein generisches Dokumentmodell. Wie oben beschrieben, nutzen die Sprachen des Semantic Web XML als Syntax.

- *Object Layer:* Diese Ebene erlaubt Interoperabilität auf Objektebene. Sie bietet eine objektorientierte (oder auch Frame-basierte) Sicht auf Informationen. Die wichtigsten Merkmale sind Objektidentität, einfache Typisierung und binäre Relationen zwischen Objekten. Das Datenmodell des RDF stellt diese bereit.

- *Semantic Layer:* Auf der semantischen Ebene erfolgt eine Interpretation der Objekte aus der Objektebene. Hierfür kommen konzeptuelle Modelle wie Schemata und Domänenmodelle wie Ontologien sowie formale Sprachen zum Einsatz. Die nötigen Modellierungsperformativen stellen RDF Schema und darauf aufbauende Sprachen wie DAML bereit.

Ontologiebasierte Wissensrepräsentation Auch bezüglich der Ausdrucksstärke der Modellierungsprimitiven lassen sich verschiedene Schichten beschreiben [HP01]. Dabei bauen die höheren Schichten jeweils auf den Modellen der darunterliegenden Schichten auf. Die Ausdrucksstärke der Modellierungssprachen wird so inkrementell erweitert.

- *Metadata Layer:* Auf dieser untersten Ebene sind einfache semantische Zuweisungen möglich, um Metainformationen zu beschreiben. RDF stellt hierfür ein generisches Datenmodell bereit. Die höheren Schichten bauen auf diesem einfachen Datenmodell auf. Bereits auf dieser Ebene ist eine Frame-basierte Wissensrepräsentation möglich.

- *Schema Layer:* Auf dieser Ebene werden einfache Ontologie-Sprachen für eine hierarchische Beschreibung von Konzepten und Properties eingeführt. Die Sprache des Semantic Web für diese Ebene ist RDF Schema. Sie stellt einige wichtige Primitiven zur Modellierung von Ontologien bereit.

- *Logical layer:* Hier werden weitere, ausdrucksstärkere Sprachen eingeführt. Sie basieren auf den einfachen Modellierungsperformativen des *Schema Layer*. Modellierungssprachen auf dieser Ebene sind z.B. DAML und OIL. Sie stellen vollwertige Sprachen zur Modellierung von Ontologien dar.

Erweiterbarkeit Das Frame-orientierte Typsystem von RDF erlaubt eine Erweiterbarkeit wie sie aus der objektorientierten Welt bekannt ist: Basierend auf Teilen bestehender RDF-Schemata ist es möglich, neue Schemata zu entwickeln. Ora Lassila spricht gar von einer Darwinschen Evolution der Metadaten, bei der gute Lösungen überleben und erweitert werden [Fen00c].

3.3 RDF

Das Ziel von RDF ist die Definition eines Mechanismus zum Beschreiben von Ressourcen. Da a priori keinerlei Annahmen über eine spezielle Anwendungsdomäne gemacht werden, erlaubt es RDF, Ressourcen beliebiger Natur zu beschreiben. Genau gesagt: Jedes Objekt, dass sich über eine URI [URI] eindeutig identifizieren lässt, ist mit RDF beschreibbar. RDF ist das Ergebnis der Zusammenarbeit verschiedener Communities und hat damit Einflüsse aus verschiedenen Quellen erfahren. Insbesondere sind hierbei das W3C selbst, die Digital Library Community, die Structured Document Community und die Knowledge Representation Community zu nennen.

3.3.1 Datenmodell und Syntax des RDF

Das Datenmodell des RDF beschreibt Relationen zwischen Entitäten. Die drei zentralen Konzepte des Datenmodells sind:

- *Resource:* Alle Objekte, die durch RDF-Ausdrücke beschrieben werden, heißen *Resource*. Dies können z.B. Dokumente im WWW, aber auch z.B. physische Objekte sein. Resources werden durch URIs [URI] benannt und eindeutig identifiziert.

- *Property:* Eine *Property* ist ein Attribut, ein Aspekt, eine Eigenschaft oder eine Relation zur Beschreibung einer Resource.

- *Statement:* Eine bestimmte Resource zusammen mit einer benannten Property und dem Wert diese Property für diese Resource heißt *Statement*. Diese drei Bestandteile eines Statement heißen *Subject*, *Predicate* und *Object*. Das Object eines Statement kann eine andere Resource oder ein Literal sein. RDF-Properties können als Attribute von Resources verstanden werden und korrespondieren damit mit traditionellen Attribut-Wert Paaren.

In objektorientierter Terminologie entsprechen Resources den Objekten und Properties den Instanzvariablen. Dadurch, dass Statements selbst wieder Resources

darstellen, ist es möglich, auch Statements über Statements zu formulieren (Reifikation).

Beispiel: Folgender Satz soll in RDF repräsentiert werden:
Peter Haase ist Autor der Resource http://www.informatik.uni-rostock.de/˜peter/.
Die Bestandteile des RDF-Statements sind also:

Subject (Resource)	http://www.informatik.uni-rostock.de/˜peter/
Predicate (Property)	author
Object (Literal)	“Peter Haase”

Die Tripel des RDF-Datenmodells lassen sich auch als gerichteten Graph darstellen, wie Abbildung 3.2 zeigt. Die Resources und Literale sind dabei die Knoten, während die Properties die Kanten sind.

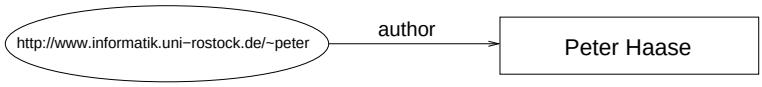

Abbildung 3.2: RDF-Beispiel in Graph-Repräsentation

Das RDF-Datenmodell ist eine syntaxunabhängige Möglichkeit zur Repräsentation von RDF-Ausdrücken. Diese Form der Repräsentation wird zur Bestimmung der Äquivalenz bezüglich ihrer Bedeutung genutzt: Zwei RDF-Ausdrücke sind äquivalent, genau dann wenn ihre Repräsentation im Datenmodell gleich sind. Diese Definition erlaubt syntaktische Variationen in Ausdrücken, ohne die Bedeutung zu ändern.

RDF nutzt XML als Syntax zur Serialisierung der Statements des abstrakten RDF-Datenmodells. Damit steht RDF nicht in Konkurrenz zu XML, sondern ist als Ergänzung zu verstehen.

Beispiel: Das RDF-Statement des obigen Beispiels in “Basic-RDF” Syntax lautet:[2]

```
<rdf:RDF>
     <rdf:Description about="http://www.informatik.uni-rostock.de/~peter/">
          <author>Peter Haase</author>
     </rdf:Description>
</rdf:RDF>
```

Wie bereits erwähnt, kann es zu äquivalenten Statements verschiedene syntaktische Repräsentationen geben. RDF sieht zwei verschiedene Serialisierungen in XML vor: Basic Syntax und Basic Abbreviated Syntax, wobei die zweite eine kompaktere Serialisierung liefert.

[2]Die Namensraumdeklarationen sind zu Gunsten der Lesbarkeit weggelassen worden.

Die ausführliche Spezifikation des RDF-Datenmodells und der RDF-Syntax ist in [LS] nachzulesen.

3.3.2 Schemata in RDF

RDF Schema [Bri], kurz RDFS, führt weitere Modellierungsprimitiven ein, mit denen es möglich ist, Schemata zu beschreiben. Über Schemata können Resources nun als Spezialisierung von anderen Resources definiert werden. Auch ist es möglich, Restriktionen für Properties einzuführen. Die wichtigsten Klassen und Properties zur Definition von Schemata sind:

- *Klassen*
  - **rdfs:Resource** Jede Entität, die durch RDF beschrieben wird, ist Instanz der Klasse Resource.
  - **rdf:Property** Die Klasse rdf:Property repräsentiert die Teilmenge von Resources, die Properties sind.
  - **rdfs:Class** rdfs:Class entspricht dem generischen Konzept des Typs, ähnlich dem Konzept der Klasse in objektorientierten Programmiersprachen. Wird in einem Schema eine neue Klasse definiert, so ist diese vom Typ rdfs:Class.
- *Properties*
  - **rdf:type** rdf:type wird verwendet, um auszudrücken, dass eine Resource Instanz einer bestimmten Klasse ist. Dabei kann eine Resource Instanz mehrerer Klassen sein.
  - **rdfs:subClassOf** Diese Property spezifiziert die Subsumptionsbeziehung zwischen Klassen. Hierüber können also Klassenhierarchien aufgebaut werden. Die rdfs:subClassOf Relation ist transitiv.
  - **rdfs:subPropertyOf** Diese Property wird benutzt, um die Subsumtionsbeziehung zwischen Properties modellieren. Es können also auch Hierarchien von Properties dargestellt werden. Natürlich ist auch die rdfs:subPropertyOf-Relation transitiv.

3.3.3 Ontologien in RDF

In Bezug auf Ontologien steuert RDFS vor allem folgende Dinge bei: Eine Standardsyntax zum Beschreiben von Ontologien sowie eine Menge von Modellierungsperformativen. Das Definieren einer Ontologie in RDF bedeutet dabei die Definition eines RDF-Schemas, welche alle Konzepte und Beziehungen zwischen diesen Konzepten der Domäne beschreibt.

Mit den Primitiven von RDF(S) ist es möglich, bestehende Ontologien in RDF zu erweitern und zu verfeinern. Man kann jedoch auch einen Schritt weiter gehen, und die Schemasprache RDFS selbst erweitern. Wie dies geschieht, wird im folgenden Abschnitt erläutert.

3.4 Erweiterungen von RDF

RDF Schema stellt bereits einige grundlegende Mittel zur Modellierung von Ontologien bereit. Dabei RDF wird nachgesagt, es enthält den kleinsten gemeinsamen Nenner aller verschiedenen Metadaten-Sprachen der Wissensrepräsentation. Dies ist sicherlich nicht viel, ist aber verständlich vor dem Hintergrund, dass es wohl keine Ontologiesprache allein geben wird, die sämtliche Anforderungen aller Anwendungen erfüllt. Jedoch ist RDF Schema mächtig genug, um reichere Sprachen aufbauend auf den relativ begrenzten Modellierungsprimitiven von RDF(S) zu definieren. Da jede Ontologie ihren eigenen Namensraum benutzt, ist es möglich, Terme aus unterschiedlichen Ontologien in einem RDF-Dokument zu benutzen. Aufgrund der klaren Objektstruktur von RDF ist es sogar möglich, Zuweisungen mit einer Sprache an ein Objekt zu machen, das in einer anderen Sprache definiert ist [DFvH$^+$00]. Die Vorgehensweise zur Erweiterung der Schemasprache von RDFS soll nun kurz (entsprechend [DFvH$^+$00]) am Beispiel der Modellierung von Ontologien mit OIL (siehe 3.4.1) skizziert werden:

1. Mittels RDF Schema werden die Modellierungsprimitiven von OIL beschrieben.

2. Mit dem so erzeugten RDF Schema wird eine spezifische Ontologie in OIL beschrieben.

3. Auf der Basis der RDF Schema Dokumente aus 1. und 2. werden die Instanzen der spezifischen Ontologie aus 2. beschrieben.

Dieselbe Vorgehensweise ist auch auf andere Erweiterungen anwendbar. Eine Anwendung, der der Namensraum der RDF-Erweiterung nicht bekannt ist, kann nun immer noch die Anteile interpretieren, die im RDF(S) Namensraum definiert sind.

Zwei konkrete RDF-Erweiterungen, OIL und DAML, werden im folgenden vorgestellt.

3.4.1 OIL

OIL ist im Rahmen des Ontoknowledge-Projektes entwickelt worden. Das Akronym OIL steht dabei für *Ontology Inference Layer* oder auch *Ontology Interchange Language* [FHvH$^+$a], [FHvH$^+$b], [BBD$^+$00]. OIL ist eine Modellierungssprache auf der Ebene des *Logical Layer* des Semantic Web.

OIL vereinigt drei wichtige Aspekte aus verschiedenen Communities: Frame-basierte Systeme, Description Logics und Webstandards [FHvH$^+$a]. Diese sollen nun kurz eräutert werden:

Frame-basierte Systeme Frame-basierte Sprachen haben eine lange Geschichte in der Künstlichen Intelligenz. Die zentralen Elemente der Modellierung sind *Frames* (Klassen) und *Slots* (Attribute). Die Frames bilden eine Klassenhierarchie,

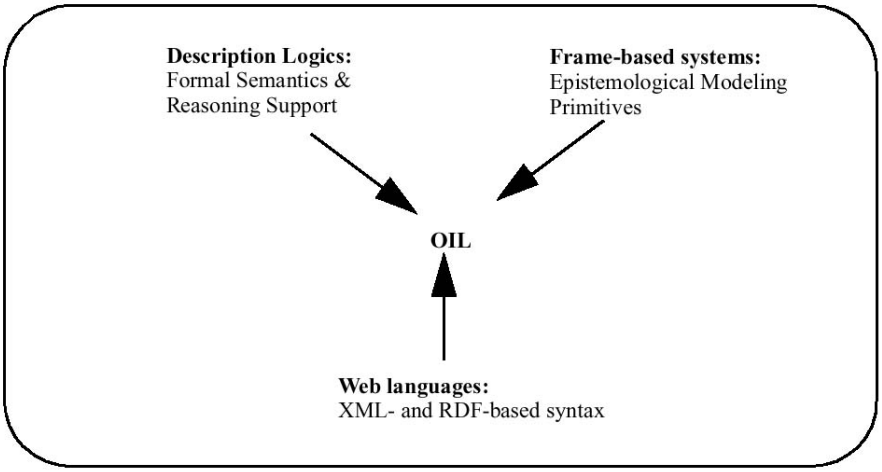

Abbildung 3.3: Die drei Wurzeln von OIL

Slots können mit zusätzlichen Einschränkungen (Restriktionen) versehen werden. Die meisten Frame-basierten Systeme bieten darüber hinaus zusätzliche Modellierungsmöglichkeiten. Viele Aspekte Frame-basierter Systeme finden sich in der objektorientierten Welt wieder. Auch OIL verwendet die grundlegenden Primitiven Frame-basierter Systeme in seiner Sprache. OIL basiert ebenfalls auf dem Konzept der Klassen und der Definition von Superklassen und Attributen. Relationen können dabei auch als eigenständige Entitäten definiert werden, so dass auch sie — wie Klassen — eigene Attribute (z.B. Domain und Range) haben und in einer Hierarchie angeordnet werden können.

Description Logics Description Logics (DL) ist eine weitere Methode der Wissensrepräsentation. Sie beschreibt Wissen mittels Konzepten und Rollen (ähnlich den Frames und Slots in Frame-basierten Systemen). Die Semantik von Ausdrücken der DL kann mathematisch präzise beschrieben werden, wodurch zum Beispiel automatisches Schließen möglich wird. OIL übernimmt diese formale Semantik aus der DL und damit auch die Unterstützung für automatisches Schließen.

Web-Standards Die Syntax von OIL ist definiert mittels Standards des W3C: Sie basiert zum einen auf einer XML DTD bzw. eines XML Schemas. Zum anderen ist OIL eine Erweiterung von RDF. Einige der Modellierungprimitiven von OIL können direkt in RDF(S) ausgedrückt werden, die anderen werden in einem eigenen Namensraum neu definiert.

Eine umfangreiche, informale Beschreibung von OIL bietet [BBD+00].

3.4.2 DAML

Eine weitere Modellierungssprache des Semantic Web ist DAML (DARPA Agent Markup Language) [dam01b]. DAML ist zwar ein von der DARPA gesponsortes Projekt, jedoch wirken bei der Standardisierung namhafte Mitglieder aus Industrie, Forschung und Organisationen wie das W3C mit. Damit ist DAML auf dem besten Weg, zum Standard der Semantic Web Community zu werden. DAML hat viele Aspekte von OIL geerbt, wird deshalb auch als DAML+OIL bezeichnet. Die Modellierungsfähigkeiten beider Sprachen sind sehr ähnlich. Für eine ausführliche Beschreibung von DAML sei auf [vHPSH01] verwiesen. An dieser Stelle seien nur ein paar Konzepte genannt, um die RDF(S) durch DAML erweitert wird:

- *Property-Restriktionen:* Es ist möglich, Kardinalitäten (`daml:cardinality`) zu beschreiben.
- *Boolesche Kombinationen bei Klassendefinitionen:* Es ist möglich, Klassen als Vereinigung (`daml:unionOf`), Schnitt (`daml:intersectionOf`) oder Komplement (`daml:complementOf`) von bestehenden Klassen zu definieren.
- *Datentypen* DAML erweitert den einzigen(!) RDF-Datentyp Literal um die XML-Schema Datentypen.
- *Inferenz:* Über spezielle Konstrukte können zusätzliche Informationen für Inferenzsysteme modelliert werden, wie z.B. inverse und transitive Relationen mit `daml:inverseOf` und `daml:TransitiveProperty`. Der nächste Schritt wird DAML-Logic sein, eine Sprache, mit der es auch möglich ist, Regeln und Axiome auszudrücken.
- *Semantik:* DAML erbt von OIL die wohldefinierte Semantik der Description Logics. In [McG] wird außerdem eine axiomatische Semantik in KIF (erweiterte Prädikatenlogik erster Stufe) angegeben. Eine modelltheoretische Semantik beschreibt [vHPSH].

3.5 Eine Ontologie für die Ausstellungs- und Konferenzdomäne

Es soll nun eine Ontologie für die Domänen Ausstellung und Konferenz erarbeitet werden. Die Gründe, diese Ontologie in DAML zu formulieren, wurden bereits erläutert, seien aber noch einmal wiederholt:

- DAML verfügt über eine hohe semantische Ausdrucksstärke für ontogiebasierten Wissensrepräsentation.
- DAML bietet ein hohes Maß an syntaktischer und semantischer Interoperabilität.

- In DAML erstellte Ontologien lassen sich einfach erweitern und kombinieren.

Da RDF und DAML durch die Standardisierungsbemühungen des W3C und der Semantic Web Community weite Verbreitung erfahren haben, gilt zusätzlich:

- Es sind bereits viele Ontologien in DAML verfügbar.
- Es existieren viele freie Tools für die Arbeit mit DAML

Die entwickelte Ontologie ist im Anhang A zu finden. Aus Gründen der besseren Lesbarkeit ist sie dort jedoch nicht in DAML, sondern in Frame-Logic-Syntax (siehe Abschnitt 4.1.1) angegeben.

Ausgangspunkt für die Entwicklung einer Ontologie für die erläuterten Domänen "Messe- und Konferenzbesuch" bildeten das bereits im Fraunhofer IGD entwickelte eGuide Exchange Format, kurz exf [KI00], sowie die "Semantic Web Research Ontology", kurz SWRO [swr]. Diese werden im folgenden kurz vorgestellt.

3.5.1 Das eGuide Exchange Format

Das eGuide Exchange Format, kurz exf, ist ein Datenaustauschformat für die Beschreibung von Ausstellungen. Diese Beschreibungen umfassen räumliche und zeitliche Strukturen wie Ausstellungspläne, Organisationsstrukturen, etc. Beim exf handelt es sich um eine XML-DTD, d.h. um eine Grammatik für XML-Dokumente. Die Datenmodellierung mit dem exf beschränkt sich damit auf die syntaktische Ebene.

3.5.2 Die Semantic Web Research Ontology

Die Semantic Web Research Ontology ist eine von der Semantic Web Community vorgeschlagene Ontologie zur Modellierung von Forschungsaktivitäten. Sie ist außer in DAML auch in anderen Sprachen wie OIL verfügbar. Hauptanwendung findet sie in der Annotation von Dokumenten, um semantischen Zugriff auf diese zu ermöglichen.

Die wichtigsten Top-Level-Konzepte der Ontologie sind **Publication**, **Person**, **Organization**, **Project**, **Product**, **Event** und **Topic**. Zu diesen Konzepten sind zahlreiche Unterkonzepte und Relationen definiert, die die semantischen Beziehungen zwischen ihnen herstellen.

Die SWRO allein genügt jedoch nicht, um die komplette Domäne von Konferenzen und Ausstellungen zu modellieren. So sind z.B. multimediale Inhalte und räumliche Konstrukte nur unzureichend berücksichtigt.

3.5.3 Modellierung der Inhalte der Bibliothek

Bei den Inhalten der Bibliothek kann es sich um beliebige semi- und unstrukturierte multimediale Inhalte handeln. Für die Modellierung solcher Inhalte, insbesondere

von wissenschaftlichen Publikationen, steht mit der SWRO bereits eine geeignete Ontologie zur Verfügung. Die Klassen **Publication**, **Person** und **Organization** sind hier für die Modellierung der Dokumente und ihrer bibliografischen Angaben von Bedeutung. Für multimediale Inhalte wird ein eigener Ansatz vorgestellt.

Wissenschaftliche Publikationen Die wichtigste Klasse von Inhalten Digitaler Bibliotheken, insbesondere im Kontext von Konferenzen, sind wissenschaftliche Publikationen. Diese sind in der SWRO mit dem Konzept **Publication** und zahlreichen Unterkonzepten umfassend berücksichtigt. Die Modellierung von Publikationen orientiert sich stark am BibTeX-Format [bib]. Es sind alle aus BibTeX bekannten Publikationstypen mit sämtlichen Attributen definiert.

Modellierung von multimedialen Inhalten Die Modellierung von multimedialen Inhalten ist nicht Gegenstand der Semantic Web Research Ontology. Daher ist hier ein anderer Ansatz notwendig. So wird zum Beispiel in [Hun01] eine Ontologie für die Darstellung von MPEG-7-Daten in RDF Schema vorgestellt. MPEG-7 dient der vorrangig inhaltsbasierten Beschreibung von multimedialen Dokumenten wie Images, Video, Audio, Sprache und Kombinationen aus diesen (z.B. Multimedia-Präsentationen). Dies soll jedoch nicht Thema dieser Arbeit sein, zum einen, da die Standardisierungsarbeiten zu MPEG-7 zu diesem Zeitpunkt noch nicht abgeschlossen sind, zum anderen da sich eine weitere Diplomarbeit [Rus01] speziell mit dieser Thematik auseinandersetzt.

Für die Modellierung von multimedialen Inhalten ist in dieser Arbeit lediglich die SWRO um das MPEG-7-Top-Level-Konzept **MultimediaContent** mit den Unterkonzepten **Audio**, **AudioVisual**, **Image**, **Video** und **Multimedia** erweitert worden. Als Properties sind die Attribute des Dublin Core [dcr01] eingeführt worden (siehe Tabelle 3.1). Als Ausnahme ist hier das Attribut *Identifier* zu nennen, welches nicht als Property modelliert wird, sondern direkt als RDF-Identifier der Resource dient. Der Dublin Core schreibt keine Restriktionen für die Nutzung der Attribute vor. In der hier modellierten Ontologie sind für die Properties sinnvolle Restriktionen für Kardinalität und Range eingeführt worden.

3.5.4 Modellierung des konkreten Szenarios

Hierbei geht um es die Modellierung einer konkreten Instanz einer Domäne, z.B. einer konkreten Ausstellung in der Ausstellungsdomäne oder einer Konferenz in der Konferenzdomäne. Dabei stehen vor allem zeitliche und räumliche Strukturen wie Veranstaltungspläne oder Ausstellungspläne im Vordergrund, aber z.B. auch Organisationsstrukturen. Auch hierfür stellt die Semantic Web Research Ontology grundlegende Konzepte bereit. Die wichtigsten für die Modellierung von Ausstellungen und Konferenzen relevanten Konzepte seien noch einmal genannt: **Person**, **Organization**, **Project**, **Product**, **Topic** und **Event**. Für die Modellierung räumlicher Strukturen wiederum wird ein eigener Ansatz vorgestellt.

Property	Range	Cardinality
title	Literal	1
subject	Topic	1..n
description	Literal	1
creator	Person	1..n
publisher	Organization	1
contributor	Person	1..n
date	Literal	1
type	Literal	1..n
format	Literal	1
source	Literal	1..n
relation	Literal	1..n
coverage	Literal	1..n
rights	Literal	1..n

Tabelle 3.1: Die Dublin Core Attribute als Properties der Klasse **MultimediaContent**

Personen Zum Konzept **Person** existiert eine weit verzweigte Konzepthierarchie mit insgesamt 14 Unterkonzepten. Zu diesen Konzepten sind Relationen definiert, über die vielfältige semantische Relationen beschrieben werden können, z.B. zu anderen Personen, Projekten, Publikationen und Events. Das Konzept **Person** eignet sich zur Modellierung von Autoren, Vortragenden, Firmenangehörigen, etc.

Organisationsstrukturen Es werden sowohl Forschungs- als auch industrielle Organisationen mit ihrer internen Struktur (z.B. **University**, **Department**, **Institute**, **Research Group**) berücksichtigt. Die interne Struktur wird dabei über die Relation **hasParts** beschrieben. Range-Restriktionen garantieren eine hierarchische Struktur. Weitere semantische Relationen werden zu Personen (z.B. **employs**), Projekten (**carriesOut**), Produkten (**develops**) und anderen Organisationen (**cooperateWith**) beschrieben.

Projekte Das Konzept **Project** dient der Modellierung von Projekten. Dabei wird zwischen industriellen und Forschungsprojekten unterschieden (**DevelopmentProject** und **ResearchProject**). Die wichtigsten Properties wie **carriedOutBy**, **isAbout**, **projectInfo** stellen semantische Beziehungen zu Organsiationen, Themen, Publikationen usw. her.

Produkte Zum Modellieren von Produkten existiert das Konzept **Product**. Über die Property **developedBy** wird die Beziehung zu Organisationen ergestellt.

Themen Das Konzept **Topic** ermöglicht die Modellierung von Themen. Zur Klassifikation der Themen werden keine Vorgaben gemacht. Sinnvoll ist jedoch die Verwendung eines Schemas wie die CR-Klassifikation oder eines spezielleren Schemas für ein konkretes Fachgebiet.

Ereignisstrukturen Zur Modellierung von Ereignisstrukturen existiert das Konzept **Event** mit den Unterkonzepten **Conference**, **Exhibition**, **Lecture**, **Meeting** und **Workshop**. Auch bei Events ist über die Relation **atEvent** und die dazu inverse Relation **hasPartEvent** eine interne Struktur darstellbar. Allerdings muss es sich hierbei nicht um eine reine hierarchische Struktur handeln.
Über die Relation **located** wird die Beziehung zu räumlichen Strukturen hergestellt. Diese Relation ist nicht Bestandteil der SWRO. Sie wurde im Zusammenhang mit der Modellierung von räumlichen Strukturen ergänzt.

Räumliche Strukturen Komplexe räumliche Strukturen sind in der SWRO nicht berücksichtigt. Daher wird hierfür ein eigener Ansatz vorgestellt: Zur Modellierung von räumlichen Strukturen wird das Konzept **PhysicalStructure** mit den Unterkonzepten **Building**, **Floor**, **Room** und **Booth** eingeführt.

Die Instanzen dieser räumlichen Strukturen stehen in einer *contains*-Relation: Ein Gebäude umfasst Etagen, eine Etage umfasst Räume, etc. Durch diese Relation wird eine hierarchische Topologie von räumlichen Strukturen beschrieben. Zur Darstellung der *contains*-Relation ist die Property **contains** definiert. Domain und Range sind in Tabelle 3.2 aufgeführt. Mit Hilfe der *contains*-Relation und dem

Domain	Range
Building	Floor
Floor	Room
Room	Booth

Tabelle 3.2: Domain und Range-Restriktionen der Property **contains**

Wissen über die Position von Objekten können neue Informationen abgeleitet werden. So könnte man z.B. wie in [Sch95] die Relationen *in* (containment), *at* (location) und *with* (co-location) definieren, um damit Fragen beantworten zu können wie: Ist Objekt X in demselben Gebäude wie Objekt Y?

Weitere Properties des Konzeptes **PhysicalStructure** sind **name** und **number**.

Die Modellierung von Koordinatensystemen, Karten, etc. wird hier nicht betrachtet. Einige Aspekte sind bereits im eGuide Exchange Format berücksichtigt. Eine wissenbasierte Modellierung würde hier keine wesentlichen Vorteile bringen. Ein interessanter Ansatz könnte darin bestehen, Routing-Informationen als Prozesse auf der Basis des 3.5.5.2 vorgestellten Prozessmodells in zu modellieren.

Hiermit könnte man beispielsweise alternative Routen darstellen. Die Routing-Informationen könnten dann in einem Planungssystem verwendet werden, um Wegbeschreibungen zu generieren.

Sollen Koordinatensysteme, Karten und Routing-Informationen des bestehenden exf-Formates verwendet werden, so wäre es z.B. möglich, diese Informationen direkt in exf-Syntax zu integrieren. In RDF [LS] ist hier für ein spezielles Attribut, `rdf:parseType`, definiert. Wird für eine Property `rdf:parseType="Literal"` gesetzt, so kann innerhalb dieser Property beliebiges Markup verwendet werden. [3]

3.5.5 Modellierung der Situation

Die Situation setzt sich aus den vier bereits erläuterten Komponenten zusammen. Das Konzept **Situation** besitzt daher die folgenden vier Properties:

- **environment** für die Beschreibung der physischen Umgebung (siehe 3.5.5.1),
- **task** für die Aufgabenbeschreibung (siehe 3.5.5.2),
- **user** für die Definition des Nutzerprofils (siehe 3.5.5.3) und
- **device** für die Gerätebeschreibung (siehe 3.5.5.4).

3.5.5.1 Physische Umgebung

Wie bereits erwähnt, spielt die Ortsangabe eine entscheidende Rolle, da aus ihr weitere Kontextinformationen abgeleitet werden können. Für die Modellierung von Ortsinformationen sind abstrakte geometrische Modelle (z.B. Euklidische Koordinaten) nicht unbedingt sinnvoll. Ein intuitives Modell, aufbauend auf den räumlichen Strukturen aus 3.5.4 erweist sich als geeigneter.

Neben der Ortsangabe ist auch die Erfassung von Objekten in der Umgebung notwendig. Bei diesen Objekten kann es sich sowohl um stationäre Objekte (z.B. Exponate), aber auch um mobile Objekte (z.B. Personen) handeln.

Um den Bezug zu zeitlichen Strukturen, z.B. Ereignisstrukturen, herstellen zu können, ist des weiteren die Angabe der aktuellen Zeit erforderlich.

Das Konzept **Environment** besitzt die folgenden Properties:

- **location** beschreibt die aktuelle Position des Nutzers. Als Wert sind Instanzen des Konzeptes PhysicalStructure zulässig. Die Kardinalität ist 1.
- **nearObject** beschreibt Objekte in der Umgebung des Nutzers. Als Werte sind mehrere Instanzen beliebigen Typs erlaubt.
- **time** gibt die aktuelle Zeit als Literal in geeigneter Kodierung an. Vorgeschlagen wird hierfür ISO 8601.

[3] Diese Technik kann natürlich auch an beliebiger anderer Stelle angewendet werden. Damit wird eine einfache Koexistenz mit dem exf-Format in der Phase der Transition zur wissensbasierten Darstellung ermöglicht.

Es sei betont, dass man sich nur an einer Position, jedoch in der Nähe von mehreren Objekten befinden kann.

3.5.5.2 Aufgaben

Für die Modellierung von Aufgabenstrukturen wird auf die DAML-S-Ontologie zurückgegriffen. DAML-S [DAM01a] ist eine spezielle Ontologie, die eigentlich für die Modellierung von Diensten im Web entwickelt wurde. Services werden hier als Prozesse modelliert. Dabei ist das zu Grunde liegende Prozessmodell jedoch so generisch, dass es sich auch hervorragend zur Modellierung von Aufgabenstrukturen eignet. Im Prozessmodell sind Einflüsse aus den Bereichen KI (Planung, Modellierung komplexer Aktionen), Workflow-Technologie, Agenten-Kommunikation, Programmiersprachen und Verteilte Systeme wiederzufinden. Die

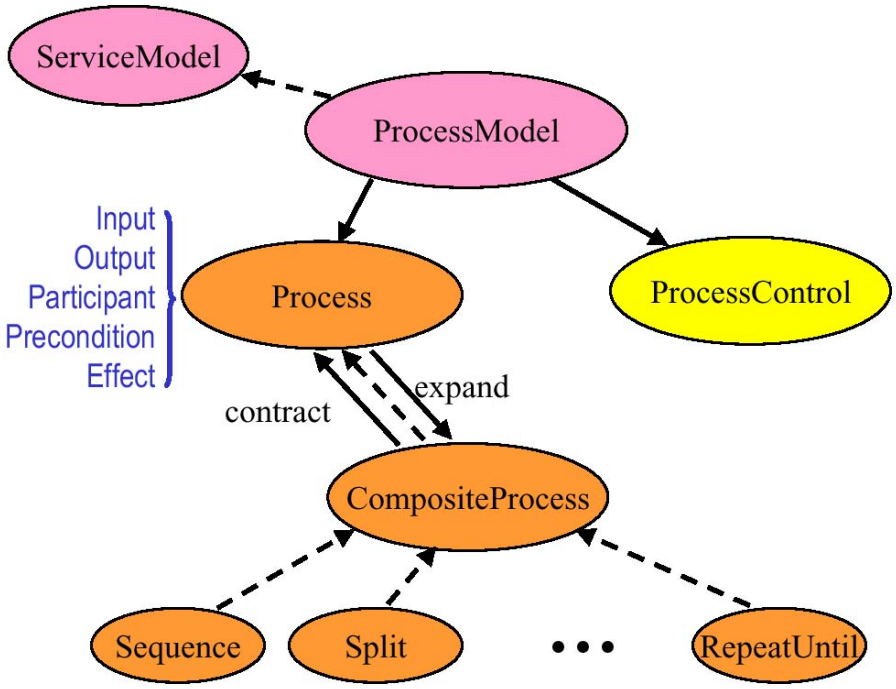

Abbildung 3.4: Das DAML-S Prozessmodell

beiden Hauptbestandteile des Prozessmodells sind der Prozess und das Prozess-Kontrollmodell (siehe Abbildung 3.4). Der Prozess beschreibt zum Beispiel eine Aufgabe basierend auf ihren Teilaufgaben und erlaubt Planung, Dekomposition, etc. Das Prozess-Kontrollmodell erlaubt die Überwachung der Ausführung (Scheduling) von Aufgaben. Hier wird also der Zustand der Aufgabe berücksichtigt. Damit genügt das Modell den in 2.3.2 genannten Anforderungen in Hinblick auf die Darstellung der internen Struktur und des Ausführungszustandes der Aufga-

ben. Das Konzept **Task** des Situationsmodells wird deshalb als Unterkonzept des Konzeptes **Process** der DAML-S-Ontologie definiert.

Leider ist in der aktuellen Version von DAML-S (Version 0.5) erst der erste Bestandteil, also der Prozess, vollständig definiert. Das Prozess-Kontrollmodell wird zwar grob beschrieben, die konkrete Definition ist aber erst für eine zukünftige Version geplant. Prozess und Prozess-Kontrollmodell sollen nun kurz entsprechend [DAM01a] beschrieben werden.

Prozess Ein Prozess kann eine beliebige Anzahl von Eingaben (für die Ausführung des Prozesses benötigte Informationen) und Ausgaben (Informationen, die der Prozess nach seiner Ausführung bereitstellt) haben. Es kann beliebig viele Vorbedingungen und Effekte geben.

Prozesse können atomar (undecomposable) oder zusammengesetzt (composite) sein. Zur Klasse **Process** gibt es deshalb die Unterklasse **CompositeProcess**, zu welcher es wiederum weitere Kontrollstrukturen als Unterklassen gibt:

- **Sequence:** Eine Liste von Prozessen, die nacheinander ausgeführt werden müssen.
- **Split:** Eine Menge von Prozessen, die parallel auszuführen sind.
- **Unordered:** Eine Menge von Prozessen, die in beliebiger Reihenfolge ausgeführt werden können.
- **Split+Join:** Prozesse, die partiell synchronisiert werden müssen.
- **Choice:** Auswahl aus einer Menge von Prozessen.
- **Condition:** Prozesse mit binärem Output zum Testen einer Bedingung.
- **If-Then-Else:** Ein Prozess, der sich aus einer Condition, einem Then- und einem Else-Prozess zusammensetzt.
- **Iterate:** Wiederholte Ausführung eines Prozesses.
- **Repeat-Until** Wiederholte Ausführung eines Prozesses mit einer untilCondition als Abbruchbedingung.

Prozess-Kontrollmodell Wie bereits erwähnt, ist das Prozess-Kontrollmodell noch nicht vollständig definiert. Es sind jedoch bereits Anforderungen an ein Modell zur Überwachung von Instanzen eines Prozesses spezifiziert, die in einer zukünftigen Definition erfüllt werden sollen:

1. Es sollte Überführungsregeln für die verschiedenen Eingangszustands-Properties (Inputs, Preconditions) auf die korrespondierenden Ausgangszustands-Properties bieten.

2. Es sollte ein Modell für temporale oder Zustands-Abhängigkeiten (Sequence, Split, Join, etc.) bieten.

3. Es sollte Repräsentationen für Nachrichten über den Ausführungszustand von atomaren und zusammengesetzten Prozessen bieten, um Ausführungsüberwachung zu ermöglichen.

Bisher ist lediglich die Klasse **ProcessControlStatus** definiert. Konkrete Prozesskontrollzustände sind: **Ready**, **Ongoing**, **Suspended**, **Aborted**, **Canceled** und **Completed**.

3.5.5.3 Nutzerprofil

Die Nutzermodellierung ist Gegenstand der Forschung im Bereich Human Computer Interaction und soll an dieser Stelle nicht tiefergehend behandelt werden. An dieser Stelle wird nur ein einfaches Modell vorgestellt, in dem sich gebräuchliche Aspekte der Nutzermodellierung wiederfinden. Für spezielle Anwendungen ist eine Erweiterung dieses einfachen Modells sicherlich sinnvoll.

Als Eigenschaften sind folgende Properties definiert:

- **interestedIn**: Über diese Property wird das Interesse des Nutzers an bestimmten Themen dargestellt. Als Werte sind entsprechend 0 bis n Instanzen der Klasse **Topic** zulässig.

- **language**: Diese Property gibt an, welcher Sprachen der Nutzer mächtig ist. Zulässig sind 0 bis n Werte vom Typ Literal. Zur Kodierung der Sprachen wird RFC 1766 verwendet.

- **expertise**: Hierüber kann die Vorbildung des Nutzers dargestellt werden, beispielsweise *Expert* oder *Novice*. Die Kardinalität dieser Property ist 1.

- **preference**: Hierüber können weitere Präferenzen des Nutzers angegeben werden. Zulässig sind 0 bis n Werte vom Typ Literal.

Oftmals werden Ziele und Aufgaben in die Nutzermodellierung einbezogen. Aufgrund ihrer besonderen Stellung im Inferenzprozess werden sie in diesem Situationsmodell gesondert behandelt.

3.5.5.4 Geräteeigenschaften

Die Eigenschaften eines Gerätes ändern sich im Allgemeinen nicht, sie sind aber trotzdem Bestandteil des Situationsmodells, da der Zugriff des Nutzers auf die Digitale Bibliothek in verschiedenen Situationen über unterschiedliche Geräte erfolgen kann. Für die Bestimmung relevanter Inhalte ist insbesondere entscheidend, welche Multimedia-Typen und Dateiformate das Gerät darstellen kann. Hierfür werden die folgenden Properties verwendet:

- **supportedContent**: Über dieses Attribut wird festgelegt, welche der in 3.5.3 beschriebenen MPEG-7-Multimedia-Typen unterstützt werden. Hierbei geht es nicht um konkrete Formate, sondern darum, ob das Gerät auf Grund seiner Hardware-Eigenschaften in der Lage ist, mit diesen Inhalten umzugehen.

- **supportedFormat**: Dieses Attribut bestimmt, welche konkreten Dateiformate unterstützt werden. Das Format der multimdialen Inhalte wird, wie beschrieben, über das Dublin-Core-Attribut *format* spezifiziert. Zur Kodierung der Formate werden die *Internet Media Types* (IMT) [imt] verwendet.

Zusätzlich werden die folgenden Geräteeigenschaften erfasst:

- **resolution** spezifiziert die Auflösung des Displays.

- **colors** spezifiziert die Farbtiefe des Displays an.

3.6 Abbildung der DAML-Ontologie auf Programmiersprachen- und Datenbankmodelle

Leider wird sich die alleinige Verwendung des wissensbasierten Datenmodells von RDF und seinen Erweiterungen für alle Komponenten der Systemarchitektur als nicht günstig erweisen. Im folgenden soll aufgezeigt werden, wie eine Abbildung auf Programmiersprachen- und Datenbankmodelle erfolgen kann.

Fensel nennt in [Fen00a] drei Aspekte der Programmierung, die für den Austausch von Information und Wissen notwendig sind:

- eine Sprache zur Beschreibung der Struktur und Semantik

- eine Sprache zur Verarbeitung der Daten

- ein Protokoll für Datenaustausch

Ergänzt sei an dieser Stelle der folgende, ebenfalls notwendige Aspekt:

- persistentes Speichern der Daten

Im Laufe dieses Kapitels wurde bereits gezeigt, dass RDF mit seinen Erweiterungen als Sprache zur Beschreibung der Struktur und Semantik der Daten geeignet ist. Auch für den Datenaustausch ist RDF mit XML als zugrundeliegender Syntax, zusammen mit HTTP als Standardtransferprotokoll, geeignet. Für die Verarbeitung von Daten hat sich Java als Programmiersprache, insbesondere im Zusammenhang mit XML, bewährt. Für das effiziente, persistente Speichern von Daten gibt es, wie später noch erläutert wird, kaum eine Alternative zu relationalen Datenbanken.

Es werden also drei Objekt- bzw. Datenmodelle eine Rolle spielen: Die zentrale Rolle spielt dabei RDF bzw. DAML, sowohl für die Beschreibung von Struktur und Semantik als auch für den Datenaustausch. Für die Verarbeitung der Daten,

insbesondere die Darstellung von Inhalten, wird das Java-Objektmodell verwendet. Für das persistente Speichern der Daten wird auf das relationale Datenmodell zurückgegriffen. Abbildung 3.5 zeigt einen Überblick über die verwendeten Daten- und Objektmodelle. Die Pfeile kennzeichen dabei Transformationen zwischen den

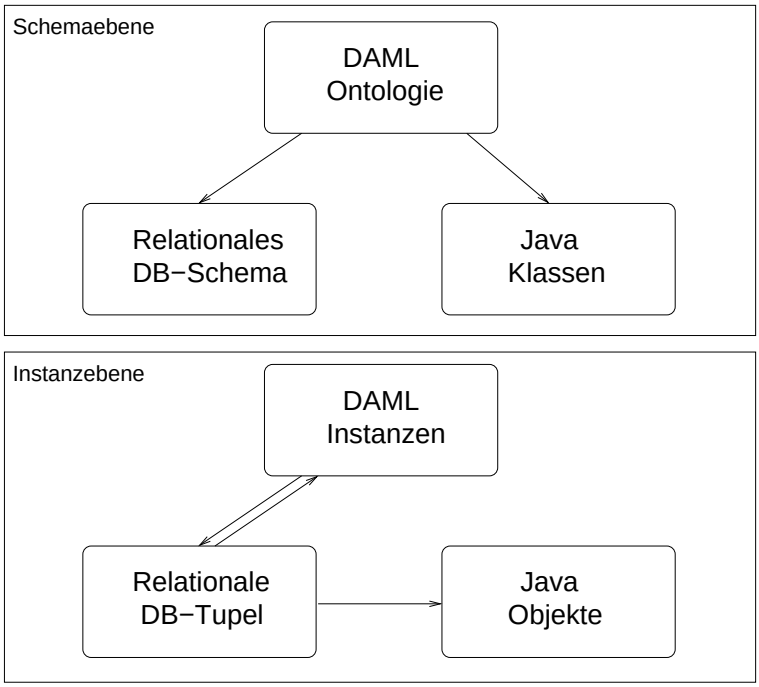

Abbildung 3.5: Daten- und Objektmodelle auf Schema- und Instanzebene

Modellen, die im folgenden vorgestellt werden.

3.6.1 Abbildung der Ontologie auf eine Java-Klassenhierarchie

Aufgrund der Affinität des Frame-basierten Datenmodells zum objektorientierten Datenmodell von Java ist die Abbildung einer DAML-Ontologie auf eine Java-Klassenhierarchie denkbar einfach:

- Die Definition einer Klasse `<classname>` mit der Oberklasse`<superclass>` wird abgebildet auf ein Interface `interface <classname> extends <superclass>` und eine Klasse `class <classname>Impl extends <superclass>Impl implements <classname>`
- Die Definition einer Property `<propertyname>` mit der Domain `<domainname>` und Range `<rangename>` wird abgebildet auf die Definition eines Attributes `<propertyname>` in der Klasse `<domainname>Impl`, in Abhängigkeit von der Kardinalität der Property mengenwertig als `Vec-`

tor <propertyname> oder einwertig als String <propertyname>. In diesen Attributen werden die Values der Properties gespeichert. Handelt es sich bei diesen Values um Literale, so werden diese direkt gespeichert, handelt es sich um Resourcen, so werden die URIs als Referenz auf diese Resourcen verwendet. In der Interface-Definition von <domainname> werden für die Attribute entsprechende "Getter-" und "Setter-Methoden" definiert.

Beispiel Die Definition der Klasse Employee einer DAML-Ontologie

```
<rdfs:Class rdf:ID="Employee">
    <rdfs:subClassOf rdf:resource="#Person"/>
        <rdfs:subClassOf>
            <daml:Restriction daml:mincardinality="1"
                              daml:maxcardinality="n">
                <daml:onProperty rdf:resource="#affiliation"/>
                <daml:toClass rdf:resource="#Organization"/>
            </daml:Restriction>
        </rdfs:subClassOf>
</rdfs:Class>
```

wird abgebildet auf:

```
public interface Employee extends Person {
    void addAffiliation(String _affiliation);
    Vector getAffiliation();
}

public class EmployeeImpl extends PersonImpl
                          implements Employee {
    private Vector affiliation = null;

    public void addAffiliation(String affiliation) {
        if(this.affiliation==null)
            this.affiliation = new Vector();
            this.affiliation.addElement(affiliation);
        }

    public Vector getAffiliation() {
        return affiliation;
    }
}
```

Die Abbildung lässt sich leicht mit Hilfe eines XSLT-Stylesheets realisieren.

3.6.2 Persistentes Speichern der RDF-Metadaten

Für das persistente Speichern der RDF-Metadaten gibt es verschiedene Alternativen. Drei Möglichkeiten mit ihren Vor- und Nachteilen werden im folgenden vorgestellt:

1. Da die RDF-Metadaten in XML-Syntax vorliegen, wäre es naheliegend, diese in einer XML-fähigen Datenbank zu speichern. Dem stehen jedoch zwei Probleme entgegen. Zum einen lässt der XML-Support heutiger Datenbanksysteme zu wünschen übrig. Zum anderen gehen durch das Speichern auf

der Ebene des XML-Dokumentmodells die Vorteile des RDF-Datenmodells verloren.

2. In [Mel00] sind verschiedene Vorschläge zu finden, wie sich die Tripel des RDF-Datenmodells direkt in relationalen Datenbanken speichern lassen. Für viele Fälle ist es z.B. ausreichend, die Tripel in einer einfachen Relation (Resource, Property, Value) zu speichern. Allerdings werden hierbei keine RDF-Schemainformationen genutzt.

3. Ein weiterer Ansatz besteht darin, das Schema der Ontologie zu nutzen und aus dieser ein Datenbankschema abzuleiten. Nach [HS97] etwa können Frame-basierte Datenmodelle ziemlich direkt auf ein objektorientiertes Datenmodell abgebildet werden. Die Klassenhierarchie der RDFS bzw. DAML Ontologie würde dabei auf ein objektorientiertes oder objektrelationales Schema abgebildet werden. Allerdings gilt auch hier wieder, dass die Unterstützung objektorientierter und objektrelationaler Konzepte entsprechend SQL3 in vielen kommerziellen Datenbanksystemen nur unzureichend ist. So bietet zum Beispiel DB2 in der aktuellen Version 7.2 keine Unterstützung für mengenwertige Attribute. Hier wären wiederum relationale Techniken zum Umgehen dieser Einschränkungen notwendig. Zudem sind insbesondere für mobile Geräte wie PDAs ausschließlich relationale Datenbanksysteme verfügbar.

Keiner der obigen Vorschläge ermöglicht ein persistentes Speichern in relationaler Form auf der Ebene des RDF-Datenmodells unter Ausnutzung der RDF-Schemainformationen. Daher wird hier eine eigene Abbildung einer DAML-Ontologie auf ein relationales Schema vorgeschlagen:

- In zwei speziellen Relationen, `classes` und `properties` werden die Schemainformationen gespeichert. Diese Relationen werden bereits vor der eigentlichen Abbildung angelegt. In der Relation `classes` mit den Attributen `classname, superclass` wird die Klassenhierarchie erfasst, in der Relation `properties` mit den Attributen `domain, propertyname, range` werden die definierten Properties der Klassen abgelegt.

- Die Definition einer Klasse `<classname>` mit der Oberklasse `<superclass>` wird abgebildet auf die unäre Relation `<classname>`, in der die URIs aller unmittelbaren Instanzen zu dieser Klasse gespeichert werden. Außerdem wird das Tupel `(<classname>, <superclass>)` in die Relation `classes` aufgenommen.[4]

- Die Definition einer Property `<propertyname>` mit der Domain `<domainname>` und Range `<rangename>` wird abgebildet auf die Relation `p_<propertyname>` mit den Attributen `subject` und `object`, in der die Attribute der Instanzen gespeichert werden. Außerdem wird das Tupel

[4]Es sei bemerkt, dass dieses Mapping nur Einfachvererbung zulässt. Für Mehrfachvererbung müssten die Superklassen entsprechend mengenwertig gespeichert werden.

(<domainname>, <propertyname>, <rangename>) in die Relation properties aufgenommen.

Im Anhang A.4 findet sich ein XSL-T Stylesheet, das genau diese Transformation von DAML nach SQL-DDL (Data Definition Language) vornimmt.

Beispiel Analog dem Beispiel aus 3.6.1 würde die Definition der DAML-Klasse Employee in folgender Schemadefinition resultieren:

```
insert into classes    (classname,  superclass)
           values      ('Employee', 'Person')

insert into properties (domain,     propertyname,  range)
           values      ('Employee', 'affiliation', 'Organization')

create table Employee (identifier varchar(100) not null,
                       primary key(identifier))

create table p_affiliation (subject varchar(100),
                            object  varchar(100))
```

Speichern einer DAML-Instanz in relationaler Form Die DAML-Instanzen können nun in dem mit der eben beschriebenen Transformation erzeugtem relationalen Schema gespeichert werden. Dies geschieht wie folgt: Für eine DAML-Instanz vom Typ <classname> mit der URI uri als Objectidentifier wird uri in die Tabelle classname eingetragen. Für jede Property <property> mit dem Wert <value> wird das Tupel (<uri>, <value>) in die Tabelle p_<property> eingetragen.

Beispiel Die Instanz

```
<FullProfessor rdf:ID="mailto:heuer@informatik.uni-rostock.de">
        <firstName>Andreas</firstName>
        <lastName>Heuer</lastName>
        <affiliation rdf:resource="org:de.uni-rostock.informatik.db"/>
</FullProfessor>
```

würde in folgender relationaler Form gespeichert:

```
insert into FullProfessor (identifier) values
                          ('mailto:heuer@informatik.uni-rostock.de')
insert into p_firstName   (subject, object) values
                          ('mailto:heuer@informatik.uni-rostock.de', 'Andreas')
insert into p_lastName    (subject, object) values
                          ('mailto:heuer@informatik.uni-rostock.de', 'Heuer')
insert into p_affiliation (subject, object) values
                          ('mailto:heuer@informatik.uni-rostock.de',
                           'org:de.uni-rostock.informatik.db')
```

Die Rekonstruktion einer DAML-Instanz aus der relationalen Form geschieht umgekehrt unter Ausnutzung der Schemainformationen.

3.6.3 Generieren von Java-Objekten

Aus den in relationaler Form gespeicherten DAML-Instanzen lassen sich auch unmittelbar Java-Objekte zu der in 3.6.1 abgeleiteten Java-Klassenhierarchie erzeugen. Hierzu wird zunächst ein Java-Objekt vom Typ `<classname>` erzeugt. Die Properties werden als Attribute über die beschriebenen "Getter-Methoden" gesetzt. Die Konstruktion der Java-Objekte kann unter Ausnutzung des Java-Konzeptes der *Reflection* geschehen, so dass zur Übersetzungszeit der konkrete Typ der Objekte noch nicht bekannt sein muss.

Analog lassen sich natürlich auch Objekte einer beliebigen anderen objektorientierten Programmiersprache erzeugen.

3.7 Zusammenfassung

Ziel dieses Kapitels war die Datenmodellierung für den situationsgesteuertern Zugriff auf Digitale Bibliotheken. Zunächst wurden die zu modellierenden Daten identifiziert. Es handelt sich hierbei um Metadaten, die die Inhalte der Bibliothek, ein konkretes Szenario und die Situation des Nutzers beschreiben. Es wurde gezeigt, dass eine wissensbasierte, ontologiegestützte Repräsentation der Metadaten sinnvoll ist. Die Verwendung der Modellierungssprachen des Semantic Web erwies sich hierfür als sinnvoll, da sie syntaktische und semantische Interoperabilität, hohe Aussdrucksstärke und Erweiterbarkeit bieten. Für die Domäne Ausstellungs- und Konferenzbesuch wurde eine konkrete Ontologie in DAML entwickelt. Dabei wurde teilweise auf das eGuide Exchange Format und auf die Semantic Web Research Ontology zurückgegriffen.

Es wurde auch gezeigt, wie die Datenmodelle des RDF auf Programmiersprachenmodelle sowie ein relationales Modell zur persistenten Speicherung abgebildet werden können.

Im nächsten Kapitel wird beschrieben, wie die wissensbasierten Metadaten in Inferenzsystemen für automatisches Schließen genutzt werden können.

Kapitel 4

Inferenz und Reasoning

In diesem Kapitel wird aufgezeigt, wie auf der Basis der wissensbasierten Darstellung von Informationen neue Informationen abgeleitet werden können. Dieser Prozess nennt sich Inferenz oder auch Reasoning. Er ist zentrale Komponente der Wissensrepräsentation.

In den Anforderungen (Kapitel 2) ist bereits angedeutet worden, an welchen Stellen in diesem System Inferenz sinnvoll ist: Zum einen geht es darum, automatisch zu schließen, welche Informationen in einer bestimmten Situation relevant sind. Zum anderen soll bestimmt werden, welche Aufgabe in einer Situation als nächstes zur Bearbeitung vorgeschlagen werden soll. Es wird sich zeigen, dass für diese beiden Anwendungen zwei unterschiedliche Inferenzsysteme erforderlich sind. Gemein ist jedoch beiden, dass sie, wie die meisten Expertensysteme, regelbasiert funktionieren. Die regelbasierte Programmierung ist das gebräuchlichste Verarbeitungsparadigma für wissensbasierte Systeme, Expertensysteme und KI-Programmiersprachen. Zu der Bedeutung regelbasierter Systeme schreibt Peter Jackson in [Jac90]: “It is probably an axiom of artificial intelligence that intelligent behaviour is rule-governed.”

Regelbasierte Expertensysteme bestehen aus den folgenden Komponenten:

- *Fakten* zur aussagenbezogenen, deskriptiven Feststellung von Sachverhalten,
- *Regeln*, auf deren Basis aus gegebenen Fakten neue Fakten abgeleitet oder Aktionen ausgeführt werden können,
- *Inferenzmaschine*, die die Ausführung der Regeln kontrolliert.

Im Groben lassen sich die Anwendungen von Inferenzsystemen in zwei Klassen einteilen: Anfragesysteme und Konsultationssysteme. Typische Eigenschaften sind dabei:

- **Anfragen** haben als einzige Form der Eingabe die Wissensbasis. Es findet eine mengenorientierte Verarbeitung statt.

- **Konsultationen** erfolgen in ständiger Interaktion mit dem Nutzer. Die Verarbeitung erfolgt schrittweise, wobei in einem Schritt jeweils *eine* (die "richtige") Regel zur Ausführung ausgewählt wird.

Entsprechend lassen sich zwei Arten von Regeln unterscheiden:

- **deduktive Regeln** für Anfragesysteme und
- **prozedurale Regeln** mit Seiteneffekten für konsultative Systeme, z.B. Produktionsregeln

Für das Bestimmen relevanter Dokumente in einer Digitalen Bibliothek wird sich ein Anfragesystem als geeignet erweisen. Dagegen erfolgt die Auswahl der Aufgaben in Interaktion mit dem Nutzer, schließlich kann der Nutzer einen Vorschlag ausschlagen und etwas anderes tun. Es ist hierfür also ein konsultatives Inferenzsystem erforderlich.

Im folgenden werden zwei konkrete regelbasierte Inferenzsysteme vorgestellt. Als erstes wird gezeigt, wie mit Frame-Logic aufbauend auf dem Frame-basierten Datenmodell des RDF Inferenz betrieben werden kann. Mit Frame-Logic als Anfragesystem wird es möglich sein, relevante Dokumente zu bestimmen. Als zweites wird ein Produktionssystem, also ein konsultatives Inferenzsystem, vorgestellt, das geeignet ist, Aufgabenselektion durchzuführen.

Es gibt auch Ansätze, die Vorteile beider Arten von Expertensystemen zu verbinden: So werden in [TS99] "Queryable consultation systems" vorgestellt, eine Kombination eines Produktionssystems mit der Anfragesprache Datalog. Diese sollen an dieser Stelle jedoch nicht behandelt werden. Bezüglich eines Vergleichs der Produktionen als Berechnungsmodell mit dem der Frames sei erwähnt, dass in [Per88] gezeigt worden ist, dass Frames und Produktionssysteme gegenseitig mit linearem Aufwand simulierbar sind, so dass von hier kein Argument für oder gegen beide Ansätze zu holen ist [Bib93].

4.1 Inferenz und Reasoning mit RDF und Frame-Logic

Obwohl sich in RDF viele Einflüsse aus der Wissensrepräsentation finden, spezifiziert RDF keinen Mechanismus für Reasoning. Ein Reasoning-Algorithmus kann jedoch auf RDF als Frame-basiertem System aufgebaut werden. Dies wurde zum Beispiel mit SiLRI (Simple Logic Based RDF Interpreter) [DBSA] realisiert.

In [SEMD00] wird ein Überblick gegeben, welche Kategorien von Axiomen häufig für Inferenz benötigt werden und wie diese in RDFS ausgedrückt werden können. Die Kategorien sind:

1. Axiome für relationale Algebra: Reflexivität, Irreflexivität, Symmetrie, Asymmetrie, Antisymmetrie, Transitivität von Relationen, sowie inverse Relationen
2. Komposition von Relationen

3. (vollständige) Partitionierungen
4. Axiome für Subrelationen
5. Axiome für "Part-whole Reasoning"

In [DBSA] werden Anforderungen an ein RDF-basiertes Anfrage- und Inferenzsystem genannt:

1. Es sollte das RDF-Datenmodell direkt unterstützen, welches eng an objektorientierte und Frame-basierte Sprachen angelehnt ist. Daher sollte eine Anfragesprache Konzepte, die aus objektorientierten Systemen bekannt sind (Klassenhierarchien, Vererbung), unterstützen.
2. Im Kontext von verteilten, heterogenen Metadaten in RDF, die für ein breites Spektrum von Anwendungen definiert sind, ist es notwendig, auf Schemainformationen für diese Vokabulare (definiert in RDFS) zugreifen und diese in Anfragen nutzen zu können.
3. Auch wenn das RDF-Datenmodell dazu geeignet ist, die Struktur von Anfragen darzustellen, ist die RDF/XML Serialisierung nicht unbedingt das beste Format um sie dem Menschen zu präsentieren. Daher sollte die RDF-Anfragesprache auf der Ebene des abstrakten Modells definiert sein, unabhängig von der Syntax. Eine Syntax angelehnt an SQL oder Datalog mag schließlich einfacher verständlich sein als eine XML Repräsentation.

Diesen Anforderungen wird das weiter unten beschriebene Frame-Logic-basierte Inferenzsystem SiLRI gerecht.

4.1.1 Einführung in Frame-Logic

Frame-Logic (kurz F-Logic) ist eine deduktive, objektorientierte Datenbanksprache. Sie vereint die deklarative Semantik und Ausdrucksstärke von deduktiven Datenbanksprachen mit den umfassenden Mitteln der Datenmodellierung des objektorientierten Datenmodells. Die theoretischen Grundlagen von F-Logic sind im F-Logic Report [KLW95] beschrieben. F-Logic ermöglicht es, über alle vorhandenen Informationen zu inferieren, indem entsprechende deklarative Regeln im Datalog-Stil angegeben werden. Die Sprache ist durch ihre Fähigkeit, auf Metadaten (das modellierte Schema) und Daten (Instanzen zu diesem Schema) in gleicher Weise zuzugreifen, prädestiniert zur Repräsentation von Ontologien. Es werden nun kurz Syntax und Objekt-Modell sowie die Semantik von F-Logic entsprechend [Erd01] vorgestellt.

Syntax und Objekt-Modell Syntaktisch gesehen entspricht F-Logic einer Obermenge der Prädikatenlogik erster Stufe (FOL): F-Logic ist um objektorientierte Modellierungsperformativen erweitert. Die Ausdrucksmächtigkeit beider Sprachen

ist jedoch äquivalent. Wie auch in FOL lassen sich die Symbole des F-Logic Alphabets in Prädikatensymbole *P*, in Funktionensymbole *F* und Variablen *V* unterteilen. Terme über $F \cup V$ werden ID-Terme genannt und bezeichnen Objekte, Methoden oder Klassen.

Atomare F-Logic-Ausdrücke (F-Atome) entsprechen einem der folgenden Schemata. Sie definieren entweder die Ergebnisse von Methodenanwendungen, die Signatur von Methoden oder Instanz- bzw. Subklassenbeziehungen zwischen ID-Termen:

- $o[m \rightarrow r]$ Dieser Ausdruck besagt, dass die Methode m angewandt auf o das Ergebnisobjekt r ergibt. m ist eine *single-valued* (oder funktionale) Methode, d.h. es gibt maximal ein Objekt, das diese Aussage wahr macht.
- $o[m \twoheadrightarrow r_1, r_2, ..., r_n]$ Dieser Ausdruck ist ebenso wie der vorherige ein *Data-Atom* und besagt, dass die r_i das Resultat der Anwendung der Methode m auf das Objekt o sind. m ist hier eine *multi-valued* (oder mengenwertige) Methode.
- $c[m \Rightarrow t]$ Dieses Atom und das folgende sind so genannte *Signatur-Atome*, die besagen, dass die Anwendung der Methode m auf Instanzen der Klasse c Objekte vom Typ t ergibt (für single-valued Methoden).
- $c[m \Rightarrow\!\!\!> t]$ Dies ist das Signatur-Atom für multi-valued Methoden.
- $o : c$ besagt, dass das Objekt o eine Instanz der Klasse c ist.
- $c_1 :: c_2$ besagt, dass die Klasse $c1$ eine Subklasse der Klasse $c2$ ist.

Die Platzhalter o,m,r,c stehen bei diesen Definitionen jeweils für ID-Terme.

Wird nicht zwischen single- und multi-valued Methoden unterschieden, so kann an Stelle der unterschiedlichen Pfeile auch ein Gleichheitszeichen "=" für das Ergebnis eines Methodenaufrufs bzw. ein "⇒" in den Signaturatomen geschrieben werden.

Konjunktiv verknüpfte Atome lassen sich zur Schreibvereinfachung zu *F-Molekülen* zusammenfassen. So ist zum Beispiel

$$X : Researcher[authorOf = Y; cooperatesWith = Z]$$

äquivalent zu

$$X : Researcher \wedge X[authorOf = Y] \wedge X[cooperatesWith = Z]$$

Da F-Logic (abwärts-)kompatibel zu Sprachen wie Datalog ist, bietet die Syntax die so genannten *P-Atome* an, die aus einem Prädikatensymbol $p \in P$ und einer Liste von ID-Termen, den Parametern, bestehen, z.B.: $Relevant(BlueView)$ mit $Relevant \in P$.

Regeln in F-Logic bestehen aus einem Regelkopf und einem Regelrumpf. Der Regelkopf besteht aus einem P-Atom oder einem F-Molekül, und der Rumpf aus

einer Konjunktion von P-Atomen und F-Molekülen, alle Variablen sind all-quantifiziert, z.B.

$$\forall X, Y\ Y : AcademicStaff[cooperateWith = X] \leftarrow \\ X : AcademicStaff[cooperateWith = Y].$$

Regeln dienen dazu, die gegebenen Fakten intensional zu erweitern, indem deklarativ neue Informationen abgeleitet werden. Im Regelrumpf können einige Atome auch negiert vorkommen. Für alle Regeln muss aber die so genannte *Safety*-Bedingung gelten, um zuzusichern, dass keine unendlichen Berechnungen notwendig werden. Für Details hierzu sei auf [FHKS96] verwiesen.

Ein *F-Logic-Programm* besteht aus einer Menge von Regeln und Fakten. Durch *bottom-up* Evaluation der Regeln und der Abschlusseigenschaften wird ausgehend von den Fakten ein Modell berechnet, das mittels Queries angefragt werden kann. *Queries* bestehen in F-Logic ähnlich zu Datalog oder Prolog aus einem Regelrumpf und einem leeren Regelkopf, z.B.

$$\forall X, Y, Z \leftarrow X : FullProfessor[affiliation = uni - rostock, \\ publication = Y] \wedge Y[title = Z].$$

Das Ergebnis besteht aus einer Reihe von Substitutionen für die Variablen der Query. Wendet man eine Substitution auf die Query an, so erhält man eine wahre Aussage, die aus den Fakten der Wissensbasis abgeleitet werden kann, z.B.

$$X = Heuer, Y = BlueView, Z ='' BlueView : Virtual Document Servers''$$

Semantik Die Semantik eines F-Logic-Programms P wird durch eine so genannte F-Struktur [KLW95] repräsentiert. Sie enthält alle gegebenen Fakten und wird durch zusätzliche Informationen vervollständigt, die durch so genannten Abschlusseigenschaften (*closure properties*) und Regelanwendungen zugesichert werden. Ein Atom t ist in einer gegebenen F-Struktur I genau dann wahr, wenn $I \models t$. Dabei enthält I ausschließlich Terme der Form:

- $p(o_1, o_2, ..., o_n)$
- $c[m \Rightarrow t]$
- $o_1[m = o_2]$
- $o : c$
- $c_1 :: c_2$

Die Platzhalter c, c_1,c_2,m,t,o und o_i stehen dabei und im folgenden für variablenfreie ID-Terme. Für jede F-Struktur I gelten folgende Zusammenhänge:

- Kommt c in I vor (an beliebiger Position in einem der Atome), dann gilt $I \models c[]$ (d.h. der ID-Term c ist eine Folgerung aus dem Modell).

- Kommt c in I vor, dann gilt $I \models c :: c$ (Reflexivität der Subklassenrelation).
- Gilt $I \models c :: c_1$ und $I \models c_1 :: c_2$, dann gilt auch $I \models c :: c_2$ (Transitivität der Subklassenrelation).
- Gilt $I \models o : c_1$ und $I \models c_1 :: c_2$, dann gilt auch $I \models o : c_2$ (Inklusionsbeziehung zwischen Subklassen).
- Gilt $I \models c_2[m \Rightarrow t]$ und $I \models c_1 :: c_2$, dann gilt auch $I \models c_1[m \Rightarrow t]$ (Vererbung von Typinformation).
- Gilt $I \models o[m \Rightarrow t]$ und $I \models t :: c$, dann gilt auch $I \models o[m \Rightarrow c]$ (Ergebnistyp-Relaxation).

Die Semantik von F-Logic orientiert sich an der Fixpunkt-Semantik von Datalog-Programmen [Ull88]. Darin wird ein Programm P bottom-up evaluiert, d.h. im ersten Schritt werden zu der initial leeren F-Struktur die Fakten aus P hinzugefügt. In den nächsten Schritten werden alle Regeln angewandt, deren Regelrümpfe durch Variablensubstitutionen zu *true* evaluiert werden können. Ist $h \leftarrow l_1 \wedge l_2 \wedge ... \wedge l_k$ eine Regel in P und es gibt eine Substitution σ, so dass für alle $i(1 \leq i \leq k)$ gilt $I \models l_i\sigma$, dann gilt auch $I \models h\sigma$. Das heißt, die Anwendung einer Regel führt zu einer Erweiterung der F-Struktur. Ebenso werden ständig die Closure-Properties sichergestellt, so dass die F-Struktur stetig wächst. Die Semantik des F-Programms ist berechnet, sobald ein Fixpunkt erreicht ist.

Diese Definition reicht aus, um die Semantik für F-Logic Programme ohne Negation zu definieren. In [KLW95] sind erweiterte Semantiken zu finden. So lässt beispielsweise die *inflationäre* Semantik Negationen zu.

Abbildung 4.1 zeigt einen Überblick über den Inferenzprozess in einem Frame-

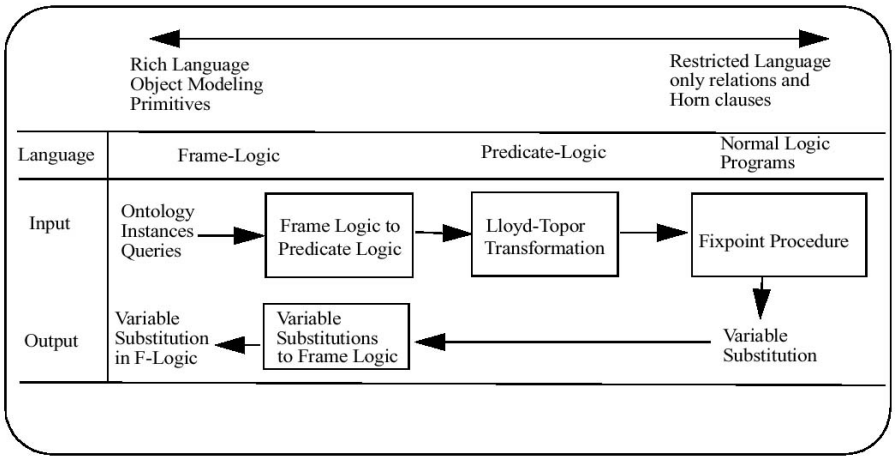

Abbildung 4.1: Der Inferenzprozess von SiLRI

Logic-basierten Inferenzsystem (SiLRI).

Für eine ausführliche Beschreibung des Prozesses sei auf [Fen00a] verwiesen.

4.1.2 Konkrete Regeln

In diesem Abschnitt sollen konkrete Regeln in F-Logic vorgestellt werden. Als Beispiel dient hierfür das in 1.2 beschriebene Szenario. Die Regeln sind bewusst einfach gehalten und sollen kein vollständiges Regelsystem darstellen, verdeutlichen aber, wie auf der Basis einer komplexen Situationsbeschreibung relevante Inhalte der Bibliothek bestimmt werden können. Zur besseren Lesbarkeit sind Variablennamen und Methoden klein, Konzepte und Prädikate groß geschrieben.

Berücksichtigung von Nutzerinteressen Mit Hilfe der ersten Regel können für den Nutzer relevante Publikationen auf der Basis seiner Interessen bestimmt werden:

$$\begin{aligned} &\forall publ, user, topic\ Relevant(publ) \leftarrow \\ &\quad user : User[interestedIn \twoheadrightarrow topic] \wedge \\ &\quad publ : Publication[keywords \twoheadrightarrow topic]. \end{aligned} \tag{4.1}$$

Nutzung von Orts- und Zeitinformation Mit Hilfe der nächsten Regel können z.B. Vortragsinformationen bereitgestellt werden, sobald der Nutzer den Raum betritt, in dem ein Vortrag gehalten wird:

$$\begin{aligned} &\forall event, physStruct\ Relevant(event) \leftarrow \\ &\quad event : Event[located \twoheadrightarrow physStruc : PhysicalStructure] \wedge \\ &\quad (In(physStruct) \vee At(physStruct)) \wedge During(event) \end{aligned} \tag{4.2}$$

Befindet sich der Nutzer an einem bestimmten Ort, an dem momentan eine Veranstaltung stattfindet, so sind für den Nutzer Informationen zu dieser Veranstaltung relevant.

Die letzte Regel verwendet zum einen das Zeitprädikat *During* und zum anderen die Ortsprädikate *At* und *In*. Die Modellierung zeitlicher Relationen soll an dieser Stelle nicht vertieft werden und wäre, wie auch im Ausblick dargelegt, sinnvoller Gegenstand einer weiterführenden Arbeit. Lediglich die Ortsprädikate sollen kurz erläutert werden: *At* steht für eine direkte Ortsangabe und wird aus der Situationsbeschreibung bestimmt:

$$\begin{aligned} &\forall physStruct, environm\ At(physStruct) \leftarrow \\ &\quad environm : Environment[location \twoheadrightarrow physStruct]. \end{aligned} \tag{4.3}$$

Das *In*-Prädikat dagegen nutzt die hierarchische Topologie der räumlichen Strukturen, die durch die *contains*-Relation definiert ist: Befindet man sich an einem

bestimmten Ort (*At*), so befindet man sich auch in den umgebenden räumlichen Strukturen (*In*). *In* lässt sich mit folgender Regel bestimmen:

$$\begin{aligned} &\forall x, y\ In(y) \leftarrow \\ &\quad (At(x)\ \vee\ In(x))\ \wedge\ y[contains \twoheadrightarrow x]. \end{aligned} \tag{4.4}$$

Befindet sich der Nutzer z.B. an einem Ausstellungsstand, dann kann über diese Regel abgeleitet werden, in welchem Raum, welcher Etage und welchem Gebäude er sich befindet.

Ein anderes Beispiel für die Nutzung von Ortsinformationen wäre:

(4.5)

$$\begin{aligned} &\forall product, project, user, topic\ Relevant(product), Relevant(project) \leftarrow \\ &\quad Near(product : Product) \wedge \\ &\quad project : Project[product \twoheadrightarrow product] \wedge \\ &\quad project[isAbout \twoheadrightarrow topic : Topic] \wedge \\ &\quad user : User[interestedIn \twoheadrightarrow topic]. \end{aligned}$$

Befindet sich der Nutzer in der Nähe eines ausgestellten Produktes, welches im Rahmen eines Projektes entwickelt wird, dessen Thema den Nutzer interessiert, so sind sowohl das Projekt als auch das Produkt für den Nutzer relevant.

Das Prädikat *Near* ist dabei analog zu *At* definiert:

$$\begin{aligned} &\forall x, environm\ Near(x) \leftarrow \\ &\quad environm : Environment[nearObject \twoheadrightarrow x]. \end{aligned} \tag{4.6}$$

Nutzung der Gerätebeschreibung Mit Hilfe der nächsten Regel wird ermittelt, ob ein Gerät bestimmte Inhalte darstellen kann.

$$\begin{aligned} &\forall multimedia, device, format\ Displayable(multimedia) \leftarrow \\ &\quad multimedia : MultimediaContent[format \twoheadrightarrow format] \wedge \\ &\quad device : Device[supportedFormat \twoheadrightarrow format]. \end{aligned} \tag{4.7}$$

Multimediale Inhalte sind darstellbar, wenn das Format der Inhalte mit einem der Formate übereinstimmt, die das Gerät des Nutzers unterstützt.

Ähnlich kann man prüfen, ob der Nutzer die Inhalte auch versteht:

$$\begin{aligned} &\forall multimedia, user, lang\ Understandable(multimedia) \leftarrow \\ &\quad multimedia : MultimediaContent[language \twoheadrightarrow lang] \wedge \\ &\quad user : User[language \twoheadrightarrow lang]. \end{aligned} \tag{4.8}$$

Berücksichtigung von Nutzeraufgaben Mit den folgenden Regeln wird gezeigt, wie aufgabenrelevante Informationen ermittelt werden können. Zunächst wird das Prädikat *Ongoing* für die momentan aktiven Aufgaben eingeführt:[1]

$$\begin{aligned} &\forall task\, Ongoing(task) \leftarrow \\ &\quad task : Task[currentStatus \twoheadrightarrow \text{“}Ongoing''].\end{aligned} \tag{4.9}$$

Ist nun beispielsweise die Aufgabe aktiv, mit einer bestimmten Person zu sprechen, so sind Informationen zu der betreffenden Person relevant:

$$\begin{aligned} &\forall talkto, person\, Relevant(person) \leftarrow \\ &\quad Ongoing(talkto : TalkTo[whom \twoheadrightarrow person : Person]).\end{aligned} \tag{4.10}$$

4.2 Inferenz und Reasoning mit Produktionssystemen

Produktionssysteme zeichnen sich durch eine spezielle Form von Regeln aus, sogenannte *Produktionen*. Produktionsregeln bestehen aus einer linken Seite, dem IF-Teil (*antecedent*), und einer rechten Seite, dem THEN-Teil (*consequent*). Die linke Seite besteht dabei aus einer Liste von *Patterns*, die rechte Seite aus einer Liste von *Actions*. Wichtig ist hierbei, dass es sich bei diesen Aktionen nicht nur um eine intensionale Erweiterung der Fakten, sondern auch um Seiteneffekte handeln kann, z.B. für die Interaktion mit dem Nutzer. Eine zentrale Rolle bei Produktionssystemen spielt die Konfliktlösung, also die Auswahl der jeweils besten Regel für den Fall, dass mehrere Regeln gleichzeitig anwendbar sind. Für die Anwendung im Aufgabenmanagement ermöglicht es die Konfliktlösung, zu einem bestimmten Zeitpunkt die jeweils beste Aufgabe zur Ausführung vorzuschlagen. Ein weiterer Vorteil der Produktionssysteme für das Aufgabenmanagement ist ihre ad-hoc-Modifizierbarkeit, das heißt, Fakten und Regeln können dynamisch ergänzt werden.

Für eine umfassende Einführung in Produktionssysteme sei auf [GR97], [Bib93] und [Jac90] verwiesen.

Im folgenden wird ein konkretes Produktionssystem, CLIPS, vorgestellt. Es wird in der Systemarchitektur als Inferenzsystem für das Task-Management zum Einsatz kommen.

4.2.1 Einführung in CLIPS

CLIPS ist ein Produktionssystem, das aufgrund seiner Funktionalität, seiner Möglichkeiten zur Wissensrepräsentation, Erweiterbarkeit und Portabilität weite Verbreitung im akademischen und industriellen Bereich gefunden hat. In dieser Arbeit kann keine komplette Einführung in CLIPS gegeben werden, hierfür sein auf [Gia98b]

[1] Wann eine Aufgabe aktiviert wird, bestimmen Produktionsregeln, die im folgenden Abschnitt erläutert werden.

und [Gia98a] verwiesen. An dieser Stelle sollen lediglich grundlegende Prinzipien der Arbeit mit CLIPS vorgestellt werden.

In CLIPS gibt es vier Wege, Wissen zu repräsentieren:

- *Fakten*, zur Darstellung von deklarativem Wissen
- *Funktionen*, zur Darstellung von prozeduralem Wissen,
- *Objektorientierte Programmierung*, ebenfalls zur Darstellung von deklarativem und prozeduralem Wissen, aber unter Verwendung der objektorientierten Konzepte von Klassen, Abstraktion, Kapselung, Vererbung und Polymorphismus,
- *Regeln*, zur Darstellung von hauptsächlich heuristischem Wissen basierend auf Erfahrungen.

Diese vier Techniken werden im folgenden vorgestellt.

Fakten Ein Fakt besteht aus einem oder mehreren Feldern, eingeschlossen durch runde Klammern. Die wichtigsten Kommandos, um Faktenwissen zu manipulieren, sind:

- **assert** fügt einen Fakt in die Faktenliste ein, z.B.

  ```
  (assert (task talk-to boss))
  ```

- **retract** löscht einen Fakt aus der Faktenliste.
- **deffacts** dient der Definition von “a-priori”-Fakten, die bei einem Rücksetzen (reset) initialisiert werden. Die Definition hat die Form

  ```
  (deffacts fact_name ‘‘optional_comment’’ (<fact>))
  ```

- **undeffacts** löscht über deffacts erstellte Faktendefinitionen.
- **clear** löscht die Faktenliste.
- **reset** löscht die Faktenliste und initialisiert die mit deffact definierten Fakten.

Funktionen Über das Kommando **deffunction** können eigene Funktionen definiert werden. Die Definition hat dabei die folgende Form:

```
(deffunction function_name "optional_comment"
  (?arg_1 ... ?arg_n)
  (action_1 ... action_n))
```

Dabei bezeichnen die `arg_i` die Funktionsparameter und die `action_i` auszuführende Aktionen.

Objektorientierte Programmierung Im Gegensatz zu F-Logic beruht CLIPS nicht auf einem Frame-basierten oder objektorientierten Datenmodell. Mit COOL (CLIPS Object-Oriented Language) sind jedoch objektorientierte Konzepte in CLIPS eingeführt worden. COOL unterstützt Mehrfachvererbung und abstrakte Klassen. Die wichtigesten Kommandos für die objektorientierte Programmierung sind:

- **defclass** dient der Definition von Klassen und Klassenhierarchien. Seine einfachste Form sieht so aus:

  ```
  (defclass class_name (is-a direct_superclass_name))
  ```

- **make-instance** erzeugt Instanzen zu einer Klasse.

  ```
  (make-instance [<instance_name>] of <class_name>)
  ```

- **definstances** definiert Instanzen zu einer Klasse, die nach jedem (reset) Kommando initialisiert werden.

Regeln Regeln bestehen aus einem Regelkopf, einer Menge von *Patterns* (Mustern), und einer Menge von *Actions* (Aktionen). Regeln werden mit *defrule* definiert. Die Definition einer Regel hat die folgende Form:

```
(defrule rule_name "optional_comment"
  (pattern_1) ... (pattern_m)
  =>
  (action_1) ... (action_n))
```

Regeln bestehen also aus Listen von Mustern `(pattern_i)` auf der linken Seite (LHS) und einer Liste von Aktionen `(action_i)` auf der rechten Seite (RHS), die ausgeführt werden, wenn die Regel "feuert". CLIPS versucht über einen Pattern Matching Algorithmus, den *Rete* Algorithmus [For82], zu den Mustern der Regeln passende Fakten in der Faktenliste zu finden. Wenn alle Muster einer Regel mit Fakten der Faktenliste übereinstimmen, wird die Regel aktiviert und bekommt einen Platz auf der *Agenda*. Die Agenda ist eine Menge von *Activations*, also aktivierten Regeln. Auf der Agenda können sich keine, eine oder mehrere aktivierte Regeln befinden. In einem Schritt wird jeweils genau eine aktivierte Regel zur Ausführung ausgewählt (eine Regel "feuert"). Sind keine aktivierten Regeln auf der Agenda wird die Ausführung des Programms angehalten. Sind mehrere aktivierte Regeln auf der Agenda, so bestimmt CLIPS automatisch welche Regel feuert. Hierzu werden die Regeln nach ihrer *Salience* (Priorität) sortiert. Wichtig bei dieser Sortierung ist die Konfliktlösung. Hier liegen die Stärken des CLIPS Produktionssystems: CLIPS verfügt über sieben verschiedene Konfliktlösungsstrategien. Nach ihrer Ausführung wird die Regel von der Agenda gelöscht, um ein erneutes Feuern zu verhindern. Dieser Vorgang nennt sich *Refraction*.

4.2.2 Konkrete Regeln

An dieser Stelle soll nun eine konkrete Regel für das im Szenario der Einleitung beschriebene Beispiel eine Produktionsregel angegeben werden: Hat sich der Nutzer die Aufgabe definiert, dass er eine bestimmte Person sprechen möchte, so will er daran erinnert werden, sobald diese Person in seiner Nähe ist.

Die dazugehörige Produktionsregel lautet:

```
(defrule remind-talk-to "Remind me to talk to someone"
        (talk-to ?person) (near ?person)
        =>
        (printout t "Talk to " ?person))
```

Als Aufgabe hat der Nutzer definiert, eine bestimmte Person zu sprechen:

```
(talk-to <Person>)
```

Ist diese Person nun in der Nähe,

```
(near <Person>)
```

so wird die Regel aktiviert. Als Ergebnis der Anwendung der Produktionsregel wird ausgegeben:

```
"Talk to <Person>"
```

Diese Regel ließe sich auch leicht in F-Logic aufschreiben. Für ein “echtes” Taskmanagement mit mehreren parallelen Aufgaben und mehreren in einer Situation anwendbaren Regeln sind jedoch Produktionssysteme aus den oben genannten Gründen besser geeignet.

4.3 Zusammenfassung

In diesem Kapitel ist gezeigt worden, wie regelbasiserte Inferenzsysteme als Basis für intelligente Assistenzsysteme genutzt werden können. Es ist auch deutlich geworden, dass nicht ein einziges Inferenzsystem sämtlichen Anforderungen intelligenter Assistenz genügen kann.

So eignet sich die deduktive Anfragesprache F-Logic zum Bestimmen situationrelevanter Inhalte. Dabei kann unmittelbar die wissensbasierte Darstellung des RDF genutzt werden. Dagegen eignen sich Produktionssysteme wie CLIPS hervorragend für das Management von Nutzeraufgaben.

In der im nächsten Kapitel vorgestellten Systemarchitektur werden diese Inferenzsysteme eine wichtige Rolle spielen.

Kapitel 5

Beschreibung der Systemarchitektur

In diesem Kapitel wird nun eine Systemarchitektur für den situationsgesteuerten mobilen Zugriff auf Digitale Bibliotheken vorgestellt, die den in Kapitel 2 erläuterten Anforderungen genügt.

Nach einem Gesamtüberblick werden die einzelnen Komponenten der Architektur im Detail vorgestellt.

Die Schnittstellen der Komponenten werden als Java-Interfaces beschrieben. Natürlich lässt sich die Implementierung auch in anderen Programmiersprachen realisieren.

5.1 Gesamtüberblick

Die Systemarchitektur setzt sich aus folgenden Komponenten zusammen:

- *Context Manager* für die Erfassung von Informationen über die Situation des Nutzers und das Propagieren von Kontextänderungen
- *Environment Monitor* für die Überwachung der physischen Umgebung des Nutzers
- *Task Manager* für die Verwaltung der Aufgaben des Nutzers
- *Profile Manager* für das Management von Nutzer- und Geräteprofilen
- *Query Manager* für die Anfragebearbeitung
- *Retrieval Manager* für das effiziente Bereitstellen von Inhalten
- *Dialog Manager* für die Darstellung von Inhalten und Interaktion mit dem Nutzer

Abbildung 5.1 stellt die Systemarchitektur im Gesamtüberblick dar. Es handelt sich um eine verteilte Architektur, die im folgenden erläutert wird.

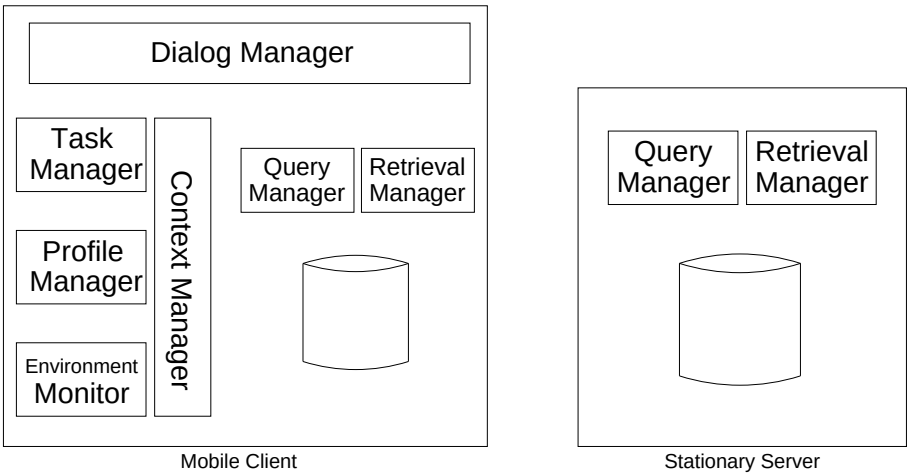

Abbildung 5.1: Komponenten der Systemarchitekur

5.2 Verteilte Architektur

Die beschriebene Systemarchitektur stellt eine verteilte Architektur mit mobilem Endgerät und stationärem Server dar.

Für die Kommunikation zwischen mobilem Endgerät und stationärem Server wird eine drahtlose Netzwerkverbindung vorausgesetzt. Die Netzwerkverbindung kann, wie in mobilen Szenarien üblich, zeitweilig unterbrochen sein. Ziel ist es, dies für den Nutzer völlig transparent zu gestalten, d.h. trotz Netzwerkunterbrechungen eine hohe Verfügbarkeit und Aktualität der Daten zu gewährleisten.

Aus diesem Grund ist die Datenhaltung sowie die Anfrage- und Retrievalbearbeitung verteilt realisiert. Damit handelt es sich aus Sicht der Datenbankarchitektur nicht um eine klassische Client-Server-Architektur wie beispielsweise in [HS99] beschrieben, in der die Funktionaltäten auf Server und Client aufgeteilt sind (beispielsweise Anfrage-Client und Anfrage-Server oder Objekt-Client und Objekt-Server). Es handelt sich vielmehr um eine erweiterte, verteilte Datenbankarchitektur, bei der die Daten — wie weiter unten eingehend beschrieben — repliziert vorliegen. Trotz dessen wird im folgenden das mobile Endgerät aufgrund seiner besonderen Rolle als Client bezeichnet.

Verteilte Datenhaltung Für die verteilte Datenhaltung werden hier zwei zentrale Annahmen getroffen:

1. Auf dem Server wird der gesamte Datenbestand gehalten. Die Speicherkapazität auf dem Client dagegen ist beschränkt. Das bedeutet es kann nur eine Teilmenge der Daten auf Client gespeichert werden. [1]

[1] Es muss sich natürlich nicht um eine echte Teilmenge handeln, d.h. es können unter Umständen

2. Änderungen am Datenbestand werden nur auf dem Server vorgenommen. Auf dem mobilen Endgerät findet ausschließlich lesender Zugriff statt.

Verteilte Anfrage- und Retrievalbearbeitung Auf der Basis der verteilten Datenhaltung können Anfragen und Retrieval wahlweise auf dem lokalen Datenbestand des Clients und auf dem vollständigen Datenbestand des Servers bearbeitet werden. Die konkreten Verfahren werden in den Abschnitten 5.4 (Query Manager) und 5.5 (Retrieval Manager) beschrieben.

5.3 Context Manager

Aufgabe des Context Manager ist die ständige Überwachung der Nutzersituation. Änderungen der Situation müssen erkannt und an andere Systemkomponenten weitergeleitet werden. Die Kommunikation von Kontextänderungen beruht auf den folgenden Konzepten:

- *ContextComponent* bezeichnet eine Einheit, die für die Überwachung eines bestimmten Situationsanteils (z.B. physische Umgebung) zuständig ist.
- *ContextEvent* bezeichnet das Ereignis einer Situationsänderung.
- *ContextEventListener* bezeichnet eine Komponente, welche *ContextEvents*, also Informationen über Situationsänderungen, empfängt und so kontextsensitive Dienste bereitstellen kann.

Der Context Manager verfügt über folgende Schnittstellen:

- `void registerContextComponent(ContextComponent c)` zum Registrieren einer *ContextComponent*,
- `void registerContextEventListener(ContextEventListener l)` zum Registrieren von einem *ContextEventListener*,
- `void contextChanged(ContextEvent e)` wird von den *ContextComponents* aufgerufen, um über Kontextänderungen zu informieren,
- `String getContext()` liefert eine komplette Beschreibung der momentanen Situation entsprechend des Situationsmodells in DAML. Dazu werden die Kontextinformationen der einzelnen *ContextComponents* aggregiert, im einfachsten Fall konkateniert.

Konkrete *ContextComponents* sind entsprechend des Situationsmodells: Der Environment Monitor, der Profile Manager und der Task Manager. *ContextEventListeners*, also Komponenten die kontextsensitive Dienste bereitstellen, sind der Task Manager und der Dialog Manager.

alle Daten auf dem Client gehalten werden. Auch eine leere Menge, also eine ausschließlich serverseitige Datenhaltung ist möglich.

Natürlich können weitere *ContextEventListeners* und *ContextComponents* integriert werden. Zum Beispiel könnte als weitere *ContextComponent* ein globaler Kontextmanager eingebunden werden, über den nicht nur lokale Kontextinformationen, sondern z.B. auch die Position anderer Nutzer bezogen werden kann.

Abbildung 5.2 zeigt den Daten- und Kontrollfluss bei Situationsänderungen. Es ist dargestellt, wie *ContextEvents* — beispielsweise eine durch den Environ-

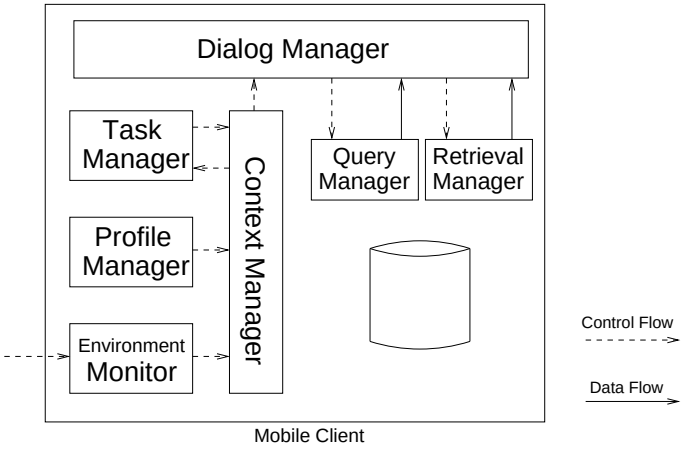

Abbildung 5.2: Daten- und Kontrollfluss bei der Situationsänderungen

ment Monitor neu erfasste Position — von den *ContextComponents* an den Context Manager und über diesen an die *ContextEventListeners* (Task Manager und Dialog Manager) propagiert werden. Der Dialog Manager beispielsweise nutzt die Situationsbeschreibung dann, um über den Query Manager situationsrelevante Inhalte zu bestimmen, diese über den Retrieval Manager anzuforden und schließlich darzustellen. Der Anfrage- und Retrievalprozess wird weiter unten beschrieben.

Die drei verwendeten *ContextComponents* werden im folgenden vorgestellt.

5.3.1 Environment Monitor

Der Environment Monitor dient der Überwachung der physischen Umgebung des Nutzers. Dazu gehören insbesondere die Erfassung von Position und Objekten in der Nähe des Nutzers. Optional können weitere Größen erfasst werden. Denkbar wäre zum Beispiel die Überwachung des Geräuschpegels, um die Lautstärke beim Abspielen von Audiodaten zu regulieren.

Der Environment Monitor sorgt für eine Abstraktion der Sensordaten der konkreten verwendeten Sensoren. Das bedeutet, die Daten werden so aufbereitet, dass sie dem in 3.5.5.1 definierten Modell genügen. Für die Abstraktion ist insbesondere die Interpretation der Sensordaten von Bedeutung. Eine einfache Form der Interpretation wäre zum Beispiel die Zuordnung eines Raumes zu einer Infrarot-Baken-ID. Unter Umständen können auch die Daten verschiedener Sensoren aggregiert

werden, wie z.B. im SensorFusion-System [Kor01].

Die Abstraktion der Daten ermöglicht, dass beliebige Sensoren in das System integriert werden können.

5.3.2 Task Manager

Der Task Manager dient der Verwaltung von Nutzeraufgaben. Dabei geht es jedoch nicht nur um die Erfassung der Aufgaben, sondern vor allem um die Auswahl von in der Situation sinnvollen Aufgabenschritten. Im Kern des Task Managers agiert dazu ein CLIPS-kompatibles Produktionssystem, wie in 4.2 beschrieben. Als Aufgabenmodell verwendet der Task Manager das Prozessmodell von DAML-S.

Der Task Manager verfügt über folgende Schnittstelle:

- `void addTask(String task)` erlaubt das Einfügen einer neuen Aufgabe. Die Aufgabenbeschreibung erfolgt in DAML. Der Aufgabenstatus neuer Aufgaben ist *Ready*.

- `void deleteTask(String identifier)` dient dem Löschen einer Aufgabe. Als Parameter wird die URI der Aufgaben erwartet.

- `void abortTask(String identifier)` bricht aktive Aufgaben ab. Der Aufgabenstatus wird von *Ongoing* auf *Aborted* gesetzt.

- `void completedTask(String identifier)` markiert aktive Aufgaben als erledigt. Der Aufgabenstatus wird von *Ongoing* auf *Completed* gesetzt.

Zusätzlich gibt es folgende Schnittstelle, die eine direkte Interaktion mit dem Produktionssystem erlaubt:

- `void loadDAMLString(String daml)` Diese Methode dient dem Einfügen von Fakten in DAML-Syntax in das Produktionssystem.

- `void executeCommand(String cmd)` Mit dieser Methode können beliebige CLIPS-Kommandos an das Produktionssystem übergeben werden. Dies ermöglicht eine flexible Steuerung des Produktionssystems.

Der Task Manager wird als *ContextEventListener* über Situationsänderungen informiert. Ermittelt das Produktionssystem, dass die Ausführung einer Aufgabe in der momentanen Situation sinnvoll ist, so wird diese zunächst dem Nutzer vorgeschlagen. Der Nutzer kann sich dann entscheiden, ob er die Aufgabe ausführen will oder nicht. (Diese Interaktion läuft über den Dialog Manager.) Entscheidet sich der Nutzer für die Ausführung, so wird der Aufgabenzustand auf *Ongoing* gesetzt, entscheidet er sich dagegen, so bleibt er auf *Ready*.

Transformation der Aufgabendefinition von DAML nach CLIPS Da die Aufgabenbeschreibung als Komponente des Situationsmodells in DAML erfolgt, ist zunächst eine Transformation von DAML nach CLIPS-Datenmodell nötig. Hier wird die in [Kop01] vorgeschlagene Transformation verwendet: Die Tripel des zugrundeliegenden RDF-Datenmodells

```
(<subject>, <predicate>, <object>)
```

werden übersetzt in CLIPS-Fakten der Form

```
(assert (PropertyValue <predicate> <object> <subject>))
```

5.3.3 Profile Manager

Der Profile Manager dient der Erfassung und Verwaltung von Nutzer- und Geräteprofilen entsprechend des Modells aus 3.5.5.3 und 3.5.5.4. Nutzer- und Geräteprofile sind Bestandteil des Situationsmodells. Der Profile Manager agiert deshalb als *ContextComponent* und informiert den Context Manager über Änderungen der Profile. Die Profile werden als Mengen von (Property, Value)-Paaren verwaltet.

Wie Nutzer- und Geräteprofile erfasst werden, wird nun erläutert.

Erfassung der Nutzerinformationen Das Nutzerprofil kann dynamisch verändert werden. Die Schnittstelle bietet hierfür folgende Methoden:

- `void addProperty(String property, String value)` fügt dem Nutzerprofil das Paar (`property` , `value`) hinzu.
- `void deleteProperty(String property, String value)` entfernt das Paar (`property` , `value`) aus dem Nutzerprofil.

Erfassung der Geräteinformationen Geräteeigenschaften werden einmalig zum Zeitpunkt der Initialisierung bestimmt und sind nicht veränderlich. Die Erfassung der Eigenschaften kann dabei durch automatisches Auslesen von Systemeigenschaften oder durch Lesen aus einer Konfigurationsdatei geschehen. Der Inhalt dieser Datei kann sich auf die Angabe eines konkreten Gerätetyps beschränken, sofern die Eigenschaften für diesen Gerätetyp vordefiniert sind.

5.4 Query Manager

Der Query Manager ist für die Bearbeitung von Anfragen zuständig. Der Prozess der Anfragebearbeitung findet verteilt über Client und Server statt. Er ist in Abbildung 5.3 exemplarisch für Anfragen an Metadaten dargestellt. Die Anfrage wird vom Query Manager auf der Client-Seite entgegengenommen. Zunächst wird versucht, die Anfrage an den Server weiterzuleiten und dort zu bearbeiten. Nur wenn

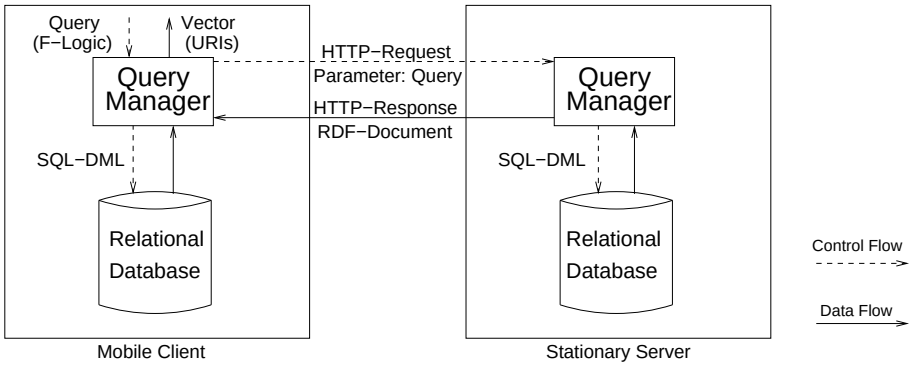

Abbildung 5.3: Daten- und Kontrollfluss beim Prozess der Anfragebearbeitung

die Netzwerkverbindung zum Server unterbrochen ist, wird die Anfrage auf dem lokalen Datenbestand des Client bearbeitet.

Die Kommunikation zwischen Client und Server erfolgt über das HTTP-Protokoll. Die Anfrageparameter werden als HTTP-Parameter kodiert. Die Ergebnismengen werden vom Server in einem RDF-Dokument gekapselt zurückgeliefert.

Ist die Anfrage frei von Negationen, Differenzen, etc. so wird das Ergebnis der Anfragebearbeitung auf dem Client stets eine Teilmenge vom Ergebnis der Anfragebearbeitung auf dem Server sein. Das bedeutet, bei unterbrochener Netzwerkverbindung umfasst das Anfrageergebnis unter Umständen nicht alle relevanten Informationsobjekte.

Der Query Manager unterstützt folgende Anfragetypen:

- Anfragen an Metadaten
- Inferenzanfragen
- Information Retrieval Anfragen

Diese werden im folgenden erläutert.

5.4.1 Anfragen an Metadaten

Für Anfragen an Metadaten wird als zugrundeliegendes Datenmodell das Frame-basierte Datenmodell von RDF(S) angenommen und entsprechend als Anfragesprache F-Logic verwendet.

Dabei wird gegenwärtig eine bestimmte Form von Anfragen unterstützt:

$\forall o \leftarrow o : c[m_1 \rightarrow r_1; ...; m_n \rightarrow r_n]$

Diese Anfrage liefert als Ergebnis alle Instanzen des Typs c deren Properties m_i bestimmte Werte r_i annehmen.

Dieser Anfragetyp ist in den allermeisten Fällen ausreichend und bietet zwei Vorteile: Bei der F-Logic-Anfrage handelt es sich um eine Regel mit leerem Regel-

kopf (siehe 4.1.1). Bei der Anfragebearbeitung kommt es also zu keiner intensionalen Erweiterung der Faktenbasis. Das bedeutet, dass die Anfrage ohne den Einsatz eines Inferenzsystems bearbeitet werden kann. Zum anderen lässt sich die Anfrage leicht in entsprechendes SQL übersetzen und kann so auf den in relationaler Form gespeicherten Daten bearbeitet werden.

Komplexere, regelbasierte Anfragen werden durch die Nutzung eines Inferenzsystems, wie SiLRI, und eine volle Unterstützung von F-Logic, möglich und in 5.4.3 beschrieben.

Die Schnittstelle des Query Managers für Anfragen an Metadaten ist:

- `Vector metaQuery(String query)` erlaubt Anfragen an Metadaten. Als Ergebnis wird die Menge der URIs aller Objekte, die die Anfrage `query` vom oben beschriebenen Typ erfüllen, zurückgeliefert.

5.4.2 Information-Retrieval-Anfragen

Information-Retrieval-Anfragen bezeichnen inhaltsbasierte Anfragen auf semi- und unstrukturierten Dokumenten. An dieser Stelle sollen keine konkreten Information-Retrieval-Verfahren oder Anfragesprachen vorgestellt werden. Hierfür sei auf [RNBY99] verwiesen. Der Query-Manager stellt eine universelle Schnittstelle für IR-Anfragen bereit:

- `Vector IRQuery(String query)` erlaubt inhaltsbasierte Anfragen auf semi- und unstrukturierten Dokumenten. Als Ergebnis wird die Menge der URIs aller Objekte, die die Anfrage `query` erfüllen, zurückgeliefert.

Die Auswahl einer konkreten Anfragesprache bleibt damit offen.

Sinnvoll ist jedoch die Verwendung einer standardierten, systemunabhängigen Anfragesprache. Denkbar ist auch eine Kombination von Information Retrieval Anfragen mit Anfragen an Metadaten. Dies soll jedoch nicht Gegenstand dieser Arbeit sein. Daher sei an dieser Stelle nur auf entsprechende Arbeiten zu IRQL, der Information Retrieval Query Language, verwiesen, bei deren Entwicklung eine enge Verknüpfung von Datenbankanfragen mit Information-Retrieval-Anfragen angestrebt wird [Tit99], [HP99].

Zu erwähnen ist auch der Ansatz des GETESS-Projektes [KBB+01]: Hier wird eine inhaltsbasierte Suche auf der Basis von Ontologien angestrebt. Dazu werden Abstracts zu den Dokumenten generiert und wissensbasiert dargestellt. Zur wissensbasierten Inhaltserschließung der Dokumente sind aufwändige linguistische Verfahren und komplexe Ontologien notwendig. Die Anwendung bleibt dadurch auf sehr eingeschränkte Domänen beschränkt. Die in dieser Arbeit vorgestellte Ontologie ist dagegen nur zur Erfassung bestimmter Metadaten geeignet, nicht jedoch zur automatischen Abbildung kompletter Dokumentinhalte.

5.4.3 Inferenzanfragen

Bei den Inferenzanfragen handelt es sich um den wichtigsten Anfragetyp. Dieser ermöglicht die eigentliche situationsgesteuerte Auswahl der Inhalte der Digitalen Bibliothek. Ziel der Inferenzanfragen ist es, allein aus der komplexen Situationsbeschreibung des Nutzers zu schlussfolgern, welche Inhalte der Bibliothek in der Situation relevant sind. Das Ermitteln der relevanten Inhalte erfolgt regelbasiert auf der Basis eines Frame-Logic-basierten Inferenzsystem, wie es in 4.1.1 vorgestellt wurde.

Die Schnittstelle des Query Managers für Inferenzanfragen lautet:

- `Vector inferenceQuery(String situation)` Als Parameter wird eine Situationsbeschreibung in DAML entsprechend des Situationsmodells aus 3.5.5 erwartet. Als Ergebnis wird die Menge der URIs aller Objekte, die in der Situation relevant sind, zurückgeliefert.

Der Inferenzprozess wurde bereits in 4.1 beschrieben. Die wesentlichsten Aspekte sind noch einmal in Abbildung 5.4 dargestellt. Als Faktenbasis dient die Be-

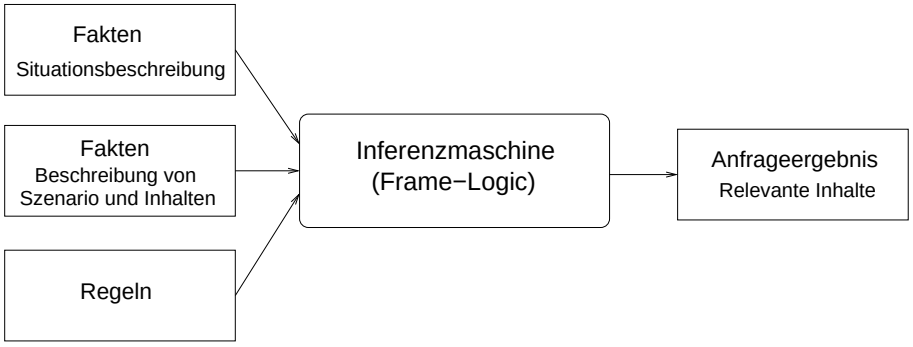

Abbildung 5.4: Bearbeitung von Inferenzanfragen

schreibung der Inhalte, die Beschreibung des Szenarios, und - als Anfrageparameter - die Beschreibung der Situation. Regeln werden verwendet, um aus diesen Beschreibungen neue Fakten abzuleiten. Konkrete Beispielregeln wurden bereits in 4.1.2 vorgestellt. Eine besondere Rolle spielen die *Queries* - spezielle Regeln ohne Regelkopf. Sie bestimmen, welche Objekte als Anfrageergebnis zurückgeliefert werden.

5.5 Retrieval Manager

Aufgabe des Retrieval Managers ist die effiziente Bereitstellung von angeforderten Informationsobjekten.

Die Effizienz beim Retrieval der Daten wird vor allem durch folgende zwei Techniken gewährleistet:

- *Caching und Replikation:* Die Informationsobjekte können auf dem mobilen Endgerät für den Nutzer transparent gecacht werden oder auf expliziten Nutzerwunsch repliziert werden. Damit wird angestrebt, auf angeforderte Objekte möglichst lokalen Zugriff zu haben.

- *Level of Detail:* Die Informationsobjekte können in unterschiedlichen Granularitätsstufen existieren. Es werden jeweils nur so viel Informationen bereitgestellt, wie momentan nötig.

Diese Techniken werden im folgenden vorgestellt.

5.5.1 Caching und Replikation

Der Vorteil des Einsatzes von Caching und Replikation besteht in der erhöhten Effizienz durch lokalen Zugriff sowie in der verbesserten Verfügbarkeit der Daten, z.B. bei einer Unterbrechung der Netzwerkverbindung.

Zunächst soll der Unterschied zwischen Caching und Replikation in dieser Systemarchitektur deutlich gemacht werden. In beiden Fällen werden Informationsobjekte lokal auf dem Client gespeichert. Bei der Replikation geschieht dies initiiert durch den Nutzer, der z.B. ein Objekt für spätere Verfügbarkeit auf seinem Gerät halten möchte. Beim Caching dagegen erfolgt das lokale Speichern automatisch und für den Nutzer transparent. Entsprechend werden beim Caching Objekte automatisch wieder verdrängt, während bei der Replikation lokale Objekte permanent gespeichert bleiben und erst durch den Nutzer initiiert gelöscht werden.

Im folgenden werden die Begriffe Fragmentierung und Allokation für den Retrieval Manager geklärt. Daraufhin wird erläutert, wie Konsistenz und Aktualität der Daten gewährleistet werden. Danach wird eine konkrete Ersetzungsstrategie für das Caching vorgeschlagen.

Fragmentierung und Allokation Unter Fragmentierung versteht man die Aufteilung von Datenbankobjekten in mehrere Teile, die verteilt gespeichert werden sollen. Werden Fragmente mehrfach gespeichert, so spricht man von replizierten Fragmenten. Die Zuordnung von Fragmenten zu konkreten Rechnern, hier also zum Server und mobilen Client, wird als Allokation bezeichnet. Die Fragmentierung erfolgt hier auf der Ebene der Informationsobjekte des weiter unten beschriebenen Level-Of-Detail-Konzeptes. Das bedeutet, dass jeweils *komplette* Informationsobjekte repliziert bzw. gecacht werden. Dies impliziert auch, dass Client und Server, wie bei verteilter Datenhaltung üblich, über dasselbe Datenbankschema verfügen. Die Annahme, dass client-seitig eine Teilmenge des gesamten Datenbestandes auf dem Server gespeichert werden kann, bedeutet, dass zu jedem Zeitpunkt alle Informationsobjekte serverseitig allokiert sind. Die Allokation zum Client dagegen wird bestimmt durch den expliziten Nutzerwunsch zur Replikation bzw. durch die Cache-Ersetzungsstrategie.

Konsistenz und Aktualität Aufgrund der Annahme, dass Änderungen am Datenbestand nur serverseitig erlaubt sind, ist der Einsatz einer Konfliktlösungsstrategie zur Konsistenzsicherung nicht notwendig. Es muss lediglich die Aktualität der Daten auf dem Client gewährleistet werden. Dies ist aber auch nur dann notwendig, wenn auf dem Server wirklich Änderungen vorgenommen werden. In einigen Fällen kann vorausgesetzt werden, dass sich der Datenbestand überhaupt nicht ändert, z.B. bei Informationssystemen, deren Einsatz auf wenige Tage beschränkt ist, wie es bei Ausstellungen oder Konferenzen üblich ist. Sind jedoch Änderungen zugelassen, so hängt die Auswahl eines geeigneten Verfahrens zur Aktualitätssicherung von verschiedenen Parametern der Änderungen ab. Hierzu gehören z.B. die Update-Häufigkeit und das Verhältnis der geänderten Daten zum Gesamtdatenbestand.

Für den Retrieval-Manager wird folgendes *optionales* Verfahren zur Sicherung der Aktualität der Daten vorgeschlagen: Wird ein Objekt angefordert, so wird zunächst überprüft, ob dieses Objekt bereits repliziert auf dem Client vorliegt. Ist dies nicht der Fall, so wird das Objekt vom Server angefordert und lokal gespeichert. Ist es bereits lokal vorhanden, so wird beim Server der Zeitpunkt der letzten Änderung erfragt und dieser mit dem Zeitpunkt der Einlagerung im lokalen Datenbestand verglichen. Ist die letzte Änderung nach dem Einlagern im lokalen Datenbetand erfolgt, so wird das Objekt erneut vom Server angefordert und lokal gespeichert. Dieses Verfahren gewährleistet eine starke Konsistenz.

Der komplette Retrieval-Prozess unter Verwendung der eben beschriebenen Strategie zur Aktualitätssicherung wird in 5.5.3 erläutert.

Werden auf dem Server keine Änderungen vorgenommen, so kann auf den Einsatz dieses Verfahrens verzichtet werden. Damit entfällt die Anfrage über den Zeitpunkt der letzten Änderung und der Vergleich mit dem Zeitpunkt der Einlagerung im Cache.

Ersetzungsstrategie Die Notwendigkeit einer Cache-Ersetzungsstrategie ergibt sich aus der Tatsache, dass der Cache aufgrund von Kapazitätsbeschränkungen zumeist nicht alle Informationsobjekte aufnehmen kann.

Eine Ersetzungsstrategie wird in zwei Phasen unterteilt: Zunächst werden die gecachten Objekte nach einem oder mehreren Schlüsseln sortiert. Dann werden aus dem Cache solange Objekte entfernt, bis ein bestimmtes Kriterium erfüllt ist. Beispiele für ein solches Kriterium könnten sein:

- Der Speicherplatz des Cache soll ausreichen, um das nächste Objekt aufzunehmen.
- Der verfügbare Platz im Cache soll eine bestimmte Grenze nicht unterschreiten.
- Die Anzahl der Objekte im Cache soll eine bestimmte Grenze nicht überschreiten.

Als Kriterium wird hier die maximale Anzahl der Objekte im Cache vorgeschlagen. Der Grund für die Auswahl dieses Kriteriums besteht darin, dass es sehr einfach zu überprüfen ist. Dagegen ist es z.B. schwierig, den vorhandenen Speicherplatz auf dem mobilen Gerät zu ermitteln. Auch die tatsächliche Größe eines Objektes ist nicht immer leicht zu bestimmen. So ist es z.B. nicht trivial, die tatsächliche Größe eines Frames, der in relationaler Form in einer Datenbank gespeichert ist, zu ermitteln. Die maximale Anzahl der im Cache speicherbaren Objekte kann heuristisch in Abhängigkeit vom verwendeten mobilen Endgerät und den Charakteristika der Objekte festgelegt werden.

Als Schlüssel für die Sortierung der Objekte im Cache könnten folgende Parameter berücksichtigt werden:

- der Zeitpunkt des Ablegens des Objektes im Cache,
- der Zeitpunkt der letzten Referenzierung des Objektes,
- die Anzahl der Referenzen auf das Objekt,
- die Größe des Objektes.

Die Sortierung nach einem oder mehreren dieser Schlüssel bestimmt die Ersetzungsstrategie. Einen Überblick über verschiedene Ersetzungsstrategien gibt [Jon99].

Der Retrieval Manager verwendet die FIFO-Strategie. Dabei wird das jeweils älteste Objekt im Cache verdrängt. Die Gründe für die Verwendung dieser Strategie sind folgende: Die FIFO-Strategie ist sehr einfach zu implementieren. Als einziger Parameter wird der Zeitpunkt des Ablegens des Objektes im Cache benötigt. Dieser Parameter ist sehr einfach zu erfassen und wird ohnehin für die Aktualisierung des Caches benötigt, wie oben beschrieben.

In einer konkreten Realisierung der Architektur kann statt FIFO jedoch auch eine andere Ersetzungsstrategie verwendet werden. Hierfür müssten eventuell andere Parameter gesondert erfasst und bewertet werden.

Für weitere Ausführungen zur Problematik der Datenreplikation in mobilen Umgebungen sei auf [LH00] verwiesen.

5.5.2 Level of Detail

Die Idee des Level-of-Detail-Konzeptes (kurz LoD) besteht darin, dass Objekte mit typischerweise multimedialen Inhalten in unterschiedlichen Detaillierungsstufen existieren können. Unter Ausnutzung dieser Levels of Detail kann ein effizienter Zugriff realisiert werden, indem nicht mehr Informationen zur Verfügung gestellt werden als momentan unbedingt notwendig. Weitere Informationen können später bei Bedarf nachgeladen werden, das lokal verfügbare Wissen über das Objekt kann also inkrementell erweitert werden.

Das Modell dieser Architektur kennt drei Granularitätstufen, oder Levels of Detail:

- **Label**: Hierbei handelt es sich um einen menschenverständlichen Bezeichner für das Objekt in Form eines Literals. Das Label wird über die Property `rdfs:Label` für alle RDF-Resources definiert. Im Falle eines Artikels könnte z.B. der Titel als Label vergeben werden.
- **Metadaten**: Hierbei handelt es sich um das Objekt beschreibende, strukturierte Daten. Dies sind im Falle eines Artikel z.B. der Autor und das Erscheinungsjahr, aber auch z.B. ein Abstract.
- **Dokument**: Hierbei handelt es sich um ein multimediales, zumeist unstrukturiertes Dokument, im Falle eines Artikels also z.B. eine Datei im PDF-Format.

Je nach Bedarf können die Daten in einer dieser Detaillierungsstufen vom Retrieval Manager angefordert werden, wie im folgenden erläutert wird.

Die Verwaltung der replizierten Informationsobjekte unterschiedlicher Levels of Detail wird über spezielle Tabellen gesteuert. Für jede LoD-Stufe wird hierfür sowohl client- als auch serverseitig eine eigene Allokationstabelle verwendet:

lod_name(`id` *varchar(100)*, `value` *varchar(100)*, `replica` *int*, `timestmp` *timestamp*)

Der Tabellenname **lod_name** ist dabei ein Bezeichner für die jeweilige LoD-Stufe. Die Funktion der Attribute wird in Tabelle 5.1 erläutert.

Attribut	Erläuterung
`id`	URI des betreffenden Objektes
`value`	Enthält das Objekt selbst oder eine Referenz, über die das Objekt rekonstruiert werden kann.
`replica`	Gibt an, ob das Objekt repliziert oder gecacht vorliegt, in anderen Worten permanent oder transient gespeichert ist.
`timestmp`	Zeitpunkt der Einlagerung bzw. der letzten Änderung des Objektes.

Tabelle 5.1: Allokationstabelle zur Verwaltung replizierter Informationsobjekte

Perspektivisch wäre es denkbar, den Retrieval Manager um weitere Konzepte für den effizienten Zugriff zu erweitern. Insbesondere Datenreduktion, Kompression und Prefetching ließen sich sinnvoll mit den hier verwendeten Konzepten Level of Detail, Replikation und Caching kombinieren. In diesem Zusammenhang sei auch auf die Ergebnisse des MoVi-Projektes hingewiesen, wie z.B. [Kir98], [LH98] und [Lub00].

Im folgenden werden die Schnittstellen des Retrieval Manager und der eigentliche Prozess des Retrievals beschrieben.

5.5.3 Retrieval von Informationsobjekten

Für jede Stufe des LoD-Konzeptes stellt die Schnittstelle des Retrieval Manager eine Methode zum Retrieval bereit:

- `String getLabel(String identifier)` liefert einen String, der das Label des Informationsobjektes enthält.
- `Object getMetaData(String identifier)` liefert ein Java-Objekt, das die Metadaten des Informationsobjektes kapselt.
- `String getDocument(String identifier)` liefert einen Dateinamen als Referenz auf das Informationsobjekt.

Als Parameter wird jeweils der Objekt-Identifier (URI) des betreffenden Objektes erwartet. Der Typ des zurückgegeben Objektes hängt von der jeweiligen LoD-Stufe ab und wird weiter unten beschrieben. Im Fehlerfall wird jeweils `null` zurückgeliefert.

Der Prozess des Retrievals wird im folgenden beschrieben und ist in Abbildung 5.5 exemplarisch für das Retrieval von Metadaten dargestellt.

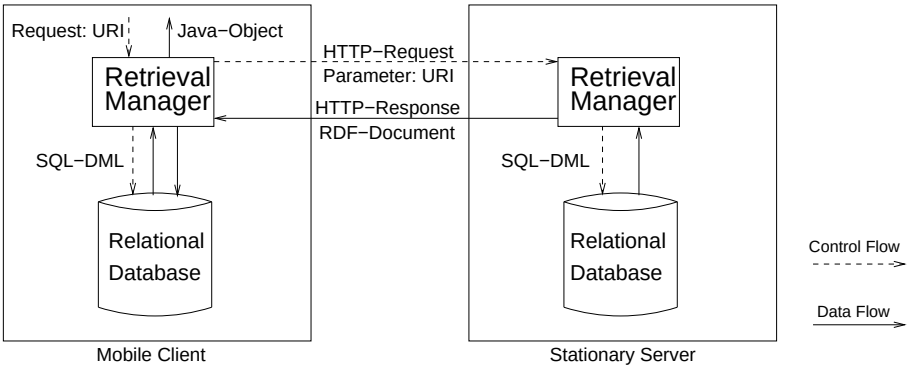

Abbildung 5.5: Daten- und Kontrollfluss beim Retrieval-Prozess

1. Anforderung eines Objektes: Die Anforderung eines Objektes wird vom Retrieval Manager auf der Client-Seite entgegengenommen. Als Parameter wird die URI des Objektes erwartet. Zunächst wird mittels der Allokationstabelle überprüft, ob das Objekt bereits im lokalen Datenbestand vorhanden ist. Falls dies nicht der Fall ist, weiter bei 3.
2. Überprüfung der Aktualität: Ist das Objekt bereits lokal vorhanden, wird beim Retrieval Manager des Servers über einen HTTP-Request der Zeitpunkt der letzten Änderung des Objektes erfragt. Liegt dieser vor dem Zeitpunkt der Einlagerung des Objektes im lokalen Datenbestand, weiter bei Schritt 4.
3. Caching des Objektes: Das Objekt wird über einen HTTP-Request vom Server angefordert, im lokalen Datenbestand gespeichert und in die Allokationstabelle (siehe Tabelle 5.1) aufgenommen. Dabei wird `timestmp` auf den Zeitpunkt der Einlagerung gesetzt und `replica` auf `0` gesetzt, um anzuzeigen, dass das Objekt transient gespeichert ist.

4. Zurückliefern des Objektes: Das Objekt wird aus dem lokalen Datenbestand zurückgeliefert.

In welcher Form die Objekte im Datenbestand gespeichert, zum Transfer serialiert und schließlich zurückgeliefert werden, hängt von der jeweiligen LoD-Stufe ab und wird im folgenden erläutert.

Retrieval der Label Die Label der Informationsobjekte werden in der Datenbank des Client und des Server direkt in der Allokationstabelle (siehe Tabelle 5.1) als `value` gespeichert. Sie werden zwischen Server und Client gekapselt in einem RDF-Dokument über HTTP kommuniziert. Der Retrieval Manager liefert das Label als String zurück.

Retrieval der Metadaten Die DAML-Metadaten werden im Datenbestand des Client und des Servers in relationaler Form gespeichert. Die Abbildung der DAML-Ontologie auf ein relationales Schema wurde bereits in 3.6.2 beschrieben. Die Abbildung einer DAML-Instanz auf eine relationale Form und umgekehrt wurde im selben Abschnitt erläutert. In der Allokationstabelle (siehe Tabelle 5.1) wird der Typ des DAML-Objektes als `value` gespeichert. Anhand des Typs und der Schemainformationen können die DAML-Objekte rekonstruiert werden. Die Metadaten werden zwischen Server und Client als DAML-Dokument kommuniziert.

Der Retrieval Manager liefert die Metadaten in Form von Java-Objekten zurück. Die entsprechende Abbildung wurde bereits in 3.6.3 beschrieben. Eine Abbildung der DAML-Ontologie auf eine Java-Klassenhierarchie wurde ebenfalls in 3.6.1 vorgestellt.

Retrieval der Dokumente Die semi- und unstrukturierten Dokumente werden als Dateien im jeweiligen Dateisystem des Clients und des Servers gehalten. [2] In der Allokationstabelle (siehe Tabelle 5.1) wird der lokale Dateiname als `value` gespeichert. Die Dokumente werden direkt als Dateien zwischen Server und Client über HTTP kommuniziert. Der Retrieval Manager liefert den Dateinamen als Referenz auf die lokale Datei zurück.

5.5.4 Replikation von Objekten

Wie auch für das Retrieval von Objekten, existieren für jede LoD-Stufe eine eigene Schnittstelle für das explizite Replizieren von Objekten.

- `boolean replicateLabel(String identifier)`
- `boolean replicateMetaData(String identifier)`

[2] Von einem Speichern der Dokumente innerhalb der Datenbank z.B. als BLOB oder als Datalink wird abgesehen, da viele DBMS, insbesondere auf mobilen Endgeräten, diese Möglichkeiten nicht bieten.

- `boolean replicateDocument(String identifier)`

Als Parameter wird auch hier die URI (als Objekt-Identifier) des zu replizierenden Objektes erwartet. Zurückgeliefert wird ein boolescher Wert, welcher indiziert, ob das Replizieren erfolgreich ausgeführt wurde. Der Prozess des Replizierens der Objekte ist mit dem des Retrievals identisch bis auf zwei Unterschiede:

- Es wird nicht das Objekt zurückgeliefert, sondern ein boolescher Wert, wie eben erläutert.
- In der Allokationstabelle (siehe Tabelle 5.1) wird das Attribut `replica` auf `1` gesetzt, um anzuzeigen, dass das Dokument permanent gespeichert ist.

Analog zu den Methoden für das Anlegen von Replikaten, gibt es drei Methoden für das Entfernen von Replikaten aus dem lokalen Datenbestand.

- `boolean removeLabel(String identifier)`
- `boolean removeMetaData(String identifier)`
- `boolean removeDocument(String identifier)`

Beim Entfernen eines Replikates wird der entsprechende Eintrag aus der Allokationstabelle entfernt und das Objekt gelöscht.

5.6 Dialog Manager

Der Dialog Managers ist für den Nutzer die Schnittstelle für den Zugriff auf die Inhalte der Digitalen Bibliothek. Die Aufgabe des Dialog Managers ist die Visualisierung von relevanten Inhalten und die Interaktion mit dem Nutzer bei möglichst geringem kognitiven Overhead.

Eine konkrete Oberfläche soll hier nicht spezifiziert werden. Es sollen jedoch wichtige, generische Konzepte hervorgehoben werden, die für den Zugriff auf multimdiale Inhalte in Digitalen Bibliotheken wichtig sind.

Nach [EF00] lassen sich drei Methoden der Informationsgewinnung unterscheiden: Suchen, Navigieren und Stöbern. Es wird nun aufgezeigt, welche Konzepte die Systemarchitektur bereitstellt, die direkt im Dialog Manager verwendet werden können, um diese Methoden der Informationsgewinnung zu unterstützen:

- *Suchen:* Beim Suchen wird im allgemeinen zwischen Anfragen an Metadaten und Anfragen an semi- und unstrukturierte Dokumente unterschieden.

  Der Dialog Manager kann für die Unterstützung dieser Anfragen auf die entsprechenden Schnittstellen des Query Manager zurückgreifen.

- *Navigieren:* Hier werden oftmals Baumstrukturen und Klassifikationsschemata zur Strukturierung der Inhalte verwendet. Das Traversieren dieser Strukturen wird als Navigieren bezeichnet.

Der Dialog Manager kann sich der Schemainformationen der Ontologie bedienen, um einen navigierenden Zugriff auf die Inhalte zu ermöglichen.

- *Stöbern / Browsing:* Diese Form der Informationsgewinnung wird auch als nicht-zielgerichtetes Suchen bezeichnet. Das Browsing entspricht in Hypermedia-Systemen dem Folgen von Hyperlinks.

  Der Dialog Manager kann für ein einfaches, intuitives Browsing hier die Relationen des zu Grunde liegenden Objektmodells ausnutzen. In diesem Sinne lassen sich die Objekte als Knoten und die Relationen als Kanten (Links) eines Hypermedia-Systems betrachten.

Eine besondere Rolle spielt natürlich die proaktive Bereitstellung von Informationen. Im Gegensatz zu den eben beschriebenen Verfahren ist hierfür keine explizite Nutzerinteraktion notwendig, da relevante Dokumente unter Ausnutzug der Kontextinformationen über die Inferenzanfragen des Query Managers bestimmt werden.

Eine weitere wichtige Technik ist die Verwendung von *Widgets*. Widgets dienen der Darstellung von Informationen von einem bestimmten Typ. In Kombination mit der Verwendung von Ontologien hat der Begriff Typ hierbei eine ganz konkrete Bedeutung: Ein Widget dient der Darstellung von Instanzen eines Konzeptes der Ontologie. Ebenso wie die Konzepte der Ontologie können Widgets in einer Hierarchie angeordnet werden, d.h. bestehende Widgets können erweitert und verfeinert werden. So kann es zum Beispiel ein Widget zur Darstellung von Publikationen geben und ein spezielleres Widget zur Darstellung von Artikeln. Die existierenden Widgets werden in einem Widget-Repository verwaltet. Zur Darstellung von Informationsobjekten wird jeweils das speziellste vorhandene und passende Widget aus dem Repository ausgewählt.

5.7 Zusammenfassung

In diesem Kapitel ist eine Systemarchitektur für den situationsgesteuerten mobilen Zugriff auf Digitale Bibliotheken vorgestellt worden.

Ein wichtiger Aspekt dieser Architektur ist die verteilte Datenhaltung und die verteilte Realisierung von Komponenten für Anfragen und Retrieval. Damit wird eine hohe Verfügbarkeit und Effizienz auch bei unterbrochener Netzwerkverbindung ermöglicht.

Weitere Komponenten erfassen die Umgebung des Nutzers, seine Aufgaben und Profile. Eine spezielle Komponente, der Context Manager dient dem Propagieren von Änderungen der Situation.

Die in Kapitel 4 vorgestellten Inferenzsysteme kommen im Task Manager für die Aufgabenverwaltung und im Query Manager für das Bestimmen situationsrelevanter Inhalte zum Einsatz.

Einige Aspekte einer prototypischen Realisierung der Architektur werden im nächsten Kapitel dargestellt.

Kapitel 6

Realisierung der Systemarchitektur

Die im vorherigen Kapitel vorgeschlagene Systemarchitektur ist in ihrer gesamten Breite prototypisch umgesetzt worden. Der Prototyp trägt den Namen *MoBibel* [1]. Einige Aspekte der Implementierung, insbesondere auch verwendete Systeme, werden in diesem Kapitel betrachtet. Dazu wird außerdem auf alternative Systeme, Protokolle und Verfahren hingewiesen, die in einer Realisierung der Systemarchitektur verwendet werden könnten.

Wie bereits dargestellt, ist die Systemarchitektur unabhängig von einer konkreten Anwendungsdomäne. Die Festlegung auf eine bestimmte Domäne ist ausschließlich durch die Verwendung einer konkreten Ontologie und konkreter Inferenzregeln gegeben. Für die prototypische Realisierung ist die in Kapitel 3 modellierte Domäne des Ausstellungs- und Konferenzbesuches verwendet worden. Wie die Systemarchitektur in anderen Domänen eingesetzt werden kann, wird in Abschnitt 7.2 beschrieben.

6.1 Gesamtüberblick

Die Realisierung der Systemarchitektur orientiert sich am Referenzmodell der J2EE (Java 2 Enterprise Edition) [j2e01]. Einen Überblick über die J2EE-Architektur zeigt Abbildung 6.1. Die in der Realisierung verwendeten Bestandteile sind farbig hervorgehoben. Zu betonen ist hierbei, dass das Diagramm die logische Aufteilung der Komponenten darstellt, nicht jedoch die physische Verteilung auf konkrete Rechner.

Die J2EE-Architektur setzt sich aus den folgenden Bestandteilen zusammen:

- **Application Components:** Die J2EE-Architektur definiert vier verschiedene Application Components: *Application Clients*, *Applet Clients*, *Servlets*

[1] Der Name MoBibel soll den Bezug zur Mobilität und zu Bibliotheken verdeutlichen.

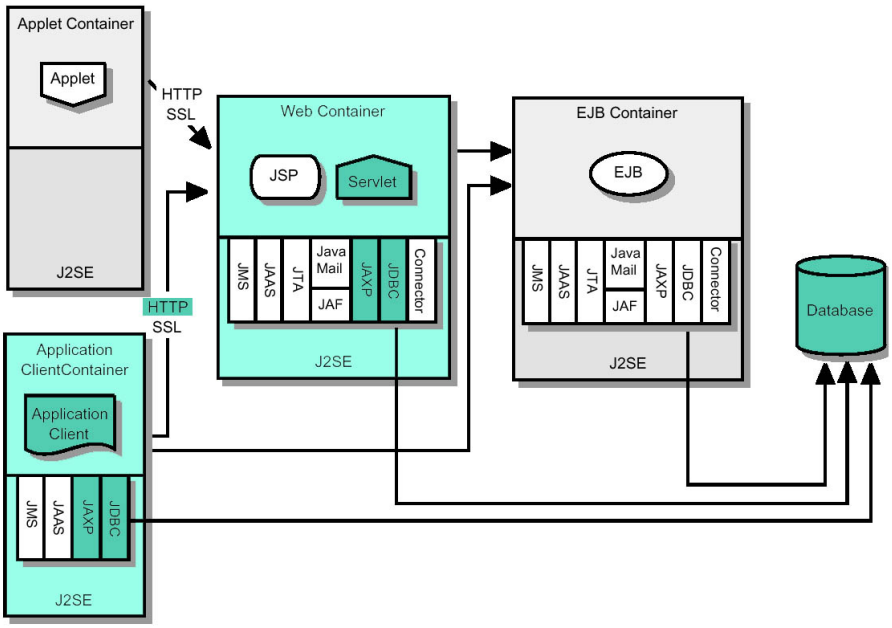

Abbildung 6.1: J2EE-Architektur-Diagramm

und *JSPs* (Java Server Pages). In der Umsetzung der Systemarchitektur wurden folgende Application Components eingesetzt:

  - Die Komponenten des mobilen Endgerätes sind als *Application Client* realisiert worden. Diese Java-Applikation setzt sich aus Java-Klassen zusammen, die die in Kapitel 5 beschriebenen Interfaces implementieren.
  - Die Server-seitigen Komponenten (Query und Retrieval Manager) sind als Servlets realisiert worden.

- **Containers:** Container sind die Laufzeitumgebungen (Runtime Environments) für die Application Components. Auf der Seite des mobilen Endgerätes wurden hierfür verschiedene Java Runtime Environments eingesetzt, auf der Server-Seite der Jakarta-Tomcat Webserver.

- **Database:** Zum persistenten Speichern von Daten sieht die J2EE-Architektur eine Datenbank vor, auf die über die JDBC-Schnittstelle zugegriffen wird. Als Besonderheit ist hierbei die verteilte Datenhaltung mit jeweils einer Datenbank auf dem Server und dem mobilen Endgerät zu erwähnen.

Das J2EE-Referenzmodell schreibt außerdem die Verwendung von Interoperabilitätsstandards vor. In der Realisierung wurden hier konkret die Protokolle HTTP und JDBC sowie das JAXP (Java API for XML Parsing) verwendet.

6.2 Verteilte Architektur

In diesem Abschnitt werden verwendete Geräteplattformen, Netzwerktechnologien und Datenbanksysteme zur Realisierung der verteilten Systemarchitektur vorgestellt.

6.2.1 Geräteplattformen

Der größte Vorteil der Verwendung der Java-Technologie besteht in der weitgehenden Plattformunabhängigkeit. Mit den Java-Varianten J2ME (Java 2 Micro Edition) [j2m01], J2SE (Java 2 Standard Edition) und J2EE (Java 2 Enterprise Edition) wird eine breite Palette von Geräten — vom Mobiltelefon und PDA bis zum Server — unterstützt, wie Abbildung 6.2 zeigt.

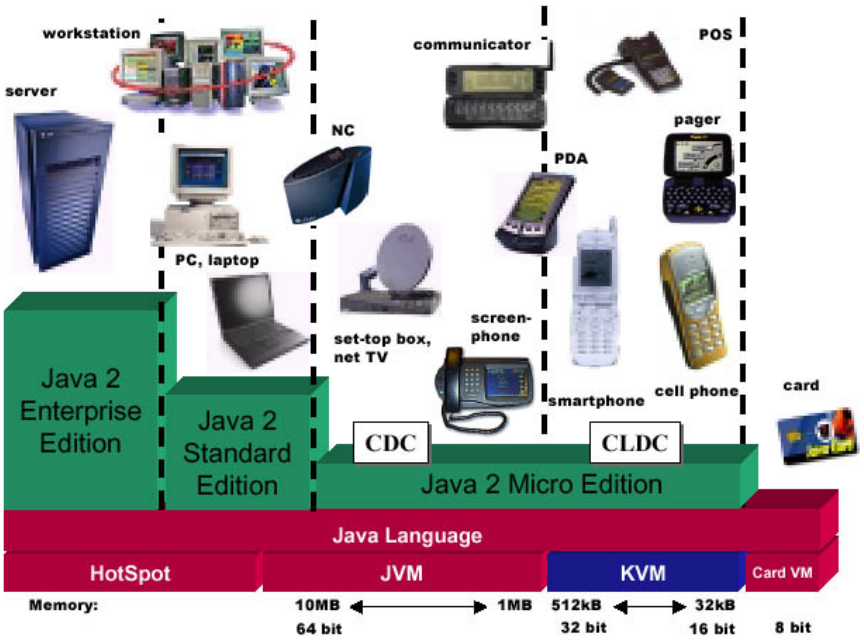

Abbildung 6.2: Die Java-Varianten und ihre Zielgeräte

Als mobile Endgeräte kommen für *MoBibel* insbesondere Notebooks und PDAs in Frage, also Geräte, die durch die J2SE und die CDC (Connected Device Configuration) der J2ME abgedeckt sind. Geräte der CLDC (Connected Limited Device Configuration) sind aufgrund ihrer Gerätemerkmale und des Funktionsumfangs der KVM (K Virtual Machine) nicht geeignet. In diese Klasse fallen z.B. aktuelle Mobiltelefone. Dies schließt jedoch — perspektivisch — den Einsatz von Mobiltelefonen nicht aus. Mit der Entwicklung der SmartPhones verfließen die Grenzen zwischen Mobiltelefonen und PDAs. Bereits heute sind leistungsfähige SmartPhones

mit dem PocketPC-Betriebssystem verfügbar. Es ist abzusehen, dass Java Virtual Machines für diese Geräte folgen werden.

Abbildung 6.3 zeigt die Hardware-Infrastruktur, die für die Realisierung von *MoBibel* konkret verwendet wurde. Als stationäre Server wurden im Fraunhofer

Abbildung 6.3: Verwendete Hardware-Infrastruktur

IGD ein Linux PC, am Lehrstuhl DBIS des FB Informatik eine Sun Solaris Workstation eingesetzt. Als mobile Endgeräte wurden ein Windows 2000 Notebook und ein PocketPC Compaq iPAQ 3660 verwendet. Als Java-Laufzeitumgebungen wurden verschiedene JREs von Sun sowie die Jeode Virtual Machine von Insignia für den Compaq iPAQ genutzt.

Die Jeode Virtual Machine entspricht in der aktuellen Version 1.7 der Spezifikation von PersonalJava 1.2, einem Vorgänger der J2ME, und bietet damit den Funktionsumfang von Java 1.1.8. Gegenwärtig ist eine Version in Entwicklung, die der J2ME-Spezifikation genügt. Die Jeode Virtual Machine wird ab sofort mit jedem iPAQ der 3800-Serie ausgeliefert. Dies ist ein Zeichen dafür, dass die Bedeutung der Java-Technologie auf ultraportablen Geräten in absehbarer Zukunft stark zunehmen wird.

6.2.2 Drahtlose Datenübertragung

Für die drahtlose Kommunikation zwischen mobilem Endgerät und Server wurde auf die bestehende Infrastruktur des Fraunhofer IGD und der Universität Rostock zurückgegriffen. In der Universität Rostock wird ein WLAN nach dem IEEE 802.11-Standard verwendet, im Fraunhofer IGD kommt das proprietäre RangeLAN der Firma Proxim zum Einsatz.

Da die Kommunikation zwischen Server und mobilem Endgerät ausschließlich auf dem HTTP-Protokoll aufsetzt, können auch andere Netzwerktechnologien eingesetzt werden. Aspekte die man bei der Auswahl einer konkreten Technologie berücksichtigen sollte, sind verfügbare Bandbreite, Reichweite und Kosten. Zu

beachten ist auch, dass einige Verfahren wie z.B. Infrarotdatenübertragung Sichtkontakt zwischen Sender und Empfänger voraussetzen.

Grundsätzlich lassen sich drei Kategorien drahtloser Netzwerktechnologien unterscheiden:

- *Zellulare Netze* auf der Basis von GSM, GPRS und in naher Zukunft UMTS, erlauben drahtlose Kommunikation im WAN (Wide Area Network).

- *Wireless LANs* sind mit ihrer mittleren Reichweite von 50-400 Metern insbesondere für den Einsatz innerhalb von Gebäuden bzw. auf Firmengeländen geeignet. Ethernet 802.11 hat sich zum Standard im Bereich der WLANs entwickelt.

- *Wireless PANs (Personal Area Network) und BANs (Body Area Networks)* erlauben die drahtlose Kommunikation zwischen Geräten in unmittelbarer Umgebung des Nutzers. Sie sind gekennzeichnet durch typischerweise kurze Reichweite, geringe Leistung und begrenzte Anzahl von Knoten (8-16). Erfolgversprechendster Vertreter von dieser Netzwerktechnologie ist Bluetooth. Daneben gibt es bewährte Standards wie z.B. IrDA für Kommunikation auf Infrarotbasis.

Ein interessanter Ansatz besteht darin, verschiedene Netzwerktechnologien zu kombinieren, um unter Berücksichtigung von Verfügbarkeit, Kosten und Bandbreite auf verschiedene Alternativen zurückgreifen zu können. So würde sich eine Kombination aus UMTS (oder GPRS) im WAN, Ethernet 802.11 im WLAN und Bluetooth im PAN als sinnvoll erweisen. Dies ist bereits heute realisierbar: So werden beispielsweise die Compaq iPAQ der 3800-Serie mit integriertem Bluetooth ausgeliefert. Als Erweiterung sind Funk-Jackets erhältlich, die eine Kommunikation sowohl über GPRS als auch Ethernet 802.11 ermöglichen.

6.2.3 Datenbanksysteme

Für die Realisierung des verteilten Datenmanagements wurden als Datenbanksysteme verschiedene Versionen von IBMs DB2 eingesetzt. Diese werden im folgenden vorgestellt.

IBM DB2 UDB DB2 UDB [db2] ist ein leistungsfähiges, objektrelationales Datenbankmanagementsystem. Es ist erhältlich für eine Vielzahl von Plattformen, darunter Windows, Solaris, Linux und natürlich AIX.

Es wurde auf der Server-Seite im Fraunhofer IGD in der Version 7.2 für Linux und im FB Informatik in der Version 7.1 unter Sun Solaris eingesetzt.

DB2 bietet mit seiner Extender-Technologie spezielle zusätzliche Funktionalitäten z.B. für XML-, Multimedia- und Textdaten.

IBM DB2 Everyplace DB2 Everyplace [db201] ist ein Datenbanksystem speziell für mobile Endgeräte. Dabei bietet es allerdings nicht alle Merkmale eines "echten" Datenbankmanagementsystems. Dies betrifft z.B. die Unterstützung von Transaktionen.

DB2 Everyplace ist momentan in der Version 7.2 für die folgenden Plattformen verfügbar: Windows CE, Palm OS, EPOC, Neutrino, Embedded Linux und Win32. In der realisierten Architektur wurde es auf der Client-Seite unter Windows CE auf dem iPAQ und unter Windows 2000 auf dem Notebook eingesetzt.

IBM DB2 Everyplace for Java DB2 Everyplace for Java ist ein neues, komplett in Java realisiertes Datenbanksystem. Es ist damit eine vollständige Plattformunabhängigkeit erreicht, allerdings auf Kosten der Performance. Im Vergleich mit den nativen DB2-Everyplace-Varianten ergeben sich für DB2 Everyplace for Java Performance-Einbußen um den Faktor 100. Außerdem ist mit ca. 10 mal mehr Speicherplatzbedarf zu rechnen. Damit ist DB2 Everyplace for Java zumindest momentan keine echte Alternative.

Alternative Datenbanksysteme Da die Anbindung der Datenbanksysteme über die JDBC-Schnittstelle erfolgt, können praktisch beliebige relationale Datenbanksysteme auf Server- und Client-Seite eingesetzt werden, sofern ein JDBC-Treiber verfügbar ist. Es werden keine besonderen Anforderungen an unterstützte Datentypen, Transaktionen oder SQL-Funktionsumfang gestellt. Es werden nur Standarddatentypen verwendet, Transaktionen spielen aufgrund des primär lesenden Zugriffs keine Rolle.

Als freies Datenbanksystem sei exemplarisch MySQL [mys] genannt, das für eine Vielzahl von Standardbetriebssystemen verfügbar ist.

6.3 Context Manager

Im folgenden wird die Realisierung der *ContextComponents*, Environment Monitor, Task Manager und Profile Manager, vorgestellt.

6.3.1 Environment Monitor

Positionsbestimmung Zur Positionsbestimmung und Objektidentifikation werden die vom Fraunhofer IGD entwickelten IrDA-Baken [Ide00] auf Infrarotbasis eingesetzt. Diese Baken sind in Versionen für Netzbetrieb und auch für Batteriebetrieb verfügbar und sind damit sowohl stationär als auch mobil einsetzbar.

Als alternative Möglichkeiten zur Positionsbestimmung sind z.B. satellitengestützte Systeme (GPS) und die Nutzung von Zellinformationen zellularer Netze zu nennen. Sinnvoll ist auch eine Kombination verschiedener Techniken zur Positionsbestimmung, wie sie z.B. im Projekt Sensorfusion [Kor01] untersucht wird.

6.3.2 Task Manager

Für den Task Manager wurde als Produktionssystem Jess [FH01] in der Version 6.0 eingesetzt. JESS ist ein zu CLIPS kompatibles[2] regelbasiertes Expertensystem komplett in Java und ermöglicht somit eine enge Kopplung von Expertensystem und Java-Applikationen. Für die Konvertierung der Situationsbeschreibungen in DAML nach Jess entsprechend der in 5.3.2 vorgeschlagenen Transformation kommt DAMLJessKB [Kop01], ein freies Tool ebenfalls in Java, zum Einsatz.

6.3.3 Profile Manager

In der prototypischen Realisierung werden die Geräteeigenschaften aus einer Konfigurationsdatei ausgelesen. Während dies für nur zwei verschiedene eingesetzte Gerätetypen durchaus sinnvoll ist, wäre für einen Einsatz auf einer größeren Anzahl verschiedener Geräte zu überprüfen, inwieweit Geräteeigenschaften automatisch bestimmt werden können, beispielsweise aus Registry-Einträgen oder unter Ausnutzung der Java System Properties.

6.4 Query Manager

Für die Bearbeitung der Inferenz- und Information-Retrieval-Anfragen werden spezielle Systeme verwendet. Einige Besonderheiten werden nun erläutert.

- *Inferenzanfragen:* Für die Realisierung der Inferenzanfragen kommt das frei verfügbare, Frame-Logic-basierte Inferenzsystem SiLRI zum Einsatz. Dieses Inferenzsystem in Java wurde ursprünglich an der Universität Karlsruhe, später bei der Ontoprise GmBH entwickelt. SiLRI ist ausführlich beschrieben in [DBSA]. Es implementiert weite Teile der Frame-Logic, wobei gegenüber der in 4.1.1 beschriebenen Standardsyntax einige Erweiterungen implementiert sind [Erd01]. SiLRI bietet direkte Unterstützung für RDF. Regeln und Anfragen können nicht direkt in RDF formuliert werden, sondern nur in F-Logic.

  Im Hinblick auf die Performanz des Inferenzsystems ist hier der Einsatz der Rollback-Funktionalität zu erwähnen. Die initiale Faktenbasis bestehend aus der Beschreibung der Inhalte der Bibliothek und Szenarios wird einmalig geladen. Mit jeder Anfrage wird die Faktenbasis um die Situationsbeschreibung und abgeleitete Fakten erweitert. Nach der Bearbeitung der Anfrage können diese neuen Fakten durch ein *Rollback* entfernt und der Initialzustand wiederhergestellt werden, so dass nicht die gesamte Faktenbasis neu geladen werden muss. Dadurch können die Antwortzeiten erheblich reduziert werden ($t \ll 1s$). Damit ist das Anfragesystem prinzipiell auch für den

[2] In der verwendeten Version gibt es einige Einschränkungen in der Kompatibilität in Bezug auf die unterstützten Konfliktlösungsstrategien (nur Depth und Breadth sind vorhanden) und die Verwendung von Quantoren in Kombination mit Negationen,

Mehrbenutzerbetrieb geeignet, auch wenn mehrere Anfragen nicht unmittelbar parallel bearbeitet werden können.

- *Information-Retrieval-Anfragen:* Für inhaltsbasierte Anfragen auf Volltexten wird der DB2 TextExtender eingesetzt. Zur Deskribierung der Dokumente wird ein linguistischer Index (Englisch) verwendet, der auf einer Stammwortreduktion basiert. Neben diesem Indextyp *linguistic* unterstützt der TextExtender die Indextypen *n-gram* und *precise*. Für die Recherche wird die linguistische Suche eingesetzt, bei der auch Variationen eines Suchbegriffes gefunden werden. Alternative Recherchefunktionen sind *Precise Search*, *Fuzzy Search*, *Thesaurus Search* und *Phonetic Search*. Für Details zu Deskribierungs- und Recherchefunktionalitäten sei auf [tex00] verwiesen.

  Da der TextExtender weder PDF- noch PS-Dokumente direkt indexieren kann, erfolgt bei diesen Dokumenttypen zunächst eine Extraktion des ASCII-Textes.

  Zu erwähnen ist, dass eine inhaltsbasierte Suche aufgrund von Ressourcenbeschränkungen auf mobilen Endgeräten wenig Sinn macht. Sie wurde deshalb nur Server-seitig umgesetzt.

  Als interessante Erweiterung wäre eine inhaltsbasierte Suche auf anderen Medientypen denkbar, z.B. durch Einsatz des DB2 Audio-, Image- und VideoExtenders.

6.5 Retrieval Manager

Für die Realisierung des Retrieval Manager sind keine speziellen Systeme zum Einsatz gekommen. Erwähnenswert ist jedoch, wie beim Retrieval der Metadaten die zurückgelieferten Java-Objekte generiert werden. Aus der Unabhängigkeit der Ontologie und der Java-Klassenhierarchie von der Systemarchitektur ergibt sich, dass der Typ der angeforderten Java-Objekte erst zur Laufzeit bekannt ist. Deshalb werden die Java-Objekte mit Hilfe des Java-Konzeptes der *Reflection* generiert.

6.6 Dialog Manager

Der Dialog Manager ist die einzige Komponente, die für die Zielplattformen PocketPC und Notebook getrennt realisiert wurde. Hierfür gibt es zwei Gründe. Zum einen ist es sinnvoll, bei der Entwicklung der Oberfläche besondere Geräteeigenschaften wie z.B. die Größe des Displays zu berücksichtigen. Der andere Grund besteht darin, dass die verwendete Java Virtual Machine *Jeode* auf dem Compaq iPAQ in der aktuellen Version nur die AWT-, nicht aber die komfortablere und umfangreichere Swing-Bibliothek bereitstellt. Daher wurde ein Dialog Manager auf der Basis des AWT (Abstract Window Toolkit) für PocketPCs und ein weiterer auf der Basis von Swing für Notebooks entwickelt.

Für die Top-Level-Konzepte der Ontologie wurden exemplarisch Widgets entwickelt. Abbildung 6.4 zeigt einen Screenshot der Oberfläche des Dialog Manager. Zu erkennen sind im linken Panel die hierarchische Struktur der Ontologie, im mitt-

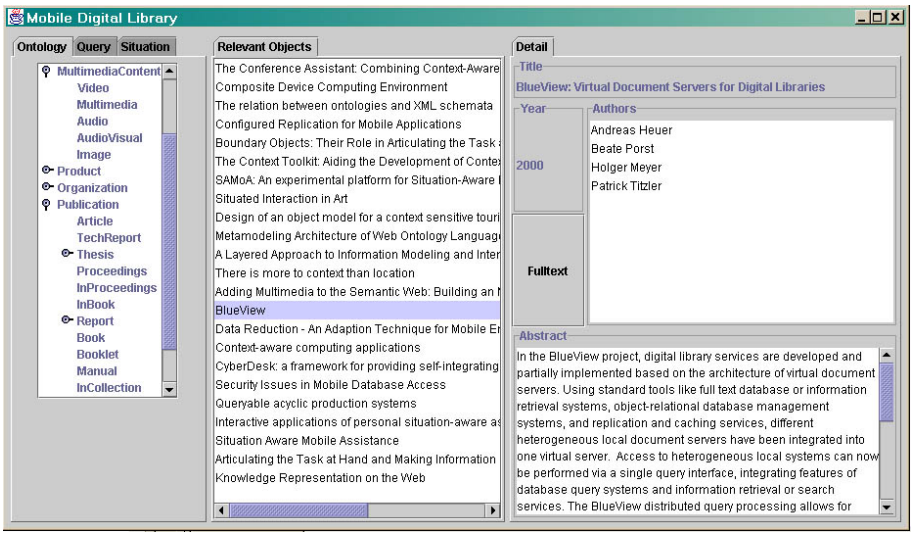

Abbildung 6.4: Screenshot von *MoBibel* (Windows 2000)

leren Panel eine Liste relevanter Objekte (im abgebildeten Fall alle Instanzen des Typs *InProceedings*) und im rechten Panel das Widget zum Darstellung von Publikationen. Erkennbar ist auch die Anwendung des Level-Of-Detail-Konzeptes: In der Liste der relevanten Objekte wird zunächst nur der Titel der Publikationen angezeigt. Nach Auswahl eines Objektes werden die Metadaten einschließlich eines Abstracts dargestellt. Der komplette Volltext des Dokumentes würde nach dem Betätigen des *Fulltext*-Buttons angezeigt werden. Für die Darstellung der semi- und unstrukturierten Dokumente wie Volltexte oder Vortragsfolien werden Standardapplikationen wie der Acrobat Reader und PowerPoint verwendet. Auch für PocketPCs sind bereits Applikationen zur Darstellung dieser Inhalte verfügbar, so. z.B. *Primer PDF Viewer* [pri01] für PDF-Dokumente und *PocketSlides* [poc01] für PowerPoint-Präsentationen.

Der implementierte Dialog Manager bietet des weiteren eine Anfrageschnittstelle für die beschriebenen Formen von Anfragen, die Möglichkeit der Definition von Nutzerprofilen und der Verwaltung von Aufgaben.

6.7 Zusammenfassung

In diesem Kapitel ist die prototypische Realisierung der Systemarchitektur, *MoBibel*, vorgestellt worden. Durch die Verwendung der Java-Technologie ist es gelungen, ein plattformunabhängiges System für den situationsgesteuerten mobilen

Zugriff auf Digitale Bibliotheken zu entwickeln. Lediglich der Dialog Manager ist für die konkreten Zielplattformen PocketPC und Notebook getrennt realisiert worden.

MoBibel ist zunächst als Assistent für Ausstellungs- und Konferenzbesuche entwickelt worden. Im nächsten abschließenden Kapitel wird erläutert, wie ein Einsatz in anderen Domäne realisiert werden kann. Außerdem wird aufgezeigt, welche Erweiterungen der Systemarchitektur sinnvoll wären.

Kapitel 7

Zusammenfassung und Ausblick

In diesem abschließenden Kapitel werden zunächst die Ergebnisse dieser Arbeit zusammengefasst. Es wird dann gezeigt, wie einfach das Gesamtsystem in einer beliebigen anderen Domäne als der hier vorgestellten angewendet werden kann.

Natürlich sind im Rahmen dieser Arbeit auch Fragen und Probleme offen geblieben. Diese werden am Ende des Kapitels angesprochen.

7.1 Ergebnisse der Arbeit

In dieser Arbeit ist es gelungen, eine Systemarchitektur für den situationsgesteuerten mobilen Zugriff auf Digitale Bibliotheken zu entwickeln und prototypisch umzusetzen. Nach einer Einführung in Konzepte Digitaler Bibliotheken und der situationsgesteuerten Assistenz wurden zunächst die Anforderungen an eine solche Systemarchitektur analysiert. Ein wichtiges Ergebnis dieser Anforderungsanalyse war, dass ein verteiltes Datenmanagement erforderlich ist, um einen fehlertoleranten und effizienten Zugriff auf die Inhalte der Digitalen Bibliothek zu realisieren. Es wurde weiterhin herausgearbeitet, dass zusätzlich zu den bekannten Anfragetypen Digitaler Bibliotheken — Anfragen an Metadaten und Information-Retrieval-Anfragen — ein weiterer Anfragetyp erforderlich ist, um situationsrelevante Inhalte proaktiv bereitstellen zu können. Dieser neue Anfragetyp der Inferenzanfragen ermöglicht es, relevante Inhalte aus der Nutzersituation abzuleiten.

Um ein solches automatisches Schließen zu ermöglichen, stellen sich besondere Anforderungen an die Datenmodellierung. Es wurde gezeigt, warum eine ontologiegestützte, wissenbasierte Datenmodellierung sinnvoll ist, und dass die Modellierungssprachen des Semantic Web sich hierfür als besonders geeignet erweisen. Exemplarisch wurde eine Ontolgie für die Domäne der Assistenz bei Ausstellungs- und Konferenzbesuchen in DAML entwickelt. Bei der Modellierung sind sowohl die Inhalte der Bibliothek, die Beschreibung des Szenarios und die Nutzersituation berücksichtigt worden.

Die im Anschluss entwickelte Systemarchitektur beschreibt Schnittstellen und Techniken, die für den situationsgesteuerten, mobilen Zugriff auf Digitale Biblio-

theken erforderlich sind und den definierten Anforderungen genügen. Eine zentrale Rolle spielt dabei die Situationsanalyse. Spezielle Komponenten dienen der Erfassung der physischen Umgebung mittels Sensoren, der Verwaltung von Nutzeraufgaben sowie von Geräte- und Nutzerprofilen.

Die Kontextinformationen werden in regelbasierten Inferenzsystemen genutzt, um situationsrelevante Informationen abzuleiten. Dabei kommt für die Aufgabenverwaltung ein Produktionssystem zum Einsatz. Für die Realisierung der Inferenzanfragen zur Bestimmung situationsrelevanter Inhalte wird die deduktive Anfragesprache Frame-Logic genutzt.

Die Verwendung der Techniken Level of Detail, Replikation und Caching erlauben eine hohe Effizienz beim Retrieval von Inhalten. Durch das verteilte Datenmanagement ist zudem eine hohe Verfügbarkeit der Daten auch bei unterbrochener Netzwerkverbindung ermöglicht.

Mit *MoBibel* wurde die gesamte Systemarchitektur unter Verwendung von Java-Technologien und damit plattformunabhängig umgesetzt und in Form eines Ausstellungs- und Konferenzassistenten demonstriert. Aufgrund ihres generischen Charakters ist die Systemarchitektur auch auf andere Domänen anwendbar.

7.2 Erweiterbarkeit und Anwendung auf andere Domänen

Ein entscheidender Vorteil der in dieser Arbeit vorgestellten Systemarchitektur und der ontologiebasierten Datenmodellierung liegt in der einfachen Anwendbarkeit auf andere Domänen. In Kapitel 3 ist exemplarisch die Domäne Ausstellungs- und Konferenzbesuch modelliert und eine Ontologie entwickelt worden. Soll die Systemarchitektur in einem anderen Gebiet angewendet werden, z.B. in einem mobilen Informationssystem für Touristen, so ist für diese neue Domäne eine neue Ontologie zu entwerfen. Natürlich muss die Ontologie nicht komplett neu entwickelt werden: Existierende Ontologien können kombiniert und erweitert werden. Die neue Ontologie wird durch einfache Transformationen auf ein relationales Datenbankschema und eine Java-Klassenhierarchie abgebildet, wie ebenfalls in Kapitel 3 dargestellt.

Neben der Ontologie sind weiterhin Regeln für den Inferenzprozess zum regelbasierten Bestimmen relevanter Inhalte zu entwickeln. Außerdem sind eventuell neue Widgets für die Visualiserung der Inhalte und für die Interaktion mit dem Nutzer notwendig. In einem existierenden System lassen sich Ontologie, Datenbankschema, Widgets, Regeln, etc. austauschen, ohne Veränderungen an den Komponenten der beschriebenen Systemarchitektur vorzunehmen.

7.3 Offene Probleme und Ausblick

Da in dieser Arbeit eine Vielzahl von Aspekten Digitaler Bibliotheken, intelligenter Assistenz, der Wissensrepräsentation und Inferenz behandelt wurden, konnten

nicht alle Aspekte konsequent im Detail dargestellt werden. Einige Bereiche, in denen weitere Arbeit sinnvoll wäre, sind:

- *Task Management:* Das in der Arbeit beschriebene Task Management setzt voraus, dass bereits eine Beschreibung der Aufgaben mit ihrer internen Struktur vorliegt. Bei komplexen Aufgaben besteht die erste Hürde aber schon darin, diese Aufgabenbeschreibung zu erzeugen. Dies könnte z.B. durch den Einsatz eines Planungssystems geschehen. Für das Scheduling der Aufgaben wäre ein vollständiges und generisches Regelwerk auf der Basis des beschriebenen Prozessmodells wünschenswert.

  Des weiteren wäre es interessant zu evaluieren, wie gut das zu Grunde liegende Prozessmodell nicht nur für die Modellierung von Aufgaben, sondern z.B. auch für die Modellierung von Routing-Informationen oder Event-Schedules geeignet ist. In diesem Zusammenhang ergeben sich auch neue Anforderungen an die Modellierung der Zeit, die in dieser Arbeit nur sehr kurz angeschnitten wurde.

- *Räumlich-zeitliche Strukturen:* Die Modellierung räumlich-zeitlicher Topologien, wie sie in dieser Arbeit vorgenommen wurde, ist sicherlich nicht ausreichend für sämtliche Aspekte mobiler Assistenz, wie z.B. für die Darstellung von Karten oder komplexen, räumlich und zeitlich verteilten Aktivitäten. Generell scheint aber eine ontologiebasierte Modellierung dieser Strukturen sinnvoll, wie beispielsweise die Time Ontology [tim] zeigt.

- *Zugriffsarchitekur:* In der der Zugriffsarchitektur des Retrieval-Manager kommen bereits Techniken wie Replikation, Caching und Level of Detail zum Einsatz. Wie schon erwähnt, wäre eine Verbindung mit Techniken wie Prefetching, Datenreduktion und Kompression sinnvoll.

Einige Themen konnten in dieser Arbeit gar nicht behandelt werden. Hierzu gehören zum Beispiel:

- *Sicherheit:* Insbesondere in mobilen Umgebungen spielen Sicherheitsaspekte eine besondere Rolle. Diese wurden in dieser Arbeit nicht berachtet.

- *Zahlungsmodelle:* In der Arbeit wurden Kostenmodelle nicht berücksichtigt. In vielen Szenarien werden diese jedoch eine Rolle spielen, z.B. bei kostenpflichtigem Zugriff auf Inhalte der Digitalen Bibliothek oder für Netzwerkverbindungen. Hier wären dann auch gesonderte Transaktionsmodelle notwendig.

Für konkrete Anwendungen ist es sicherlich wünschenwert, das vorgestellte System um Funktionalität zu erweitern, die nicht unmittelbar mit dem Zugriff auf Digitale Bibliotheken zu tun haben, sehr wohl aber in einem mobilen Assistenzsystem wünschenswert wären. Solche Funktionalitäten könnten zum Beispiel sein:

- *Routing:* Bei räumlich verteilten Aktivitäten ist eine dynamische Berechnung von Wegbeschreibungen sinnvoll.
- *Scheduling:* Die Verwaltung von persönlichen Terminkalendern wäre denkbar.

Der letzte Punkt betrifft weniger offene Fragen dieser Diplomarbeit, sondern ein allgemeines Problem der Softwaretechnologie, nämlich die Verwendung unterschiedlicher Datenmodelle. Im Kapitel 3 wurde bereits angesprochen, wie die Datenmodelle des Semantic Web auf Programmiersprachen- und Datenbankmodelle abgebildet werden können. Konkret wurden Abbildungen einer Ontologie auf ein relationales Datenbankschema und eine Java-Klassenhierarchie vorgestellt. Eine noch bessere Integration der Datenmodelle der Wissensrepräsentation, Programmiersprachen und Datenbanken wäre aber sehr hilfreich gewesen. Sinnvolle Ansätze für standardisierte Objektmodelle gab es bereits, z.B. mit dem ODMG-Standard. Zum einen haben diese aber nicht die nötige Verbreitung gefunden. Zum anderen werden sie nicht allen Anforderungen gerecht, denn die Probleme gehen über die des klassischen *Impedance Mismatch* hinaus. Dies betrifft vor allem die beschränkte Aussdrucksstärke dieser Modelle, zum anderen die mangelnde Interoperabilität auf semantischer Ebene.

Anhang A

Ontologien

A.1 Ontologie für die Ausstellungs- und Konferenzdomäne

```
Publication::null.
Article::Publication.
Book::Publication.
Booklet::Publication.
InBook::Publication.
InCollection::Publication.
InProceedings::Publication.
Manual::Publication.
Misc::Publication.
Proceedings::Publication.
Report::Publication.
TechReport::Publication.
Thesis::Publication.
Unpublished::Publication.
MasterThesis::Thesis.
PhDThesis::Thesis.
ProjectReport::Report.

MultimediaContent::null.
Audio::MultimediaContent.
AudioVisual::MultimediaContent.
Image::MultimediaContent.
Multimedia::MultimediaContent.
Video::MultimediaContent.

Product::null.
SoftwareComponent::Product.

Project::null.
DevelopmentProject::Project.
ResearchProject::Project.
SoftwareProject::DevelopmentProject.

Topic::null.
ResearchTopic::Topic.

Person::null.
Employee::Person.
AcademicStaff::Employee.
FacultyMember::AcademicStaff.
AssociateProfessor::FacultyMember.
```

```
AssistantProfessor::FacultyMember.
FullProfessor::FacultyMember.
Lecturer::AcademicStaff.
Manager::Employee.
TechnicalStaff::Employee.
Student::Person.
Graduate::Student.
PhDStudent::Graduate.
Undergraduate::Student.

PhysicalStructure::null.
Booth::PhysicalStructure.
Building::PhysicalStructure.
Floor::PhysicalStructure.
Room::PhysicalStructure.

Organization::null.
Association::Organization.
Department::Organization.
Enterprise::Organization.
Institute::Organization.
ResearchGroup::Organization.
University::Organization.

Event::null.
Conference::Event.
Exhibition::Event.
Lecture::Event.
Meeting::Event.
ProjectMeeting::Meeting.
Workshop::Event.

Publication[
year=>>STRING;
note=>>STRING;
abstract=>>STRING;
author=>>Person;
title=>>STRING;
keywords=>>STRING
].
Article[
volume=>>STRING;
month=>>STRING;
number=>>STRING;
pages=>>STRING;
journal=>>STRING
].
Book[
editor=>>Person;
isbn=>>STRING;
price=>>STRING;
volume=>>STRING;
series=>>STRING;
month=>>STRING;
number=>>STRING;
edition=>>STRING;
source=>>STRING;
publisher=>>Organization;
address=>>STRING
].
Booklet[
howpublished=>>STRING;
```

```
month=>>STRING;
address=>>STRING
].
InBook[
chapter=>>STRING;
editor=>>Person;
volume=>>STRING;
series=>>STRING;
month=>>STRING;
number=>>STRING;
edition=>>STRING;
pages=>>STRING;
publisher=>>Organization;
address=>>STRING;
type=>>STRING
].
InCollection[
chapter=>>STRING;
editor=>>Person;
volume=>>STRING;
series=>>STRING;
month=>>STRING;
number=>>STRING;
edition=>>STRING;
pages=>>STRING;
publisher=>>Organization;
booktitle=>>STRING;
address=>>STRING;
type=>>STRING
].
InProceedings[
editor=>>Person;
volume=>>STRING;
series=>>STRING;
month=>>STRING;
number=>>STRING;
pages=>>STRING;
publisher=>>Organization;
booktitle=>>STRING;
address=>>STRING;
organization=>>Organization
].
Manual[
month=>>STRING;
edition=>>STRING;
address=>>STRING;
organization=>>Organization
].
Misc[
howpublished=>>STRING;
month=>>STRING
].
Proceedings[
editor=>>Person;
volume=>>STRING;
series=>>STRING;
month=>>STRING;
number=>>STRING;
publisher=>>Organization;
address=>>STRING;
organization=>>Organization
].
```

```
ProjectReport[
describesProject=>>Project
].
TechReport[
institution=>>Organization;
month=>>STRING;
number=>>STRING;
address=>>STRING;
type=>>STRING
].
Thesis[
month=>>STRING;
address=>>STRING;
school=>>University;
type=>>STRING
].
Unpublished[
month=>>STRING
].

MultimediaContent[
description=>STRING;
format=>STRING;
date=>STRING;
title=>STRING;
language=>STRING;
type=>STRING;
subject=>>Topic;
creator=>>Person;
contributor=>>Person;
publisher=>>Organization;
coverage=>>STRING;
rights=>>STRING
].

Topic[
name=>>STRING
].
ResearchTopic[
dealtWithIn=>>Project;
isWorkedOnBy=>>AcademicStaff
].

Product[
developedBy=>>Organization;
name=>>STRING
].
SoftwareComponent[
hasPrice=>STRING
].

Project[
member=>>Employee;
isAbout=>>ResearchTopic;
name=>>STRING;
projectInfo=>>Publication;
title=>>STRING;
head=>>Employee;
financedBy=>>Organization;
carriedOutBy=>>Organization
].
SoftwareProject[
```

```
product=>>Product
].

Person[
lastName=>STRING;
age=>STRING;
name=>STRING;
firstName=>STRING;
middleInitial=>STRING;
homepage=>>STRING;
phone=>>STRING;
email=>>STRING;
fax=>>STRING;
address=>>STRING;
photo=>>STRING
].
Employee[
affiliation=>>Organization
].
AcademicStaff[
headOf=>>Project;
headOfGroup=>>ResearchGroup;
editor=>>Publication;
memberOfPC=>>Event;
worksAtProject=>>Project;
supervises=>>PhDStudent;
cooperateWith=>>AcademicStaff;
publication=>>Publication
].
Student[
studiesAt=>>University
].
PhDStudent[
supervisor=>>AcademicStaff;
worksAtProject=>>Project;
publication=>>Publication
].

Organization[
carriesOut=>>Project;
publishes=>>Publication;
develops=>>Product;
location=>>STRING;
employs=>>Person;
finances=>>Project;
name=>>STRING
].
Department[
hasParts=>>Institute
].
Institute[
hasParts=>>ResearchGroup;
cooperateWith=>>Institute
].
ResearchGroup[
member=>>Employee;
head=>>Employee
].
University[
student=>>Student;
hasParts=>>Department
].
```

```
Event[
located=>PhysicalStructure;
atEvent=>>Event;
hasPartEvent=>>Event;
location=>>STRING;
name=>>STRING;
eventTitle=>>STRING;
date=>>STRING;
publication=>>Publication;
presentation=>>MultimediaContent
].
Conference[
series=>>STRING
].
Workshop[
series=>>STRING
].
Meeting[
participant=>>Person;
date=>>STRING;
title=>>STRING
].

PhysicalStructure[
number=>STRING;
name=>>STRING;
].
Building[
contains=>>Floor
].
Floor[
contains=>>Room
].
Room[
contains=>>Booth
].

// F-Logic Axioms :
FORALL X,Y Y[cooperateWith->>X] <- X[cooperateWith->>Y].
FORALL X,Y,Z X[hasPartEvent->>Z] <- X[hasPartEvent->>Y] AND Y[hasPartEvent->>Z].
FORALL X,Y Y[carriedOutBy->>X] <- X[carriesOut->>Y].
FORALL X,Y Y[carriesOut->>X] <- X[carriedOutBy->>Y].

FORALL X,Y Y[atEvent->>X] <- X[hasPartEvent->>Y].
FORALL X,Y Y[hasPartEvent->>X] <- X[atEvent->>Y].

FORALL X,Y Y[publishes->>X] <- X[publisher->>Y].
FORALL X,Y Y[publisher->>X] <- X[publishes->>Y].

FORALL X,Y Y[head->>X] <- X[headOfGroup->>Y].
FORALL X,Y Y[headOfGroup->>X] <- X[head->>Y].

FORALL X,Y Y[isWorkedOnBy->>X] <- X[worksAtProject->>Y].
FORALL X,Y Y[worksAtProject->>X] <- X[isWorkedOnBy->>Y].

FORALL X,Y Y[describesProject->>X] <- X[projectInfo->>Y].
FORALL X,Y Y[projectInfo->>X] <- X[describesProject->>Y].

FORALL X,Y Y[developedBy->>X] <- X[develops->>Y].
FORALL X,Y Y[develops->>X] <- X[developedBy->>Y].
```

```
FORALL X,Y Y[supervises->>X] <- X[supervisor->>Y].
FORALL X,Y Y[supervisor->>X] <- X[supervises->>Y].

FORALL X,Y Y[affiliation->>X] <- X[employs->>Y].
FORALL X,Y Y[employs->>X] <- X[affiliation->>Y].
```

A.2 Ontologie für das Situationsmodell

```
User::null.
Device::null.
Task::Process.
Environment::null.
Situation::null.
Situation[
user=>User;
device=>Device;
environment=>Environment;
tasks=>>Task
].
User[
expertise=>STRING;
language=>>STRING;
preference=>>STRING;
interestedIn=>>Topic
].
Device[
displaySize=>STRING;
resolution=>STRING;
supportedFormat=>>STRING;
supportedContent=>>STRING
].

Environment[
location=>STRING;
nearObject=>>null;
time=>STRING
].
```

A.3 F-Logic-Regelwerk

```
// (1) Finde relevante Publikationen auf Basis von Interessen
FORALL publ,user,topic Relevant(publ) <-
       publ:Publication[keywords->>topic] AND user:User[interestedIn->>topic].

// (2) Finde relevante Events auf Basis von Ortsinformation
FORALL event, physStruct Relevant(event) <-
       event:Event[located->>physStruct] AND (In(physStruct) OR At(physStruct)).

// Publikationen zu einem relevanten Event sind auch relevant
FORALL event,publ Relevant(publ) <-
       Relevant(event:Event[publication->>publ:Publication]).

// Ebenso MultimediaContent, aber nur wenn das Geraet dieses darstellen kann
FORALL event, mmcontent Relevant(mmcontent) <-  Relevant(event:Event) AND
       event[presentation->>mmcontent] AND Displayable(mmcontent:MultimediaContent).

// (3) At Praedikat
FORALL physStruct,environm At(physStruct) <-
       environm:Environment[location->>physStruct:PhysicalStructure].

// Informationen ueber den aktuellen Aufenthaltsort sind relevant
FORALL  x Relevant(x) <- At(x).

// (4) In Praedikat
FORALL x,y In(y) <-
       (At(x) OR In(x)) AND y:PhysicalStructure[contains->>x:PhysicalStructure].

// (5) Bestimme relevante Projekte und Produkte auf der Basis von Interessen
FORALL prod, project, user, topic Relevant(prod), Relevant(project)  <-
       Near(product:Product) AND
       project:Project[product->>prod] AND
       project[isAbout->>topic:Topic] AND
       user:User[interestedIn->>topic].

// (6) Near Praedikat
FORALL Sit,Res,Loc Near(Res) <- Sit[location->>Loc]
       AND Res[location->>Loc].

// (7) Finde MultimediaContent, welches das Geraet darstellen kann
FORALL multimedia, device, format Displayable(multimedia) <-
       multimedia:MultimediaContent[format->>format] AND
       device:Device[supportedFormat->>format].

// (8) Bestimme MultimediaContent in einer Sprache, die der Nutzer versteht
FORALL multimedia, user, lang Understandable(multimedia) <-
       multimedia:MultimediaContent[language->>lang] AND
       user:User[language->>lang].

// (9) bestimme aktive Aufgaben
FORALL task Ongoing(task) <- task:TaskcurrentStatus->>literal("Ongoing")].

// (10) In einem Gespraech mit einer Person sind Informationen zu dieser relevant
FORALL task, person Relevant(person) <- Ongoing(task:TalkTo[whom->>person:Person]).
```

A.4 Transformation der DAML-Ontologie nach SQL-DDL

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/XSL/Transform/1.0"
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
 xmlns:daml="http://www.daml.org/2001/03/daml+oil#">

<xsl:template match="/">
<xsl:apply-templates/>
</xsl:template>

<xsl:template match="rdf:RDF">
    create table labels (id varchar(100) not null, label varchar(100) not null,
        replica int not null default 0,
        timestmp timestamp not null default current timestamp, primary key(id));
    create table resources  (id varchar(100) not null, type  varchar(100) not null,
        replica int not null default 0,
        timestmp timestamp not null default current timestamp, primary key(id));
    create table files  (id varchar(100) not null, filename varchar(100) not null,
        replica int not null default 0,
        timestmp timestamp not null default current timestamp, primary key(id));

    create table classes (id varchar(100) not null, subclassof varchar(100),
        primary key(id));
    create table properties (domain varchar(100),
        propertyname varchar(100), range varchar(100));
<xsl:apply-templates select="rdfs:Class"/>
<xsl:apply-templates select="//daml:Restriction"/>
</xsl:template>

<xsl:template match="rdfs:Class">
    insert into classes values (’<xsl:value-of select="@rdf:ID"/>’,
    <xsl:choose>
        <xsl:when test="rdfs:subClassOf[@rdf:resource]">
        ’<xsl:value-of select="substring-after(rdfs:subClassOf/@rdf:resource,’#’)"/>’
        </xsl:when>
        <xsl:otherwise>null</xsl:otherwise>
    </xsl:choose>);
    create table <xsl:value-of select="@rdf:ID"/>
        (identifier varchar(100) not null, primary key(identifier));
</xsl:template>

<xsl:template match="daml:Restriction">
    insert into properties values (’<xsl:value-of select="../../@rdf:ID"/>’,
    ’<xsl:value-of select="substring-after(daml:onProperty/@rdf:resource,’#’)"/>’,
    ’<xsl:value-of select="substring-after(daml:toClass/@rdf:resource,’#’)"/>’ );
    <xsl:choose>
    <xsl:when test="substring-after(daml:onProperty/@rdf:resource,’#’)=’abstract’">
        create table
            p_<xsl:value-of select="substring-after(daml:onProperty/@rdf:resource,’#’)"/>
            (subject varchar(100), object varchar(3800));
    </xsl:when>
    <xsl:otherwise>
        create table
            p_<xsl:value-of select="substring-after(daml:onProperty/@rdf:resource,’#’)"/>
            (subject varchar(100), object varchar(100));
    </xsl:otherwise>
    </xsl:choose>
</xsl:template>
</xsl:stylesheet>
```

Literaturverzeichnis

[AF00] Ernesto G. Arias and Gerhard Fischer. Boundary Objects: Their Role in Articulating the Task at Hand and Making Information Relevant to It. In *Intelligent Systems and Applications, 2000*, 2000. `http://www.cs.colorado.edu/~gerhard/papers/icsc2000.pdf`.

[BBD+00] Sean Bechhofer, Jeen Broekstra, Stefan Decker, Michael Erdmann, Dieter Fensel, Carole Goble, Frank van Harmelen, Ian Horrocks, Michael Klein, Deborah McGuiness, Enrico Motta, Peter F. Patel-Schneider, Steffen Staab, and Rudi Studer. An informal description of Standard OIL and Instance OIL, 2000. `http://www.ontoknowledge.org/oil/downl/oil-whitepaper.pdf`.

[BG01] Gerald Bieber and Martin Giersich. Personal Mobile Navigation Systems - Design Considerations and Experiences. In *Computers and Graphics*, pages 563–570, 2001.

[bib] The BibTeX Format. `http://www.ecst.csuchico.edu/~jacobsd/bib/formats/bibtex.html`.

[Bib93] Wolfgang Bibel. *Wissensrepräsentation und Inferenz*. Vieweg Verlag, 1993.

[blua] The BlueRose Project Homepage. `http://wwwdb.informatik.uni-rostock.de/global-info/`.

[blub] The BlueView Project Homepage. `http://wwwdb.informatik.uni-rostock.de/blueview/`.

[Bri] Dan Brickley. Resource Description Framework (RDF) Schema Specification 1.0. `http://www.w3.org/TR/PR-rdf-schema`.

[CK00] Guanling Chen and David Kotz. A Survey of Context-Aware Mobile Computing Research. Technical Report TR2000-381, Dept. of Computer Science, Dartmouth College, 2000. `http://citeseer.nj.nec.com/390713.html`.

[DA99] Anind K. Dey and Gregory D. Abowd. Towards a Better Understanding of context and context-awareness. Technical Report GIT-GVU-99-22, GIT, College of Computing Georgia Institute of Technology, College of Computing, 1999.

[DAM01a] DAML-S: Semantic Markup For Web Services, 2001. `http://www.daml.org/services/daml-s/2001/05/daml-s.pdf`.

[dam01b] DAML+OIL, DARPA Agent Markup Language Program, 2001. `http://www.daml.org/2001/03/daml+oil-index`.

[db2] IBM DB2 Universal Database. `http://ibm.com/software/data/db2`.

[db201] IBM DB2 Everyplace: A Small Footprint Relational Database System, 2001. `http://www-4.ibm.com/software/data/pubs/papers/db2e/db2e.pdf`.

[DBSA] Stefan Decker, Dan Brickley, Janne Saarela, and Juergen Angele. A Query and Inference Service for RDF. `http://www.w3.org/TandS/QL/QL98/pp/queryservice.html`.

[dcr01] Expressing Qualified Dublin Core in RDF / XML, 2001. `http://dublincore.org/documents/2001/08/29/dcq-rdf-xml/`.

[Dey99] Anind K. Dey. The Conference Assistant: Combining Context-Awareness with Wearable Computing. In *Proceedings of the 3rd International Symposium on Wearable Computers*, 1999.

[Dey00] Anind K. Dey. *Providing Architectural Support for Building Context-Aware Applications*. PhD thesis, GIT, College of Computing Georgia Institute of Technology, College of Computing, 2000.

[DFvH+00] Stefan Decker, Dieter Fensel, Frank van Harmelen, Ian Horrocks, Sergey Melnik, Michael Klein, and Jeen Broekstra. Knowledge Representation on the Web. In *Proceedings of the 2000 International Workshop on Description Logics (DL2000)*, pages 89–97, 2000. `http://www.cs.vu.nl/~dieter/ftp/paper/DL00-oil.pdf`.

[DM00] Stefan Decker and Sergey Melnik. A Layered Approach to Information Modeling and Interoperability on the Web. In *Proceedings of the ECDL 2000 Workshop on the Semantic Web*, 2000. `http://www-db.stanford.edu/~melnik/pub/sw00/sw00.pdf`.

[EF00] Albert Endres and Dieter W. Fellner. *Digitale Bibliotheken*. dpunkt.verlag, Heidelberg, 2000.

[Erd01] Michael Erdmann. *Ontologien zur konzeptuellen Modellierung der Semantik von XML*. PhD thesis, University of Karlsruhe, 2001.

[Fen00a] Dieter Fensel. Ontologies: Silver Bullet for Knowledge Management and Electronic Commerce, 2000.
`http://www.cs.vu.nl/~dieter/ftp/paper/silverbullet.pdf`.

[Fen00b] Dieter Fensel. Relating Ontology Languages and Web Standards, 2000.
`http://www.cs.vu.nl/~dieter/ftp/spool/mod2000.pdf`.

[Fen00c] Dieter Fensel. The Semantic Web and its languages. *IEEE Intelligent Systems*, 2000.
`http://www.cs.man.ac.uk/~horrocks/Publications/download/2000/faqs-on-oil.pdf`.

[FH01] E. Friedmann-Hill. JESS the Java Expert System Shell, 2001.
`http://herzberg.ca.sandia.gov/jess/`.

[FHKS96] Juergen Frohn, Reiner Himmeroeder, Paul-Th. Kandzia, and Christian Schlepphorst. How to Write F-Logic Programs in FLORID, 1996.
`ftp://informatik.uni-freiburg.de/pub/florid/tutorial.ps.gz`.

[FHvH+a] Dieter Fensel, Ian Horrocks, Frank van Harmelen, Stefan Decker, Michael Erdmann, and Michael Klein. OIL in a Nutshell.
`http://www.cs.vu.nl/~dieter/ftp/spool/oil.nutshell.pdf`.

[FHvH+b] Dieter Fensel, Ian Horrocks, Frank van Harmelen, Stefan Decker, Michael Erdmann, and Michael Klein. A brief Introduction to OIL.
`http://www.cs.vu.nl/~dieter/ftp/paper/oil.cuba.pdf`.

[Fis01] Gerhard Fischer. Articulating the Task at Hand and Making Information Relevant to It. In *HCI Journal Context Aware Computing*, 2001.
`http://www.cs.colorado.edu/~gerhard/papers/hci2001.pdf`.

[For82] C. L. Forgy. RETE: A fast match algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence*, 1982.

[Gia98a] Joseph C. Giarratano. CLIPS Reference Manual, 1998.
`http://www.ghg.net/clips/download/documentation/bpg.pdf`.

[Gia98b] Joseph C. Giarratano. CLIPS User Guide, 1998.
`http://www.ghg.net/clips/download/documentation/usrguide.pdf`.

[GR97] Joseph C. Giarratano and G. Riley. *Expert Systems: Principles and Programming*. PWS Publishing Company, Boston, 1997.

[Haa00] Peter Haase. Heterogeneous Data Replication. Thesis, University of Rostock, Germany, 2000.

[HD01] Peter Haase and Michael Drews. Komplexe Softwaresysteme: Digitale Bibliotheken - Ein Virtueller Dokumentenserver, 2001. `http://www.informatik.uni-rostock.de/~peter/ksws.pdf`.

[HMTP00] Andreas Heuer, Holger Meyer, Patrick Titzler, and Beate Porst. BlueView: Virtual Document Servers for Digital Libraries. In *Proceedings of the IEEE Conference on Advances in Digital Libraries*, 2000. `http://wwwdb.informatik.uni-rostock.de/blueview/paper/adl00.pdf`.

[HP99] Andreas Heuer and Denny Priebe. IRQL – Yet Another Language for Querying Semi-structured Data? Technical Report CS-01-99, University of Rostock, Germany, 1999. `http://e-lib.informatik.uni-rostock.de/fulltext/1999/preprint/cs-01-99-1999.ps`.

[HP01] Ian Horrocks and Jeff Z. Pan. Metamodeling Architecture of Web Ontology Languages. In *Proceedings of SWWS 2001*, 2001. `http://www.semanticweb.org/SWWS/program/full/paper11.pdf`.

[HS97] Andreas Heuer and Gunter Saake. *Datenbanken: Konzepte und Sprachen*. International Thomson Publishing, 1997.

[HS99] Andreas Heuer and Gunter Saake. *Datenbanken: Implementierungstechniken*. International Thomson Publishing, 1999.

[Hun01] Jane Hunter. Adding Multimedia to the Semantic Web: Building an MPEG-7 ontology. In *Proceedings of SWWS 2001*, 2001. `http://www.semanticweb.org/SWWS/program/full/paper59.pdf`.

[Ide00] Ruediger Ide. IrDA Beacon (TM) Transmitter, 2000. `http://www.rostock.igd.fhg.de/fhg_igd/abteilungen/a3/projects/irda/`.

[imt] Internet Media Types. `http://www.graphcomp.com/info/specs/mime.html`.

[j2e01] Java 2 Platform Enterprise Edition Specification, v1.3, 2001. `http://java.sun.com/j2ee/`.

[j2m01] Java 2 Platform Micro Edition (J2ME) Technology for Creating Mobile Devices, 2001. `http://java.sun.com/products/cldc/wp/KVMwp.pdf`.

[Jac90] Peter Jackson. *Introduction to Expert Systems*. Addison Wesley, 1990.

[Jon99] Michael Jonas. Ein Replikationsdienst für komplexe Dokumente. Master's thesis, University of Rostock, Germany, 1999.

[KBB+01] Meike Klettke, Mathias Bietz, Ilvio Bruder, Andreas Heuer, Denny Priebe, Günter Neumann, Markus Becker, Alexander Maedche, Steffen Staab, and Rudi Studer. GETESS - Ontologien, Objektrelationale Datenbanken und Textanalyse als Bausteine einer Semantischen Suchmaschine. *Datenbankspektrum*, 2001.

[KCI98] Thomas Kirste, Esteban Chavez, and Ruediger Ide. SAMoA: An experimental platform for Situation-Aware Mobile Assistance. In *Proceedings of the Workshop IMC 98*, pages 29–36, 1998. `http://www.rostock.igd.fhg.de/fhg_igd/abteilungen/a3/files/pdf/imc98-samoa.pdf`.

[KCI99] Thomas Kirste, Esteban Chavez, and Ruediger Ide. Interactive applications of personal situation-aware assistants. In *Computers and Graphics*, number 6, pages 903–915, 1999. `http://www.elsevier.nl/gej-ng/10/13/20/24/35/45/article.pdf`.

[KI00] Thomas Kirste and Ruediger Ide. The eGuide Exchange Format, 2000.

[Kir98] Thomas Kirste. Skizze der MOVI-Zugriffsarchitektur: Das generische Level-of-Detail-Model. Technical Report 98i022-FEGD, Fraunhofer Institute for Computer Graphics, Rostock, 1998.

[Kir00] Thomas Kirste. Ein Modell für situationsgesteuerte Mobile Assistenz, 2000.

[KLW95] Michael Kifer, Georg Lausen, and James Wu. Logical Foundations of Object-Oriented and Frame-Based Languages. *Journal of the ACM*, 1995. `http://www.acm.org/pubs/articles/journals/jacm/1995-42-4/p741-kifer/p741-kifer.pdf`.

[Kop01] Joe Kopena. DAMLJessKB, 2001. `http://plan.mcs.drexel.edu/projects/legorobots/design/software/DAMLJessKB/`.

[Kor01] Malte Korten. SensorFusion. Master's thesis, Hochschule Mittweida (FH), 2001.

[LH98] Astrid Lubinski and Andreas Heuer. Data Reduction - An Adaption Technique for Mobile Environments. In *Proceedings of the Workshop IMC 98*, pages 139–144, 1998.
`http://wwwdb.informatik.uni-rostock.de/~lubinski/artikel/imc98.ps`.

[LH00] Astrid Lubinski and Andreas Heuer. Configured Replication for Mobile Applications. In *4th IEEE International Baltic Workshop*, 2000.
`http://wwwdb.informatik.uni-rostock.de/~lubinski/artikel/baltic00.ps`.

[LS] Ora R. Lassila and Ralph R. Swick. Resource Description Framework (RDF) Model and Syntax Specification W3C Recommendation.

[Lub00] Astrid Lubinski. Small Database Answers for Small Mobile Resources. In *Proceedings of the Workshop IMC 2000*, pages 142–149, 2000.
`http://wwwdb.informatik.uni-rostock.de/~lubinski/artikel/imc00.ps`.

[McG] Deborah McGuiness. An Axiomatic Semantics for RDF, RDF-S, and DAML+OIL.
`http://www.daml.org/2001/03/axiomatic-semantics.html`.

[Mel00] Sergey Melnik. Storing RDF in a relational database, 2000.
`http://www-db.stanford.edu/~melnik/rdf/db.html`.

[mys] MySQL.
`http://www.mysql.com/`.

[Per88] M. Perlin. On the computational equivalence of frames and rules. Technical report, Carnegie-Mellon University, 1988.

[poc01] Pocket Slides, 2001.
`http://www.conduits.com/products/slides/`.

[pri01] Primer PDF Viewer, 2001.
`http://www.ansyr.com/products/windows/primer31/`.

[RNBY99] Berthier Ribeiro-Neto and Ricardo Baeza-Yates. *Modern Information Retrieval*. Addison Wesley, 1999.

[Rus01] Matthias Rust. MPEG7: Metadaten in Multimediadatentypen (work in progress). Master's thesis, University of Rostock, Germany, 2001.

[SBG98] Albrecht Schmidt, Michael Beigl, and Hans-W. Gellersen. There is more to context than location. In *Proceedings of the International*

Workshop on Interactive Applications of Mobile Computing (IMC98), Rostock, Germany, number 6, pages 893–901, 1998.

[Sch94] Bill N. Schilit. Context-aware computing applications. In *Proceedings of IEEE Workshop on Mobile Computing Systems and Applications*, pages 85–90, 1994.
`http://www.fxpal.com/people/schilit/wmc-94-schilit.pdf`.

[Sch95] Bill N. Schilit. *A System Architecture for Context-Aware Mobile Computing*. PhD thesis, Columbia University, 1995.

[SDA99] Daniel Salber, Anind K. Dey, and Gregory D. Abowd. The Context Toolkit: Aiding the Development of Context-Enabled Applications. In *Proceedings of CHI99*, pages 434–441, 1999.
`http://citeseer.nj.nec.com/salber99context.html`.

[sem01] The Semantic Web Project Homepage, 2001.
`http://www.semanticweb.org/`.

[SEMD00] Steffen Staab, Michael Erdmann, Alexander Maedche, and Stefan Decker. An Extensible Approach for Modeling Ontologies in RDF(S), 2000.
`http://www.aifb.uni-karlsruhe.de/~sst/Research/Publications/onto-rdfs.pdf`.

[swr] The Semantic Web Research Ontology.
`http://www.semanticweb.org/ontologies/`.

[tex00] *DB2 Text Extender Administration and Programming.* IBM Corporation, 2000.

[tim] The Time Ontology.
`http://www.ai.sri.com/daml/ontologies/sri-basic/1-0/Time.daml`.

[Tit99] Patrick Titzler. Entwurf und Implementierung eines Virtuellen Dokumenten-Servers. Master's thesis, University of Rostock, Germany, 1999.

[TS99] David Tanzer and Dennis Shasha. Queryable acyclic production systems. In *Proceedings of the eighth international conference on Information knowledge management*, pages 284–291, 1999.
`http://www.acm.org/pubs/articles/proceedings/cikm/319950/p284-tanzer/p284-tanzer.pdf`.

[Ull88] Jeffrey D. Ullmann. *Principles of Database and Knowledge-Base Systems*. Computer Science Press, 1988.

[URI] Uniform Resource Identifiers (URI): Generic Syntax. `http://www.ietf.org./rfc/rfc2396.txt`.

[vHPSH] Frank van Harmelen, Peter F. Patel-Schneider, and Ian Horrocks. A Model-Theoretic Semantics for DAML+OIL (March 2001). `http://www.daml.org/2001/03/model-theoretic-semantics.html`.

[vHPSH01] Frank van Harmelen, Peter F. Patel-Schneider, and Ian Horrocks. Reference description of the DAML+OIL (March 2001) ontology markup language, 2001. `http://www.daml.org/2001/03/reference.html`.

Abbildungsverzeichnis

Tabellenverzeichnis

Thesen

1. Digitale Bibliotheken spielen eine herausragende Rolle in der Versorgung mit Wissen. Um den Zugang zu digitalen Inhalten auch in mobilen Umgebungen adäquat zu gestalten, sind neue Konzepte notwendig.

2. Durch Ausnutzung von Kontextinformation kann eine neue Qualität in der Interaktion zwischen Mensch und Computer erreicht werden, indem situationsrelevante Informationen proaktiv bereitgestellt werden.

3. Für die Beschreibung der Inhalte der Bibliothek und der Situation des Nutzers ist der Einsatz von Techniken der ontologiegestützten Wissensrepräsentation sinnvoll.

4. Die Sprachen des Semantic Web sind aufgrund ihrer Modellierungsfähigkeiten, Interoperabilität und Erweiterbarkeit für eine ontologiegestützte, wissensbasierte Datenmodellierung besonders gut geeignet.

5. Regelbasierte Anfragesysteme wie Frame-Logic sind geeignet, um auf der Basis einer wissensbasierten Beschreibung von Situation und Inhalten, situationsrelevante Inhalte zu bestimmen.

6. Eine vollständige Analyse der Nutzersituation erfordert die Erfassung der physischen Umgebung des Nutzers, seiner Aufgaben sowie die Berücksichtigung von Nutzer- und Geräteprofilen.

7. Verteilte Datenhaltung und eine verteilte Realisierung des Anfrage- und Retrievalprozesses ermöglichen eine hohe Verfügbarkeit der Daten z.B. auch bei unterbrochenen Netzwerkverbindungen.

8. Die Verwendung der Techniken von Level of Detail, Replikation und Caching erlauben ein effizientes Retrieval von Informationen.

9. Die prototypische, Java-basierte und damit plattformunabhängige Umsetzung der Systemarchitektur *MoBibel* demonstriert den möglichen Einsatz eines mobilen Assistenzsystems bei Ausstellungs- und Konferenzbesuchen.

10. Durch die ontologiebasierte, von der Systemarchitektur unabhängige Datenmodellierung ist die Anwendung des Systems in anderen Domänen sehr einfach: Es müssen lediglich die Ontologie und die Inferenzregeln neu entworfen werden.

Erklärung

Ich erkläre, dass ich die vorliegende Arbeit selbständig und nur unter der Vorlage der angegebenen Literatur und Hilfsmittel angefertigt habe.

Rostock, den 7.11.2001

Peter Haase