Universität des Saarlandes
Fachbereich 6.2 Informatik
Lehrstuhl für Künstliche Intelligenz

**Dissertation**

# DOMeMan: Repräsentation, Verwaltung und Nutzung von digitalen Objektgedächtnissen

Dissertation
zur Erlangung des Grades
Doktor der Ingenieurwissenschaften (Dr.-Ing.)
der Naturwissenschaftlich-Technischen Fakultät I
der Universität des Saarlandes

Jens Haupert
(jens.haupert@dfki.de)

Saarbrücken, 5. Februar 2013

**Dekan:**
Prof. Dr. Mark Groves

**Vorsitzender des Prüfungsausschusses:**
Prof. Dr. Philipp Slusallek

**Berichterstatter:**
Prof. Dr. rer. nat. Dr. h.c. mult. Wolfgang Wahlster
Prof. Dr. Wolfgang Maass
Dr. Alexander Kröner

**Akademischer Beisitzer:**
Dr. Boris Brandherm

**Tag des Kolloquiums:**
16. Januar 2013

# Eidesstattliche Erklärung

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus anderen Quellen oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet.

Die Arbeit wurde bisher weder im In- noch im Ausland in gleicher oder ähnlicher Form in einem Verfahren zur Erlangung eines akademischen Grades vorgelegt.

Saarbrücken, den 5. Februar 2013 _____

Unterschrift

# Danksagung

Diese Arbeit entstand im Rahmen der vom Bundesministerium für Bildung und Forschung (BMBF) geförderten Projekte SemProM und RES-COM und des vom saarländischen Ministerium für Wirtschaft und Wissenschaft geförderten Projekts PIZZA am Deutschen Forschungszentrum für Künstliche Intelligenz (DFKI).

Zu der Entstehung dieser Arbeit haben viele Personen durch ihre direkte und indirekte Unterstützung beigetragen. Ihnen gebührt mein Dank!

Ich danke meinem Doktorvater Prof. Wolfgang Wahlster für die Möglichkeit, in seiner Gruppe am Deutschen Forschungszentrum für Künstliche Intelligenz mitarbeiten und meine Doktorarbeit anfertigen zu können. Vor allem in den letzten Phasen meiner Arbeit unterstütze er mich mit konstruktiven Diskussionen in ausgesprochen angenehmer Atmosphäre und lieferte wertvolle Denkanstößen zur Verbesserung meiner Arbeit. Prof. Wolfgang Maass danke ich für seine Bereitschaft, das Zweitgutachten für diese Arbeit zu verfassen. Dr. Alexander Kröner danke ich für seine Bereitschaft, das Drittgutachten für diese Arbeit zu verfassen.

Des Weiteren bedanke ich mich bei meinen Arbeitskollegen am DFKI für die angenehme Arbeitsatmosphäre, für interessante und anregende Diskussionen und für die spannende und produktive Zusammenarbeit. Zusätzlich danke ich ihnen für das Verständnis und ihre Rücksichtnahme während des Niederschreibens und der heißen Schlussphase meiner Arbeit. Ohne die von Kollegen generierten Freiräume wären die zugrundeliegende Forschung sowie das Verfassen der Arbeit in dieser Form nur schwer möglich gewesen.

Für das Teilen meines Leids in schwierigen Phasen danke ich allen ehemaligen und aktuellen Doktoranden und Mitarbeitern am Lehrstuhl von Prof. Wahlster und seiner Arbeitsgruppe am DFKI.

Ein besonderer Dank gilt zum Schluss meiner Familie und meinen Freunden, die mich jederzeit nach Kräften unterstützt haben. Ohne ihre moralische und praktische Unterstützung hätte diese Arbeit nicht in dieser umfassenden Form entstehen können.

Saarbrücken, Februar 2013
Jens Haupert

# Zusammenfassung

Diese Arbeit beschäftigt sich mit der Frage, wie eine *Infrastruktur für digitale Objektgedächtnisse* aussehen muss. Ziel der Arbeit ist die Erstellung eines Architekturkonzepts für die Repräsentation und die Verwaltung von, sowie den Zugriff auf digitale Objektgedächtnisse inklusive abgestimmter Werkzeuge, die die Erstellung neuer und die Migration bestehender Systeme erleichtern.

Aufgrund der heterogenen Datenlandschaft in einem „open-loop"-Szenario, werden an ein Objektgedächtnis besondere Anforderungen gestellt. Zum einen muss es flexible genug ausgelegt sein, um mit den unterschiedlichsten Daten umgehen zu können. Zum anderen muss ein einfacher und strukturierter Zugriff auf die Daten ermöglicht werden, der je nach Anwendungsfall auch mit Konzepten des Rechte- und Rollen-basierten Zugriffs und der Versionsverwaltung ergänzt werden muss.

Diese Arbeit zeigt daher zuerst ein *Datenmodell* zur Strukturierung von Objektgedächtnisdaten. Dieses erlaubt mit Hilfe von Metadaten auch in heterogenen Szenarien einen Domänen-übergreifenden Datenaustausch. Im Anschluss wird eine Softwarearchitektur vorgestellt, die eine *Speicher- und Zugriffslösung für Objektgedächtnisse* anbietet und die Gedächtnisse zusätzlich mit einer *Aktivitätskomponente* ausstattet. Abgerundet wird das Konzept mit einem Satz an Werkzeugen, die es Entwicklern erlauben, zum Beispiel bestehende Datenbestände zu migrieren, neue Strukturen auf Basis von semantischen Daten aufzubauen und diese mit *anwendungsspezifischen Visualisierungen* dem Benutzer wieder zugänglich zu machen.

# Abstract

This thesis addresses the research question, how an *infrastructure for digital object memories* has to be designed. Primary goal of this thesis is to identify and develop components and processes of an architecture concept particularly suited to represent, manage, and use digital object memories. In order to leverage acceptance and deployment of this novel technology, the envisioned infrastructure has to include tools for integration of new systems, and for migration with existing systems.

Special requirements to object memories result from the heterogeneity of data in so-called open-loop scenarios. On the one hand, it has to be flexible enough to handle different data types. On the other hand, a simple and structured data access is required. Depending on the application scenario, the latter one needs to be complemented with concepts for a rights- and role-based access and version control.

First, this thesis provides a *data model* for structuring object memory data by means of meta data, which enables cross-domain data exchange in heterogeneous scenarios. Then, a software architecture concept will be introduced, which provides means of *storing and accessing memory data*, and integrates an *activity component* into object memories. Finally, the concept is completed by a toolset that enables developers to migrate existing datasets, to create new structures based on semantic information, and to support user interaction with this data by means of *application-specific visualizations*.

# Inhaltsverzeichnis

KAPITEL 1

# Einleitung

## 1.1 Motivation

Entwicklungen der letzten Jahre im Bereich der Kommunikations- und Informationstechnologien erlauben es heute realen Objekten eine digitale Identität zu geben, das sogenannte *Internet of Things* (IoT) oder *Internet der Dinge* war geboren (aktuelle Entwicklungen siehe [AIM10, Cha10]). In der ersten Phase des IoT wurden Objekte der realen Welt lediglich mit einer eindeutigen Identifikationsnummer (ID) ausgestattet, welche zum Beispiel über optische Marker (Strichcode oder 2D-Barcode) oder funkbasierte Technologien (z.B. NFC oder RFID) realisiert wurden.

Somit war eine automatische Erfassung und Verarbeitung möglich, um an vielen Stellen im Warenfluss genutzt zu werden. Doch die Ansprüche des Internets der Dinge forderten weitergehende Anwendungen [UHM11]. Sogenannte *Smart Labels* erlauben nun eine Objektzentrierte Mensch-Maschine-Interaktion und Maschine-zu-Maschine (M2M) Kommunikation. Dazu werden über die bisher erwähnte Möglichkeiten der eindeutigen Identifikation auch Sensoren direkt am Objekt mit Hilfe des Labels angebracht. Daher kann das Objekt während seines gesamten Lebenszyklus (von der Produktion, über die Benutzung bis zum Lebensende und Recycling) auch ohne eine vorhandene intelligente und instrumentierte Umgebung selbstständig seinen Zustand überwachen.

Diese Entwicklung gipfelt schließlich in der Ausprägung des *digitalen Produktgedächtnisses* (DPG) [WKSB08], welches das mit Sensorik versehene Smart Label mit einer zusätzlichen Gedächtnisfunktionalität ausstattet und somit alle Aspekte bietet, um den Weg frei zu machen für sogenannte *open-loop* Szenarien, bei denen das Objekt mitsamt seines DPGs auf viele unterschiedliche Teilnehmer der Produktlebenszykluskette trifft (wie zum Beispiel Hersteller und Zulieferer, Logistiker und Transporteure, Verkäufer und Einzelhändler sowie Benutzer und Verbraucher und schließlich Entsorger und Verwerter), mit denen es Daten austauschen kann. Somit werden Alltagsgegenstände durch die neuen Möglichkeiten des

DPG veredelt. Beispielhafte Szenarien finden sich dabei in den Bereichen der industriellen Produktion [SMK+10], der kontinuierlichen Überwachung des $CO_2$-Verbrauchs von Produkten [KKS+10] sowie der mobilen Kundenunterstützung [KGB+11].

## 1.2 Ziele dieser Arbeit

### 1.2.1 Problemdarstellung

Das Ziel dieser Arbeit ist es, eine Architektur zu entwickeln, die es erlaubt physische Objekte mit digitalen Gedächtnissen auszustatten. Dadurch ist es auf der einen Seite möglich, dass Objekte relevante Daten in ihrem Gedächtnis sammeln können und auf der anderen Seite ändert sich das Kommunikationsparadigma für diese Objekte dergestalt, dass die Daten nicht mehr via Backend von Partner zu Partner transferiert werden, sondern das physische Objekt seine Daten für jeden Partner bereitstellt. Dies kann sowohl über eine direkte Ausrüstung des Objektes mit einem eingebetteten System oder durch die Anbringung einer Referenz zu einem Backendsystem erfolgen. Aufbauend auf dieser Architektur wird ein Software-Rahmenwerk erstellt, welche eine Infrastruktur für Objektgedächtnisse ermöglicht. Zusätzliche Werkzeuge erlauben es, sowohl bereits bestehende Daten leicht in das neue Kommunikationsparadigma zu überführen als auch gänzliche neue Daten auf einer semantischen Grundlage zu generieren. Darauf aufbauend erhält das Gedächtnis eine sogenannte aktive Komponente, die eigenständige Verarbeitungslogik innerhalb des Gedächtnisses zur Ausführung zu bringen.

### 1.2.2 Wissenschaftliche Fragen

Folgende Aufstellung zeigt die wesentlichen wissenschaftlichen Fragestellungen, die in dieser Arbeit betrachtet werden:

1. **Wie kann ein strukturelles Datenmodell den speziellen Anforderungen von digitalen Objektgedächtnissen Rechnung tragen?**
   Stattet man Objekte mit einem digitalen Gedächtnis aus, so sammeln diese (eigenständig oder mit Hilfe einer externen Infrastruktur) in einem open-loop Prozess Daten über sich selbst auf. Allerdings können keine Annahmen bezüglich vorhandener Daten über den vorherigen und den folgenden Partner getroffen werden. Daher führt jeder Zugriff auf ein unbekanntes Gedächtnis nicht zu einem direkten Zugriff auf die Daten, sondern immer zu einem Suchvorgang nach Daten, die den gewünschten Kriterien entsprechen. Aus diesem Grund ist ein spezielles Strukturmodell notwendig,

welches die heterogenen Gedächtnisdaten aufnimmt und mit Hilfe von zusätzliche Metadaten diesen Suchprozess ermöglicht.

2. **Wie kann ein Benutzer mit der Absicht digitale Objektgedächtnisse zu nutzen durch Werkzeuge bei seiner Aufgabe unterstützt werden?**
   Zur Nutzung digitaler Objektgedächtnisse ist zum einen eine gewisse Grundinfrastruktur und zum anderen auch ein Satz an zusätzlichen Hilfswerkzeugen notwendig. Dabei soll, basierend auf dem erwähnten Strukturmodell, sowohl eine Smart Label-basierte als auch eine Server-basierte Infrastrukturlösung erstellt werden. Zusätzlich ist ein Satz an Werkzeuge notwendig, mit deren Hilfe bestehende Daten (zum Beispiel aus Datenbanken) konvertiert, semantische Daten leicht erstellt, Struktur- und Syntaxprüfungen durchgeführt und Daten für Endbenutzer visualisiert werden können.

3. **Wie lassen sich Objektgedächtnisse mit Aktivität ausstatten, so dass diese extern angestoßene oder eigenständige Operationen durchführen können?**
   Durch ihren heterogenen Datenbestand kann es vorteilhaft sein, wenn digitale Objektgedächtnisse die Logik für gewisse Operationen selbst mitführen und diese Operationen auch eigenständig durchführen. Zu diesem Zweck soll die angedachte Infrastrukturlösung ein weiteres Modul erhalten, welches es erlaubt jedes Gedächtnis mit zusätzlichen Skripten zu erweitern, um solche eigenständigen Operationen durchführen zu können. Diese Operationen sollten sowohl von extern gestartet werden als auch eigenständig durch einen Timer oder ein Ereignis (zum Beispiel durch eine Änderung am Gedächtnis) ausgelöst werden.

4. **Wie sieht eine ganzheitliche Architektur zur Speicherung von digitalen Objektgedächtnissen aus?**
   Führt man die bisher beschriebenen Ansätze zur Erstellung eines Strukturmodells mit der erstellten Infrastruktur samt Werkzeugen und aktiven Komponenten zu einer Architektur zusammen, in der alle beteiligten Komponenten aufeinander abgestimmt sind, steht dem Nutzer ein ganzheitlicher Ansatz zur Verfügung, um bisherige konventionelle oder zukünftige Prozesse mit den Vorteilen und Möglichkeiten der offenen und flexiblen Objektgedächtnisse auszustatten.

## 1.3 Aufbau der Arbeit

Diese Arbeit besteht aus acht einzelnen Kapiteln, deren Zusammenhang in der Abbildung 1.1 dargestellt sind. Die folgende Aufzählung vermittelt einen kurzen Eindruck, welche Informationen im jeweiligen Kapitel präsentiert werden:

1. Das erste Kapitel *Einleitung* beinhaltet eine einführende Motivation für digitale Objektgedächtnisse, bietet eine Problemdarstellung, listet den wissenschaftlichen

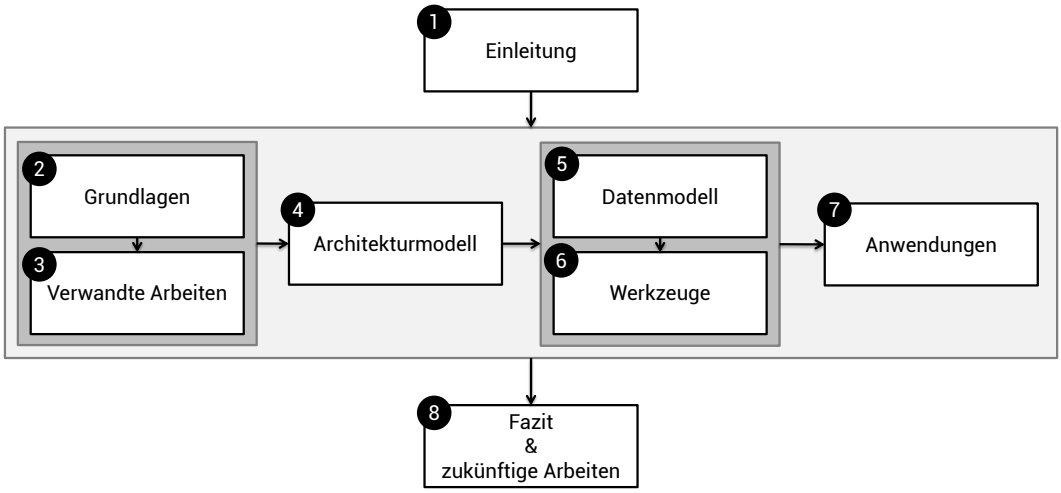



Abbildung 1.1: Gliederung dieser Arbeit

Beitrag dieser Arbeit auf und schließt mit einer Übersicht über die Gliederung der Arbeit.

2. Das folgende Kapitel *Grundlagen* zeigt dem Leser eine Übersicht über wichtige Fachtermini, die für das Verständnis dieser Arbeit notwendig sind, zum Beispiel aus dem Bereich des „Ubiquitous Computung“ oder des semantischen Webs. Anschließend werden die Anforderungen an die Architektur, die in dieser Arbeit entwickelt wird, definiert.

3. Eine Auswahl an aktuellen *verwandten Arbeiten* wird in diesem Kapitel präsentiert. Diese sind zum leichteren Verständnis in die Kategorien: „Forschungsprojekte“, „Industrieprojekte“, „Ontologien und Metadaten“, „Ontologie-Editoren“, „Datenspeicherung“ und „Datenvisualisierung“ gegliedert. Das Kapitel schließt mit einer Bewertung dieser verwandten Arbeiten und einer Schlussfolgerung, die sich direkt auf das Architekturmodell dieser Arbeit auswirkt.

4. Im Kapitel *Architekturmodell* wird das im Rahmen dieser Arbeit entwickelte *DOMe-Man*-Konzept (Digital Object Memory Managment) eingeführt. Dazu werden zuerst zwei grundlegende Szenarien beschrieben, die als Ausgangsbasis für die Entwicklung der Architektur dienen. Anschließend werden die einzelnen Komponenten in einer abstrakten Übersicht vorgestellt.

5. Dieses Kapitel widmet sich dem strukturierenden *Datenmodell*, welches verwendet wird um Gedächtnisdaten für Objekte zu partitionieren. Dazu werden die Strukturelemente beschrieben, bereits definierte Formate näher erläutert und unterschiedliche Repräsentationen vorgestellt. Zusätzlich werden die im Rahmen dieser Arbeit entwickelten Ontologien für Produktdaten und Personalisierung beschrieben. Das

Kapitel schließt mit einem kurzen Exkurs zur Aufnahme von regelbasierten Daten in Objektgedächtnissen.

6. Im Kapitel *Werkzeuge* werden die in dieser Arbeit entwickelten Softwaremodule vorgestellt, mit deren Hilfe eine Infrastruktur für digitale Objektgedächtnisse aufgebaut werden kann. Die einzelnen Module befassen sich mit den Anforderungen Daten für Objektgedächtnisse (sowohl aus bestehenden als auch aus neuen Quellen) zu generieren, die Konsistenz der Daten sicherzustellen, Daten sowohl serverbasiert als auch direkt am Produkt speichern zu können und diese anschließend für den Benutzer zu visualisieren.

7. Die im Rahmen dieser Arbeit entwickelten und umgesetzten *Anwendungen* werden in diesem Kapitel dargestellt. Die Arbeiten zeigen Demonstratoren und Exponate aus dem Projekt SemProM, welches mit einem Informationskiosk dem Benutzer den Zugang zu Objektgedächtnisdaten im Rahmen unterschiedlicher Szenarien ermöglicht und die Nutzung von aktiven Komponenten in der Fabrik der Zukunft. Des Weiteren werden Arbeiten im Bereich des Supermarkts der Zukunft gezeigt, die ebenfalls auf der in dieser Arbeit vorgestellten Architektur basieren.

8. Im abschließenden Kapitel *Fazit & zukünftige Arbeiten* werden die in dieser Arbeit vorgestellten Ansätze und Konzepte noch einmal zusammenfassend präsentiert und auf die wissenschaftlichen und praktischen Beiträge hingewiesen. Zusätzlich werden Veröffentlichungen dargelegt, die im Rahmen dieser Arbeit erfolgt sind. Das Kapitel schließt mit einer Übersicht über zukünftige Arbeiten.

KAPITEL 2

# Grundlagen und Anforderungen

In diesem Kapitel werden die grundlegenden Begrifflichkeiten und Konzepte erläutert, bevor in den folgenden Kapiteln verwandte Arbeiten und die Ergebnisse dieser Arbeit dargelegt werden. Da sich ein Großteil dieser Arbeit mit Vorgängen in instrumentierten Umgebungen beschäftigt, wird dieser Begriff im Kapitel 2.1 definiert. Eine Grundvoraussetzung zur Erfassung und Nutzung von Objekten in solchen Umgebungen ist die eindeutige Identifizierung der Objekte. Daher werden im Kapitel 2.2 unterschiedliche Konzepte zur Annotation von Objekten vermittelt. Eine wichtige Zielrichtung für solche intelligenten Systeme ist die vollständige Integration in bestehende Objekte und Abläufe. Dieser dem Thema „Ubiquitous Computing“ zugeordnete Bereich wird in Kapitel 2.3 diskutiert. Aktuelle Bestrebungen versuchen ein intelligentes und aktives Verhalten nicht nur in die Umgebung, sondern auch direkt in Objekte zu integrieren, die mit eigener Sensorik ihre Umgebung selbst überwachen. Details zu solchen sogenannten intelligenten Objekten (Smart Objects) werden in Kapitel 2.4 präsentiert. Solche intelligente Systeme sollen ihre Daten nach Möglichkeit so ablegen, dass diese maschinell verstanden werden können. Diese Anforderung kann mit Hilfe von Techniken aus dem semantischen Web umgesetzt werden, die in Kapitel 2.5 beschrieben werden. Die so generierten Daten müssen nun auch effizient in Datencontainer abgelegt werden, so dass es externen Anwendungen erleichtert wird, passende Daten zu finden und zu nutzen. Aus diesem Grund ist die Art der Datenspeicherung von Interesse und wird daher in Kapitel 2.6 diskutiert. Die gesamte Architektur bildet somit ein Rahmenwerk zur Unterstützung des gesamten Produktlebenszyklus, welcher in Kapitel 2.7 näher definiert wird. Kapitel 2.8 zeigt eine Taxonomie unterschiedlicher Ausprägungen von Objektgedächtnissen. Abschließend wird in Kapitel 2.9 eine Liste von Anforderungen definiert, die eine Zielarchitektur erfüllen sollte. Diese Anforderungen werden dann im folgenden Kapitel 3 mit bestehenden und verwandten Arbeiten abgeglichen.

## 2.1 Instrumentierte Umgebungen

Der Begriff der *Instrumentierten Umgebung* beschreibt einen Ausschnitt der realen physischen Welt, der samt einiger oder aller sich darin befindenden Objekten instrumentiert wurde, um dem Benutzer einen Mehrwert anbieten zu können (siehe [Sch10]). Um diese Anforderung zu erreichen werden *Sensoren* installiert, die in der Lage sind den Zustand und die Veränderung der Umgebung zu erkennen und die Anwesenheit und die Handlungen von Benutzern festzustellen. Zusätzlich stehen Ausgabegeräte zur Verfügung, die entweder Informationen an Benutzer übermitteln oder direkt die Umgebung verändern können (z.B. Bildschirme, Lautsprecher oder haptische Bedienelemente). Diese werden als *Aktuatoren* bezeichnet. Zusätzlich zu diesen realen Objekten ist eine virtuelle Zwischenschicht notwendig, die die Verarbeitung von Daten übernimmt, um die Mehrwerte realisieren zu können. Diese, in der Regel in Software umgesetzte Schicht, besteht aus unterschiedlichsten Verarbeitungsmodulen. Zusätzlich werden sowohl intern ermitteltes als auch von extern bezogenes Wissen und externe Dienste verwendet, um Verarbeitungsschritte durchführen zu können. Die eigentliche Umgebung und die virtuelle Verarbeitungsschicht sind dabei über eine Kommunikationsverbindung miteinander verbunden, um Daten und Befehle austauschen zu können (siehe Abbildung 2.1). Die Übergänge zwischen instrumentierten Objekten, der Umgebung und der Verarbeitungsschicht sind dabei sehr fließend. Ein Beispiel am anderen Ende der Skala wäre ein intelligentes Objekt, welches Sensoren, Verarbeitung und Aktuatoren in einem Objekt vereint (siehe Kapitel 2.4).

## 2.2 Objektidentifizierung

Ein zentrales Element von intelligenten Umgebungen ist die eindeutige Identifizierbarkeit solcher physischen Objekte (im Sinne des Internets der Dinge, siehe [AIM10, Cha10]). Dabei ist zum einen eine eindeutige Kodierung für jede physische Instanz notwendig, um zum Beispiel *Lampe123* von *Lampe124* unterscheiden zu können, obwohl beide Inkarnationen des identischen Lampentyps darstellen. Zusätzlich muss eine solche eindeutige Identifikationsnummer direkt an den Objekten angebracht sein, damit die Umgebung die Objekte identifizieren kann, um sie in diese zu integrieren. Zur Umsetzung einer solchen Auszeichnung stehen verschiedene Verfahren zur Verfügung. Unabhängig von der technischen Umsetzung unterscheiden sich diese in der Fähigkeit einmal geschriebene Daten nur noch lesen oder auch wieder verändern und löschen zu können. Einfachere Verfahren sind in der Regel nur noch lesbar, bei aufwendigeren Lösungen ist ein Schreiben möglich. Diese Fähigkeit kann aber je nach Anwendungsfall eingeschränkt beziehungsweise deaktiviert sein, um gestellte Anforderungen zu erfüllen. Im Folgenden werden nun die für diese Arbeit wichtigsten Identifizierungstechniken vorgestellt.

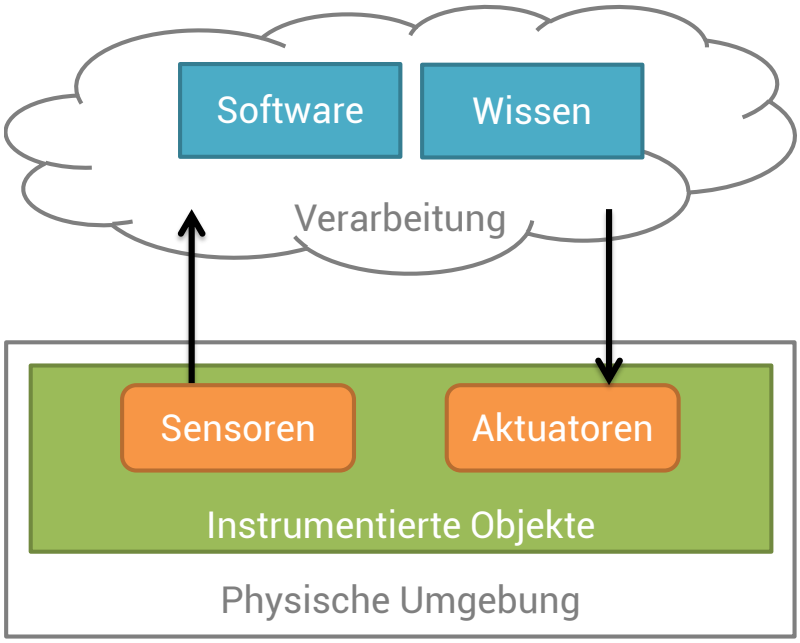



Abbildung 2.1: Aufbau und Zusammenhang von Hard- und Software in intelligenten Umgebungen nach [Sch10]

### 2.2.1 Identifikationscode

Die einfachste Form der Objektidentifizierung stellt ein Code dar, der auf dem Objekt angebracht wird, so dass ein Mensch diesen lesen kann und mit Hilfe eines Datenbestands, der konventionell in Papierform (zum Beispiel eine Tabelle) oder digital vorliegen kann (zum Beispiel eine Datenbank), eine Abbildung von diesem Code zu bestimmten Eigenschaften des Objekts herstellen kann. Diese Codes werden bereits seit mehreren Jahrzehnten in der Industrie eingesetzt, zum Beispiel in Form von Chargennummern auf Medikamentenverpackungen, Bauteilen oder Lebensmitteln (siehe Abbildung 2.2).

Ebenfalls als ein eindeutiger Identifikationscode angedacht ist eine IPv6-Adresse [DH98]. Durch den großen Namensraum ist es theoretisch möglich $\approx$ 340 Sextillionen Geräte zu adressieren, so dass jedes Objekt mit einer eindeutigen IPv6-Adresse ausgestattet werden könnte. Dieses Verfahren bietet allerdings nur eine Adressierung der Objekte und keinen direkten Hinweis über welchen Kanal und welches Protokoll mit diesen kommuniziert werden kann.

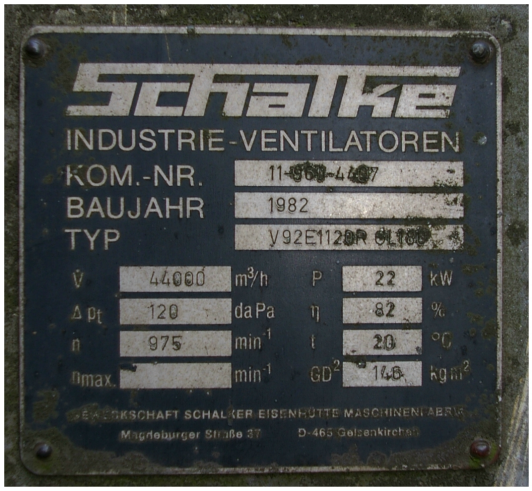



Abbildung 2.2: Konventionelles und nicht maschinenlesbares Typenschild[1]

### 2.2.2 Maschinenlesbarer Code

Ein Barcode (engl. bar „Balken“) ist eine optoelektronisch lesbare Schrift, die aus verschieden breiten, parallelen Strichen und Lücken besteht, mit dessen Hilfe Daten abgebildet werden können. Diese Daten können leicht auf den unterschiedlichsten physischen Objekten platziert werden, zum Beispiel durch direktes Bedrucken oder durch Anbringen eines Aufklebers. Der Barcode kann im Gegensatz zu den bereits genannten einfachen Identifikationscodes, die nur menschenlesbar sind, dann von Barcodelesern erfasst und von entsprechenden System elektronisch verarbeitet werden ([Ros97].


Es existieren unterschiedliche Formen der Codierung solcher Barcodes. Man unterscheidet zum einen nach dem repräsentierten Inhalt und zum anderen nach der Darstellungsform, wobei beide Merkmale eng miteinander verwandt sind. Der klassische eindimensionale (1D) Barcode findet sich heute auf praktisch allen Produkten. Er stellt in der Regel eine Zahl in Form von Balken dar und wird vor allem im Handel genutzt. Typische Vertreter sind die in Europa verwendeten *EAN-8* und *EAN-13* Codes mit der Angabe einer Produktklasse als 8- oder 13-stelligen nummerischen Wert. Zusätzlich lassen sich heute beliebige alphanumerische Werte als 1D-Barcode darstellen (siehe zum Beispiel *Code128*).

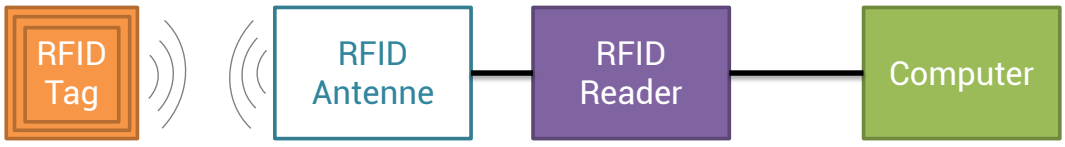
[1] Quelle: `http://de.wikipedia.org/w/index.php?title=Datei:Typschild-Schalke.jpg` [Letzter Zugriff: 26.11.2012]

Eine Weiterentwicklung stellen zweidimensionale (2D) Barcodes dar, bei denen die Daten über eine Fläche verteilt werden, die aus verschieden breiten Strichen oder Punkten und dazwischen liegenden Lücken mit möglichst hohem Kontrast bestehen [Len00, FCD05]. Mit solchen Codes lassen sich beliebige Informationen codieren, wodurch sich diese Technik für die unterschiedlichsten Anwendungen eignet. Typische Vertreter sind der aus Japan stammende *Quick-Response (QR)-Code* [Int06a] oder der *DataMatrix-Code* [Int06b]. Beiden Systemen gemein ist der hohe Grad an Redundanz, wodurch die Robustheit des Systems stark erhöht wird (im Vergleich zu 1D-Barcodes ohne Redundanz).

Ein anderes Verfahren zur maschinenlesbaren Identifizierung stellen sogenannte Systeme zur sicheren Identifikation dar, die auf eindeutigen physikalischen Gegebenheiten der Objekte beruhen, die nach der Herstellung nicht mehr verändert oder dupliziert werden können. Mit dem Verfahren von Tailorlux[2] werden geringe Mengen eines anorganischen, ungiftigen Leuchtpigments in das zu schützende Produkt selbst oder in seine Oberflächenbeschichtung eingebracht. Die Leuchtpigmente basieren auf den Elementen seltener Erden. Diese sind chemisch stabil, säurefest und hitzebeständig. Die mit dem bloßen Auge nicht erkennbaren Marker erzeugen jeweils ein individuelles Lichtspektrum, welches unverwechselbar wie ein Fingerabdruck ist. Beim alternativen Verfahren von Bayer Technology Services[3] können Oberflächen eindeutig authentifiziert werden, ohne dass irgendeine zusätzliche Kennzeichnung erforderlich ist. Das Objekt selbst wird zur Markierung sogar bei unterschiedlichen Oberflächen (z. B. Papier, Kunststoff, Holz, Leder und Metalle). Diese Art der Identifikation wird in dieser Arbeit allerdings nicht weiter betrachtet.

### 2.2.3 Radio Frequency Identification / Near Field Communication

Radio Frequency Identification (RFID) ist ein Verfahren zur kontaktlosen Identifikation von physischen Objekten mit Hilfe einer Funkübertragung. Die Objekte werden zu diesem Zweck mit sogenannten *RFID-Tags* ausgestattet, welche aus einer Antenne und einem Controllerbaustein bestehen, wodurch diese immer eine eindeutige ID enthalten und zusätzlich je nach Bauform auch weiteren Speicher bereitstellen können. Diese Tags können dann mit Hilfe von *RFID-Readern* ausgelesen und beschrieben werden.




Zur Datenübertragung ist keine direkte Sichtverbindung notwendig, was es erlaubt die Tags auch innerhalb von Objekten zu platzieren, wodurch diese unter Umständen besser

[2] http://www.tailorlux.com/ [Letzter Zugriff: 26.11.2012]
[3] http://www.protexxion.de/ [Letzter Zugriff: 26.11.2012]

gegen Beschädigungen geschützt werden können. Weiterhin ist es nicht notwendig die Tags gegenüber dem Leser korrekt auszurichten; die Anwesenheit im elektromagnetischem (EM) Feld des RFID-Reader ist ausreichend. Darüber hinaus können mit einem Reader auch mehrere Tags gleichzeitig erfasst werden. Demgegenüber steht der Nachteil aller funkbasierten Systeme, durch externe Strahlung gestört werden zu können und in der Regel in der Nähe von metallischen Objekten nur eine verminderte Übertragungsleistung bieten. Zusätzlich ist es möglich, dass je nach verwendeter Übertragungsfrequenz eine Störung durch die Bestandteile eines Objektes erfolgen kann (z.B. Objekte mit hohem Wassergehalt bei der Verwendung von UHF-Tags mit 2,4 GHz).

RFID-Systeme können durch die Art der Energieversorgung in zwei unterschiedliche Klassen aufgeteilt werden. Kostengünstige (< 0.10 Euro pro Tag) *passive Tags* werden durch die EM-Übertragung des RFID-Readers mit Energie versorgt. Dadurch ist keine Energiequelle zum Betrieb des Tags notwendig, wodurch die Betriebszeit nahezu unbegrenzt ist. Die Leistungsfähigkeit solcher Tags ist allerdings eingeschränkt, da das EM-Feld nur in sehr begrenztem Maße Energie liefern kann. Dieses Problem verstärkt sich insbesondere dann, wenn sich mehrere Tags im Feld des Readers befinden. Als Alternative bieten sich *aktive Tags* an, welche durch eine eigene Energiequelle versorgt werden (z.B. eine Batterie oder ein Akkumulator). Dieser muss allerdings in regelmäßigen Abständen getauscht beziehungsweise geladen werden. Hinzu kommen noch deutlich höhere Kosten für solche Tags (ca. 20 Euro pro Tag), aber auch die Möglichkeit diese Tags direkt mit integrierter Sensorik zu fertigen (z.B. zur Temperaturüberwachung). RFID wird bereits heute im großen Umfang in der Industrie und vor allem in der Logistik eingesetzt [SKSM07]. Weitere Anwendungsfelder sind zum Beispiel die Positionierung innerhalb von Gebäuden ([BS05]) oder in instrumentierten Umgebungen [NGK+05].

Eine mit RFID verwandte Technik ist der Nahfunkstandard *Near Field Communication* (NFC) [Int10], welcher auf der RFID-Funktechnik (ISO/IEC 14443) basiert, diese aber um ein Datenaustauschformat ergänzt. Dieses Format (NFC Data Exchange Format, NDEF[4]) benutzt eine Binärcodierung und teilt die Daten in kleine Blöcke auf. Jeder dieser Blöcke besitzt eine eindeutige ID, einen Typ und den eigentlichen Nutzinhalt. Mit Hilfe des Typs kann dabei erkannt werden welcher Inhalt und welche Kodierung in den Nutzdaten zu erwarten sind (zum Beispiel ein Bild oder eine URL). Die üblicherweise verwendeten NFC-Tags und Lesegeräte arbeiten mit einem Abstand zwischen Tag und Antenne von maximal 5 cm. Häufige Anwendungsfälle sind das sogenannte Micropayment (zum Beispiel elektronische Tickets oder bargeldloses Bezahlen) und ID-Karten (zum Beispiel für Sicherheitslösungen oder Zeitkontenüberwachung).

[4] `http://www.nfc-forum.org/specs/` [Letzter Zugriff: 26.11.2012]

### 2.2.4 Eingebettete Systeme

Die komplexestes Stufe einer Instrumentierung von physischen Objekten stellen die eingebetteten Systeme dar. Dabei werden alle drei Schritte der Verarbeitungskette: Erfassen - Bewerten - Reagieren direkt am Objekt realisiert. Dazu kommen in der Regel kleine Rechnereinheiten zum Einsatz, die mit Verarbeitungs- und Speicherkapazität bestückt sind. Darüber hinaus können sie mit eingebauter Sensorik auch die Veränderungen ihrer Umgebung feststellen. Schließlich können sie mit Hilfe ihrer Kommunikationskanäle neues Wissen in die Umgebung oder an andere eingebettete Systeme propagieren und diese somit beeinflussen (siehe Abbildung 2.3). Dadurch können Objekte, die mit dieser Technik ausgestattet sind, ihre Aufgaben vollständig ohne eine instrumentierte Umgebung (siehe Kapitel 2.1) durchführen.


Abbildung 2.3: Medikamentendose mit eingebettetem System zur Überwachung der Lagerbedingungen und der Einnahme durch den Patienten

## 2.3 „Ubiquitous Computing“

Die Vision des „Ubiquitous Computing“ leitete einen Paradigmenwechsel im Bereich der Computertechnologie ein und basiert auf der Idee von Mark Weiser aus dem Jahr 1991, dass die profundesten Technologien die sind, die verschwinden und im Alltag aufgehen und von diesem nicht mehr zu unterscheiden sind [Wei95]. Rechen- und Verarbeitungsleistung in Form von Desktop Computern wird mehr und mehr verschwinden und in die Alltagsumgebung integriert werden. Dies hat zur Folge, dass Computer nicht mehr als eigenständige Entität und als zentrales Terminal wahrgenommen werden, sondern die Informationen an den jeweils passenden Stellen zur Verfügung stehen [Kaw09]. Der Mensch fokussiert sich nur noch auf die eigentliche Aufgabe und beschäftigt sich explizit nur noch dann mit Computern, wenn dies für die Tätigkeit notwendig ist. Der offensichtliche Ansatz um

Computer in die Umgebung zu integrieren ist die drastische Verkleinerung der Systeme, so dass diese direkt in Alltagsgegenstände integriert werden können. Dieses offensichtliche Vorgehen wird beispielsweise durch das Mooresche Gesetz [Moo65] unterstützt, dass besagt, dass sich die Rechenleistung von Systemen ca. alle 18 Monate verdoppelt. Ähnliche Ansätze gelten auch für Kapazitäten von Speichersystemen und die Übertragungsbandbreite von Kommunikationssystemen. Ausgehend von diesen Entwicklungen eröffnet sich nun die Möglichkeit, leistungsstarke System in Alltagsgegenständen zu integrieren und diese zu *intelligenten Objekten* zu machen (engl. „Smart Objects"). Dieses Vorgehen stellt eine kostengünstige Lösung dar und verstärkt die Vernetzung der realen und der digitalen Welt [BGS01]. Somit ist man der Lösung eines unsichtbaren Computers sehr nahe und kann eine auf den Menschen fokussierten Zugriff auf Informationen realisieren [Nor99]. Die heutige Forschung konzentriert sich auf die Suche nach einem ausgewogenen Verhältnis zwischen Leichtigkeit der Interaktion, Komplexität der Informationsaufbereitung, Belastung des Benutzers und sozialer Akzeptanz solcher Systeme [Kaw09]. Ein interessanter Ansatz um diese Balance zu ermöglichen, ist die Einbeziehung von angemessenem Timing, Örtlichkeiten, Identitäten und anderen kontextabhängigen Attributen in Interaktion mit dem Benutzer.

### Kontext und kontextsensitive System

Daher spielen solche Kontext-sensitiven Systeme eine immer größere Rolle in der Forschungslandschaft. Viele unterschiedliche Arbeiten beschäftigen sich mit dem Thema [Sch95, Dey00, Pas01, Sch02], wobei jeder Ansatz das Konzept „Kontext" etwas unterschiedlich definiert [Kaw09]. In der Regel konvergieren alle Definitionen zu der folgenden Aussage (vgl. [Dey00]).

> **Definition 2.1** (**Kontext**)**.** Der Kontext umfasst alle Informationen, die genutzt werden können, um den Zustand einer Entität genauer charakterisieren zu können.

Daraus folgend spricht man von einem kontextsensitiven System, wenn es Kontextinformationen nutzt um dem Benutzer relevante Daten oder Dienste anbieten zu können, wobei die Relevanz von der jeweiligen Aufgabe des Benutzers definiert wird. Da mit diesem Ansatz ein großer Mehrwert für den Benutzer entsteht, bildet die Thematik der Kontexterfassung und -verarbeitung einen großen Bestandteil der Forschung im Bereich „Ubiquitous Computing".

## 2.4 Intelligente Objekte

Als deutsche Übersetzung des Begriffs „*Smart Objects*“ wird in der Regel der Ausdruck *intelligente Objekte* verwendet, womit bereits sichtbar wird, wodurch sich dieses Konzept auszeichnet. Es beschreibt die Verknüpfung von Intelligenz mit physischen Objekten. Trotz der Umsetzung vieler prototypischer Ansätze, die teilweise auch weiterführende Konzepte wie Nachhaltigkeit mit einbeziehen (siehe zum Beispiel [BKH+12b]), ist die Art wie sich diese Intelligenz darstellt und auszeichnet noch nicht klar abgegrenzt. Vermutlich ist aus diesem Grund eine Marktdurchdringung solcher Objekte bis heute nicht erfolgt, da sowohl die Entwicklung solcher Objekte als auch umgebender Anwendungen nicht nach klaren Regeln und Strukturen erfolgen kann, solange die Definition der Möglichkeiten solcher Objekte nicht fixiert ist.

[Kaw09] betrachtet und vergleicht eine Vielzahl unterschiedlicher Definitionen, welche durch ihre chronologische Reihenfolge mit der Zeit gewachsen sind. Zentrales Element der meisten Definitionen von intelligenten Objekten ist die Fähigkeit der Kommunikation nach außen (und daher mit anderen Objekten und der Umgebung/dem Benutzer) sowie die Fähigkeit selbständig Daten verarbeiten zu können (z.B. [BGS01]). Darauf aufbauend wurde die Fähigkeit selbstständig den eigenen Kontext zu ermitteln ergänzt, womit diese Objekte durch Sensoren die Umgebung erfassen [KAB+07] und feststellen können, wo sie sich befinden, wer noch in der Umgebung anwesend ist und wie sich die Umgebung mit der Zeit verändert. Als orthogonale Betrachtungsweise führt Kawsar noch an, dass auch die Art der Umsetzung von Intelligenz zwei verschiedene Facetten abbilden kann. So kann man Intelligenz verstehen als die Fähigkeit von Objekten eigenständig Operationen durchführen zu können oder eine alternative Sichtweise in Betracht ziehen, bei der das Objekt dadurch Intelligenz ausdrückt, in dem es den Benutzer auf angemessene Weise anleitet oder bei der Ausführung von Operationen [SKjBe+98] unterstützt.

Eine sehr interessante zusätzliche Dimension fügt Kawsar ein, die sich mit der Frage der Granularität der Intelligenz beschäftigt. Ein Objekt kann dabei nach außen (zum Beispiel für den Benutzer) intelligent wirken, obwohl es selbst nur eine geringe oder gar keine eigene Intelligenz besitzt. Die notwendige „Restintelligenz“ wird bereitgestellt von der Umgebung oder von Geräten, die zur Interaktion mit dem Objekt genutzt werden.

## 2.5 Semantisches Web

Das *semantische Web* beschäftigt sich mit der Aufgabe der Annotation von Dokumenten im World Wide Web (WWW) mit semantischen Informationen wie z.B. Ontologien [FHLW05], um das Problem zu lösen, dass sich gegenwärtige Suchalgorithmen nur auf eine syntaktische Analyse des Inhalts von Dokumenten stützen können. Besonders multimediale Elemente

(wie zum Beispiel Grafiken oder Videos) können syntaktisch nicht sinnvoll erschlossen werden. Daher wurden eigene Sprachen definiert (wie zum Beispiel RDF[5], siehe [MM04]), die solche Inhalte mit semantischen Modellen beschreiben. Dazu werden in solchen Sprachen Aussagen formuliert, die die Eigenschaften von Objekten und die Beziehungen von Objekten zu anderen Objekten beschreiben und die Erstellung von Wissen in lesbarer Form sowohl für Maschinen als auch für den Menschen ermöglichen (siehe [SAHS06]). Somit kann z.B. das Web als eine große und verteile Wissensdatenbank genutzt werden.

### 2.5.1 Ressourcenkonzept als Ausgangsbasis

Die heute genutzte Webarchitektur bietet bereits eine elegante Möglichkeit Ressourcen zu identifizieren. Ursprünglich handelt es sich bei einer Ressource um eine Informationseinheit, die im World Wide Web von einer anderen Entität verlinkt beziehungsweise vernetzt werden kann [Hec05]. In HTML-Seiten werden beispielsweise andere Informationen über sogenannte *Uniform Resource Locators* (URLs) verlinkt. Genereller betrachtet kann eine Ressource allerdings alles Mögliche sein, über das man sprechen möchte, sowohl physische Objekte (wie zum Beispiel ein Tisch, ein Mensch oder eine Wolke) als auch abstrakte Konzepte (zum Beispiel Zustände oder Emotionen). Im Folgenden werden die W3C-Definitionen aus [BLFM98] genutzt (siehe auch [Hec05]).

> **Definition 2.2** (**Ressource**)**.** Eine *Ressource* kann alles Mögliche sein, solange es eine Identität besitzt.

> **Definition 2.3** (**Identifier**)**.** Ein *Identifier* ist ein Objekt, welches als Referenz zu etwas, das eine Identität besitzt, fungiert.

Da die bereits erwähnten URLs nur für die Verlinkung im WWW gedacht waren, existiert ein generisches Konzept zur Identifikation von Ressourcen: die Uniform Resource Identifier (URI).

> **Definition 2.4** (**Uniform Resource Identifier (URI)**)**.** Ein *Uniform Resource Identifier* ist ein Identifier, bei der das Objekt aus einem String besteht, der durch eine beschränkte Syntax definiert wird (siehe [BLFM05]).

[5] Resource Description Framework

URIs benutzen eine hierarchische Struktur bestehend aus einzelnen Konzepten, um Ressourcen simultan zu identifizieren und zu klassifizieren. Diese Art der Darstellung erlaubt es, konzeptuelle Gleichheit von Objekten daran zu erkennen, dass bei URIs bis zu einem gewissen Teil übereinstimmen. Sind beide URIs identisch, so handelt es sich um exakt das gleiche Objekt.

> **Definition 2.5** (**Konzept**). Ein *Konzept* ist ein formuliertes Gedankengerüst zur Realisierung von etwas (siehe [PBdWdD89]).

Eine Beispiel-URI, die eine Ressource beschreibt, könnte wie folgt lauten: Dabei ist die Ressource eine Instanz mit der Nummer 12345 vom Konzept c, das ein Unterkonzept von Konzept b ist, welches wiederum ein Unterkonzept von Konzept a darstellt.

$$uri:a:b:c:12345$$

### 2.5.2 RDF - Resource Description Framework

Das Resource Description Framework (RDF) ist eine Sprache zur Repräsentation von Ressourcen im WWW [MM04] und wurde speziell entwickelt, um Ressourcen im Web mit Metadaten auszustatten, wie zum Beispiel: Titel, Autoren, Copyright-Informationen. Abstrakter betrachtet können mit RDF alle Ressourcen, die einen Identifier im Web besitzen, beschrieben werden. Eine Erreichbarkeit dieser Ressource über das Web ist nicht zwingend erforderlich (es genügt den Identifier zu kennen).

Klassischerweise verwendet RDF als Struktur ein sogenanntes Subjekt-Prädikat-Objekt-Tripel, welches in der Regel in der Form $P(S,O)$ oder als $S \xrightarrow{P} O$ dargestellt wird. Subjekte und Objekte können dabei ausgetauscht werden, das heißt jede Entität kann sowohl als Subjekt als auch als Objekt fungieren (und zusätzlich auch als Prädikat), wobei beide als Ressource oder als einfaches Literal manifestiert sind. Zusätzlich ist es möglich, dass ein solches Tripel selbst als Subjekt oder Objekt fungiert. Dieses Verfahren wird als *Reifikation* bezeichnet und erlaubt es Aussagen über Aussagen zu treffen (zum Beispiel $(S \xrightarrow{P} O) \xrightarrow{P'} O'$).

In [SvH05] wird dargelegt, dass für die Beschreibung von Ressourcen über Metadaten und die Definition dieser Metadaten XML nicht ausreicht. Es wird daher empfohlen diese Aufgabe mit Hilfe von RDF zu realisieren. Nichtsdestotrotz wird das RDF-Modell in der Regel als XML-Datei repräsentiert. In Analogie zur XML Schemadefinition existiert in RDF auch ein Ansatz zur Schemadefinition von RDF-Dokumenten mit dem Namen *RDF Schema*

(RDFS). Mit Hilfe dieses Schemas kann festgelegt, werden welche Eigenschaften (und damit welche Prädikate) für Objekte definiert werden können und welchen Wertebereich diese Eigenschaften haben. Dazu wird ein hierarchisches Klassenmodell verwendet. Zum Beispiel kann eine Relation `hasWritten` lauten, die ein Subjekt der Klasse `Writer` und ein Objekt der Klasse Book erhalten muss.

### 2.5.3 OWL - Web Ontology Language

Die *Web Ontology Language* (OWL) bietet im Vergleich zu XML, RDF und RDFS deutlich mehr Möglichkeiten um Semantik auszudrücken [MvH03], sowie eine bessere Interpretierbarkeit durch automatische System [SvH05]. Zu den weiterführenden Möglichkeiten gehören die Definition einer hierarchischen Klassenstruktur, Relationen zwischen Klassen (zum Beispiel Disjunktheit), Eigenschaften, Kardinalitäten (1..n) und Charakterisierung von Eigenschaften (z.B. Symmetrie). OWL entstand als Nachfolger der DAML+OIL Ontologiesprache. Für weiterführende Informationen zur Benutzung von OWL in intelligenten Umgebungen siehe [Ebe03] und die verwandten Arbeiten in Kapitel 3.

## 2.6 Dokumentenmanagement und Datenablage

Als *Dokumentenmanagement* bezeichnet man den Vorgang der Ablage elektronischer Dokumente mit Hilfe einer Datenbank-gestützten Verwaltung [Kam99]. Im Folgenden wird der Begriff (im sogenannten *engeren Sinn*) ausschließlich für exakt diesen Bereich der elektronischen Archivierung verwendet und Punkte wie Dateneingabe (zum Beispiel durch Scanner), darüber hinausgehende Kommunikation und Workflowverwaltung nicht weiter betrachtet (dies wäre dann der *weitere Sinn*). Folgende Kerneigenschaften bleiben bei einer solchen Definition über:

- **Speicherung** - Die eigentliche Hauptfunktionalität ist die Speicherung der notwendigen Dokumente. Dies umfasst den Vorgang der Persitierung und Fixierung der Dokumente und der Generierung einer eindeutigen Identifikationsnummer (ID), mit deren Hilfe das Dokument direkt wiedergefunden werden kann.
- **Retrieval und Metadaten** - Durch die Annotation von Dokumenten mit Hilfe von Metadaten (zum Beispiel das Datum der Erstellung, der Ersteller oder Schlagworte bezüglich des Inhalts) kann das Dokument auch mit Hilfe dieser Attribute wiedergefunden werden. Dies ist insbesondere dann von Interesse, wenn der Benutzer nicht genau nach einem expliziten Dokument sucht (dessen Existenz ihm bekannt ist), sondern auf der Suche nach beliebigen Dokumenten ist, die eine oder mehrere Eigenschaften besitzen müssen. Sind diese Eigenschaften als Metadaten verfügbar, ist

eine Suchvorgang möglich, ohne den eigentlichen Dokumenteninhalt betrachten zu müssen.

- **Indizierung** - Mit Hilfe eines Index kann die Suche nach Dokumenten beschleunigt werden. Dazu werden die Dokumente bei der Ablage klassifiziert. Diese Klassifikation kann in Kombination mit den Metadaten in eine sehr effiziente Datenstruktur abgelegt werden, die zusätzlich zu den eigentlichen Dokumenten vorgehalten wird. Dadurch, dass Suchvorgänge nun auf dieser Datenstruktur durchgeführt werden können, ist der Vorgang mit weniger Rechenleistung und somit schneller möglich.

- **Sicherheit** - Für eine Teilmenge aller abgelegten Dokumente kann es zusätzliche Anforderungen bezüglich der Sicherheit geben. In der Regel bezieht sich diese Anforderung auf die Tatsache, dass nur eine beschränkte Nutzergruppe die Rechte besitzt ein Dokument lesen beziehungsweise schreiben zu können. Diese Anforderung wird mit Hilfe eines Rechtesystems realisiert, welches mit unterschiedlichen Verfahren (zum Beispiel Passwort, Zertifikate oder biometrische Merkmale) den Benutzer identifiziert und dann die passenden Rechte einräumt.

- **Versionsverwaltung** - Für Dokumente, die häufigen Änderungen unterworfen sind, ist es zusätzlich notwendig exakt festzuhalten welche Entität welche Änderung durchgeführt hat. Zusätzlich ist es jederzeit notwendig noch Zugriff auf eine ältere Version zu haben. Daher werden Versionsverwaltungssystem eingesetzt, die jede Änderung protokollieren und jederzeit den Zugriff auf alle älteren Versionen erlauben.

Zur Speicherung der Dokumente wird in der Regel eine Datenbank eingesetzt. Heutige Datenbank-Implementierungen decken bereits einen großen der Teil der obigen Anforderungen in ihren Funktionseinheiten ab [EN06]. Darüber hinausgehend existieren zusätzliche Anforderungen, die durch die konkrete Implementierung einer Datenbank entstehen. Jeder Zugriff auf die Datenbank wird in Form einer Transaktion durchgeführt. Je nach Art des Zugriffs ist diese Transaktion keine atomare Operation, das heißt sie besteht aus mehreren Bestandteilen, die nacheinander abgearbeitet werden. Daher kann der Fall eintreten, dass die Transaktion nur teilweise bearbeitet wurde und ein inkonsistenter Zwischenzustand erreicht wird. Zwei der häufigsten Konzepte zur Vermeidung dieses Zustandes, die auch auf Systeme mit Transaktionen abseits von Datenbanken angewendet werden, sind *ACID* und *BASE*, welche im Folgenden beschrieben werden.

- **ACID** (Atomicity, Consistency, Isolation, Durability) - Das *ACID*-Modell beschreibt die klassischen vier Eigenschaften einer Datenbank bezüglich ihrer Transaktionssicherheit. Dabei muss jede Transaktion für den Benutzer als *atomar* wahrgenommen werden können. Das bedeutet, dass sichergestellt ist, dass jede Transaktion entweder vollständig oder überhaupt nicht durchgeführt wird. In der Regel wird dies dadurch erreicht, dass im Fehlerfall der letzte gültige Stand genutzt wird. Dadurch wird die *Konsistenz*-Anforderung unterstützt, die besagt, dass die Datenbank immer nur von

einem konsistenten in einen anderen konsistenten Zustand überführt werden darf. Inkonsistente Zwischenzustände müssen vom System unterbunden werden. Zusätzlich gilt die Anforderung der *Isolation*. Das bedeutet, dass mehrere Transaktionen sich nicht gegenseitig beeinflussen dürfen. Dies wird in der Regel entweder durch Sperren (nur eine Transaktion pro Objekt ist erlaubt) oder eine Versionsverwaltung realisiert. Insgesamt ist als letzter Punkt die *Beständigkeit* sicherzustellen, das heißt erfolgreiche Transaktionen müssen bestand haben, auch im Falle eines Fehlers.

- **BASE** (Basically Available, Soft state, Eventual consistency) - Einen alternativer Ansatz, der diametral zu ACID ausgerichtet ist, bietet das BASE-Modell. Im Gegensatz zur pessimistischen Annahme von ACID ist BASE eher optimistisch orientiert mit dem Ziel der größtmöglichen Verfügbarkeit bei ständig fluktuierenden Zuständen. Es gilt dabei die Annahme, dass die Datenbank *im Wesentlichen verfügbar* ist, das heißt auch bei Ausfällen ist zumindest noch ein mehr oder weniger großer Teil verfügbar. Zusätzlich ist die Datenbank immer in einem *weichen Zustand*, das heißt der Anwender ist sich dessen bewusst und kann die unterschiedlichen Zustände handhaben. Somit erreicht man das Ziel der *letztendlichen Konsistenz*, in dem die Datenbank in der Regel konsistent ist. Dabei ist die Verfügbarkeit wichtiger als die immer gegebene Konsistenz. Auf den Fall von Konflikten reagiert man aktiv mit Lösungsstrategien, zum Beispiel mit *ausweichen* (man akzeptiert unterschiedliche Konfliktzustände) oder einer *extrem einfachen Lösung* (der letzte Zugriff gewinnt).

## 2.7 Produktlebenszyklus

Der Begriff des *Produktlebenszyklus* stammt aus dem Bereich der Betriebswirtschaftslehre und bezeichnet den Zeitabschnitt eines Produktes zwischen dessen Design und dessen Recycling. Dieser Zeitbereich ist in mehrere Phasen eingeteilt, die das Produkt (in der Regel ein Konsumgut) durchläuft. Die Verkettung dieser einzelnen Phasen bildet somit einen Prozess, den das Produkt durchläuft [Sä08, Eig09] (siehe Abbildung 2.4). Als *open-loop* wird eine Lebenszykluskette charakterisiert, wenn zu Beginn die Liste und Reihenfolge der einzelnen Partner noch nicht feststeht und sich somit nicht klar abschätzen lässt, welche weiteren Partner an der Kette teilnehmen werden. Somit müssen alle Partner darauf vorbereitet sein, dass das Produkt nicht das erwartete „Vorleben“ besitzt. Sowohl bei starren Abläufen als auch im open-loop Bereich beschreibt das *Produktlebenszyklusmanagement (PLM)* ein eigenständiges Konzept zur Verwaltung aller Abläufe, die während eines Lebenszyklus auftreten. Dabei handelt es sich um keine fertige Softwarelösung, sondern um einen Verbund aus unterschiedlichen Teilkomponenten, die in ihrer Gesamtheit das PLM-System formieren.

Abbildung 2.4 zeigt diese Subsystem und ihr Einsatzgebiet entlang der Lebenszykluskette. Da ein großer Teil der Systeme mit der Eigenschaft „Computer Aided (CA)“ gekennzeichnet

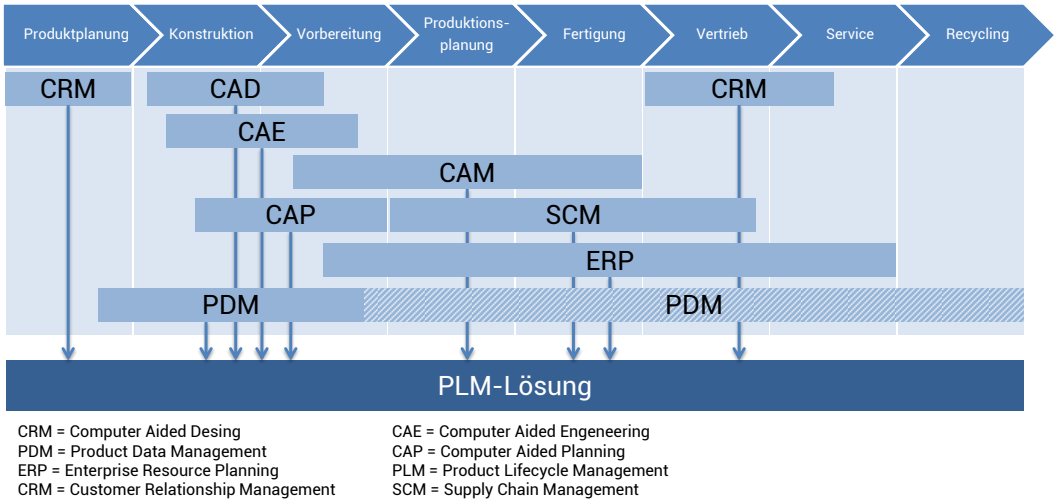



Abbildung 2.4: Acht Phasen einer Produktlebenszykluskette mit IT-Lösungen für ein vollständiges Lebenszyklusmanagement [Aac07]

wird, spricht man in der Summe auch von *CAx*-Systemen. Eine wichtige Eigenschaft ist, dass alle Subsysteme auf eine einheitliche und gemeinsame Datenbasis zugreifen. Diese wird in der Regel von einem *Produktdatenmanagement (PDM)* zur Verfügung gestellt, das mit Hilfe von Schnittstellen mit den Unterschiedlichen *CAx* und *ERP*-Systemen kommuniziert. PDM-Lösungen, die sich vom ERP unterschieden, fußen in der Regel auf einem Dokumentenmanagement-System oder überführen alternativ Daten aus dem *Computer Aided Design (CAD)* in spätere Phasen. Zur Kommunikation verwenden viele Systeme Schnittstellen, die die Norm STEP/ISO10303 erfüllen [ISO04]. Je nach genutzter Softwarelösung ist der Übergang von PDM zu ERP flexibel geregelt. Die Abbildung 2.4 zeigt den Fall, dass das ERP-System ab der Produktionsplanung alle Aufgaben des PDM übernimmt. Es ist aber auch möglich das PDM-System über den gesamten Lebenszyklus zu nutzen. Der Vorteil solcher Systeme, die es erlauben Informationen über Lebenszyklusphasen hinaus weiterzugeben, stehen im Gegensatz zu der Tatsache, dass diese Lösungen in der Regel industrie-/unternehmensspezifisch ausgelegt sind.

## 2.8 Digitale Objekgedächtnisse

Analoge Gedächtnisse, bei denen man in der Regel von einem Materialverhalten spricht, welches sich nach einer Deformation wieder in seinen Ursprungszustand zurückversetzen lässt, bewegen sich im Bereich der Nano- und Materialwissenschaften. Ganz im Gegensatz dazu führen die digitalen Gedächtnisse, im Bereich der Informationstechnologie, die Idee

des Internets der Dinge, physische Objekte mit einer Internet-basierten virtuellen Identität auszustatten [Ash09], fort [Wah13b].

> **Definition 2.6 (Digitales Objektgedächtnis (DOMe)).** Ein digitales Objektgedächtnis eines physischen Objekts ist ein digitaler Speicher, der alle relevanten Daten, die im Verlauf der Lebensdauer dieses Objekts ermittelt wurden, dauerhaft kapselt.

Diese Gedächtnisse fügen sich in eine Gesamttaxonomie ein, die auf der Idee des Internets der Dinge basiert (siehe Abbildung 2.5). Dabei fungieren als Grundlage für solche Objektgedächtnisse in der Regel mobile cyber-physische Systeme, die die physischen Objekte mit Sensorik und Verarbeitungskapazität ausstatten. Je nach Art der Ausstattung der Objekte spricht man von sogenannten passiven Gedächtnissen (nur Identifizierung am physikalisches Objekt, Sensorik und Verarbeitung in der Umgebung) und aktiven Gedächtnissen (direktes Messen und Verarbeiten am Objekt). Grundsätzlich können solche generischen Objektgedächtnisse für beliebige Objekte genutzt werden, zum Beispiel für natürlich vorkommende Objekte oder auch bei Benutzertagebüchern (siehe [KHW06] und [WKSB08]).

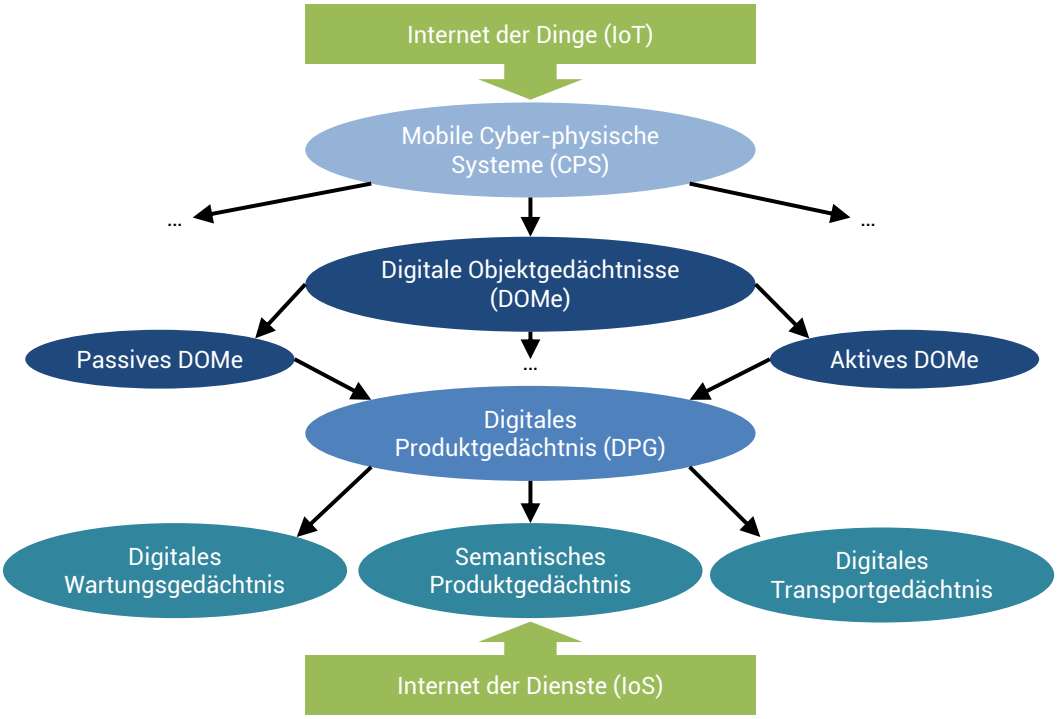



Abbildung 2.5: Eine Taxonomie von Digitalen Objektgedächtnissen [Wah13b]

Digitale Produktgedächtnisse stellen eine Unterklasse dar, die explizit für produzierte und hergestellte Produkte definiert wurde und die Aufgabe hat, die Daten eine Produktes während der typischen Lebensspanne von Definition über Herstellung bis zur Entsorgung abzubilden. Semantische Produktgedächtnisse bilden eine weitere Spezialisierung, da diese im Gegensatz zu digitalen Produktgedächtnissen nicht nur maschinen-lesbare, sondern semantische maschinen-verstehbare Informationen enthalten. Dadurch können diese Daten von jeder Anwendung genutzt werden, die Zugriff auf die entsprechenden Beschreibungen der epistemologischen Primitive und den entsprechenden Ontologien hat [Wah13b]. Kombiniert werden diese Informationen noch mit Funktionen aus dem Internet der Dienste.

## 2.9 Anforderungen

Um die in Kapitel 1.2 definierten Ziele dieser Arbeit erreichen zu können, ist es notwendig eine fixierte Anforderungsliste zu erstellen, die es auf der einen Seite erlaubt bestehende Lösungen auf ihre Anwendbarkeit hin zu überprüfen und gleichzeitig als Leitlinie zu fungieren, welche die noch durchzuführenden Entwicklungen definiert. Daher werden im Folgenden eine Reihe von Anforderungen definiert, die eine Zielarchitektur erfüllen sollte. Zur besseren Übersicht werden diese in unterschiedliche Kategorien aufgeteilt, die sich mit der eigentlichen Produktinformation, der Speicherarchitektur, der Visualisierung und der benötigten Werkzeuge und dem Thema Migration befassen.

### 2.9.1 Digitale Produkt-Informationen

Eine digitale Beschreibung eines physischen Objekts, deren Aufgabe es ist, die Kommunikation zwischen Produkt und Kunde auf der einen Seite, als auch zwischen den einzelnen Partnern der Lebenszykluskette dieses Produkts zu verbessern (im Sinne des *Produktdatenmanagement (PDM)*), muss folgende spezifische Eigenschaften erfüllen (siehe auch [JM08]):

- **Abbildung auf gegenwärtige Standards ($R_D1$)** - Zur Vermeidung der Entwicklung einer isolierten Anwendung sollte ein Konzept zur Darstellung von Objektdaten in der Lage sein, sowohl eine Abbildung von bestehenden Standards auf eigene Konzept abzubilden, als auch die vollständige Integration bestehender Standards als Ersatz für ein eventuelles eigenes Datenmodell. Dieser flexible Ansatz ist, zur Realisierung von Anwendungen mit sehr heterogenen Daten in „open-loop“-Szenarien, notwendig.
- **Semantische Struktur ($R_D2$)** - Eine digitale Objektbeschreibung benötigt eine klar definierte semantische Struktur, um auf bestehende Techniken im Bereich des semantischen Webs zurückgreifen zu können und eine automatische Verarbeitung und eine

Erweiterung der Daten zu ermöglichen. Die Struktur bezieht sich dabei sowohl auf die eigentlichen Daten an sich, als auch auf deren Art der Bereitstellung.

- **Offene und flexible Datenstrukturen ($R_D3$)** - Ein Konzept zur Aufnahme von Objektdaten sollte einen offenen Ansatz verfolgen und jederzeit durch neue Daten und Konzepte erweiterbar sein. Dadurch kann das System im laufenden Betrieb an neue Gegebenheiten angepasst werden und bleibt jederzeit voll flexibel in Bezug auf die aufzunehmenden Daten und Modelle.

- **Unterstützung beim Wiederauffinden von Daten ($R_D4$)** - Durch den offenen Ansatz und die Unterstützung flexibler Datenstrukturen (siehe $R_D3$) folgt der Zugriff auf Daten keinem festen Schema. Vielmehr muss der Anwendung eine Hilfsstruktur zur Verfügung gestellt werden, die es erlaubt die Datenbasis nach bestimmten Kriterien zu durchsuchen bzw. Daten filtern zu können, um die gewünschten Informationen aufzufinden.

### 2.9.2 Speicherarchitektur

- **Vernetzung der Daten ($R_S1$)** - Die Möglichkeit Produktdaten zu vernetzen erlaubt es zum einen Daten aus verschiedenen Quellen zu einer Gesamtdarstellung zu verbinden und zum anderen einzelne Daten auszulagern, wodurch die Kommunikation einzelner Partner über die Daten des Objekts erleichtert wird.

- **Verteilte Datenspeicherung ($R_S2$)** - Um es Anbietern von Daten zu erlauben die Kontrolle über eigene Daten zu behalten, ist es notwendig die Datenspeicherung an verschiedenen Orten zu erlauben, so dass die Nutzung von einem Server der Wahl (der unter der eigenen Kontrolle steht) ermöglicht wird. Diese Fähigkeit verbessert zusätzlich die Skalierbarkeit und Verfügbarkeit solcher Systeme. Durch die Anforderung der Vernetzung kann auch aus diversen Quellen ein Gesamtbild generiert werden.

- **Eingebettete und Server-basierte und Systeme ($R_S3$)** - Die Speicherarchitektur muss so ausgelegt sein, dass es möglich ist diese sowohl direkt an Objekten in Form von eingebetteten Systemen zu betreiben, als auch eine Verlagerung der Speicherlogik auf einen dedizierten Server zu erlauben und Nutzung eines Verweises zum Server, welches am Objekt angebracht ist.

- **Einheitliche Zugriffsschnittstelle aller Implementierungen ($R_S4$)** - Unabhängig von der konkreten Realisierungsform muss der Zugriff auf die Gedächtnisfunktionalität stets gleich geartet sein. Die Schnittstelle sollte darüber hinaus in der Lage sein, die unterschiedlichen Leistungsfähigkeiten einzelner Systeme darstellen zu können.

- **Versionsverwaltung und Rechte- bzw. Rollen-basierter Zugriff ($R_S5$)** - Um es Anwendungen zu ermöglichen auch ältere Gedächtnisstände in Betracht ziehen zu können, sollte die Architektur eine Versionsverwaltung besitzen. Diese Verwaltung soll im Zusammenspiel mit einem ebenfalls geforderten Rechte- und Rollen-basierten Zugriffssystem eine Darstellung von Historischen Daten (Provenance) ermöglichen.

- **Integration aktiver Komponenten ($R_S6$)** - Um auf flexible Art und Weise sogenannte Aktivität im Ökosystem des Objektgedächtnisses realisieren zu können, sollte es die Architektur erlauben Verarbeitungslogik, die z.B. aus dem Gedächtnis oder von einer externen Quelle stammen kann, direkt im Gedächtnis ausführen zu können. Als Aktivität wird hierbei die Fähigkeit angesehen, vorgefertigte und von Extern injizierte Codefragmente ausführen zu können, die das Gedächtnis manipulieren oder Ergebnisse zurück liefern. Diese Ausführung soll dabei sowohl von außen angestoßen werden können, als auch ereignisorientiert bzw. zeitgesteuert erfolgen.

### 2.9.3 Visualisierung

Im Bereich von open-loop Anwendungen (siehe Kapitel 2.7) muss ein allgemeingültig nutzbares Framework, welches möglichst unabhängig von der jeweiligen Anwendung genutzt werden soll, einige Anforderungen erfüllen. Der folgende Abschnitt erläutert die vier wichtigsten Anforderungen:

- **Erweiterbarkeit ($R_V1$)** - Eine wichtige Eigenschaft von Objektgedächtnissen ist, dass diese beliebige Daten enthalten können. In der Regel sind die verwendeten Daten, Formate und Kodierungen nicht im Voraus bekannt und können sich während der Lebenszeit eines Objekts weiter verändern. Ein Framework muss daher in der Lage sein, mit unterschiedlichsten Daten umzugehen und so flexibel gestaltet sein, dass auch im Nachhinein eine Erweiterung zur Unterstützung neuer Daten und Datenformate integriert werden kann.

- **Laufzeitanpassung($R_V2$)** - Mit der Möglichkeit als Datenspeicher zu fungieren sind Objektgedächtnisse auch in der Lage Anwendungsmodule und Filteralgorithmen mitzuführen, die die Darstellung der im Speicher abgelegten Inhalte ermöglicht. Eine Visualisierung sollte daher in der Lage sein, zur Laufzeit solche Module integrieren und nutzen zu können.

- **Wiederverwendbarkeit ($R_V3$)** - Weiterhin soll das Framework Entwickler in die Lage versetzen Komponenten zur Datenverarbeitung und zur Visualisierung in den verschiedensten Anwendungsfällen wiederverwenden zu können, so dass sowohl der Entwicklungsaufwand minimiert wird und gleichzeitig eine konsistente Darstellung aller Anwendungen zur Verfügung steht.

- **Anpassbarkeit ($R_V4$)** - Es gibt allerdings Einzelfälle, in denen man von konsistenten Standarddarstellungen abweichen möchte. Dies kann zum Beispiel die explizite Darstellung von bestimmten Objekteigenschaften sein, ein sogenanntes Hersteller-„Branding“ oder auch die Anpassung an spezielle Bedürfnisse oder Eigenschaften von bestimmten Benutzern oder Benutzergruppen.

### 2.9.4 Werkzeuge und Migration

- **Modularer Aufbau des Frameworks ($R_T1$)** - Die Zielarchitektur soll durch kein monolithisches Gesamtsystem realisiert werden, da durch die offenen Szenarien eine Vorkonfiguration schwierig ist. Durch die unterschiedlichen Anforderungen ist ein Zuschneiden auf den jeweiligen Anwendungsfall notwendig. Daher ist ein modularer Ansatz gefordert, der je nach Anwendungsdomäne und Leistungsfähigkeit der Hardware die passenden Module in einer Art Baukastensystem zu einer funktionalen Einheit zusammenfasst.

- **Migration bestehender Daten ($R_T2$)** - Eine direkte Umstellung auf eine neue Systeminfrastruktur ist nur in den seltensten Fällen realistisch. Aus diesem Grund soll das System es erlauben eine schrittweise Migration durchführen zu können und so beispielsweise bestehende Datenbestände mit Hilfe einmalig erstellter Vorlagen automatisch in die Objektgedächtniswelt zu transferieren.

- **Geringe Einstiegshürde zur Nutzung semantischer Daten ($R_T3$)** - Der Zielhorizont wird durch eine vollständige Nutzung semantischer Daten definiert, mit deren Hilfe eine optimale Maschinenverarbeitbarkeit gewährleistet werden kann. Zusätzlich unterstützt Semantik die Kommunikation in offenen Szenarien, da durch Abstraktion und wohl definierte Klassifikation eine Harmonisierung der zu kommunizierenden Daten erfolgt. Um eine Einführung solcher Daten zu erleichtern soll das System den Anwender bei der Generierung semantischer Daten unterstützen und es zusätzlich erlauben semantische Daten händisch, auch von Nicht-Experten, anzulegen.

## 2.10 Fazit

In diesem Kapitel wurden grundlegende Begriffe und Konzepte eingeführt. Dazu wurde die instrumentierte Umgebung vorgestellt, Methoden zur Objektidentifizierung gezeigt, Ubiquitous Computing und intelligente Objekte eingeführt und Methoden des semantischen Webs, des Dokumentenmanagements und des Projektlebenszyklus dargelegt. Ebenso wurde eine Taxonomie unterschiedlicher Objektgedächtnisausprägungen präsentiert. Anschließend wurden Anforderungen an eine Architektur für digitale Objektgedächtnisse definiert. Im

nächsten Kapitel wird nun eine Auswahl an verwandten Arbeiten und Konzepten vorgestellt, die in direktem Bezug zu dieser Arbeit stehen. Abschließend werden diese gegen die in diesem Kapitel definierten Anforderungen abgewogen.

KAPITEL 3

# Verwandte Arbeiten

## 3.1 Einleitung

Im Bereich der sogenannten „smarten“ bzw. intelligenten Objekte finden sich diverse, meist unterschiedlich fokussierte Herangehensweisen zur Generierung, Verarbeitung und Speicherung von Objekt-relevanten Daten. Ein Überblick im Rahmen dieser Arbeit kann daher nicht erschöpfend beziehungsweise vollständig sein. Aus diesem Grund wird der Schwerpunkt dieses Kapitels darauf liegen, ausgewählte Forschungs- und Industrieansätze vorzustellen (die besonderen Bezug zu Schwerpunkten dieser Arbeit aufweisen), deren Stärken und Schwächen auszuloten und diese in Bezug auf die Konzepte dieser Arbeit zu setzen.

Da im Grunde bereits kleine Webseiten oder verschiedenste Datenbank-basierte Systeme als Grundlage für die Speicherung von Objektwissen betrachtet werden können, wird in dieser Arbeit ein verwandtes Konzept erst dann als relevant angesehen, sobald dieses grundlegende Unterstützung bietet, um Daten und Prozesse im Ablauf des Produktlebenszyklus von Objekten strukturiert ablegen zu können und eine Infrastruktur bereitstellen, um diese Daten effizient weiterverarbeiten zu können.

Die verwandten Arbeiten werden nun wie folgt beschrieben. Die Kategorisierung der einzelnen Arbeiten erfolgt dabei basierend auf den wissenschaftlichen Fragen und der daraus resultierenden Anforderungen an eine Architekturkonzeption (siehe Kapitel 2.9). Die Kapitel 3.2 und 3.3 zeigen vollständige Systeme mit Verwandtschaft zu den Bereichen der intelligenten Objekte und digitalen Objektgedächtnisse. Kapitel 3.2 zeigt dabei Arbeiten aus forschungsorientierten Entwicklungen während Kapitel 3.3 Systeme aus der Industrie diskutiert. Anschließend werden verwandte Arbeiten im Bereich einzelner Module solcher Systeme betrachtet. Kapitel 3.4 zeigt dabei Arbeiten zu Ontologien und Metadaten. Kapitel 3.5 betrachtet ergänzend dazu unterschiedliche Ontologie-Editoren. Anschließend werden in Kapitel 3.6 Ansätze und Systeme betrachtet, die genutzt werden können, um Daten effizient abzulegen. Schließlich zeigt Kapitel 3.7 Arbeiten zur Visualisierung von Daten,

die in instrumentierten Umgebungen oder mit Hilfe von intelligenten Objekten generiert wurden. Alle diese verwandten Arbeiten werden dann final in Kapitel 3.8 bewertet und Schlussfolgerungen generiert, die die Grundlage für die Werkzeuge dieser Arbeit dienen.

## 3.2 Ansätze basierend auf Forschungsprojekten

### 3.2.1 Smart Product Networks

Das Projekt Smart Product Networks (SmaProN), gefördert vom Bundesministerium für Bildung und Forschung (BMBF), untersucht die Umsetzbarkeit einer dynamischen Bündelung, Klassifizierung und Vernetzung von smarten Produkten. Zur Umsetzung wird eine Infrastruktur entwickelt, welche konzeptionell auf der sogenannten Tip 'n Tell-Architektur [MF07] sowie dem semantischen Produktbeschreibungsmodell SPDO [JM08] basiert.

Bei den betrachteten „smarten" Produkte handelt es sich um physische Objekte, welche mit digitalen Produktbeschreibungen ausgestattet wurden [MB06] und analog zu den Schlüsselcharakteristika der „Ambient Intelligence" ([Wah03]) wie folgt charakterisiert werden: (1) situiert, (2) personalisiert, (3) adaptiv, (4) pro-aktiv, (5) geschäftsrelevant und (6) netzwerkfähig. Basierend auf dieser Charakterisierung wird eine semantische Repräsentation der Daten, in Kombination mit einer vernetzten Umgebung, als notwendig angesehen und es werden folgende Anforderungen an eine Infrastruktur gestellt (vergleiche Darstellung in Kapitel 2.9.1):

- **Semantische Struktur** - Klar definierte semantische Struktur, um eine automatische Verarbeitung und eine Erweiterung der Daten zu ermöglichen.
- **Vernetzung der Daten** - Möglichkeit Produktdaten zu vernetzen, zur Verbindung von Daten zu einer Gesamtdarstellung zur Auslagerung von Daten.
- **Abbildung auf gegenwärtige Standards** - Zur Vermeidung einer isolierten Anwendung soll eine Abbildung von bestehenden Standards auf eigene Konzept abbildbar und integrierbar sein.
- **Verteilte Datenspeicherung** - Kontrolle über die eigene Daten möglich und verbesserte Skalierbarkeit und Verfügbarkeit der Systeme.
- **Natürlichsprachliche Verarbeitung** - Zur Verbesserung der Kommunikation und der Akzeptanz von Produkten (z.B. durch Dialogsysteme und eine Frage-Antwort-Datenbasis).
- **Erweiterbarkeit** - Offener Ansatz zur Erweiterbarkeit durch neue Daten und Konzepte während der Laufzeit des Systems.

**Semantic Product Description Objects** Nach Aussagen des Projekts [JM08] können keine der gegenwärtig etablierten Standards für Produktdaten, alle diese Kriterien erfüllen, so dass im Rahmen des Projekts eine eigene Infrastruktur geschaffen wurde. Die Daten werden dabei in *Semantic Product Description Object* (SPDO) genannten Containern abgespeichert (siehe Abbildung 3.1).

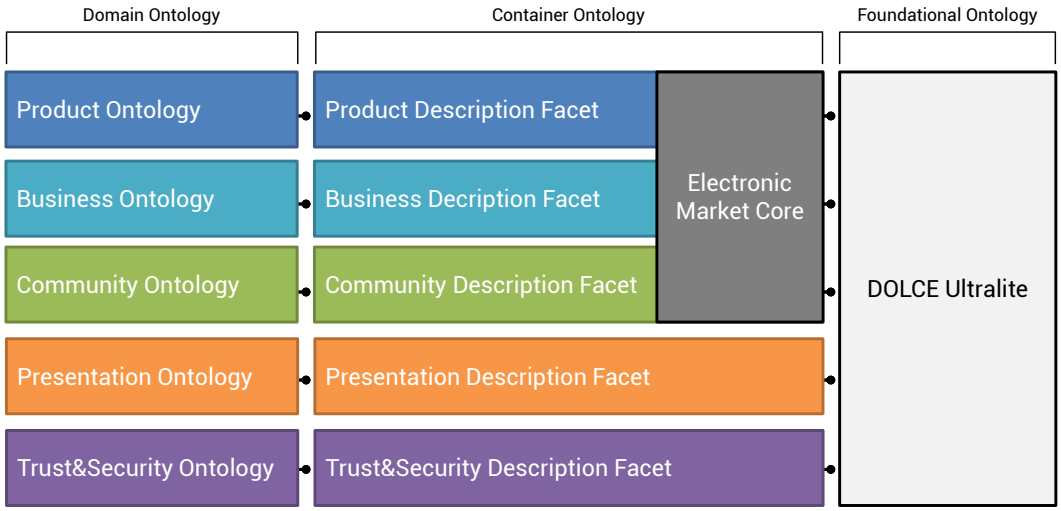



Abbildung 3.1: *Simple Product Description Object* Facetten und Ontologien

Diese Container nutzen das sogenannte Facettenmodell, wodurch die verfügbaren Daten in verschiedene Teile partitioniert werden. Jeder Teil stellt dabei eine bestimmte Facette oder Betrachtungsweise der Daten dar und legt den Fokus auf einen bestimmten Aspekt. Die verwendeten Facetten lauten dabei wie folgt:

1. **Produktbeschreibung/Product Description**: Verbindung zu etablierten Standards für Produktmetadaten und aussagenlogische Beschreibungen der funktionalen Eigenschaften eines Produkts auf einem abstrakten Niveau.

2. **Geschäftsbeschreibung/Business Description**: Verhandlungsprotokoll, Bepreisungsschema und Vertragsdaten eines Produkts.

3. **Daten der Community/Community Description**: Beschreibungen über Pläne, Aufgaben, Rollen und Ziele im Kontext einer Community.

4. **Präsentationsbeschreibung/Presentation Description**: Beschreibung der Darstellung und Präsentation von Produktdaten für einen Endbenutzer.

5. **Daten zur Vertrauensebene und zur Sicherhei /Trust and Securiy Description**: Daten über den „Ruf“ des Produkts und zu Zugriffsbeschränkungen.

6. **Selbstbeschreibung/Self Description**: Spezifikation eines SPDO in maschinen-lesbarer Form.

Als Grundlage für die semantische Modellierung werden drei zusätzliche Bausteine verwendet: (1) die Basisontologie repräsentiert durch DOLCE Ultralite (DUL) [1], (2) die Domänenontolgien und (3) der sogenannte *Electronic Market Core* (EMC) (siehe Abbildung 3.1). Dabei stellt das Containermodell eine Spezialisierung des DOLCE-„information object“ dar und basiert auf dem Ansatz „Description and Situation“ (DnS) der DOLCE Ultralite, wobei die Relationen der DUL durch OWL-Restriktionen an die Bedürfnisse der SPDOs angepasst werden.

Die Integration der SPDO-Konzepte in das Rahmenwerk der DUL ist wie folgt motiviert: (1) DOLCE ist eine weit verbreitete und akzeptierte Ontologie und drauf aufbauende Ansätze erlauben ein offenes und portables System. (2) Die ontologischen SPDO-Aussagen (Statements) sind direkt verbunden mit Konzepten eines gemeinsamen Verständnisses, so dass Kommunikationsdienste von physischen Objekten auf Weltwissen aufbauen können und daher einfacher intelligent handeln können. (3) Die Weiterverarbeitung des DnS-Ansatzes legt die Grundlage für ein Schlussfolgern (Reasoning) über die DOLCE-Strukturen.

Der sogenannte „Electronic Market Core“ (EMC) ist ein generisches Ontologieframework für elektronische Märkte (eCommerce) und stellt maschinenlesbare Begriffe bereit, die von Diensten der Märkte verarbeitet werden können [MBG07]. Der EMC basiert ebenfalls konzeptuell auf DOLCE und verknüpft Konzepte aus dem Markt (z.B. Käufer, Verkäufer und Geld) mit den ontologischen Konzepten der DOLCE-Ontologie nach dem DnS-Ansatz.

Auf Implementierungsseite wird für jede Facette der SPDOs eine formalisierte, semantische Repräsentation in Form von OWL-DL genutzt. Zusätzliche werden Anfragen an die Ontologien über ein SPARQL-basiertes Protokoll kommuniziert.

Zur Abfrage von produktspezifischen Beschreibungen wird bei SmaProN die eigens entwickelte Abfragesprache *PQL* genutzt. Diese auf P-RDF, einer Produktbeschreibungssprache, basierende SPARQL-ähnliche Sprache, bietet einen Web-basierten Zugriff auf Produktdaten. Sie erlaubt die Abfrage von statischen Produktdaten als auch von Relationen zu anderen Produkten (z.B. Zubehör) im RDF-Triple-Format.

Der folgende Quellcodeausschnitt 3.1 zeigt eine Beispielanfrage mit dem Ziel, den Namen aller Produkte zu ermitteln, die als kompatibel zu einem speziellen anderen Produkt markiert sind.

[1] `http://www.disi.unige.it/person/MascardiV/Download/DISI-TR-06-21.pdf` [Letzter Zugriff: 26.11.2012]

**Quellcode 3.1: PQL-Anfrage (Zeile 1) und Antwort (Zeilen 3-6)**

```
1 compatibleWithProduct GET INSTANCES GET "Name"
2
3 <PQLresults>
4   <PQLresult PQLtype="Name" PQLValue="Comp product name A" />
5   <PQLresult PQLtype="Name" PQLValue="Comp product name B" />
6 </PQLresults>
```

**Tip 'n Tell** Als Backendsystem wird die Web-basierte *Tip 'n Tell*-Middleware genutzt [MF06]. Das gesamte System besteht dabei aus dem Tip 'n Tell-Webservice und verschiedenen RDF-basierten SPDO Stores.

Diese in C# implementierte Middleware implementiert unterschiedliche Funktionen. Zum einen dient sie als Vermittler zwischen PQL-Anfragen und P-RDF-Daten, die von den einzelnen SPDO Stores bereitgestellt werden. Zusätzlich wird ein Cache-Mechanismus bereitgestellt, so das wiederholte Anfragen mit identischem Inhalt beschleunigt verarbeitet werden können. Das System stellt dabei sicher, dass die Datenintegrität gewährleistet ist.

Zur Anwendungsseite hin stellt ein sogenannter *Communication Manager* die Schnittstelle bereit. Zusätzlich bietet der Manager eine Menge von Mustern natürlichsprachlicher Fragen, welche der SPDO Informationen in Kombination mit den ursprünglichen PQL-Ergebnissen instanziiert werden können. Mit zusätzlichen PQL-Anfragen kann ein Kunde weitere Produktbeschreibungsdaten und Präsentationsdaten erkunden. Dabei werden schematische W-Fragen (Was, Wo, Wer) auf Produktinformationen abgebildet, die von ermittelten P-RDF Beschreibungen bereitgestellt werden.

In einer überarbeiteten Tip'n Tell Architektur [MF07] wurden einige Veränderungen am bisher beschriebenen Konzept durchgeführt (siehe Abbildung 3.2). Die Daten aus den URL-referenzierten SPDOs werden nun über einen sogenannten SPDO-Broker aus dem Netz ermittelt, wodurch es möglich wird, diese Daten erst dann zusammenzustellen, wenn diese auch benötigt werden. Somit stellen die Daten stets den zum Abrufzeitpunkt aktuellsten Stand dar. Die Informationen des Brokers werden anschließend über ein sogenanntes *Dynamic Product Interface* dem Benutzer zur Verfügung gestellt [MJ07]. Diese Schnittstellen werden entweder auf einem mobilen Gerät implementiert oder direkt in ein *„tangible Product"* integriert (mit Hilfe von eingebetteten Systemen).

### 3.2.2 Prottoy Middleware / FedNet Framework

Prottoy ist eine weitere Middleware für Systeme, die auf intelligenten Objekten [KFN05] basieren. Diese nutzt eine Schichtenarchitektur, um Anwendungen eine vereinheitlichte

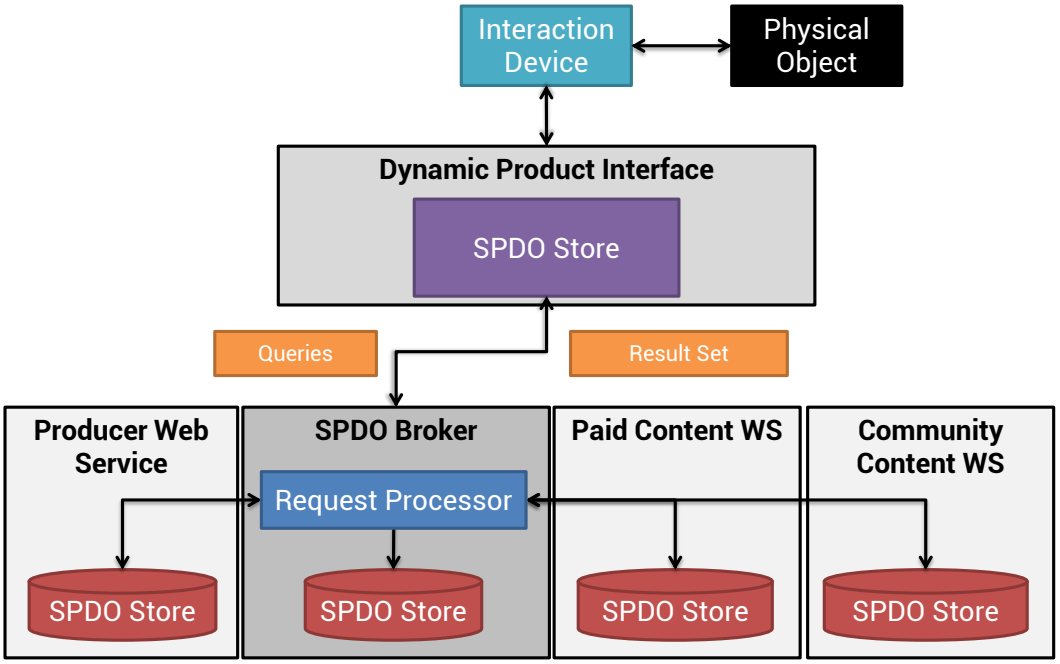



Abbildung 3.2: Überarbeitete *Tip'nTell* Architektur

Sicht auf die darunterliegende intelligente Umgebung zu geben und eine Hardwareabstraktion zu leisten. Die häufigsten Funktionen von intelligenten Objekten werden dabei zu einem gemeinsamen Kern zusammengefasst und eine Sammlung zusätzlicher Diensteprofile repräsentiert die den gemeinsamen Kern umgebende, Plugin-basierte „Wolke" von Spezialisierungsmodulen. Anwendungsentwickler erhalten von der Middleware eine einheitliche Schnittstelle, um bidirektional mit den intelligenten Objekten interagieren zu können, unabhängig von deren Typ und Eigenschaften. Im System wird ein Dokumenten-basierter Ansatz genutzt, bei dem die Anforderungen von Anwendungen und die Dienste von intelligenten Objekten durch einzelne Dokumente beschrieben werden [KNPJ08]. Die Laufzeitumgebung ist, basierend auf diesen Dokumenten, in der Lage die intelligente Objekte und die jeweiligen Anwendungen spontan zu funktionalen Einheiten zu gruppieren.

**Intelligente Objekte** Im Hinblick auf die Entwicklung der Middleware wurden von Prottoy folgende Eigenschaften von intelligenten Objekten besonders hervorgehoben:

1. **Aufforderungscharakter und Variationen der digitalen Anreicherung:** Jedes physische Objekt, unabhängig von Größe oder Form, hat bestimmte Merkmale, die beeinflussen wie Benutzer mit dem Objekt umgehen.

2. **Erscheinungsbild mit Wahrnehmungsrückmeldung:** Eine wichtige Anforderung bei der Ausstattung von Alltagsgegenständen mit digitalen Funktionen ist die Tatsa-

chen, dass das Objekt danach immer noch das gleiche Erscheinungsbild haben muss wie vor der Ausstattung. Zusätzliche ist eine Rückmeldung der digitalen Funktionen an den Endbenutzer notwendig, damit die Interaktion in jedem Zustand korrekt durchgeführt werden oder zumindest entsprechend reagiert werden kann (z.B. sollte ein intelligentes Objekt den Benutzer darauf hinweisen, falls die „Intelligenz“ durch eine Störung nicht oder nur teilweise verfügbar ist).

3. **Push-Pull Methode:** Im Bereich der intelligenten Umgebungen spielen sogenannte proaktive Dienste eine entscheidende Rolle. Dabei gibt es zwei grundlegende Ansätze: Sensornetzwerke und intelligente Objekte. Sensornetze benutzen einen Top-Down-Ansatz, denn sie dienen der reinen Datenbeschaffung im „Feld“, wohingegen die Verarbeitung an zentraler Stelle geschieht, wodurch eine Trennung zwischen reiner Datengenerierung und nachgelagerter Datenverarbeitung entsteht. Intelligente Objekte nutzen einen Bottom-Up-Ansatz, denn sie kombinieren Sensoren und Verarbeitungslogik in einer physischen Einheit.

4. **Objektgedächtnis:** Um eine Verarbeitung im intelligenten Objekt selbst durchführen zu können, muss dieses auch mit einem Gedächtnis ausgestattet werden, welches z.B. statische Daten zur Selbstbeschreibung als auch Logging- und Sensorinformationen enthalten kann [Sch07].

**Architektur** Basierend auf den genannten Entscheidungen und Anforderungen wurde nun eine Middleware entwickelt, welche grundsätzlich aus zwei Komponenten besteht: dem Artefakt Wrapper und dem virtuellen Artefakt. Der Wrapper kapselt dabei das eigentliche intelligente Objekt, während das virtuelle Artefakt sein Gegenstück auf Anwendungsseite darstellt.

Der **Artefakt Wrapper** stellt eine Schichtenarchitektur bereit, in der fundamentale Funktionalitäten von intelligenten Objekten zu einer „Core“-Komponente kombiniert werden. Das *Kommunikationsmodul* bietet die Möglichkeit der Kommunikation in die Umgebung und kapselt gleichzeitig alle Transportvorgänge. Das *„Discovery“-Modul* erlaubt die Anmeldung und Bekanntmachung von Diensten. Das *Benachrichtigungsmodul* erlaubt es allen anderen Modulen ihren Status zu verbreiten. Der *Artefaktspeicher* beinhaltet Konfigurationsdaten, Profilbeschreibungen und andere temporäre Daten. Der *„Client-Handler“* verteilt Anfragen an die entsprechenden Module und Profile. Schließlich bietet das *„Profilrepository“* die Möglichkeit, die einzelnen Profile zu speichern und bei Bedarf zu starten. Jedes Profil repräsentiert dabei eine spezifische Funktionalität und implementiert die zur Umsetzung notwendige Logik. Jedes Profil kann entweder einen Sensor darstellen und somit Daten bereitstellen oder als Aktuator fungieren und Aktionen ausführen.

Ein **virtuelles Artefakt** bietet eine Infrastrukturunterstützung für Anwendungen. Entgegen vieler zentralistischer Ansätze läuft das virtuelle Artefakt im Bereich der eigentlichen An-

wendung. Das *Kommunikationsmodul* stellt dabei die Verbindung zum jeweiligen Artefakt her. Das *„Locator"-Modul* fungiert als Gegenstelle für das Discovery-Modul des Artefakt Wrappers. Das *Speichermodul* erlaubt es Anwendungen Logginginformationen zwischenzuspeichern (in einer XML-basierten Datenbank). Das *Proxymodul* wird verwendet, um die gespeicherten, historischen Informationen zu verändern. Zusätzlich kann mit diesem Modul auch ein intelligentes Objekt simuliert werden, in dem historische Daten eines anderen Objektes zweckentfremdet werden. Schließlich stellt das *Datenverarbeitungsmodul* Methoden zur Datenfilterung, zur -aggregation und zur -interpretation bereit. Entwickler können diese Module implementieren, um die jeweiligen Verarbeitungslogik bereitzustellen.

Anwendungen, die mit intelligenten Objekten arbeiten, bestehen in dieser Architektur aus einer Sammlung von funktionalen Aufgaben (**Aufgabenorientiertes Anwendungsmodell**). Die einzelnen Aufgaben einer Anwendung werden (analog zu den Profilen der Objekte) in einem eigenen XML-Dokument beschrieben. Innerhalb dieses Dokuments werden alle Aufgaben gelistet und sowohl die Profile definiert, die notwendig sind um diese Aufgabe zu erledigen, als auch die Schnittstelle zur Kommunikation zwischen Anwendung und Artefakt. Die eigentliche Kommunikation wird anschließend über eine Http-basierte REST-Schnittstelle und durch den Austausch von XML-Dateien abgewickelt.

Sowohl die bisher beschriebenen Artefakte als auch die Anwendungen sind vollständig unabhängig von einer Infrastruktur und werden beide durch deklarative Dokumente (Aufgaben- und Profilbeschreibung) beschrieben. **FedNet** bietet nun die Möglichkeit beide Seiten zu verbinden, basierend auf den jeweiligen Dokumenten. FedNet stellt mit Hilfe der Dokumenten-Semantik eine Verbindung zu Artefakten her und kann von Anwendungen über eine REST-Schnittstelle angefragt werden (siehe Abbildung 3.3). FedNet besteht dabei aus den folgenden 4 Komponenten:

1. **Anwendungs-Repository:** Beinhaltet alle Anwendungen, die auf FedNet laufen. Dabei wird die Anwendungsbeschreibung automatisch mit den aktuellen Zugangsdaten aktualisiert, sobald die Anwendung auf FedNet gestartet wurde.

2. **Artefakt-Repository:** Verwaltet alle Artefakte, die sich in der FedNet Umgebung befinden. Dabei werden die Profile der einzelnen Artefakte im FedNet System abgelegt und ebenfalls um die jeweiligen Zugangsdaten ergänzt.

3. **FedNet Core:** Stellt alle Kernfunktionalitäten von FedNet bereit. Beim Start einer Anwendung werden alle Tasks bereits als Template angelegt und erst beim Zugriff mit der jeweiligen Verbindung zum Artefakt gefüllt.

4. **Zugangspunkt:** Repräsentiert die physische Umgebung bzw. die physische Verbindung zwischen FedNet und den jeweiligen Artefakten.

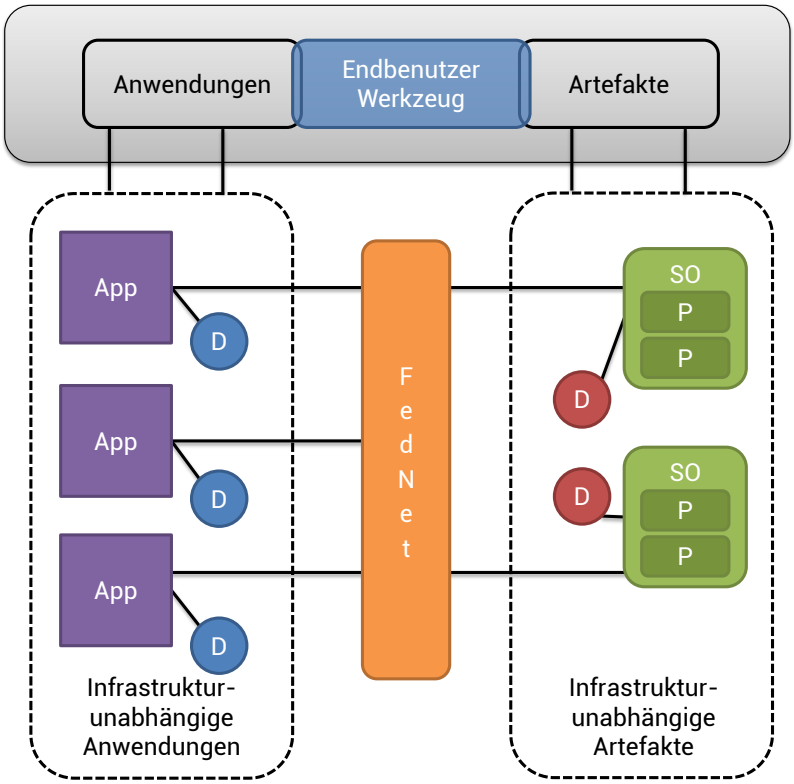



Abbildung 3.3: Abstrakte Darstellung des FedNet Workflows (P: Profil, SO: Intelligentes Objekt, App: Anwendung, D: Dokument)

**Beispielszenarios** Im Rahmen der Entwicklung der beiden Systeme Prottoy und FedNet wurden einige Anwendungen entwickelt, die die Systeme nutzten. **RoonRoon** ist ein ansteckbarer Teddybär, der als Benutzerschnittstelle zu Informationsdiensten fungiert. Er überwacht die Aktivität des Benutzers (z.B. laufen, gehen, sitzen) und kann kontextabhängige, personalisierte Benachrichtigungen ausgeben.
Das zweite Szenario zeigt in einem **kooperativen Wohnzimmer** proaktives Verhalten (z.B. „Das Licht wird automatisch eingeschaltet, sobald jemand den Raum durch die Tür betritt". Verschiedene Objekte im Raum wurden instrumentiert (Tür, Telefon, Lampe, TV, usw.) und bieten alle ein Profil zur Zustandsüberwachung. Die Zustände von TV und Lampe könnten zusätzlich über ein zweites Profil verändert werden.
Das dritte System (**Virtuelles Aquarium**) dient zur Verbesserung der Mundhygiene, in dem dem Benutzer das korrekte Putzen der Zähne gezeigt wird. Dem Benutzer wird dazu ein Aquarium mit Fischen auf einem Display im Badezimmer angezeigt. Die Qualität des Putzvorgangs wird dabei direkt in der „Lebendigkeit" der Fische im Aquarium reflektiert.

### 3.2.3 SmartProducts

Das Projekt *SmartProducts*, gefördert vom 7. Rahmenprogramm der Europäischen Union, besteht aus zehn Partnern aus Industrie und Forschung mit einer Laufzeit von 2009 bis 2012 [Sma12].

SmartProducts entwickelt die wissenschaftliche und technologische Basis zur Erstellung intelligenter Objekte (sogenannter SmartProducts) mit integriertem *pro-aktivem Wissen* [Müh07]. Diese SmartProducts helfen Kunden, Entwicklern und Arbeitern mit der immer stärker ansteigenden Komplexität und dem immer größeren Variantenreichtum moderner Produkte besser umzugehen. SmartProducts nutzen dazu sogenanntes pro-aktives Wissen, um die Kommunikation und die Zusammenarbeit zwischen Menschen, anderen Produkten und der Umgebung zu verbessern. Dieses pro-aktive Wissen beinhaltet dabei Wissen über das eigentliche Produkt (z.B. Eigenschaften, Funktionen, Abhängigkeiten, Benutzung, usw.), die Umgebung (z.B. physischer Kontext, andere Produkte in der Umgebung, usw.) und die Benutzer (Eigenschaften, Fähigkeiten, Absichten, usw.). Zusätzlich beinhaltet das pro-aktive Wissen auch Arbeitsabläufe und Interaktionswissen, um es dem SmartProduct zu ermöglichen pro-aktiv multimodale Dialoge mit dem Benutzer zu initiieren. Somit können SmartProducts mit Entwicklern, Arbeitern und Konsumenten, die mit ihnen interagieren, „reden“, diese „anleiten“ und ihnen „assistieren“. Einige dieser Daten werden dabei bereits zusammen mit dem Produkt entwickelt, andere hingegen werden erst während des Produktlebenszyklus mit Hilfe der eingebauten Sensor- und Kommunikationsfähigkeiten ermittelt. Die angestrebte Zielgruppe umfasst dabei den Endbenutzerbereich, sowie die Produktions- und Automatisierungsindustrie als auch die Luftfahrtindustrie.

Innerhalb des Projekts werden dabei folgende Ziele verfolgt:

- Erstellung einer Plattform zur Bereitstellung von SmartProducts, die pro-aktives Wissen besitzen und nutzten
- Entwicklung von Werkzeugen zur Akquise und zur Wartung von pro-aktivem Wissen
- Analyse der Mehrwerte solcher Systeme
- Showcase der Möglichkeiten von SmartProducts in 3 Anwendungsszenarien: „Smart Kitchen“, „Smart Car“ und „Aircraft Manufacturing“

Das Projekt SmartProducts erforscht dabei alle Aspekte, die in Bezug zu intelligenten Objekten stehen, wie z.B. Akquise, Modellierung, automatisches Schließen und Verwaltung von pro-aktivem Wissen, um eine technologische Basis zu schaffen um pro-aktives Wissen in intelligente Objekte zu integrieren. Zu diesem Zweck ist eine ausgiebige und umfassende Forschung in diversen wissenschaftlichen Bereichen von Nöten. SmartProducts hat dabei Beiträge in den folgenden Bereichen geleistet [Sma12]:

1. **Produktzentrisches Wissensmanagement**: *SmartProducts* bringen das Konzept des ontologisch modellierten Wissensmanagements weiter, von der Unterstützung verteilter Communities bis zur Unterstützung des Produktlebenszyklus, vom Design bis zur tatsächlichen Nutzung, von Entwicklern zu Endbenutzern. Jede Gruppe wird dabei von speziellen maßgeschneiderten Werkzeugen unterstützt, welche alle auf das gemeinsame Wissen zugreifen können, unabhängig von dessen Ersteller.

2. **Wissensmodellierung und logisches Schließen**: Bisher wurden keine Ansätze versucht dieses passive Wissen mit aktiven Wissenskomponenten (z.B. Aufgabenmodelle oder Arbeitsabläufe) zu verbinden, welche essentiell sind um den pro-aktiven Charakter der *SmartProducts* umsetzten zu können. Aus diesem Grund wurde ein übergreifendes Modellierungsrahmenwerk entwickelt, welches es erlaubt passives Kontextwissen mit einer deklarativen Beschreibung von aktivem Wissen zu verbinden.

3. **Intelligente Objekte und Umgebungen**: Forschung im Bereich von intelligenten Objekten und Umgebungen fokussiert sich in der Regel auf eine spezielle Anwendungsdomäne (z.B. Smart Home oder Smart Office) mit Bezug auf den Endbenutzer oder einen Arbeiter. Im Gegensatz dazu fokussiert sich *SmartProducts* auf industrielle Fertigung und den gesamten Lebenszyklus von Massenware.

4. **SOA-Plattform**: In aktuellen Systemen ist die Dienstekomposition in der Regel fest vorgegeben, basierend auf einfacher Schnittstellenübereinstimmung oder generierten Arbeitsschritten. *SmartProducts* unterstützt größere Freiheitsgrade und selbstorganisierende Konzepte.

5. **Kontextbezogene, pro-aktive, multimodale Interaktion**: Die Kombination von kontextbezogener, multimodaler Interaktion mit pro-aktivem Wissen erfordert substanzielle Forschung in der Methodik und in Werkzeugen zur Erstellung solcher Objekte. Dabei müssen die Objekte eine natürliche, leicht zu benutzende und an den Kontext angepasste Benutzerschnittstelle besitzen.

### MundoCore

Als Basis für die Kommunikation einzelner intelligenter Objekte wird im Projekt die Java-basierte *MundoCore* Plattform genutzt [AKM07]. Deren modulare Architektur unterstützt eine große Anzahl unterschiedlicher Geräte von kleinen Sensorknoten bis zu zentralen Servern. Durch eine strikte Abstraktion konnten die einzelnen Module der Plattform vollständig entkoppelt werden, wodurch die eigentlichen Dienste völlig losgelöst von Funktionen wie Orchestrierung, Verbindung und Fehlerbehandlung ablaufen können. Die verwendete Schichtenarchitektur organisiert die Kommunikation in Gruppen in Anlehnung an die verwendeten Adressierungsschemata und Nachrichtenstrukturen. Dadurch lassen sich die

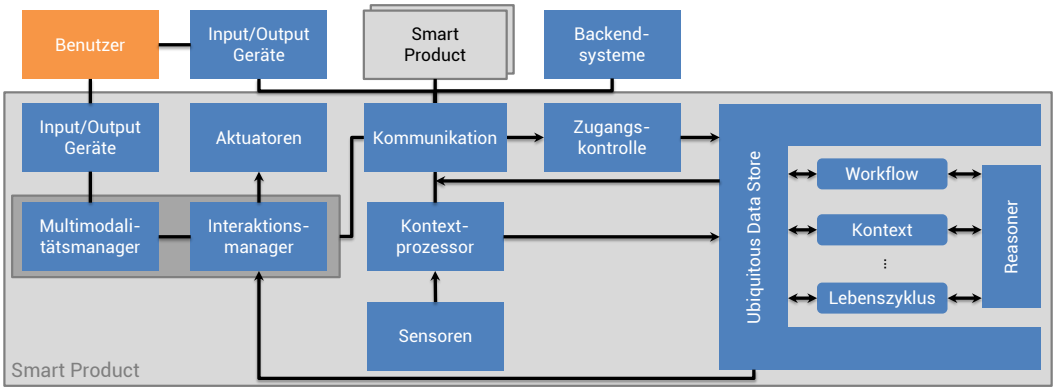



Abbildung 3.4: Vereinfachter Überblick über die SmartProducts Architektur

Welten der verteilten, objektorientierten Programmierung (DOOP), Publish/Subscribe, Peer-to-Peer (P2P) und Multimediadatenverteilung zusammenführen.

### Kernarchitektur von *SmartProducts*

Basierend auf der MundoCore Architektur wurde für das Projekt die SmartProducts Kernarchitektur entwickelt (siehe Abbildung 3.4). Jedes SmartProduct setzt sich dabei aus Eingabe- und Ausgabefähigkeiten, aus Aktuatoren, aus Sensoren und aus produktspezifischen Daten zusammen [SAH+11a]. Um die gesteckten Ziele (1) der Unterstützung einer natürlichen Kommunikation mit dem Endbenutzer und (2) der Nutzung anderer SmartProducts und Ressourcen in der Umgebung (welches für SmartProducts essentiell ist, da diese in der Regel zu wenig eigene Ressourcen besitzen, um eigenständig Aufgaben durchführen zu können) zu erreichen, besitzt jedes Produkt ein Kommunikationsmodul basierend auf der MundoCore Architektur [AKM07]. Im Anschluss folgt eine Übersicht, wie andere Komponenten mit Hilfe der Kommunikationskomponente die gesetzten Ziele (1+2) erreichen [MSM09].

**Kontext** Um auf natürliche Art und Weise mit dem Benutzer kommunizieren zu können, müssen sich Produkte ihres Kontextes bewusst sein. Im Projekt werden dabei zwei Facetten der Kontextbezogenheit betrachtet: ermitteln von Kontext und reagieren mit Hilfe von Kontext. Daher besitzt jedes SmartProduct Regeln, wie es auf Kontext reagiert. Als extrem einfaches Beispiel für ein solches Produkt wird eine Kaffeemaschine angeführt, die die Heizplatte automatisch abschaltet, sobald ein Temperatursensor eine Überschreitung eines Temperaturschwellwerts festgestellt hat. Das Projekt befasst sich daher mit höherwertigem Kontext, wie z.B. dass die Kaffeemaschine bereits Kaffee zubereitet, da sie weiß, dass ihr Besitzer in Kürze eine Pause machen wird. Dies setzt eine große Zahl an (sowohl physischen als auch virtuellen) Sensoren voraus. Da nicht jedes Produkt damit ausgerüstet sein kann

werden viele dieser Daten aus der Umgebung extrahiert oder abgerufen. Die Komponente *Kontextprozessors* übernimmt diese Aufgabe.

**Interaktion** Ein Hauptbestandteil des Prozesses zur Ausstattung von Objekten mit „Intelligenz“ ist eine natürliche Interaktion mit dem Benutzer. Das Projekt definiert dabei drei wichtige Bestandteile: (1) automatisierte Prozesse zur Vermeidung jeglicher unnötiger Interaktion, (2) pro-aktive Führung des Benutzers durch die nicht-automatisierbaren Aufgaben und (3) eine natürliche Interaktion mit dem Objekt und der Umgebung. Die Umsetzung dieser Anforderungen übernehmen der *Interaktionsmanager* und der *Multimodalitätsmanager*.

**Ubiquitous Data Store** Während ihres gesamten Lebenszyklus benötigen intelligente Objekte eine große Menge an Informationen, wie z.B. technische Daten zur Herstellung oder Handbücher für Endbenutzer. Die Objekte könnten aufgrund ihrer geringen Kapazitäten in der Regel diese Daten nicht alle lokal vorhalten. Auch wird empfohlen nicht alle Daten entfernt zu speichern, da in diesem Falle ohne Datenverbindung überhaupt keine Informationen vorliegen. Das Projekt schlägt zu diesem Zweck einen Speicher vor, der gewisse Daten vorab bereitstellt, die in der nächsten Phase des Lebenszyklus benötigt werden. Dieses Verhalten wird durch ein passendes Lebenszyklusmodell ermöglicht. Zusätzlich wird ein Speichersystem propagiert, welches die gesamte Umgebung als Datenspeicher nutzt. Dazu wurden genaue Metriken definiert, die bestimmen in welcher Art und Weise SmartObjects als (temporärer) Speicher genutzt werden können [SAH+11b]. Da viele dieser Objekte mobil sind, achtet der Algorithmus darauf, dass möglichst Speicher genutzt werden, die sich in der Nähe befinden, um die Verfügbarkeit und die Geschwindigkeit des Zugriffs zu optimieren. Diese Funktionalitäten werden unter dem Begriff *mobile cache cloud* zusammengefasst. In der SmartProducts Architektur werden diese Funktionalitäten vom *Ubiquitous Data Store* realisiert. Detailliertere Informationen zur Arbeitsweise des Store sind als vertraulich eingestuft und liegen nicht öffentlich vor.

**Datenmodell** Das ontologisch modellierte Wissen von SmartProducts teilt sich in zwei große Bereiche auf: passives und aktives Wissen [NdT11]. Die Abbildung 3.5 zeigt alle Komponenten, die zusammen das pro-aktive Wissen bilden. Die Unterscheidung zwischen aktiv und passiv stellt sich wie folgt dar:

- **Passives Wissen:** Ontologien, die Datenstrukturen bereitstellen um die unterschiedlichen Konzepte, die relevant für SmartProducts sind, zu beschreiben.
- **Aktives Wissen:** Prozedurales Wissen, die das Produkt nutzt um Entscheidungen zu treffen und Aktivitäten auszuführen. Dies beinhaltet Problemlösungsstrategie und Aufgabenkontrollmechanismen.

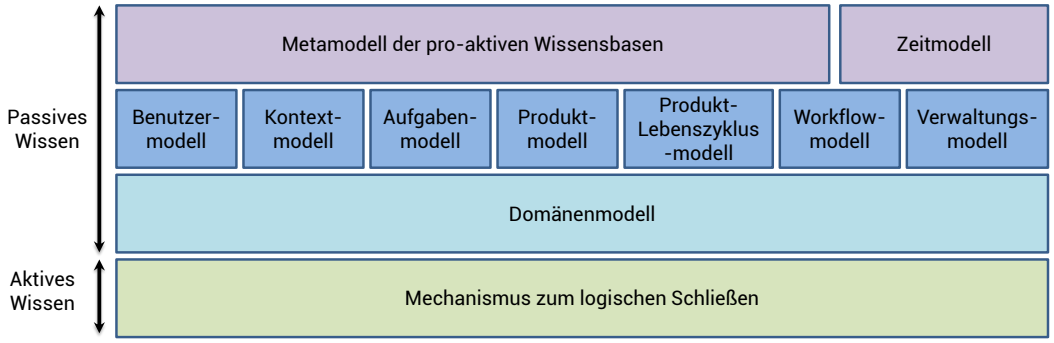



Abbildung 3.5: Konzeptuelle Modelle von SmartProducts

Die konzeptuellen Ontologien sind als Menge von OWL-Dateien organisiert. Die Dateistruktur spiegelt dabei nicht zwingend die Ordnung der Module aus Abbildung 3.5 dar. Da die generischen Modelle stark miteinander vernetzt sind, würde eine Aufteilung in viele verschiedene Dateien die Wartung erschweren ohne einen größeren Mehrwert zu generieren, daher wurde darauf verzichtet. Die Modell selbst stehen online[2] und bestehen aus den folgenden Modulen:

1. *Universelle Ontologien:* Beinhalten alle gemeinsamen und anwendungsunabhängigen Konzepte.

2. *Domänenontologien:* Erweiterungen der universellen Ontologien um anwendungsspezifische Daten.

Alle Konzepte dieser Ontologien wurden selbst entwickelt, da existierende Ansätze wie z.B. DOLCE[3] oder [KKM06] als zu schwergewichtig für SmartProducts angesehen wurden, und werden im Prinzip von den folgenden vier Hauptkonzepten abgeleitet (welche alle Subkonzepte von owl:Thing sind:

- *Abstract* - Beinhaltet alle Entitäten, die nicht einer klaren Position in der Raumzeit zugeordnet werden können, wie z.B. Materialien, Einheiten oder physische Eigenschaften.

- *Agent* - Alle Entitäten, die als Agenten eine Rolle spielen.

- *SpatialThing* - Alle Entitäten, die eine räumliche Position besitzen (z.B. geografische Koordinaten).

- *TemporalThing* - Alle Entitäten, die einen Bezug zur Zeit haben und sich beispielsweise über die Zeit hinweg verändern.

[2] `http://kmi.open.ac.uk/projects/smartproducts/ontologies`, [Letzter Zugriff: 26.11.2012]
[3] `http://www.loa.istc.cnr.it/DOLCE.html`, [Letzter Zugriff 16.04.2012]

**Datenverwaltung und -zugriff** Basierend auf den in den vorherigen Abschnitten erläuterten Ontologien wurde eine Verwaltungsinfrastruktur für diese semantischen Daten für Android-Geräte entwickelt [NdGV11]. Zusätzlich zu den SmartProducts Strukturen wurde ein System geschaffen, welches für beliebige generische semantische Anwendungen auf der Android-Plattform genutzt werden kann. Um es diesen Geräten zu ermöglichen auf semantische Daten anderer Quellen zuzugreifen, wurde eine SPARQL-Engine für verteilte Anfragen entwickelt, die Kontakt mit Webservern auf den jeweiligen Geräten aufnimmt, wodurch es möglich ist, die Anfragen auch auf mehreren Geräten zu starten. Um einfache und einheitliche Schnittstellen zu nutzen, werden die SPARQL-Anfragen über HTTP zwischen den Geräten transportiert. Die Datenspeicherung der ermittelten semantischen Daten in Form von RDF-Trippeln wird dabei der verwendeten Softwarelösung (*Sesame*[4]) überlassen.

### 3.2.4 Tales of Things and Electronic Memory

*Tales of Things and Electronic Memory* (TOTeM) ist ein Forschungsprojekt zwischen fünf akademischen Instituten im Vereinigten Königreich mit dem Ziel das Gedächtnis und den Wert von „alten“ (Alltags-)Objekten zu erforschen [BHSdB10]. Nach Aussagen des Projekts besteht für das Internet der Dinge bereits im Domänenmanagement der Lieferkette und Energieverbrauch ein breites Verständnis, während hingegen der Bereich des sogenannten *Augmented Memory* noch kaum diskutiert wurde. Tales of Things widmet sich daher der Idee, Alltagsgegenstände mit Geschichten ihrer Benutzer auszustatten [dJBHS11].

Die Kernkomponente des Systems bildet der Tales of Things Dienst, der es Benutzern erlaubt eine virtuelle Präsenz von Alltagsgegenständen zu erstellen und es somit anderen zu erlauben ebenfalls alle Daten über dieses Objekt betrachten zu können und diese sogar selbst ergänzen zu können. Dazu kann entweder das TOTeM-Internetportal[5] genutzt werden oder mit QR-Codes ausstattete Objekte „in der Wildnis", die dann mit Hilfe einer App für Android oder iPhone gelesen werden können und als Link zu den eigentlichen Daten über das Objekt fungieren [Kin02]. Der eigentliche Service besteht somit aus zwei Komponenten:

1. **TOTeM Webseite:** Die Webseite (siehe Abbildung 3.6) stellt ein Portal bereit, mit dessen Hilfe man Geschichten über das Objekt (z.B. Namen, Bilder, Erzählungen) hochladen und bearbeiten und mit Multimediainhalten verknüpfen kann. Zusätzlich lässt sich ein QR-Code generieren, der sich ausgedruckt am Objekt befestigen lässt. Des Weiteren werden die Informationen auch als RDFa-Format in die Webseite integriert, so dass diese auch maschinell weiterverarbeitet werden können.

[4] `http://www.openrdf.org/` [Letzter Zugriff: 26.11.2012]
[5] `http://talesofthings.com/`, [Letzter Zugriff: 26.11.2012]


Abbildung 3.6: *Tales of Things and Electronic Memory* Webseite

2. **TOTeM mobile Anwendung:** Die mobile Applikation dient hauptsächlich der Betrachtung der gespeicherten Geschichten nach dem „Auffinden“ des Objektes. Es ist allerdings auch möglich eigene Daten, z.B. in Form von Geschichten oder Geokoordinaten, hinzuzufügen. Die Daten für die mobile Anwendung werden über eine REST-Schnittstelle vom TOTeM-Dienst geladen.

Wenn eine neue Identität generiert wird, haben Benutzer die Möglichkeit einen Namen, eine Beschreibung, einen Status (öffentlich oder privat) und Schlüsselworte anzugeben. Optional können weitere Daten wie das Kaufdatum, das Herstellungsjahr, ein Bild, die ID eines RFID-Tags und eine Positionsinformation festgelegt werden. Nun können beliebige Geschichten für das Objekt angelegt werden. Eine Geschichte besteht dabei aus einem Titel, einem Textblock und zusätzlichen Schlüsselworten. Optional können auch hier eine Position und Links zu Mediendateien angegeben werden.

Zusätzlich zu dem bereits beschriebenen Anwendungsfall werden im Projekt auch spezialisierte Anwendungen betrachtet [dJBHS11]. Für den Oxfam's Charity Store[6] (**Retail**

[6] `http://www.oxfam.org.uk/` [Letzter Zugriff: 26.11.2012]

**Charity Shop**), in dem Second-Hand-Objekte verkauft werden, wurden einzelne Objekte mit TOTeM ausgestattet, wobei eine Audionachricht vom Vorbesitzer für diese Objekte abrufbar ist. Die Objekte wurden zusätzlich zu den QR-Codes auch mit RFID-Tags versehen, um es Kunden zu ermöglichen auch ohne die mobile Anwendung mit Hilfe eines Kiosksystems auf die Audionachricht zuzugreifen. Des Weiteren wurden Studien mit anderen Gruppen (**Community Groups**) erstellt, z.B. älteren Menschen, Menschen mit Gedächtnisschwierigkeiten oder Minderheiten. Zu diesem Zweck wurde das System genutzt, um Videotagebücher der Teilnehmer festzuhalten. Für den Museumsbetrieb (**Museums**) stellt TOTeM eine Ergänzung dar. Dem Projekt zufolge können so die Exponate mit virtuellen TOTeM-Entitäten und QR-Codes ausgestattet werden, wodurch die Besucher des Museums mit Hilfe der TOTeM-App auf weitere Informationen zugreifen können.

### 3.2.5 UbisWorld

Das Ziel des Systems *UbisWorld*, welches am Deutschen Forschungszentrum für Künstliche Intelligenz (DFKI) entwickelt wurde, ist die Konzeption, Entwicklung und Anwendung von Werkzeugen zur ubiquitären (allgegenwärtigen) Benutzermodellierung, um laut [Hec05]

> „(...) einerseits die neuen Möglichkeiten des verändert-instrumentierten Benutzerumfeldes miteinzubeziehen, andererseits aber auch die gestiegenen Anforderungen an Transparenz, Privatsphäre und Introspektion zu berücksichtigen."

**Situational Statements** Zentrales Element eines solchen Ansatzes ist die Abbildung des aktuellen Zustands des Benutzers in einer Art und Weise, die ein System verarbeiten kann. Dazu werden sogenannte *situative Aussagen* (Situational Statements) genutzt, um zum Beispiel eine Informationseinheit über den aktuellen Zustand eines Objekts oder eines Benutzers zu beschreiben. Die Gesamtheit aller Aussagen definiert dann das vollständige Wissen, dass ein System über die aktuelle Situation hat. Die Aussagen basieren auf RDF-Trippeln und folgenden deren Syntax *Subjekt - Prädikat → Objekt*. Ein Beispiel für ein solches Tripel könnte lauten Subjekt=*Peter*, Prädikat=*KognitiveLast* und Objekt=*hoch* für die Aussage, dass der Benutzer Peter im Augenblick eine hohe kognitive Last hat. Diese Tripel wird im UbisWorld-System erweitert um für jede Aussage zusätzliche Informationen bereitstellen zu können [Hec02] (siehe Abbildung 3.7).

Zum einen können dadurch sowohl die zeitlichen als auch die räumliche Gültigkeit der Aussage eingeschränkt und somit exakter spezifiziert werden. Des Weiteren lassen sich Informationen über den Besitzer bzw. den Ersteller dieser Aussage festlegen und gleichzeitig kann auch die Sichtbarkeit der Aussage eingeschränkt werden, so dass diese zum Beispiel

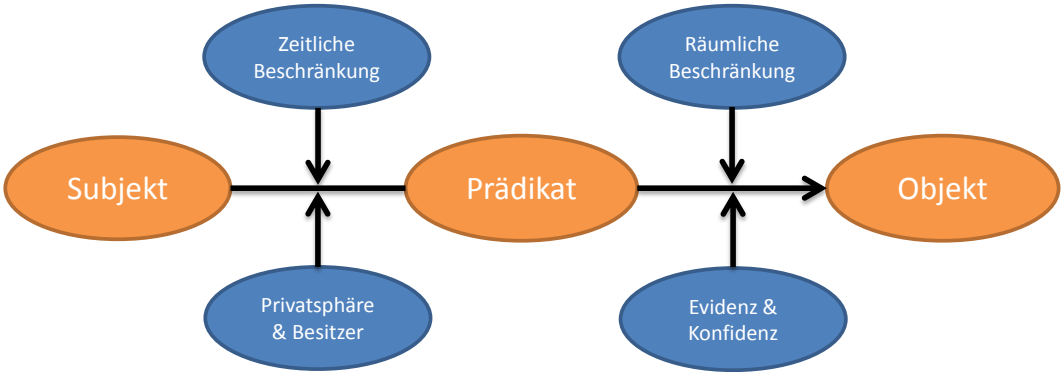



Abbildung 3.7: UbisWorld: Erweiterte RDF-Tripel

nur für bestimmte Systeme genutzt werden kann. Auf Implementierungsseite werden diese Aussagen in der eigens dafür definierten Sprache *SituationML* kodiert.

**Extended Resource Identifiers** Um die Subjekte und Objekte in den beschriebenen Aussagen korrekt identifizieren zu können, müssen diese eine eindeutige ID besitzen. Da das System UbisWorld mit Webtechnologien umgesetzt wurde, bietet es sich an URIs zu verwenden [BLFM05]. Um die Darstellung möglichst kurz und für den Menschen lesbar zu halten, werden sogenannte *UbisIdentifier* (Ubid) verwendet. Da sich alle Konzepte innerhalb der UbisWorld-Ontologie wiederfinden, kann auch die Angabe eines Namensraums entfallen. Für Benutzer werden zum Beispiel die Klartextnamen des Benutzers verwendet. Zusätzlich werden allen solchen Strings eine sechsstellig Zahl angefügt, um auch bei Namensgleichheit eine Eindeutigkeit zu gewährleisten (zum Beispiel `Peter.210004`).

**GUMO - the General User Model Ontology** UbisWorld bringt zusätzlich eine eigene Ontolgoie zur Benutzermodellierung mit [HSB+05]. Deren Aufbau wird durch die situativen Aussagen beeinflusst. Das Benutzermodell ist dabei in drei Bestandteile *Zusatz* (auxiliary), *Prädikat* (predicate) und *Wertebereich* (range) partitioniert.

$$Subjekt \left\{ \begin{array}{l} Zusatz \\ Praedikat \\ Wertebereich \end{array} \right\} Objekt$$

Ein Beispiel, welches das Interesse an Fußball in drei konkreten Werten darstellt, könnte lauten:

$$Subjekt \left\{ \begin{array}{l} Zusatz = hatInteresse \\ Praedikat = Fussball \\ Wertebereich = wenig - mittel - viel \end{array} \right\} Objekt$$

Für den Zusatz sind folgende Werte in der GUMO vorgesehen: hasProperty, hasInterest, hasBelieve, hasKnowledge, hasPreference, hasRegularity, hasPlan, hasGoal, hasDone, hasLocation. Diese Liste ist dabei nicht als vollständig anzusehen, sondern dient als Startpunkt für eine erste Umsetzungen und bietet daher ein breites Spektrum an häufig genutzten Eigenschaften von Benutzern.

Zur Darstellung von Daten zum Benutzer in unterschiedlichen Domänen finden sich in der GUMO Konzepte zur folgenden Kategorien: (1) Demographische Daten und Kontaktinformationen, (2) Persönlichkeit und Charakteristiken, (3) Stimmung und Emotionen, (4) mentaler und physiologischer Zustand, (5) Rollen und Beruf und (6) Örtlichkeit und Orientierung.

**UbisWorld** Auf diesen beiden Konzepten aufbauend wurde die UbisWorld entwickelt. Diese besteht aus einer Sammlung von sechs additiven Ontologien. Diese umfassen die Domänen physische Objekte, räumliche und zeitliche Verortung, Aktivitäten, Situationen und Inferenz. Zusätzlich werden N-äre Relationen zur Integration der situativen Aussagen genutzt (siehe Abbildung 3.8).

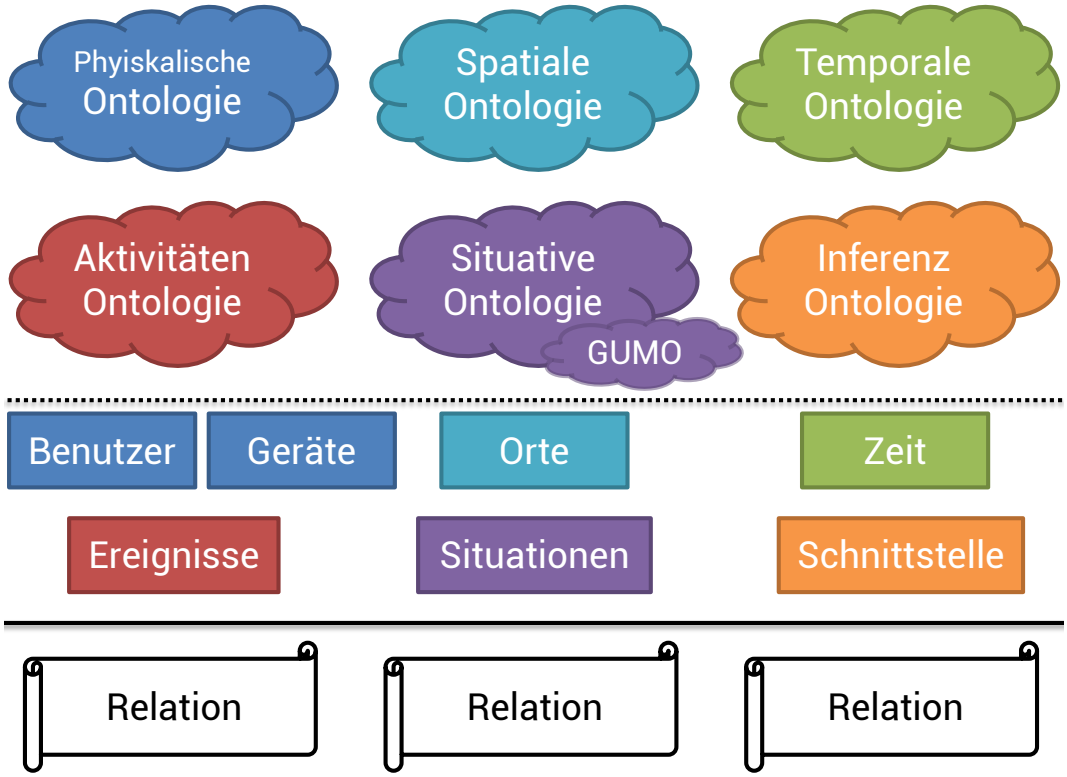



Abbildung 3.8: UbisWorld mit grundlegenden Ontologien (oben), zugehörigen Instanzen (mitte) und Relationen (unten)

**UbisWorld Toolset** Um die Nutzung dieser UbisWorld zu erleichtern wurden einige Werkzeuge erstellt, die es einem Benutzer erlauben mit dem System zu interagieren [HLM+09]. Der *UbisBroker* ist ein Dienst, mit dessen Hilfe Benutzerdaten aus unterschiedlichen Quellen harmonisiert werden (durch eine Abbildung auf die GUMO) und anschließend ausgetauscht werden. Dies erlaubt eine anwendungsübergreifende Kommunikation mit Hilfe semantisch annotierter Daten. *UbisSearch* ist eine Suchmaschine zum Auffinden von Informationen in der UbisWorld. Das System verwendet Indizes zur effizienten Suche und nutzt UbisIdentifier zur Datenrepräsentation. Der *UbisEditor* erlaubt es dem Benutzer direkten Einfluss auf die Daten der einzelnen Ontologien zu nehmen. Über eine Webanwendung kann der Benutzer die Ontologie verändern oder Daten eingeben. Zusätzlich lässt sich die Sicht auf einzelnen Bereiche der Ontologie einschränken, um einen besseren Überblick zu erhalten. *UbisAccess* stellt ein Modell zur Zugriffskontrolle für UbisWorld dar. Dabei können einzelne Rollen definiert werden, die jeweils unterschiedliche Rechte haben. Jedem Benutzer kann somit eine Menge an Rollen zugewiesen werden.

### 3.2.6 RFID-Based Automotive Network (RAN)

Das bis Ende 2012 laufende und vom Bundesministerium für Wirtschaft und Technologie (BMWI) geförderte Forschungsprojekt *RFID-Based Automotive Network* (RAN), hat zum Ziel die Entwicklung einer RFID-basierten hybriden Steuerungsarchitektur und Bewertungsmethode für Wertschöpfungsketten [REG+11] (siehe Abbildung 3.9). Am Beispiel der Automobilindustrie wird gezeigt, dass diese im Regelfall prozessbezogene Informationen nicht zeitnah an alle Beteiligten der Lieferkette weitergibt. Aufgrund der Komplexität des Gesamtprozesses ist dies jedoch eine wünschenswerte Eigenschaft, um die Planung und Steuerung der folgenden Prozesse frühzeitig beeinflussen zu können. Zu diesem Zweck wird ein kombiniertes Datenmanagement entwickelt, welches produktspezifische Daten (wie zum Beispiel Qualitätsinformationen) dezentral über RFID-Tags austauscht. Zusätzlich werden auftragsspezifische Daten (wie zum Beispiel der Auftragsstatus) zentral in unternehmensspezifischen Datenbanken abgelegt. Ein sogenannter Informationsbroker übernimmt nun die überbetriebliche Kommunikation von prozessrelevanten Daten. Mit dieser Architektur erfolgt der Transport von produktspezifischen Daten direkt am jeweiligen Objekt und der Transport von prozessrelevanten Daten echtzeitnah über ein Backend [SRLRT11].

Somit kann zusätzlich zur Informationsübermittlung an spätere Partner der Lieferkette auch eine direkte Prozessanpassung mit Hilfe der an Objekte angebrachten Tags erfolgen. Die Partner des Projekts erstellen einen generischen Modulbaukasten, mit dessen Hilfe die optimalen Werkzeuge (zum Beispiel für Track&Trace) für das jeweilige Unternehmen (zum Beispiel ein Fertigungsunternehmen) gefunden werden können. Da allerdings die Integration eines solchen Ansatzes in eine bestehende Infrastruktur mit hohen Investitionskosten verbunden ist, wird seitens des Projekts auch eine Wirtschaftlichkeits- und

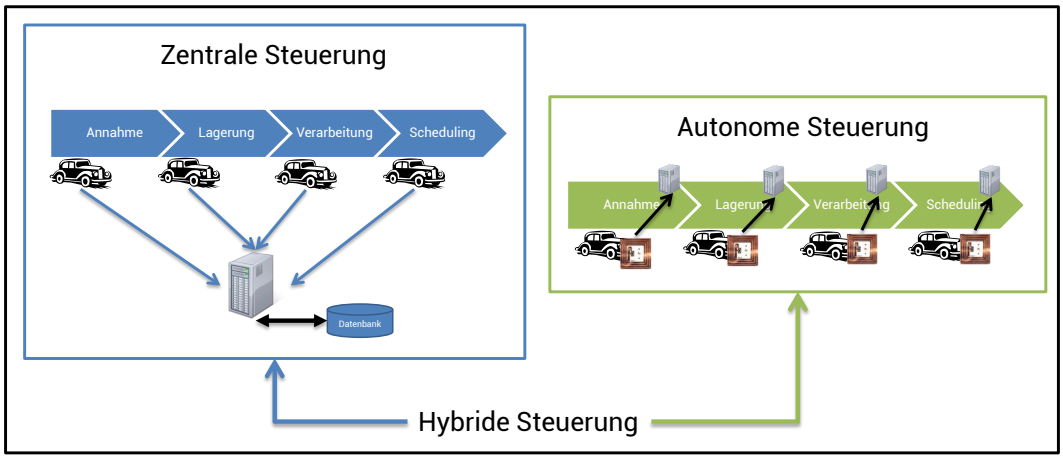



Abbildung 3.9: RAN: Hybride Steuerung

Ressourceneffizenzuntersuchung durchgeführt. Mit Hilfe von Modellfabriken und Logistiklager wird die gesamte Wertschöpfungskette nachgebildet und die hybride RFID-basierte Produktionssteuerung demonstriert.

## 3.3 Ansätze basierend auf Industrieprojekten

### 3.3.1 Physical Markup Language

Die *Physical Markup Language (PML)*[7] entwickelt am Auto-ID Center des Massachusetts Institute of Technology als gemeinsame Sprache zur Beschreibung von physischen Objekten, Prozessen und Umgebungen [BMKL01]. Bei der Entwicklung lag der Schwerpunkt auf der Schaffung eines einfachen, ausreichenden und effektiven Standards mit dem Ziel die Schwachstellen und Nachteile bisheriger Ansätze zu eliminieren und mit Hilfe von Erweiterungen trotzdem flexibel auf Anforderungen aus vielen Domänen reagieren zu können (siehe Abbildung 3.10). Es existiert eine XML-basierte Darstellung der PML-Elemente. Folgende wichtige Eigenschaften wurden bei der Entwicklung der PML berücksichtigt:

- **Allgemeingültigkeit:** Die PML soll als universeller Standard so viele Domänen wie möglich abdecken. Aus diesem Grund wurde auf eine feingranulare Detaillierung verzichtet. Vielmehr wurden die Möglichkeiten darauf ausgerichtet auch Domänen mit unterschiedlicher Darstellung der gleichen Eigenschaften zu einer Vereinheitlichung der Darstellung über die PML zu bewegen.

[7] `http://web.mit.edu/mecheng/pml/index.htm` [Letzter Zugriff: 26.11.2012]

- **Einfachheit:** Im Gegensatz zu anderen Spezifikationen, die sich durch ihre hohe Komplexität nicht durchsetzten konnten (zum Beispiel SGML), soll die Nutzung der PML so einfach wie möglich erfolgen, um eine hohe Akzeptanzquote zu erreichen (zum Beispiel die SGML-Ableger HTML und XML).

- **Einführungspfad:** Die erste Version der PML soll so einfach wie möglich gehalten sein, um die Einstiegshürde gering zu halten. Die Spezifikation soll aber offen für Erweiterungen sein, die sich oft erst in der alltäglichen Nutzung als notwendig herauskristallisieren.

- **Umfassende Datentypen:** Die PML definiert verschiedene Sichten auf die jeweiligen Datentypen, welche als *statisch*, *temporal*, *dynamisch* und *algorithmisch* betrachtet werden können. Statische Daten sind in der Regel über die gesamte Lebenszeit eines Objekts konstant (zum Beispiel physische Form), im Gegensatz zu temporalen Daten, die sich während der Lebenszeit verändern (zum Beispiel die Position des Objekts). Zusätzlich werden Daten als dynamisch bezeichnet, wenn sich diese regelmäßig verändern (zum Beispiel durch Sensoren gemessene Werte). Abschließend definieren algorithmische Daten das Verhalten des Objekts (zum Beispiel mit Modellen oder Prozessbeschreibungen). Diese Sichten sind allerdings nie als absolut und endgültig zu betrachten, da z.B. die Form eines Objekts statisch ist. Falls das Objekt allerdings unerwarteter Weise beschädigt wird und sich die Form verändert, kann diese auch als temporal angesehen werden.

- **Abstrakte Nomenklatur:** Da die PML viele verschiedene Domänen abdecken soll und diese oft für identische Sachverhalte unterschiedliche Begriffe nutzten, wird mit der PML eine abstrakte Nomenklatur definiert, um einen sicheren Datenaustausch zwischen den Domänen zu ermöglichen.

- **Robuster Betrieb:** Da sich PML-Daten häufig ändern und sehr stark mit anderen Dokumenten vernetzt sind, müssen Anwendungen, die die PML einsetzen, auch mit unvollständigen oder fehlerhaften Daten umgehen können und einen robusten Betrieb sicherstellen, da diese nicht durch das Format selbst gewährleistet werden kann.

- **Unterstützung von Datenarchiven:** Die PML unterstützt genaue Zeitstempel für alle periodischen und temporalen Daten, damit diese auch später noch ausgewertet werden können und im Sinne eines Archivs längerfristig vorgehalten werden können.

- **Standard für Maßeinheiten:** Da große Teile der PML-Daten durch Einheiten definiert werden, ist eine genormte Anwendung von Einheiten ein entscheidender Faktor um Daten sicher domänenübergreifend nutzen zu können. Dazu wird das sogenannte

SI-System (*Le Système International d'Unités*) eingesetzt [BIP06] und durch weitere Einheiten der NIST[8] (National Institute of Standards and Technology) ergänzt.

- **Grundlegende und abgeleitete Daten:** Zur Vermeidung von Inkonsistenzen werden in der PML so weit als möglich keine redundante Daten abgelegt, die aus anderen Daten abgeleitet oder berechnet werden können.
- **Standardisierte Syntax & Globale Sprache:** Zur Darstellung der PML wird auf das weit verbreitete und standardisierte XML-Format zurückgegriffenen, welches mit Hilfe von Techniken wie XML-Query auch effizient angesprochen werden kann. Da die PML weltweit zum Einsatz kommen soll, wird zusätzlich auf nationale Ausdrücke verzichtet.
- **Erleichterte Anwendungsentwicklung:** Alle bisherigen Anforderungen dienen zum großen Teil dazu, die Entwicklung von Anwendungen zu erleichtern. Zusätzlich wird die syntaktische Struktur der PML so angelegt, dass eine maschinelle Verarbeitung verbessert wird.

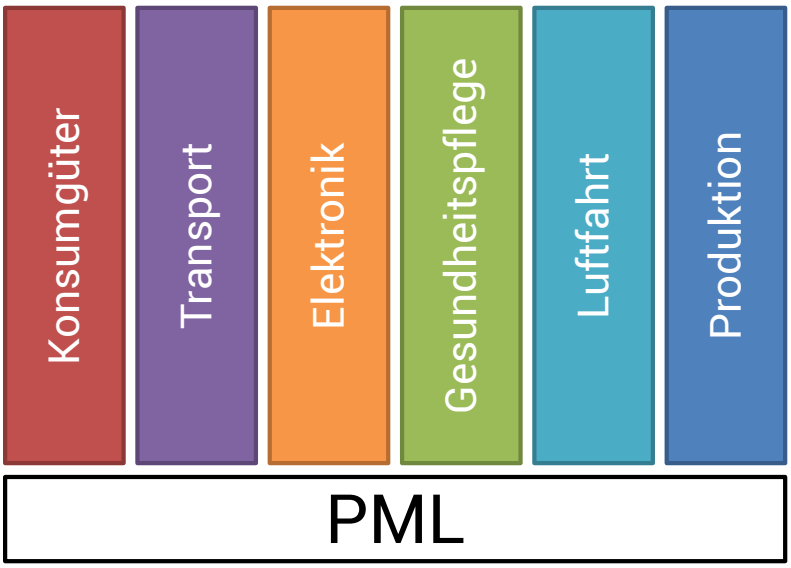



Abbildung 3.10: PML-Komponenten aus verschiedenen Domänen

Ein wichtiger Schwerpunkt der PML ist die *Klassifikation* von Objekten. Da jede Klassifikation eines Objekts immer nur eine subjektive Sichtweise darstellt, erlaubt es die PML einzelne Objekte mehrfache zu klassifizieren (z.B. kann ein Apfel sowohl zur Klasse „Frucht" als auch zur Klasse „Kreisförmig" gehören). Dazu besteht eine Klassendefinition der PML aus den drei Attributen *system* (identifiziert das Klassifikationsschema), *type* (gibt den Typ oder die Ebene im Schema an) und *name* (Bezeichnung der konkreten Instanz innerhalb

[8] `http://www.nist.gov` [Letzter Zugriff: 26.11.2012]

des Schemas). Zur Definition der Hierarchie nutzt eine PML-Klasse die Attribute *subscribe* (andere Klassen von denen diese Klasse abgeleitet wurde) und *members* (andere Klassen die von dieser Klasse ableiten). Siehe dazu auch Codebeispiel 3.2.

**Quellcode 3.2: PML-Teilebeziehung am Beispiel eines Bestecks**

```
<part label="Besteck" epc="01.0001AC.0000114.000019800">
  <part label="Gabel" epc="01.0001AC.0000114.000019851">
    <link type="logical"></link>
  </part>
  <part label="Messer" epc="01.0001AC.0000114.000019852">
    <link type="logical"></link>
  </part>
  <part label="Löffel" epc="01.0001AC.0000114.000019853">
    <link type="logical"></link>
  </part>
</part>
```

Da sehr viele Objekte aus einer Menge von kleineren Einzelobjekten zusammengesetzt sind, kann mit Hilfe der PML, in Ergänzung zur Klassendefinition, auch der Aufbau von Objekten und die Beziehung zwischen Objekten definiert werden. Zu diesem Zweck werden Objekte mit der Eigenschaft *Part* versehen. Jedes Einzelteil wird durch eine eigene Part-Definition dargestellt. Mit Hilfe des *Link*-Attributs lässt sich die Teile-Beziehung näher definieren durch die drei unterschiedlichen Werte *logical* (eine konzeptuelle Beziehung, die nicht unbedingt eine physische Entsprechung haben muss), *physical* (eine tatsächliche physische Verbindung) und *mechanical* (eine mechanische Verbindung, die auch genauer in der Art und Weise des Zusammenspiels definiert werden kann). Durch die Möglichkeit der Schachtelung von Part-Angaben lassen sich hierarchische Modelle von Objekten generieren; eine „Ladung“ besteht z.B. aus „Paletten“, die wiederum z.B. aus „Boxen“ bestehen (siehe Quellcode 3.3).

**Quellcode 3.3: PML-Hierarchie einer Containerladung**

```
<part label="Ladung" epc="01.0001AC.0000115.000019851">
  <part label="Palette" epc="01.0001AC.0000115.000019852">
    <link type="physical">Base</link>
    <part label="Box A" epc="01.0001AC.0000115.000019853">
      <link type="physical">On</link>
    </part>
  </part>
</part>
```

### 3.3.2 GS1 EPCglobal

Das *EPCglobal*-Framework der non-profit Organisation *GS1*[9] dient der Implementierung und Integration des elektronischen Produktcode (EPC) in Unternehmen. Das Framework wurde vom *Auto-ID Center* des Massachusetts Institute of Technology entwickelt. GS1 selbst entstand durch einen Zusammenschluss der Betreiber der Systeme *EAN* (European Article Number) und *UPC* (Universal Product Code). Gleichzeitig wurden die beiden konkurrierenden Modelle zur Produktidentifikation über Barcodes EAN-13 (13-stellig) und UPC-A (12-stellig) in das gemeinsame Format *GTIN* (Global Trade Item Numbers) überführt.

#### Elektronischer Produktcode (EPC)

Der elektronische Produktcode hat laut GS1 die Funktion einer akkuraten, unmittelbaren und kosteneffizienten Sichtbarmachung von Informationen (siehe [TAB+10]). Das Ökosystem, welches um diesen Code herum geschnürt wurde, besteht aus einer Sammlung in gegenseitiger Beziehung stehender Hardware, Software und Datenstandards in Kombination mit gemeinsamen Netzwerkdiensten, welche von EPCglobal und anderen betrieben werden. Nutzer dieser Aktivitäten sollen Endbenutzer (Organisationen, die EPCglobal-Standards und Netzwerkdienste, im Rahmen ihres Businessmodells nutzen) und Lösungsanbieter (Entwickler von Systemen für Endbenutzer basierend auf EPGglobal-Standards und Diensten) sein.

Die Aktivitäten des Frameworks beschäftigen sich mit drei unterschiedlichen aber verwandten Themenfeldern (siehe Abbildung 3.11):

- **EPC-Standards zum Austausch physischer Objekte**: Endbenutzer tauschen physische Objekte aus, welche mit EPCs versehen sind. Der Austausch erfolgt in der Regel in Form von Handelsgütern, die sich in einem Produktlebenszyklus befinden. Weitere Anwendungsfälle für den Einsatz von eindeutigen Produktcodes (aber außerhalb des Bereichs der Handelsgüter) sind zum Beispiel Bibliotheken oder Assetmanagement. EPC definiert hierfür nun einen Standard zur eindeutigen Identifikation und zum sicheren Austausch von Objekten.
- **EPC Datenaustausch**: Zusätzlich zum physischen Objekt tauschen Endkunden weitere Daten untereinander aus. Das Ziel ist die Visibiliät des Produkts auch außerhalb der vier Wände eines Partners in der Lebenszykluskette zu erhöhen. EPC definiert hier nun Standards zum Datenaustausch zwischen speziellen Gruppen untereinander und zwischen einem Partner und der Öffentlichkeit, welche zum Beispiel mit Hilfe von Netzwerkdiensten auf diese Daten zugreifen können.

[9] http://www.gs1.org/ [Letzter Zugriff: 26.11.2012]

- **EPC-Infrastruktur zur Datenerfassung**: Bevor allerdings Daten zwischen Partnern ausgetauscht werden können, müssen diese erst intern erfasst und verwaltet werden. Daher führt jeder Endkunde Operationen wie zum Beispiel das Erstellen von EPCs für neue Produkte, das Verfolgen von Produkten mittels Sensoren und das Erfassen von Daten innerhalb seines Unternehmens durch. Das EPC-Framework definiert auch hier Standards zur Datenerfassung und Schnittstellen zum Austausch und zur Datenablage.

Basierend auf dieser Dreiteilung wurden ca. 20 Standards definiert, wovon einige ausgewählte Standards im Folgenden näher betrachtet werden. Dabei handelt es sich zum einen um den „EPC Tag Data Standard“, der den Übergang vom physischen Objekt zur Infrastruktur darstellt sowie um die Standards „EPC-Information Services“ (EPCIS), „Pedigree“ und „Object Naming Service“ (ONS), die im Bereich des Datenaustausches angesiedelt sind.

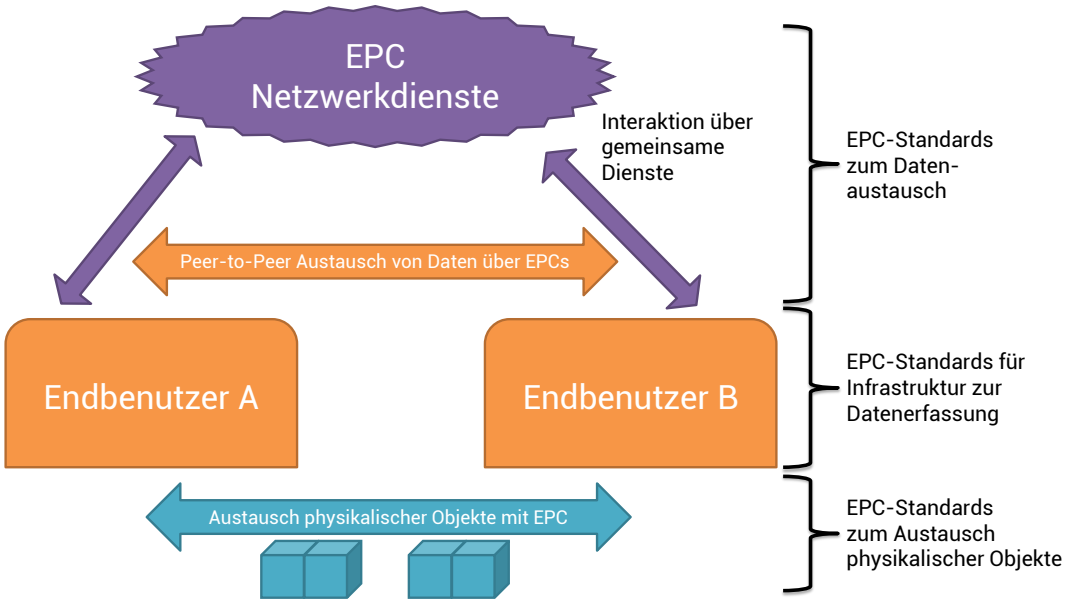



Abbildung 3.11: Architektur des EPCglobal Frameworks

### EPC Tag Data

Der EPC Tag Data Standard definiert sowohl den eigentlichen *Electronic Product Code*™ als auch den Inhalt von sogenannten Gen 2 RFID Tags [EPC11]. Bei diesen Tags handelt es sich um RFID-Tags, die der EPCglobal UHF Klasse 1 Generation 2 Luftschnittstelle Version 1.2 oder größer genügen [EPC08].

Der EPC stellt einen universellen Identifikator für beliebige physische Objekte dar und wird in Informationssystemen genutzt, um Objekte verfolgen oder anderweitig adressieren zu

| EPC Schema | GS1 Schlüssel | Typischer Anwendungsfall |
|---|---|---|
| sgtin | SGTIN | Handelsgüter |
| sscc | SSCC | Logistikeinheiten |
| sgln | GLN | Örtlichkeiten |
| grai | GRAI | Mehrwegprodukte |
| giai | GIAI | Sachwerte |
| gdti | GDTI | Dokumente |
| gsrn | GSRN | Dienstebeziehungen |
| gid | [n. vorhanden] | [nicht festgelegt] |
| usdod | [n. vorhanden] | US Depot of Defence |
| adi | [n. vorhanden] | Luftfahrt/Militär |

Tabelle 3.1: EPC URI Schemata

können. Dabei nutzt ein großer Teil der Anwendungen RFID-Tags als Datenträger. Dies hat zur Folge, dass sich ein großer Teil des Tag Data Standards mit der Kodierung von EPC Daten (und anderen nicht-EPC Daten) auf RFID-Tags beschäftigt. EPC und RFID-Tags sind daher eng miteinander verwandt, stellen aber keine Synonyme dar. EPC ist ausschließlich der Identifikator und die RFID-Tags der Datenträger. Daher können RFID-Tags auch verwendet werden um andere Informationen zu transportieren, genauso wie EPCs auch mit anderen Medien übertragen werden können. Im Folgenden werden nur einige Details zur EPC-Darstellung erläutert, die Kodierung auf RFID-Tags wird nicht betrachtet.

Bei der maschinellen Verarbeitung tritt der EPC in der Regel als *URI* (Uniform Resource Indetifier, siehe [BLFM05]) in Erscheinung und zwar unabhängig davon, ob der EPC über RFID-Tags oder ein anderes Medium transportiert wird. Diese URI wird nun als *Pure Identity EPC URI* bezeichnet und hat folgende beispielhafte Gestalt:

`urn:epc:id:`*`schema:komponente1.komponente2....`*

Jede EPC URI beginnt somit mit dem Indikator `urn:epc:id`, darauf folgt die Angabe eines Schemas (siehe Tabelle 3.1). Anschließend werden einzelne Komponenten benannt, deren Darstellung vom verwendeten Schema abhängt.

Eine Beispielhafte EPC URI vom Type SGTIN (entspricht der 14-stelligen EAN-Nummer plus Seriennummer) mit der EAN-Nummer `0614141 112345` und der Seriennummer `400` zeigt folgende Zeile:

`urn:epc:id:sgtin:0614141.112345.400`

Innerhalb des EPC Architekurframeworks wird nun je nach Verwendung des EPC zwischen drei unterschiedlichen Ebenen unterschieden:

- **Pure Identity EPC URI**: Die primäre Repräsentation des EPC erfolgt als URI. Diese URI wird in allen EPC Anwendungen genutzt, unabhängig davon wie diese an Objekten angebracht und weiter transportiert wird.
- **EPC Tag URI**: Ein EPC Class-1 Generation-2 RFID-Tag besitzt ein spezielles Feld um die EPC URI plus zusätzliche Kontrollinformationen (die Einfluss auf die Datenaufzeichnung nehmen) abzulegen. Der Begriff EPC Tag URI bezeichnet nun den Inhalt dieses Feldes als Textstring.
- **Binary Encoding**: Da die Daten im RFID-Tag aber nicht als Klartext-String abgelegt werden, sondern in einer komprimierten Binärrepräsentation, bezeichnet man den Binärstrom, der auf einem EPC RFID-Tag abgelegt wird, als Binary Encoding. Für diese Darstellung existiert eine 1-zu-1 Abbildung zur EPC Tag URI.

### EPC Information Services (EPCIS)

Das Ziel der *EPC Information Services* (EPCIS) ist es unterschiedlichste Anwendungen Zugang zu und den Austausch von EPC Daten zu ermöglichen, sowohl innerhalb von Unternehmen, als auch über Unternehmensgrenzen hinweg (siehe [EPC07a]). Daher kann EPCIS auch als EPC Information Sharing betrachtet werden und beinhaltet eine Standardschnittstelle um EPC-Daten aufzuzeichnen und wieder abzurufen, basierend auf festgelegten Dienstoperationen und EPC Daten Standards (vergleiche vorhergehender Abschnitt), sowie eines Mechanismus zur Datensicherheit und eine oder mehrere persistente Datenbanken, wobei die Kommunikation auch direkt von Anwendung-zu-Anwendung ohne die Nutzung einer Datenbank erfolgen kann. Abbildung 3.12 zeigt den typischen EPCIS-Datenfluss.

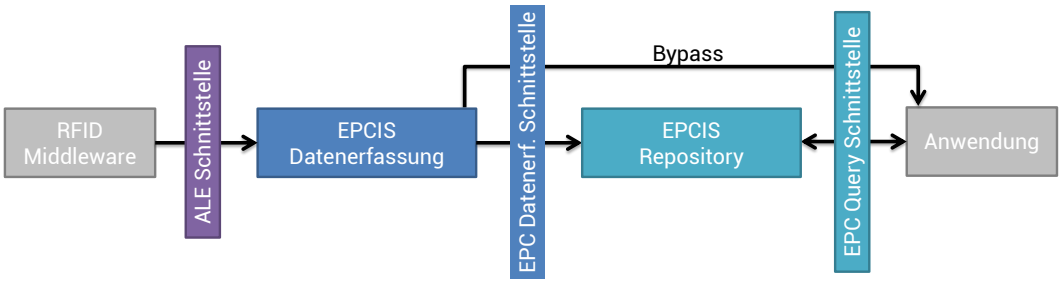



Abbildung 3.12: EPCIS Datenfluss

Neue Daten werden dabei über RFID-Reader erkannt und über Filter vorverarbeitet und anschließend mit Hilfe der RFID Middleware über die sogenannte ALE-Schnittstelle (Application Level Events) zur Verfügung gestellt. Diese Daten werden nun von der EPCIS

Datenerfassung mit Daten anderen Quellen fusioniert und mit neuen Kontextinformationen angereichert und weiter über die EPC Datenerfassungsschnittstelle bereitgestellt. Je nach Anwendungsfall können diese Daten direkt über die EPC Query Schnittstelle von Applikationen abgerufen werden („Echtzeit"-Betrieb) oder die Daten werden zuerst in einem Repository zwischengespeichert und später („bei Bedarf") abgerufen.

*Erweiterbarkeit/Modularität*: Das gesamte Framework ist zusätzlich in mehrere Layer aufgeteilt (siehe Abbildung 3.13):

- Das **abstrakte Datenmodell** beinhaltet alle generischen EPCIS Datenstrukturen, die benötigt werden um eigene Datendefinitionen zu erstellen. Dieser Layer kann als einziger nicht erweitert werden.
- Die **Datendefinitionsschicht** spezifiziert die eigentlichen Daten, die mit Hilfe von EPCIS ausgetauscht werden. Dies schließt sowohl die syntaktische Struktur als auch die semantische Bedeutung der Daten mit ein. Bereits im Standard definiert sind die grundlegenden Typen für Ereignisse aus dem Bereich der Datenerfassung.
- Die **Dienste Schicht** definiert Schnittstellen über die Anwendungen Daten via EPCIS austauschen. Der Standard sieht dabei zwei Module in dieser Schicht vor: die Datenerfassungsschnittstelle und die Query-Schnittstelle (vergleiche auch deren Nutzung in Abbildung 3.12).
- Für die beiden letzten Schichten gibt es über die Definition hinaus auch eine **konkrete Implementierung**. Die Basis-Ereignis-Typen sind dabei als XSD[10]-Datei jeweils für die Datendefinitions- und die Dienste-Schicht definiert. Die Query-Schnittstelle besitzt Implementierungen für die Übertragungsarten SOAP[11], HTTP(S)[12] und AS2[13]. Die Datenerfassungsschnittstelle ist als MsgQ[14] und HTTP-Implementierung verfügbar. Eine Erweiterung kann hier leicht durch neue Implementierungen bestehender Definitionen erfolgen.

*Datenmodell*: Das EPCIS Datenmodell macht eine klare Trennung zwischen Ereignisdaten und sogenannten „Masterdaten". Ereignisdaten entstehen bei der Ausführung von Businessprozessen und werden von der EPCIS Datenerfassungsschnittstelle erstellt und über die Query-Schnittstelle verfügbar gemacht. Sie bestehen aus einzelnen Ereignissen, die jeweils aus einem Ereignistyp und einer Liste von Ereignisfeldern (Schlüssel-Wert-Paare) bestehen. Die Schlüssel werden dabei in der Datendefinition (oder einer Erweiterung) festgelegt. Die Werte sind entweder primitive Daten (zum Beispiel ein Integer-Wert), eine Eintrag aus dem Vokabular oder eine Liste dieser beiden Typen. Masterdaten sind zusätzliche Daten, die es

[10] XML Schema Definition
[11] Simple Object Access Protocol
[12] Hyper Text Tranport Protocol
[13] Applicability Statement 2
[14] Message Queueing

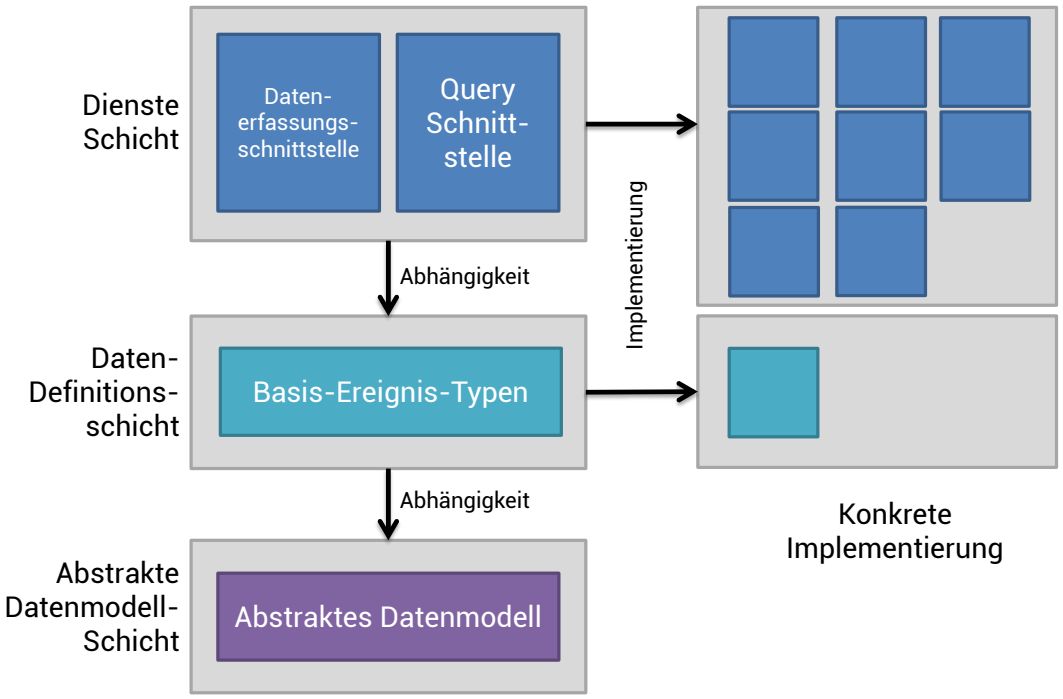



Abbildung 3.13: EPCIS Datenmodell

Anwendungen ermöglichen den Kontext zu ermitteln, um die Ereignisdaten interpretieren zu können (der Inhalt dieser Daten ist nicht in EPCIS spezifiziert). Zu jedem Eintrag eines Vokabulars gibt es zusätzlich eine Liste von Attributen (siehe Abbildung 3.14).

### Pedigree Standard

Der Pedigree (= Herkunft/Stammbaum) Standard spezifiziert eine Architektur für die Handhabung und den Austausch von elektronischen Herkunftsdokumenten zur Nutzung im Bereich von pharmazeutischen Lieferketten [EPC07b]. Die Architektur zielt dabei auf die Nutzung von Dokumenten-basierten, rechtsgültigen Herkunftsnachweisen ab. Als finale Zieldarstellung wird dabei der Zustand gesehen, bei dem alle Komponenten eines Nachweises über das Netzwerk zwischen Partnern verteilt werden und es genügt eine einzige Anfrage zu stellen, um den vollständigen Satz aller Dokumente zu erhalten. Dies ist zum heutigen Zeitpunkt allerdings unrealistisch, so dass man davon ausgeht, das die Dokumente nur zwischen einzelnen Partner ausgetauscht werden und dies entweder über deren Kommunikationssysteme erfolgt oder Web-basierte Techniken (wie zum Beispiel FTP) zum Einsatz kommen.

Ein Herkunftsnachweis (Pedigree) ist dabei ein zertifizierter Beleg der Informationen über jede Verteilung von verschreibungspflichtigen Medikamenten enthält. Dabei wird der Ver-

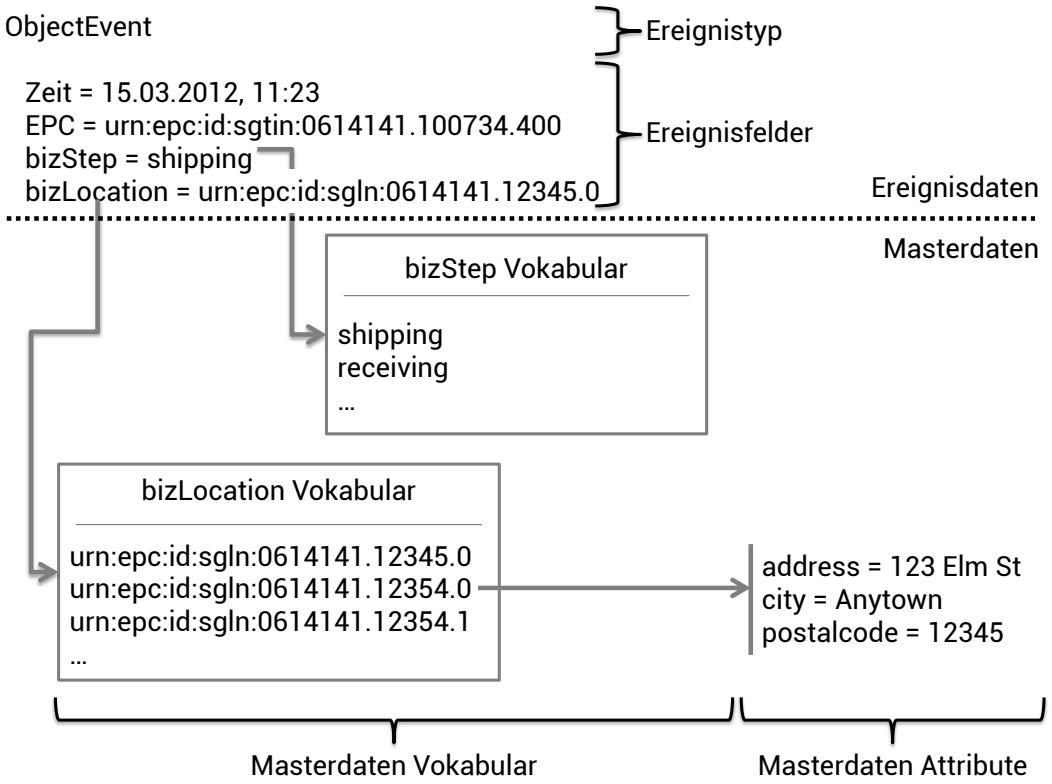



Abbildung 3.14: EPCIS Ereignis- und Masterdaten

kauf durch einen pharmazeutischen Hersteller vermerkt, jeder An- und Weiterverkauf durch Großhändler und Umverpackern und schließlich der Ankauf von Apotheken oder anderen Einrichtungen. Der Nachweis beinhaltet dann Informationen über das Produkt, die Transaktionen, die Verteilung und den Empfänger sowie zusätzliche Signaturen. Ein typischer Ablauf zur Nutzung der Herkunftsnachweise läuft wie folgt ab:

1. Erstellung des Herkunftsnachweises
2. Hinzufügen von ersten Informationen
3. Nachweis zertifizieren (Dokument wird digital signiert)
4. Nachweis wird zum Empfang des zugehörigen Produkts geschickt
5. Empfänger erhält Nachweis
6. Elektronische Authentifizierung des Dokuments
7. Manuelle Authentifizierung aller nicht-elektronischen Bestandteile
8. Abgleich zwischen gelieferten Produkten und Herkunftsnachweis

9. Nachweis wird vom Empfänger digital signiert

Da die Umsetzung des Herkunftsnachweises von den Gesetzmäßigkeiten des jeweiligen Landes abhängt, ist auch die Implementierung oder zumindest Teile davon jeweils von Land zu Land unterschiedlich. Grundlage bilden elektronische Signaturen basierend auf dem ITU-T Standard X.509 (bzw. ISO|IEC 9594-8) [ITU97]. Darauf aufbauend werden dann vom jeweiligen Partner Signaturen auf Basis des RSA-Algorithmus und dem Hash-Algorithmus SHA1 erstellt [BBF+02, U.S00]. Die eigentlichen Daten werden in einer XML-Datei abgelegt, die dann mit X.509 signiert wird. Dies ermöglicht es diesen XML-Abschnitt in einen neue Datei einzubetten (welche dann wieder komplett signiert wird) und damit eine Nachweiskette aufzubauen (siehe Abbildung 3.15).

### Object Naming Service (ONS)

Der *Object Naming Service* (ONS) hat die Aufgabe Informationen über ein EPC aus dem Internet abzurufen. Der Service bedient sich dabei dem Domain Name System (DNS), welches im Internet verwendet wird um Domänennamen in IP-Adressen zu übersetzen (siehe [Moc87a, Moc87b]). In gleicher Art und Weise soll der ONS EPCs in Dienste-Adressen konvertieren.

Der DNS-Dienst fungiert auf Clientseite wie ein Blackblox, das heißt die eigentliche Arbeitsweise wird vor der Anwendung verborgen. In der Regel wird der Client durch das Betriebssystem realisiert, kann aber je nach Umgebung auch direkt angesprochen werden. Dabei wird der Name der gewünschten Domäne sowie der gewünschte Rückgabetyp an den DNS-Server gesendet, der dann mit dem geforderten Typ antwortet. Die Serverseite hingegen ist hierarchisch organisiert, das heißt es gibt DNS-Server auf verschiedenen Ebenen. Diese Hierarchie spiegelt sich direkt im jeweiligen DNS-Namen wieder. Zum Beispiel bedeutet `oms.dfki.de`, dass ein Server aus der Root-Domäne `de` zuständig ist. Auf zweiter Ebene folgt nun der eigentliche DNS-Server `dfki.de` welcher nun die IP-Adresse des Rechners mit dem Namen `oms.dfki.de` kennt und zurückgibt. Der ONS verarbeitet analog dazu eine EPC ID zu einer Diensteadresse, wie im Folgenden Ablauf beschrieben ist (siehe auch Abbildung 3.16):

1. EPC wird aus RFID-Tag gelesen.
   (`10 000 000...`)

2. Reader sendet EPC an lokale Anwendung.
   (`10 000 000...`)

3. Konvertierung des Binärstroms in eine EPC URI.
   (`urn:epc:id:sgtin:0614141.000024.400`)

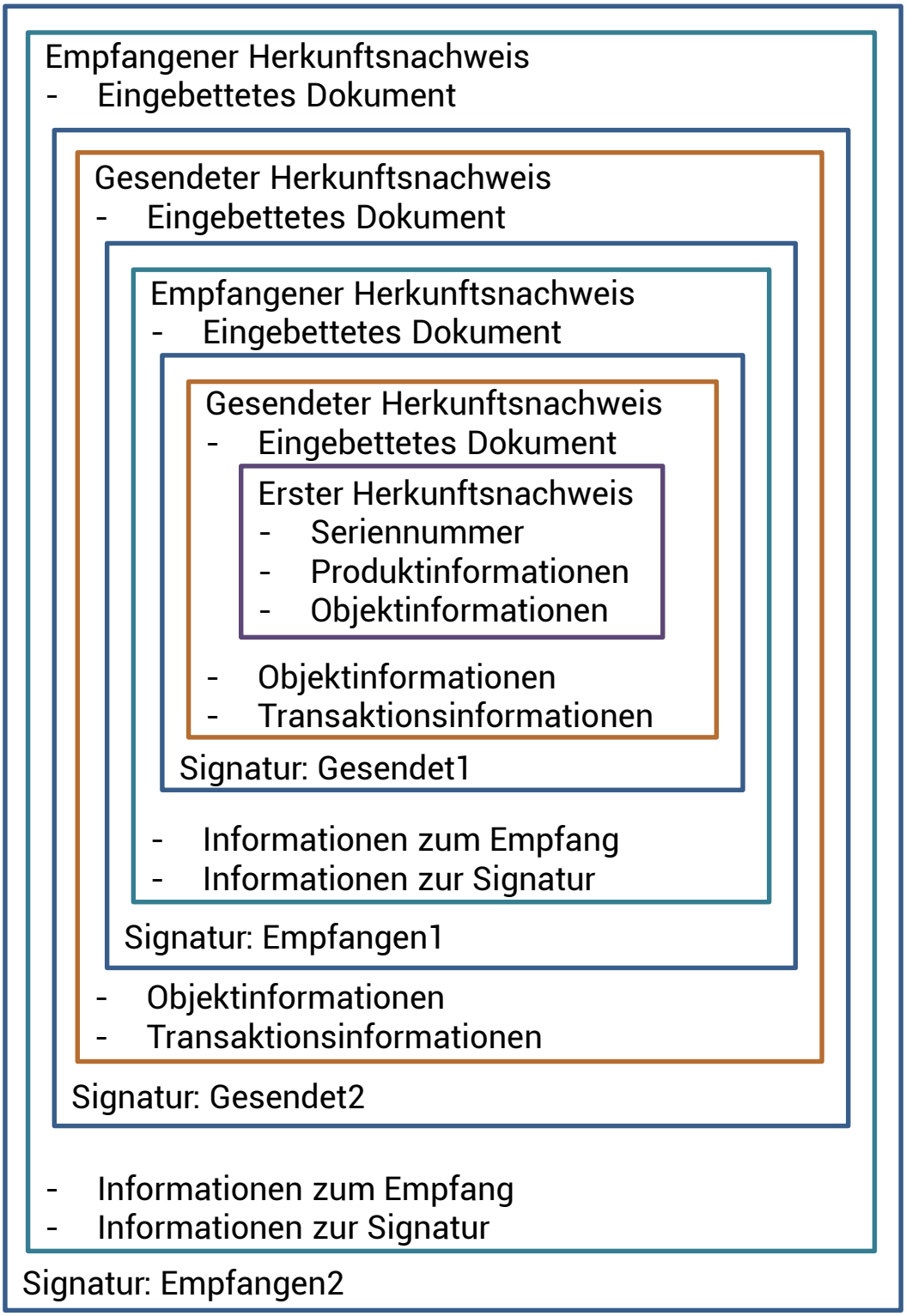



Abbildung 3.15: EPCglobal Pedigree: Nachweiskette über 3 Partner

4. Diese URI wird an den ONS-Resolver weitergeleitet.
   (`urn:epc:id:sgtin:0614141.000024.400`)

5. Der Resolver konvertiert die URI in einen EPC-DNS-Eintrag und stellt mit diesem eine DNS-Anfrage.
   (`000024.0614141.sgtin.id.onsepc.com`)

6. Die DNS-Infrastruktur liefert eine Reihe von Antworten, die auf Dienste zeigen, die zu diesem Produkt gehören.

7. Die lokale Anwendung extrahiert aus diesen Antworten die passende URL.
   (`http://epc-is.example.com/epc-wsdl.xml`)

8. Abschließend kontaktiert die Anwendung die entsprechenden Server im Netz.

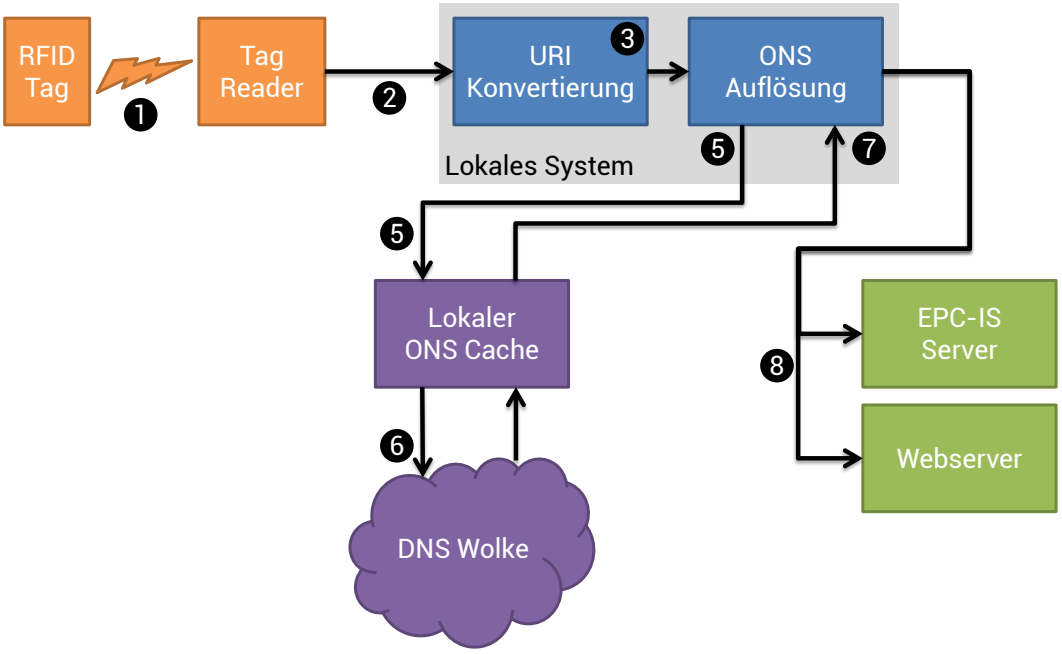



Abbildung 3.16: Typische Anfrage an den Object Naming Service (ONS)

## 3.4 Ontologien und Metadaten

Der Idee des semantischen Webs [BLHL01] folgend, stellt eine Datengrundlage, basierend auf der inhaltlichen Beschreibung digitaler Dokumente mit standardisierten Vokabularen, die eine maschinell verstehbare Semantik haben, einen hohen Mehrwert dar [Wah08a, Wah08b]. Den Kern der semantischen Technologie bilden Auszeichnungssprachen, die eine

formale Syntax und Semantik haben und in Form einer Ontologie eine standardisierte Begrifflichkeit zur Beschreibung digitaler Inhalte bereitstellen. Die semantische Markierung durch Metadaten stellt neben der Markierung für das Layout (HTML) und die Struktur (XML) eines Dokumentes die dritte Ebene einer Dokumentbeschreibung dar. Beide Konzepte werden im Folgenden näher erläutert.

### 3.4.1 Suggested Upper Merged Ontology

Die SUMO (Suggested Upper Merged Ontology) [NP01, PNL02] wurde im Rahmen der *IEEE Standard Upper Merged Ontology Working Group* entwickelt, mit dem Ziel der Erstellung einer übergeordneten Ontologie zur Verbesserung und Förderung der Dateninteroperabilität, der Informationssuche, Inferenzbildung und Verarbeitung von natürlicher Sprache. Entwickelt wurde die SUMO in einer Variante des KIF (Knowledge Interchange Format) und somit in einer Prädikatenlogik erster Stufe definiert [Gen91]. Die Ontologie umfasst aufgrund ihres übergeordneten Charakters nur sehr generische und abstrakte Konzepte und keine domänenspezifische Eigenschaften. Sie bietet aber einen Rahmen um verschiedene Domänenontologien zusammenzufassen. Die Abbildung 3.17 zeigt die Konzepte der ersten 3 Ebenen.

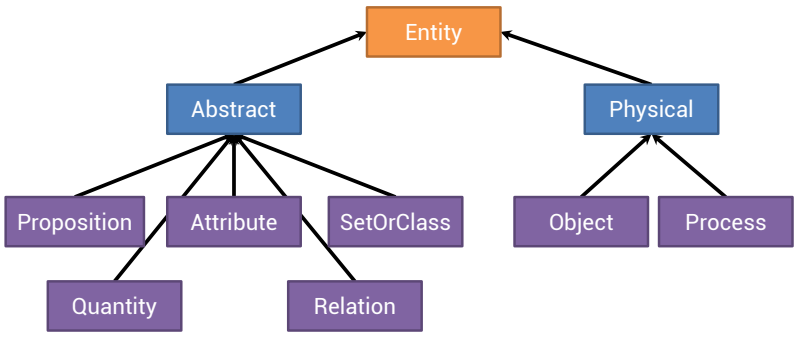



Abbildung 3.17: SUMO-Konzepte der ersten 3 Ebenen nach [Sev03]

Nach [Sev03] hat die SUMO den entscheidenden Nachteil der recht geringen Überdeckung, wodurch zwar eine Vernetzung auf oberster Ebene ermöglicht wird, aber die Darstellung von konkreten Daten in domänenübergreifenden Anwendungen nicht ausschließlich mit der SUMO möglich ist. Dies kann nur durch die Einbindung von tiefer liegenden und domänenspezifischen Ontologien (wie z.B. die Mid-Level Ontology (MILO)) erreicht werden, welche zum Beispiel zurzeit über 27.000 Terme abdeckt[15]. Dadurch verlagert sich allerdings das generelle Problem der Abbildung verschiedenen Ontologien auf tieferen Ebenen.

[15] `http://www.ontologyportal.org` [Letzter Zugriff: 26.11.2012]

### 3.4.2 WonderWeb/DOLCE

Einen anderen Ansatz verfolgt das Projekt *WonderWeb*[16], welche im Rahmen des Programms *European Commission information society technologies (IST)* entwickelt wurde [MBG+01]. Laut WonderWeb ist die Nutzung der Ontologien entscheidend. Sie können zum Beispiel im Sinne eines *semantischen Zugriffs* auf eine spezifische Ressource genutzt werden. Dazu muss allerdings die Bedeutung der genutzten Terme vorab bekannt sein. Dadurch lassen sich aber sehr schlanke Ontologien nutzen, die oftmals nur aus einer Taxonomie bestehen. Eine andere Nutzung ergibt sich zum Zwecke der Aushandlung eines gemeinsamen Verständnisses, um die Kommunikation zwischen mehreren Partnern zu verbessern oder einen Konsens zwischen menschlichen Benutzern und maschinellen Agenten herzustellen. WonderWeb konzentriert sich auf den letztgenannten Typ und bezeichnet diesen als *grundlegende Ontologien (foundational ontologies)*. Ziel des Projekts ist es eine ganze Bibliothek solcher Ontologien bereitzustellen, im Sinne eines modularen Konzepts und im klaren Gegensatz zu großen monolithischen Ontologien.

Das erste Modul dieser Bibliothek bildet die *Descriptive Ontology for Linguistic and Cognitive Engineering (DOLCE)*[17]. Dem beschriebenen Ansatz folgend ist DOLCE keine universelle Standardontologie, sondern stellt einen Startpunkt dar, um Vergleiche und Erläuterungen der Beziehungen zu anderen Modulen zu ermöglichen. Dabei liegt der Fokus der Ontologie auf der kognitiven Schiene mit dem Ziel der Darstellung von natürlicher Sprache und menschlichem Allgemeinverständnis. DOLCE ist eine Ontologie der „Einzelheiten“ (particulars), die im Gegensatz zur „Allgemeinheiten“(universals) keine Instanzen haben. Die DOLCE Taxonomie ist in der ersten Unterebene in vier Konzepte unterteilt (siehe Abbildung 3.18):

- **Endurant und Perdurant:** Die grundsätzliche Unterscheidung zwischen Endurant und Perdurant erfolgt nach den philosophischen Konzepten Kontinuierlichkeiten (continuants) und Ereignissen (occurents) [Sim87]. Endurants sind dabei ganzheitlich vorhanden, so lange sie vorhanden sind. Perdurants treten nur eine kurze Zeit auf oder stellen einen zeitlich begrenzten Zustand dar.

- **Quality:** Alle Entitäten, die wahrgenommen oder gemessen werden können, werden als Quality bezeichnet (z.B. Farbe, Größe, Geruch). Für jede Quality existiert in der Regel eine Menge an möglichen Werten (quale == quality value).

- **Abstract:** Alle Entitäten, die selbst keine räumliche oder zeitliche Eigenschaft haben und selbst keine Quality darstellen, werden als Abstract bezeichnet.

[16] `http://wonderweb.semanticweb.org/` [Letzter Zugriff: 26.11.2012]
[17] `http://www.loa.istc.cnr.it/DOLCE.html` [Letzter Zugriff: 26.11.2012]

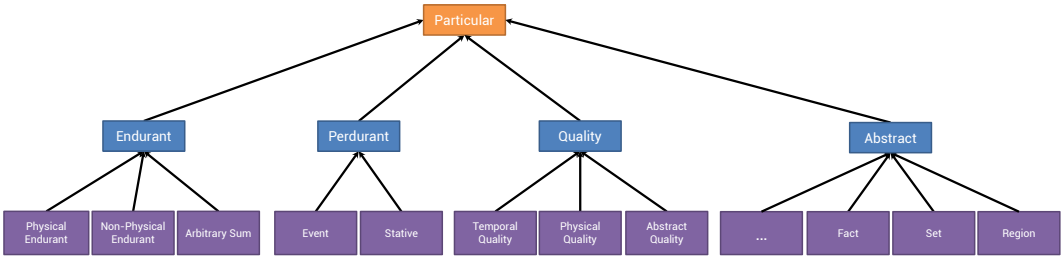



Abbildung 3.18: DOLCE-Konzepte der ersten 3 Ebenen nach [MBG+01]

### 3.4.3 Dublin Core

Die *Dublin Core Metadata Initiative* (DCMI) beschäftigt sich mit der Architektur und Modellierung von Produktmetadaten in Form von Dokumenten [MM04, DCM12]. *DCMI Metadata Terms* ist eine Spezifikation aller Metadaten, die von der DCMI verwaltet werden, welche Eigenschaften, Schlüsselwörter für Kodierungsschemata, Syntaxschemata und Klassen beinhalten. Dabei werden von der DCMI 15 Kernfelder empfohlen und als *Dublin Core Metadata Element Set* bezeichnet. Diese umfassen folgende Bereiche:

- **ID:** beinhaltet eine Möglichkeit dem beschriebenen Dokument eine eindeutige Identifizierung zu geben, die auf bestehenden Standards basiert (z.B. ISSN, URL, URN oder DOI).
- **Technische Definition:** definiert zum einen das Format des Dokuments (format) in Form der Angabe eines MIME-Typs und zum anderen kann die verwendete Sprache angegeben werden (language). Zusätzlich besteht die Möglichkeit einen generischen Typ (type) des Dokuments aus einer vorgegeben Liste zu definieren (z.B. Kollektion, Ereignis, Bild, Ton, Text).
- **Beschreibung von Inhalt:** wird definiert durch die beiden Attribute Titel (title) und Beschreibung (description), welche eine kurze und eine lange Definition des Inhalts bereitstellen. Dies erleichtert es z.B. Endbenutzern passende Dokumente zu finden. Ergänzt werden diese Attribute durch das Feld Überdeckung (coverage), welches eine inhaltliche Abdeckung in Bezug auf räumliche bzw. zeitliche Angaben definiert und das Feld Thema (subject), welches es erlaubt eine Menge an Schlüsselwörtern zu definieren, die das Dokument charakterisieren.
- **Entitäten und Rechte:** definiert den Ersteller (creator), weitere Autoren (contributor) und die veröffentlichte Instanz (publisher). Die Autoren müssen dabei keine natürliche Personen sein. Zusätzlich können sowohl der Rechteinhaber (rightsHolder) als auch die jeweiligen Rechte (rights) definiert werden. Abgeschlossen wird dieser Bereich durch die Angabe zur Echtheit und Nachverfolgbarkeit (provenance) des Dokuments (z.B. eine Signatur).

- **Relationen:** können definiert werden um die Quelle des Dokuments (source) oder Beziehungen zu anderen Dokumenten herzustellen (relation). Zusätzlich kann die Zielgruppe des Dokuments definiert werden (audience).
- **Lebenszyklus:** beinhaltet in der Regel zeitliche Angaben (Datum oder Zeitspanne), die in Bezug zu gewissen Ereignissen gesetzt werden können (z.B. Erstelldatum, letzte Änderung, Inkrafttreten).

Diese Attribute haben bereits diverse Anwendungsgebiete. Es existiert eine dedizierte RDF/XML-basierte Darstellung, die in eigenen Anwendungen genutzt werden kann. Bereits in Verwendung ist diese Möglichkeit z.B. im *OpenDocument*-Format [Ope12] oder im *RSS*-Standard [RSS09]. Zusätzliche könnten die Dublin Core Metaattribute auch in gewöhnlichen HTML-Seiten über das ⟨*meta*⟩-Tag eingebunden werden (siehe Quellcode 3.4 mit den Attributen *subject*, *title*, *format* und *type*).

**Quellcode 3.4: Dublin Core HTML Sample**

```
1 <meta name="DC.subject" xml:lang="en-GB" content="seafood" />
2 <meta name="DC.title" content="First title" />
3 <meta name="DC.format" scheme="DCTERMS.IMT" content="text/html" />
4 <meta name="DC.type" scheme="DCTERMS.DCMIType" content="Text" />
```

## 3.5 Ontologie-Editoren

Zur Erstellung und Bearbeitung von Ontologien sind spezielle Editoren nötig, um die semantisch modellierten Daten komfortabel bearbeiten zu können. Diese Editoren bieten je nach geforderter Anwendung einen unterschiedlichen Schwerpunkt in der Bearbeitung. Einige Editoren stellen die durch Ontologien bereitgestellte Komplexität vollständig dem Anwender zur Verfügung, andere fokussieren sich auf eine beschränkte, aber leichter zu bedienende Ansicht. Im Folgenden werden die wichtigsten Editoren vorgestellt.

### 3.5.1 Protégé

Der Ontologie-Editor Protégé[18] wurde an der Standford University am Center for Biomedical Information Research zur Erstellung von OWL-Ontologien im medizinischen Umfeld entwickelt und stellt heute den de-facto Standard dar. Der Editor bietet einen großen Funktionsumfang und ist in unterschiedlichen Entwicklungslinien verfügbar, die jeweils den Fokus auf andere OWL-Teilaspekte legen. Der Editor bietet über ein Plugin-System die

[18] http://protege.stanford.edu/ [Letzter Zugriff: 26.11.2012]

Möglichkeit den Funktionsumfang dynamisch zu erweitern. Zusätzlich können in einigen Versionen die sichtbaren Relationen ein- und ausgeblendet werden, wodurch die Übersichtlichkeit bei der Bearbeitung erhöht werden kann. Der Editor trennt die Darstellung von Konzepten, Relationen und Instanzen in drei eigenständige Tabs, wobei jedes mit einer Baumdarstellung der jeweiligen Hierarchie ausgestattet ist (siehe Abbildung 3.19). Bei der Bearbeitung von Relationen kann Protégé automatisch neue Instanzen (die als Ziel der Relation fungieren) erstellen und die Ansicht direkt zu deren Bearbeitung umschalten (alternativ in einem eigenen Fenster öffnen).

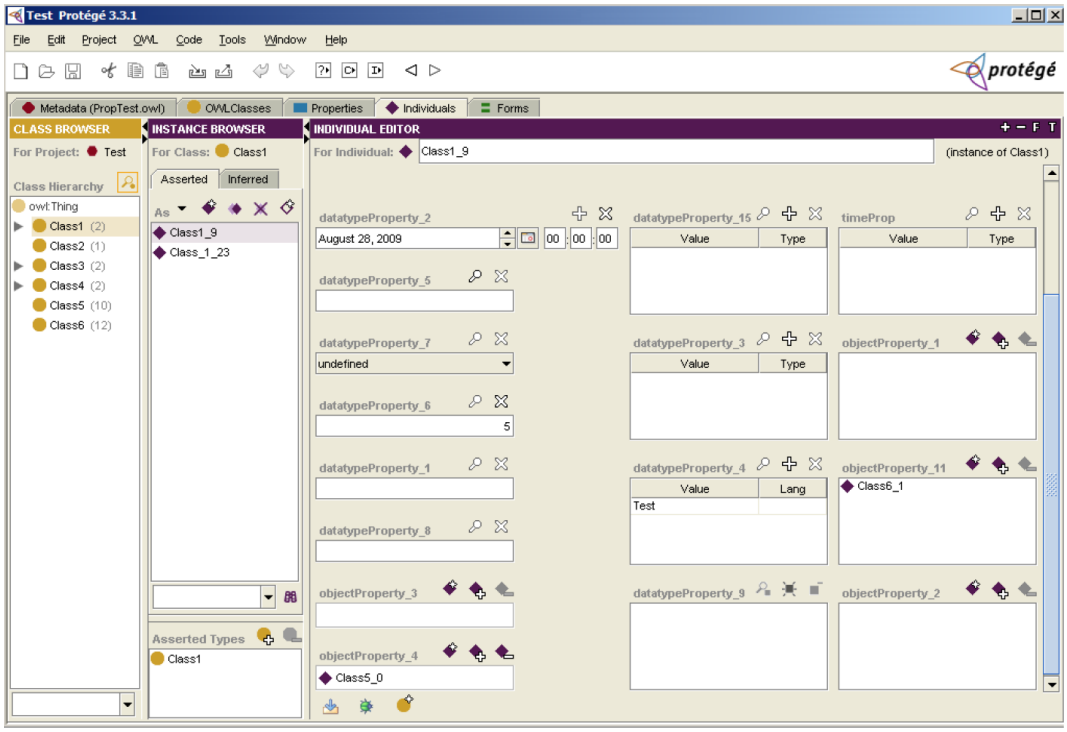



Abbildung 3.19: Graphische Oberfläche des Ontologie-Editors: *Protégé*

### 3.5.2 Swoop

Der Editor Swoop[19] wurde am Maryland Information and Network Dynamics Lab der Universität von Maryland im Rahmen des Web Agents Project entwickelt. Eine Weiterentwicklung findet seit dem Jahr 2007 nicht mehr statt, allerdings ist der Quellcode frei verfügbar. Der Funktionsumfang gleicht der gemeinsamen Schnittmenge aller Protégé-Versionen. Zur Darstellung und Bearbeitung von Ontologien generiert der Editor Webseiten, die er in einem eigenen Browser-Fenster anzeigt (siehe Abbildung 3.20). Trotz dieses flexiblen

[19] `https://code.google.com/p/swoop/` [Letzter Zugriff: 26.11.2012]

Ansatzes hat der Benutzer keine Möglichkeit die Oberfläche anzupassen. Im Unterschied zu Protégé werden Konzepte, Relationen und Instanzen nicht getrennt dargestellt und in einer gemeinsamen Baumhierarchie zusammengefasst. Zusätzlich ist es nicht möglich im gleichen Vorgang eine Relation zu bearbeiten und dabei die notwendige Relation zu erstellen. Die Instanz muss in einem eigenen Schritt vorab angelegt werden.

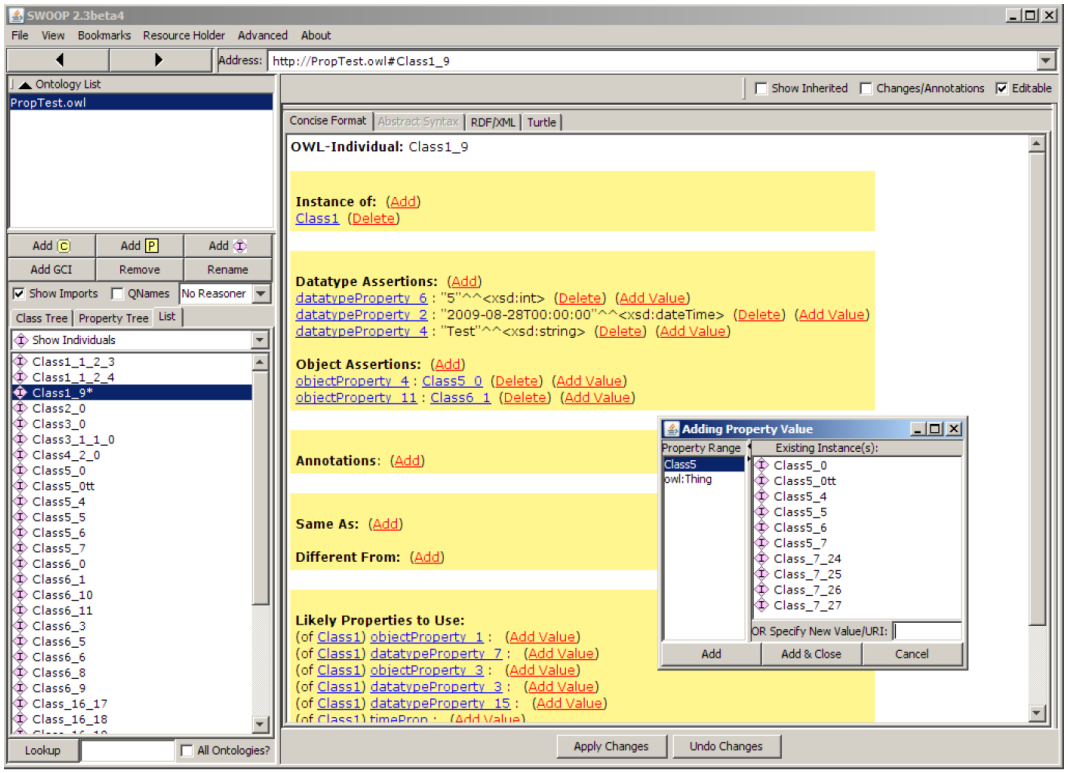



Abbildung 3.20: Graphische Oberfläche des Ontologie-Editors: *Swoop*

### 3.5.3 Neon-Toolkit

Das Neon-Toolkit[20] wurde im 6. EU-Rahmenprogramm (Sixth Framework Programme) für Information Society Technologies entwickelt und basiert auf der Eclipse[21]-Plattform. Es kann ähnlich wie Protégé mit diversen Plugins im Funktionsumfang stark erweitert werden. Die Eingabe erfolgt ebenfalls getrennt nach Konzepten und Instanzen, bietet aber keine Möglichkeit der Anpassung oder Personalisierung (siehe Abbildung 3.21). Bei der Erstellung und Verlinkung von Instanzen kann der Benutzer die Werte aus einer Auswahlliste festlegen, welche allerdings erst angezeigt wird, sobald einige Buchstaben eingegeben wurden

[20] `http://neon-toolkit.org/` [Letzter Zugriff: 26.11.2012]
[21] `http://www.eclipse.org/` [Letzter Zugriff: 26.11.2012]

(automatische Vervollständigung). Daher muss der Benutzer bei der Bearbeitung bereits wissen wie der exakte Name der Relation lautet.

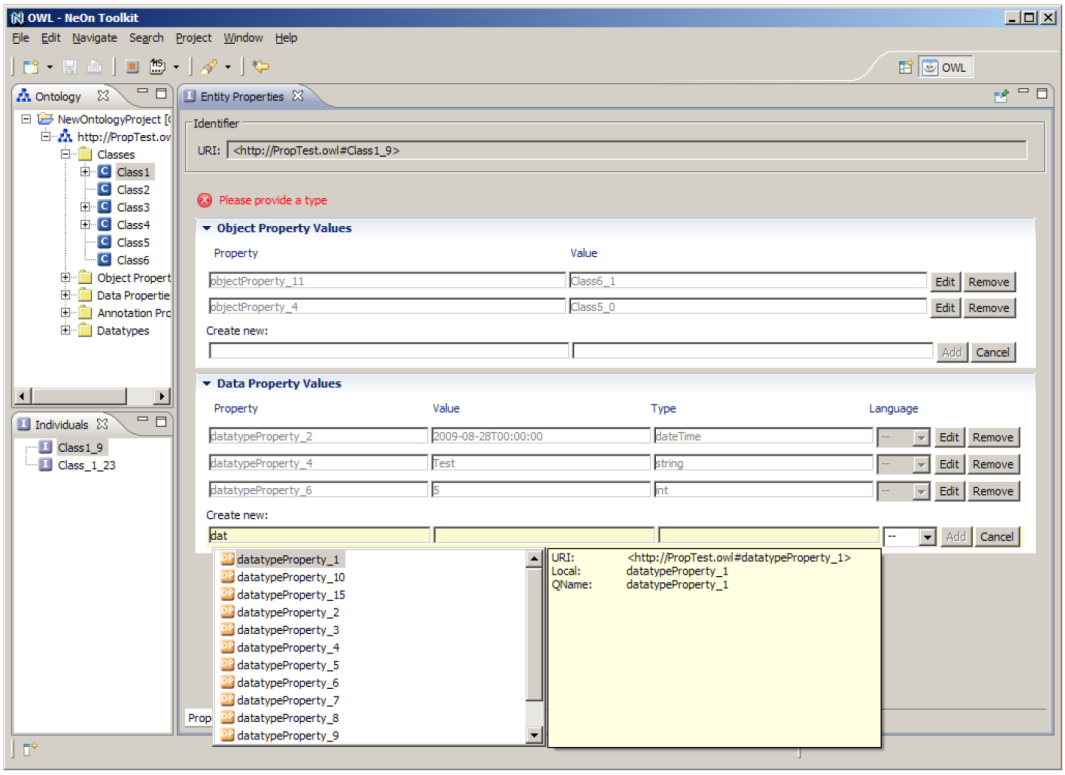



Abbildung 3.21: Graphische Oberfläche des Ontologie-Editors: *Neon-Toolkit*

## 3.6 Datenspeicherung und -bereitstellung

Aufgrund der Kernmetapher des digitalen Gedächtnisses für Objekte stellt die Speicherung von Informationen, die sich in einem solchen Gedächtnis befinden, ein zentraler Aspekt dieser Arbeit dar. Im Folgenden werden daher die aktuell gebräuchlichsten Verfahren und Ansätze beschrieben, um Daten mit Hilfe von digitalen Medien ablegen zu können.

### 3.6.1 Digitale Informationsdienste

Im Zeitalter der digitalen Information finden sich diverse Systeme die vormals als physisches Objekt vorliegende Daten mit der Hilfe von digitalen Diensten elektronisch verfügbar machen. Eine bekannte Anwendung ist zum Beispiel die digitale Bibliothek. Grundsätzlich

haben alle Systeme gemeinsam, dass Informationen mit einer eindeutigen ID versehen werden, anschließend in einem System abgelegt werden und somit jederzeit wieder abgerufen werden können. [KW06] skizzieren zu dieser Domäne einen Architekturentwurf, der im Folgenden näher beschrieben wird.

Grundsätzlich stellt sich die Funktionsweise des Systems wie folgt dar: ein *Ersteller*, zum Beispiel ein Benutzer, stellt dem System ein *digitales Objekt* zur Verfügung. Dieses Objekt setzt sich dabei aus den eigentlichen Informationen (den *Daten*) und einer Menge an *Metadaten* zusammen. Bestandteil der Metadaten ist ein eindeutiger *Identifikator* für dieses Objekt. Der Identifikator wird in der Regel entweder vom System selbst oder von einem externen *Identifikationsgenerator* erstellt. Schließlich werden die Daten selbst in einer sogenannten *Datenablage* gespeichert. Diese Ablage wird durch ein Speichersystem realisiert, welches über das Netzwerk angesprochen werden kann. Das Speichersystem stellt dazu Methoden zur Objektablage und zum Objektzugriff bereit.

Ein universeller Zugriff auf das digitale Objekt erfolgen mit Hilfe eines *Zugangsprotokolls für die Datenablage* (siehe auch [DK10]). Dieses Protokoll bildet dabei die funktionalen Methoden der Datenablage ab: Ablage und Zugriff. Das Ablage-Kommando unterstützt dabei unterschiedliche Argumente. Zum Beispiel kann ein vollständiger Datensatz, bestehend aus Daten, Metadaten und Identifikator abgelegt werden. Zusätzlich ist es möglich den Identifikator, automatisch vom System generieren zu lassen. Je nach Typ der Daten ist es sogar möglich die Metadaten automatisch ableiten zu lassen. Der Zugriff bietet eine ähnliche Flexibilität. So ist ein dedizierter Zugriff auf Einzelbestandteile eines Datensatzes möglich, z.B. nur die Metadaten oder nur die Nutzdaten selbst. Des Weiteren unterstützt das Protokoll Erweiterungen bezüglich Verschlüsselung und Datenkompression, sowie Zugriff auf Teilbereiche der Daten. Schließlich werden auch vom Datentyp abhängige Dienste angeboten, die mit Hilfe des Protokolls gestartet werden können.

### 3.6.2 Relationale Datenbanken

Der einfachste und anschaulichste Weg Daten und Informationen zu sammeln ist eine Tabelle, die sich als Matrix mit $n$ Zeilen und $m$ Spalten zusammensetzt. Die dabei typischen Merkmale sind der eindeutige Tabellenname, ein Namen für jede einzelne Spalte, der die dort hinterlegten Attribute beschreibt, sowie die darunter befindlichen Daten, jeweils ein Datensatz pro Zeile [Mei07]. Einzelne Einträge in solchen Tabellen können mehrfach in verschiedenen Zeilen auftreten. Um Einträge in Tabellen eindeutig unterscheiden zu können werden sogenannte (Identifikations-)Schlüssel verwendet. Ein solcher Schlüssel ist ein Merkmal oder eine minimale Merkmalkombination, um Einträge (=Zeilen) einer Tabelle eindeutig zu identifizieren. Wird die Tabelle als Relationenmodell aufgefasst, wird der Inhalt der Tabelle als Menge ungeordneter Tupel angesehen. Daher darf jede Tupelkombination in der Tabelle nur einmal auftreten. Man spricht dann von relationalen Datenbanken. Mit

Hilfe der Schlüssel ist es darüber hinaus möglich, Informationen aus mehreren unterschiedlichen Tabellen, die in einem Zusammenhang stehen, zu einem gemeinsamen Datensatz zu verbinden (siehe Abbildung 3.22). Dadurch und mit Hilfe von Filteroperationen können sogenannte Sichten (Views) generiert werden, die eine für den jeweiligen Anwendungsfall zugeschnittenen Datenauswahl präsentieren.

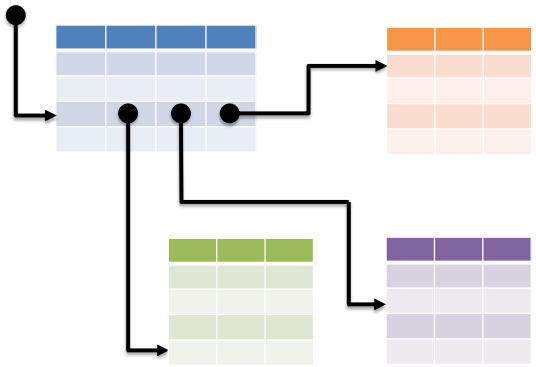


Abbildung 3.22: Kombination mehrerer Tabellen in einer relationalen Datenbank

Solche Datenbanken zeichnen sich dadurch aus, dass sich eine sehr streng schematisierte Struktur besitzen. Dadurch ist die Ablage von Daten jederzeit eindeutig möglich. Die Art der Daten und der Datenablage müssen dabei in der Regel während der Entwicklung der Datenbank bekannt sein. Beide Informationen dienen als Ausgangsbasis zum Entwurf der einzelnen Tabellenschemata, welche eine umfassende vorausgelagerte Analyse der abzulegenden Daten bedingt. Dadurch ist eine Anpassung im Nachhinein nur noch schwer möglich. Die Interaktion mit der Datenbank erfolgt in der Regel über eine Anfragesprache. Damit und mit Hilfe der Relationenalgebra werden Anfragen formuliert, die Daten innerhalb der Datenbank hinzufügen, ändern, löschen oder ausgeben. Um bei der Abfrage komplexe Zusammenhänge in einfache Rückgabewerte zusammenführen zu können, werden mit Hilfe von Tabellenschlüsseln und Mengenoperationen Teilbereiche mehrerer Tabellen zu einer gemeinsamen Darstellung verknüpft. Zur Formulierung der Anfragen wird heutzutage von praktisch allen Systemen die Sprache SQL (Structured Query Language) genutzt [ISO11, ISO07] (siehe Quellcode 3.5).

**Quellcode 3.5: SQL**

```
1 SELECT <Merkmale>
2 FROM <Tabellen>
3 WHERE <Selektionsprädikat>
```

Eine Weiterentwicklung der letzten Jahre erlaubt es nun auch direkt XML-Dateien in Datenbanken abzuspeichern [PCS+04, HS06]. Man unterscheidet dabei zwischen Systemen, die intern eine eigene Tabellen-orientierte Struktur verwenden und die Daten nur mit

Hilfe von XML importieren und exportieren können und nativen Systemen, die die Daten direkt in XML-Dokumenten abspeichern. Der Vorteil letzterer Systeme liegt darin, dass eine Konvertierung in das XML-Format entfallen kann, wenn Daten via XML ausgetauscht werden sollen, was bei Webanwendung häufig vorkommt. Des Weiteren können XML-Datenbanken aufgrund der hierarchischen Struktur von XML-Dokumenten leicht serialisiert werden. Nachteilig wirkt sich der Fakt aus, dass die Performanz von XML-Datenbanken in der Regel deutlich geringer ist. Der Zugriff auf Daten kann, abgesehen von der bereits erwähnten Methode SQL, auch mit XML-konformen Anfragesprachen wie XPath [CD99] oder XQuery [EFG07] erfolgen.

### 3.6.3 No-SQL Datenbank: CouchDB

Der heute übliche Standard zur Speicherung von Daten bilden Datenbanken. Der überwiegende Teil davon ist als relationale Datenbank ausgeführt, bei der Informationseinheiten in einzelnen Tabellen definiert sind. Durch die Verkettung von Einträgen unterschiedlicher Tabellen über Indizes können auch komplexere Daten erfasst werden. Diese Datenbanken sind stark strukturiert und definieren sich über ihre Schemata. Diese sind zum einen aufwändig zu entwerfen und im Nachhinein nur noch schwer an neue Gegebenheiten anzupassen. Dies steht im Widerspruch zu den sehr unstrukturierten, sich ständig ändernden und eher dokumentenorientierten Daten aus dem Web und dem Internet der Dinge. Aus diesen Gründen einstehen zurzeit eine neue Generation von Datenbanken, die sich vom starren relationalen Ansatz lösen. Ein Vertreter solcher NoSQL-Datenbanken (Not Only SQL) ist *CouchDB* (Cluster Of Unreliable Commodity Hardware DataBase), welches im Folgenden stellvertretend für alle anderen verwandten Systeme kurz beschrieben wird [ALS10].

CouchDB ist eine nicht-relationale NoSQL Datenbank ohne Schemata. Die Kommunikation läuft vollständig über Webschnittstellen ab. Dadurch ergeben sich sehr gute Skalierungseigenschaften und ermöglichen verteilte Systeme. Inhaltlich wird ein Dokumentenorientiertes Datenmodell verwendet. Das Dokument ist eine Abstraktion von Schlüssel-Wert-Paaren und stellt eine in sich geschlossene und sinnvolle Informationseinheit dar. In der Regel findet eine Abbildung von physischen Objekten auf Dokumente statt, zum Beispiel in Form von Visitenkarten oder Rechnungen. Diese Dokumente werden intern als Json-Objekte dargestellt. Dabei besitzt jedes Dokument eine eindeutige ID (UUID) mit dem Namen `_id` und eine Revisionsnummer `_rev` (siehe Quellcodebeispiel 3.6). Jedes Dokument kann über die Json-Struktur hinaus mit einer gewöhnlichen Datei versorgt werden.

**Quellcode 3.6: CouchDB Beispieldokument**

```
1 {
2   _id: "69...13",
3   _rev: "8a...06",
4   Name: "Haupert",
```

```
5   Vorname: "Jens",
6   E-Mail:
7   [
8     "jens.haupert@dfki.de",
9     "test@dfki.de"
10  ],
11  Telefon Büro: "12345",
12  Stadt: "Saarbrücken"
13 }
```

Dokumente können nur vollständig aktualisiert werden, das heißt die Anwendung lädt das Dokument herunter, ändert es ab und lädt es anschließend wieder in die CouchDB. Diese aktualisiert dann das Dokument, behält aber den alten Stand bei. Somit sind gleichzeitige Lese- und Schreibzugriffe möglich. Eine vollständige Versionsverwaltung ist allerdings nicht möglich, da nicht sichergestellt werden kann, dass alle älteren Versionen verfügbar sind. Das System bietet eine integrierte *Replikationsfunktionalität*. Dadurch lassen sich die Daten auf verschiedene Instanzen der Datenbank sichern. Es werden allerdings immer nur die aktuellsten Versionen synchronisiert, was die Umsetzung einer vollständigen Versionsverwaltung verhindert. Dokumente können über ihre `_id` abgerufen werden. Dynamische Anfragen sind allerdings nicht möglich. Diese müssen explizit in Form von *Views* definiert werden, wobei diese entweder fest in der Datenbank gespeichert werden können oder dynamisch mit der Anfrage mitgeliefert werden. Diese Anfragen werden in JavaScript definiert. Mit Hilfe einer *Reduzierungsfunktion* ist es möglich die Anzahl der zurückgelieferten Ergebnisse einer View zu vermindern und somit zu filtern.

Da CouchDB selbst als Webserver fungiert, lassen sich darüber hinaus auch Anwendungen direkt ohne Middleware innerhalb des CouchDB-System ausführen. Sowohl der Code als auch die benötigten Daten werden als reguläre Dokumente betrachtet. Diese Anwendungen können direkt auf dem Datenbestand der DB arbeiten und zusätzlich Daten aus dem Netz nachladen. Diese Anwendungen werden als HTML und JavaScript-Code entwickelt und direkt im Browser ausgeführt.

### 3.6.4 Semantische Datenbank: Sesame

Die Sprachen RDF und RDFS sind im Web verbreitete Techniken zur Annotation von Informationen in Form von semantischen, maschinen-lesbaren Daten (siehe Kapitel 2.5.2). Solche Daten werden häufig als XML-Dateien persitiert. Um einen effizienteren Zugriff zu erhalten und auch in großen RDF-Graphen schnell benötigte Daten zu finden, wurde das System *Sesame* im Rahmen des europäischen IST[22]-Projekts On-To-Knowledge [SS02] entwickelt. Sesame erlaubt eine persitente Speicherung von RDF Daten und Schemainformationen und

[22] Information Society Technologies, `http://cordis.europa.eu/ist/` [Letzter Zugriff: 26.11.2012]

erlaubt den Zugriff auf solche Daten über Export- und Anfragemodule. Zusätzlich sind Konzepte wie Zwischenspeicherung und gleichzeitige Zugriffe berücksichtigt [BKvH02].

Der Zugriff auf RDF-Daten kann auf drei unterschiedlichen Ebenen erfolgen:

- **Syntaktische Ebene**: Durch die Darstellung von RDF-Modellen in Form von XML-Dateien, können auch XML-basierte Anfragesysteme wie zum Beispiel XQuery [EFG07] genutzt werden. Dadurch lassen sich allerdings die erweiterten Datenstrukturen und Modelle von RDF nicht adressieren. Erschwerend kommt hinzu, dass das gleiche RDF-Modell in unterschiedlichen Ausprägungen als XML repräsentiert werden kann.
- **Strukturelle Ebene**: Auf dieser Ebene besteht ein RDF-Modell aus einer Menge an Subjekt-Prädikat-Objekt-Tripeln. Für solche Tripel existieren bereits Anfragesprachen, zum Beispiel Squish [Mil01] und SPARQL [ACD+12], mit deren Hilfe sich das RDF-Datenmodell direkt ansprechen lässt. Allerdings lassen sich ohne externen Reasoner keine impliziten Aussagen mit Hilfe dieser Sprachen abfragen.
- **Semantische Ebene**: Die semantische Ebene beinhaltet das Wissen, welches explizit und implizit durch RDF- und RDFS-Daten definiert ist. Dieses Bild kann dabei entweder durch die Berechnung des Abschluss (Closure) für den gesamten Graph oder durch Inferenz neuer Aussagen im Zuge einer Anfrage erzeugt werden. Die Anfragesprache RQL [KAC+02] trägt dieser Tatsache Rechnung. Ihre Syntax basiert auf OQL [CB00] und besteht im Wesentlichen aus einer Menge von Kernanfragen, Basisfiltern und der Möglichkeit zur funktionalen Komposition und Iteration.

Laut [BKvH02] ist eine Anfrage nur auf semantischer Ebene sinnvoll, welche von Sesame unterstützt werden. Abbildung 3.23 zeigt die Bestandteile und Module der Sesame-Architektur. Die Speicherung der eigentlichen Daten erfolgt in einer Datenbank (DB). Da Sesame keine Festlegung auf ein bestimmtes DB-System macht, wurde der eigentliche DB-Zugriff abstrahiert und wird mit Hilfe des *RDF Speicher- und Inferenzmoduls* (SAIL) gekapselt. Die funktionalen Module von Sesame bauen auf SAIL auf. Das RQL-Querymodul verwendet RQL als interne Anfragesprache. Nach dem Parsen einer Anfrage und der Generierung eines Anfragebaums wird dieser zusätzlich optimiert. Dabei wird der Bau in eine effizientere Darstellung mit identischer Bedeutung überführt. Diese Anfrage wird anschließend über die SAIL-API aufgelöst. Die Aufgabe des Adminmoduls ist es neue Daten in das System aufzunehmen oder das System vollständig zu löschen. Diese neuen Daten werden in der Regel in Form von RDF(S)-Dateien übergeben und mit Hilfe von SAIL gespeichert. Das Exportmodul dient schließlich dazu das gesamte vorliegende Wissen in eine gesamte RDF-Datei zu exportieren. Der Anfragerouter leitet die über HTTP oder SOAP angekommenen Anfragen an die jeweils zuständigen Module weiter (siehe Abbildung 3.23).

Basierend auf diesem und ähnlichen RDF-Speichersystemen wurden in den letzten Jahren verfeinerte Frameworks konzipiert, deren Fokus darauf lag, Anfragen von externen

Clients so effizient wie möglich zu gestalten. [DVDNGP11] beschreibt eine der letzten Entwicklungen in diesem Bereich.

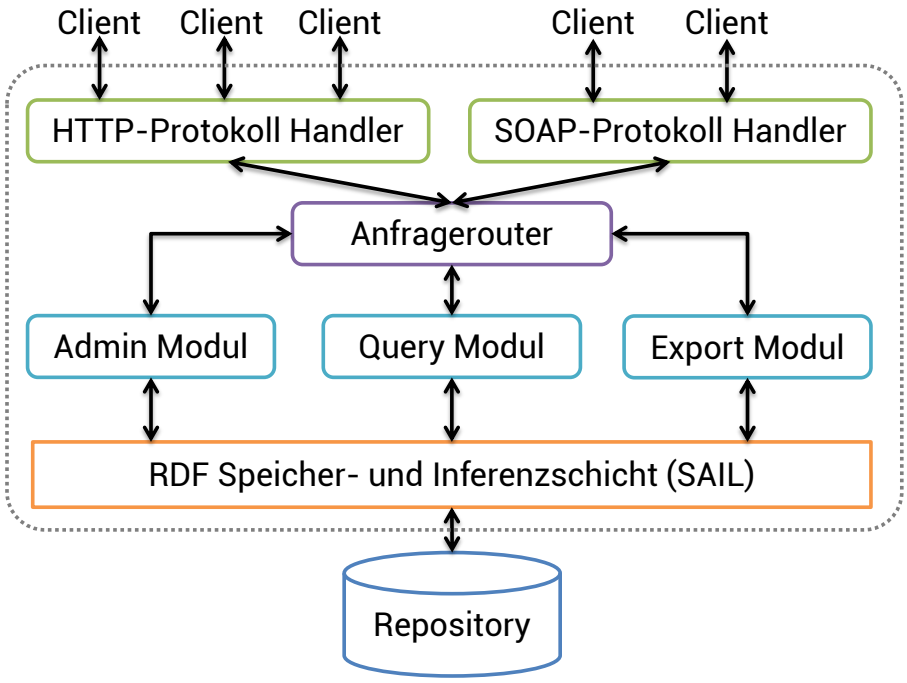



Abbildung 3.23: Architektur von *Sesame*

## 3.7 Datenvisualisierung

Alle bisher beschriebenen Ansätze sammeln Daten über Objekte oder Menschen. Diese möchte man zwar primär maschinell weiter verarbeiten, aber zusätzlich auch einem Benutzer, sei es eine technische Fachkraft oder ein Endverbraucher, zugänglich machen. Zu diesem Zweck ist ein Visualisierungssystem notwendig, welches an die besonderen Anforderungen und Möglichkeiten dieser Datenbasen angepasst ist. Im Folgenden sind nun einige ausgewählte Arbeiten beschrieben, die diese Aufgabe übernehmen könnten.

### 3.7.1 Visualisierung von intelligenten Umgebungen basierend auf Agenten

Die Johannes Kepler Universität Linz präsentiert ein Konzept zur Visualisierung von intelligenten Umgebungen basierend auf Agenten [GIK04]. Dieser Ansatz folgte dem „Model-View-Controller“-Prinzip [Ber04] und trennt innerhalb der Agenten die Datenvisualisierung

von der eigentlichen Prozesslogik. Je nach Typ der geforderten Interaktion bietet der Agent eher explizite (wie z.B. klassische, Desktop-ähnliche Interaktion mit Maus und Tastatur) oder implizite (z.B. über Sensoren oder Intentionserkennung) Interaktionsformen an (siehe Abbildung 3.24).

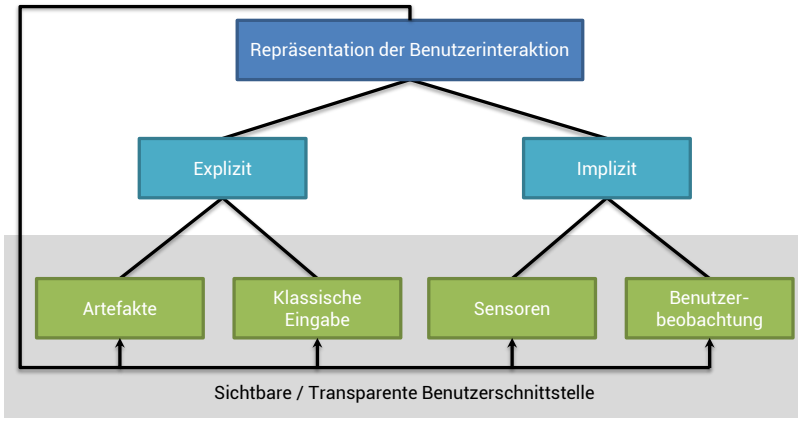



Abbildung 3.24: Durch Agenten unterstützte Visualisierungen

Zusätzlich unterscheidet der Ansatz verschiedene Kardinalitäten in den Beziehungen zwischen einem Agent und einer Benutzerschnittstelle (siehe Tabelle 3.2). Bei einer „eins zu eins“-Beziehung verwendet jeder Agent genau eine Benutzerschnittstelle über die er mit dem Benutzer kommuniziert, z.B. über einen Bildschirm. Bei einer „eins zu viele“-Relation schreibt eine sogenannte multimodale Schnittstelle, bei der der Agent über verschiedene Kanäle gleichzeitig kommuniziert (z.B. visuell, per Sprache und per Geste). Bei einer „viele zu eins“-Beziehung interagiert der Benutzer mit vielen Agenten gleichzeitig über die gleiche Schnittstelle, was den Vorteil bietet, dass der Benutzer nur mit dieser Interaktionsform vertraut sein muss. Schließlich kombiniert die „viele zu viele“-Relation die Möglichkeiten der multimodalen Interaktion mit einem auf mehreren Agenten verteiltem System.

| | Benutzerschnittstelle | |
|---|---|---|
| Agenten | eins - eins | eins - viele |
| | viele - eins | viele - viele |

Tabelle 3.2: Beziehungen zwischen Benutzer und Agenten

Basierend auf diesen Konzepten wurde ein modulares Java-basiertes Framework erstellt, welches möglichst wiederverwendbar, skalierbar und zuverlässig sein soll ([FSJ99]). Diese Agenten kümmern sich dabei um die Aufgaben der Datenbeschaffung, Datenverarbeitung und der Datenvisualisierung. Ihre Architektur besteht aus drei horizontalen Ebenen: Dateneingabe, funktionale Komponenten und Visualisierung. Die funktionalen Komponenten werden dabei durch eine Kontext- und Kommunikationsschicht ergänzt.

### 3.7.2 Adaptive Visualization over the Internet

Einen ebenfalls Agenten-basierten Ansatz präsentiert die Technische Universität Wien mit ihrem *AVAM* (Adaptive Visualization over the Internet) System [TG00], welches eine 3D-Visualisierung über das Internet verfügbar machen möchte. Die Herausforderungen für einen solchen Ansatz liegen in der beschränkten Kapazität der Netzwerkübertragung, der hohen Komplexität von verteilten Anwendungen und eine oft nicht an verteilte Szenarien angepasste Client/Server-basierte Visualisierung. Zu diesem Zweck werden hier Agenten eingesetzt, um die verschiedenen Module eines Systems umzusetzen. Dies erlaubt eine hohe Unabhängigkeit und eine lose Kopplung verschiedener Module, verbesserte Unterstützung von verteilten Systemen und eine hohe Wiederverwendbarkeit der Module bei der Entwicklung verschiedener Anwendungen.

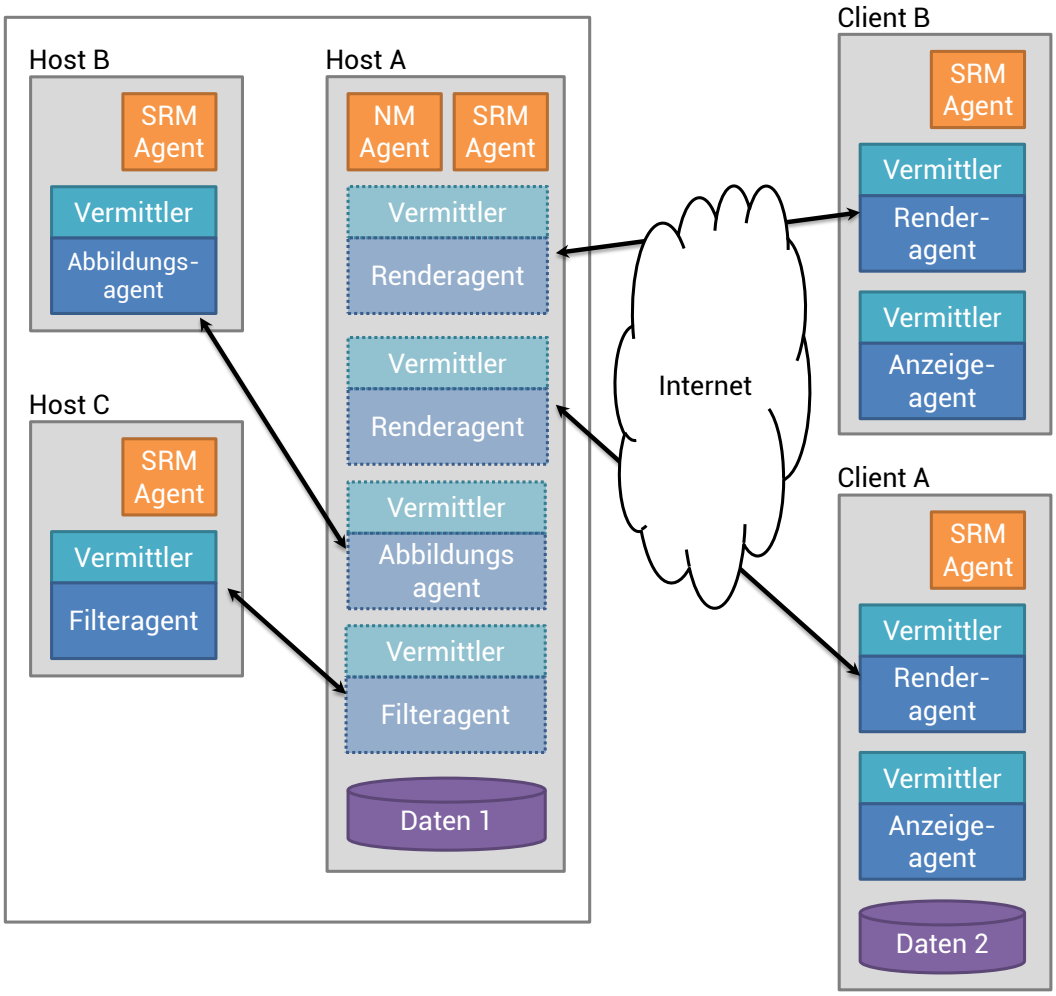



Abbildung 3.25: AVAM-Architektur mit verteilten Komponenten

Das eigentliche System besteht logisch aus drei Komponenten: ein *Sensor* (1), welcher die verfügbare Netzwerk- und Ressourcenbandbreite misst, Benutzereingabe aufnimmt und Informationen über die Datenquelle bereitstellt. Die nächste Komponente ist der *Vermittler* (arbitrator) (2), welcher die Entscheidung trifft, ob ein Visualisierungschritt lokal oder auf einem anderen Host ausgeführt werden soll. Die notwendigen Daten für diese Entscheidung liefert der Sensor. Die letzte Komponente ist der Visualisierungsagent (3), welcher die eigentliche Visualisierung beherbergt und Entscheidungen vom Vermittler ausführt. Abbildung 3.25 zeigt eine beispielhafte Anwendung basierend auf dem AVAM-System. Das Beispielsystem besteht aus einem Host-Rechner und zwei Client-Rechnern, die mit dem Host über das Internet verbunden sind. Der Host selbst besteht aus drei separaten Instanzen, wobei Host B und C Verarbeitungsfunktionen (Datenabbildung und -filterung) von Host A übernehmen. Der eigentliche Rendervorgang wurde an die beiden Clients ausgelagert. Diese Verteilung wird kontinuierlich über den Netzwerkmonitor (NM) und den Systemressourcenmonitor (SRM) überwacht und bei Bedarf angepasst.

### 3.7.3 Multi-Agenten Ansatz

Der dritte Agenten-basierte Ansatz wurde vom Deutschen Forschungszentrum für Künstliche Intelligenz (DFKI) und der Universität Kaiserslautern entwickelt [HBE+00]. Viele der heute verwendeten Systeme dienen nur einem speziellen Zweck und bieten daher nur eine sehr geringe Flexibilität in Bezug auf ihren Visualisierungsprozess; sie bieten daher entweder eine hohe Renderinggeschwindigkeit mit begrenzter Interaktion oder eine echtzeitfähige Visualisierung mit eingeschränkter Renderingqulität. Selbst wenn Systeme in der Lage sind, beliebige, aber feste Positionen zwischen diesen beiden Extremen einzunehmen, ist doch der Benutzer in der Pflicht die notwendige Einstellung manuell vorzunehmen. Das vom DFKI entwickelte System verwendet Agenten, um diese Einstellungen automatisch durchführen zu können. Das gesamte System ist dabei in viele verschiedene Module unterteilt, die sich in drei Ebenen eingliedern lassen (siehe Abbildung 3.26).

Die Kernebene beinhaltete zentrale Funktionen zur Visualisierung, zum Beispiel die Verwaltung der Szene, Geometrie, Beleuchtung und Rendering. Um die gesamte Verarbeitung von der eigentlichen Plattform zu abstrahieren, kommt eine Hardwareabstraktionsschicht zum Einsatz (HAL), welche die generische Visualisierung mit speziellen Treibern auf die jeweilige Plattform abbildet. Zu Kontrolle und Anpassung der Visualisierung, welche Notwendig ist, da sich die Randbedingungen häufig ändern, werden zwei unterschiedliche Agenten eingesetzt, um die zu verarbeitende Komplexität in kleinere Aufgaben aufzuteilen. Die sogenannten „beratenden“ Agenten (deliberative agents) können komplexe Probleme verarbeiten, in der Regel jedoch nicht unter Echtzeitbedingungen, aufgrund einer Vielzahl von notwendigen Berechnungen. „Reaktive“ Agenten verarbeiten im Gegensatz dazu nur einfache Aufgaben, dies aber mit hoher Geschwindigkeit, so dass diese problemlos in eine Verarbeitungskette eingefügt werden können.

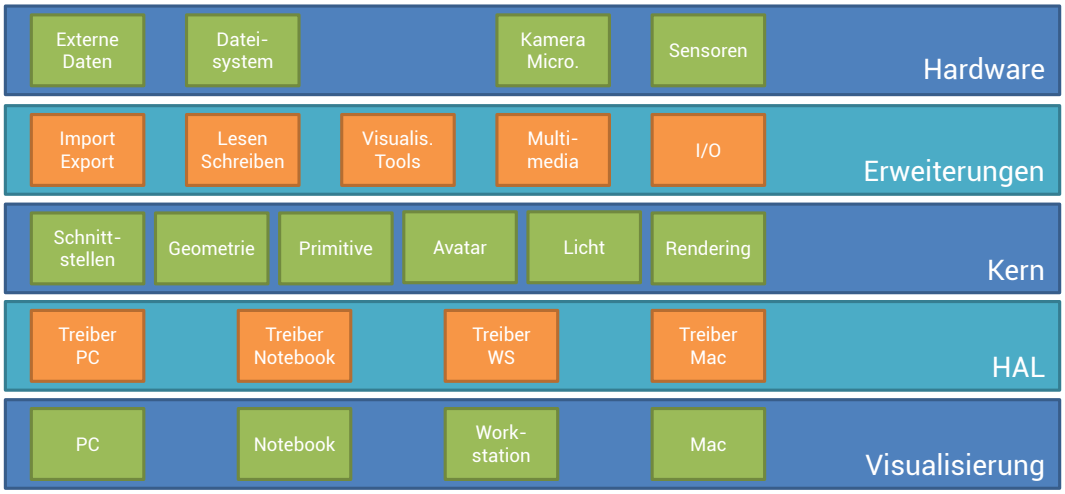



Abbildung 3.26: Multi-Agenten-basiertes Schichtenmodell

Die gesamte Verarbeitung erfolgt in einer sogenannten Pipeline, bei der einzelne Module hintereinander geschaltet sind und ihre Daten immer an das jeweils nächste Glied der Kette weitergeben (siehe Abbildung 3.27).

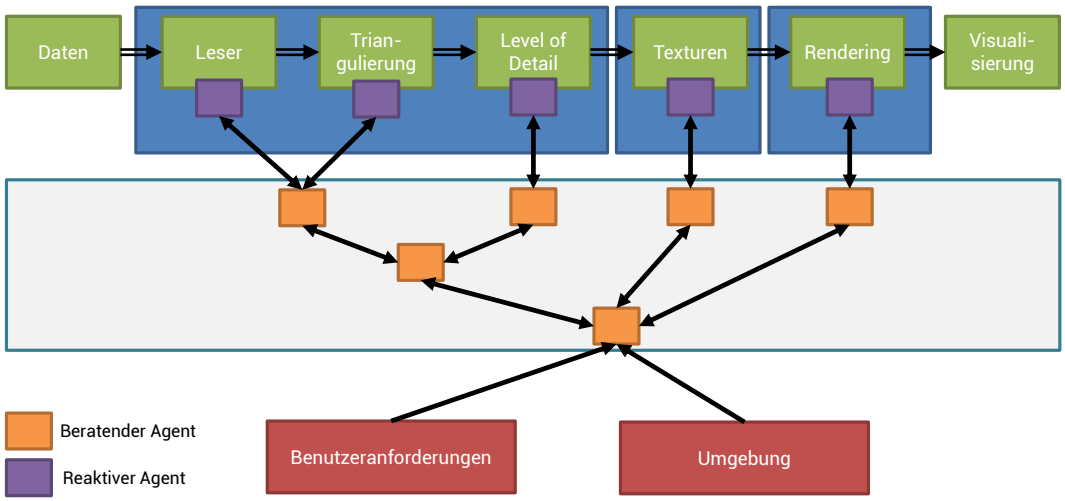



Abbildung 3.27: Multi-Agenten Visualisierungs-Pipeline

Hier wird nun jedem Modul der Kette ein reaktiver Agent zur Verfügung gestellt, welcher direkt Einfluss auf die Verarbeitung nimmt (z.B. den Grad der Detaillierung anpasst). Dieser Agent erhält seine Daten von beratenden Agenten, die im Hintergrund Entscheidungen für den reaktiven Agenten fällen, z.B. auf Daten aus der Umgebung oder durch Anforderungen des Benutzers.

Das System organisiert alle Module in einem zentralen Repository. Jede Komponente gibt dabei über ein öffentliches Interface ihre Funktionalität nach Außen bekannt und kapselt die eigentliche Implementierung nach Innen. Dieses Verfahren erlaubt die Entwicklung von unabhängigen und wiederverwendbaren Komponenten. Implementiert wurde das System in Java, wobei die Komponenten auf der Java Beans Technologie basieren.

### 3.7.4 Multiplatform Universal Visualization Architecture

Das Projekt MUVA (Multiplatform Universal Visualization Architecture) der Universität von Zagreb in Zusammenarbeit mit Ericsson beschreibt eine flexible Architektur für unterschiedliche Client-Plattformen [SKKM+05]. Dabei ist es das erklärte Ziel multimodale Inhalte (zum Beispiel Text, Hypertext, Bilder, 3D Grafiken, Audio und Video) derart aufzubereiten, dass diese auf unterschiedlichen Plattformen dargestellt werden können und die Fähigkeiten der jeweiligen Geräte bestmöglich ausnutzen. Dabei werden drei unterschiedliche Plattformen unterschieden:

- **Vollständige Clients (full clients)**: Plattformen mit genügend Rechenleistung um Rohdaten, die von einem Server geliefert werden, lokal visualisieren zu können. Zusätzlich stehen verschiedenste Interaktionsformen zur Verfügung (z.B. Maus/Tastatur, Gesten, Sprache). Als Softwarebasis können vollwertige Webbrowser mit Java und VRML-Umgebung genutzt werden.
- **Midi Clients**: Plattformen mit eingeschränkter Leistung, die nur einen Teil der Aufgaben lokal verarbeiten können (z.B. PDAs, Smartphones oder Tablets).
- **Mini Clients**: Plattformen, welche über nicht genügend Leistung verfügen, um komplexe Daten lokal verarbeiten zu können und nur einfache Textdaten darstellen können (z.B. konventionelle Mobiltelefone).

Die MUVA-Architektur erlaubt es nun Anwendung zu entwickeln, die auf allen drei Plattformen genutzt werden können. Zu diesem Zweck werden die Datenverarbeitung und die eigentliche Visualisierung getrennt und in einzelne Softwaremodule aufgeteilt. Je nach Leistungsfähigkeit des Endgeräts kann die Visualisierung direkt auf diesem durchgeführt werden oder muss extern auf einem Server erstellt werden.

Die Architektur verwendet dabei die in Abbildung 3.28 gezeigte Aufteilung in vier serverseitige logische Blöcke:

- **Visualisierungstools** sind zuständig für die Visualisierung einer bestimmten Information in einer bestimmten Art und Weise (z.B. eine Baumdarstellung oder ein Kuchendiagramm). Daher hat jedes Tool ein genau spezifizierten Dateneingang, genau definierte Befehle zur Steuerung und eine festgelegte Darstellungsform.

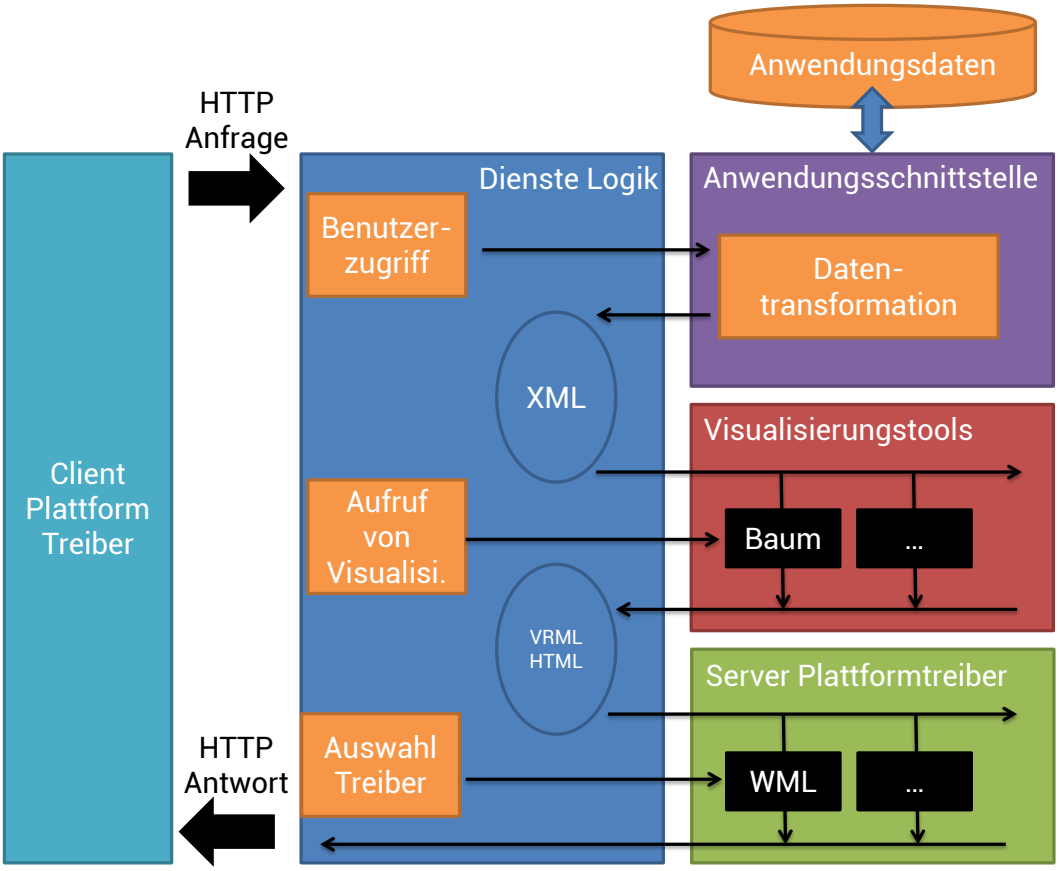



Abbildung 3.28: MUVA: Systemarchitektur

- **Server Plattformtreiber** rendern die benötigte Darstellung auf der jeweiligen Plattform. Dabei stehen unterschiedliche Implementierungen zur Verfügung (C++, HTML, Java, VRLM). Je nach Plattform wird ein Teil des Rendering auf dem Server durchgeführt oder vollständig auf den Client verlagert.

- **Anwendungsschnittstelle** kümmert sich um die Beschaffung der Daten über eine standardisierte Schnittstelle, welche für jede Anwendung angepasst werden muss.

- **Dienste Logik** ist zuständig für die Verwaltung und den Datenfluss innerhalb der bisher genannten Module. Zusätzliche Aufgaben sind eine Benutzerzugriffssteuerung und die Auswahl der passenden Plattformtreiber.

### 3.7.5 SOA-basierter Ansatz

Eine Dienste-orientierte Architektur zur Visualisierung von Daten wurde von der Universität Rostock entwickelt [TTS09]. Der anvisierte Anwendungsfall beschreibt die Visualisierung von Daten in intelligenten Umgebungen, die sich unter anderem dadurch auszeichnen, dass die verfügbaren Ein- und Ausgabegeräte sehr schnell wechseln können. Daher stellt sich die Frage, wie man die gerade verfügbaren Geräte bestmöglich für die geforderte Aufgabe nutzten kann. Da alle Geräte nur sehr lose gekoppelt sind, wurde der Ansatz einer Dienste-orientierten Anwendung gewählt, bei dem die einzelnen Dienste dynamisch miteinander verknüpft werden. Dabei wurde kein Agenten-basierter Ansatz, sondern ein Bausteinkonzept gewählt, da sich je nach Nutzung der intelligenten Umgebungen die Anforderungen verändern. Daher wird es als sinnvoller erachtet eine definierte Kontrollebene für jede Anwendung zu nutzen, um den Verwaltungsaufwand zu minimieren.

Zur Identifikation der benötigten Dienste wird das „data state reference model“ (DSRM) genutzt [Chi00], welches eine weit verbreitete Architektur zur Informationsvisualisierung repräsentiert und die Visualisierungspipeline als statisches Netzwerk von Transformationsoperationen auf Daten darstellt. Diese Idee wird nun angepasst, in dem die hartverdrahtete Pipeline aufgebrochen wird und einzelne Operationen durch Dienste dargestellt werden. Bei der Darstellung von Diensten wird zwischen Schnittstellen und Implementierungen unterschieden. Eine Schnittstelle definiert eine Menge von Ein- und Ausgabedaten mit den jeweiligen Kontrollparametern. Implementierungen stellen eine konkrete Umsetzung einer Diensteschnittstelle dar, das heißt zu einer Schnittstelle kann es mehrere Implementierungen geben.

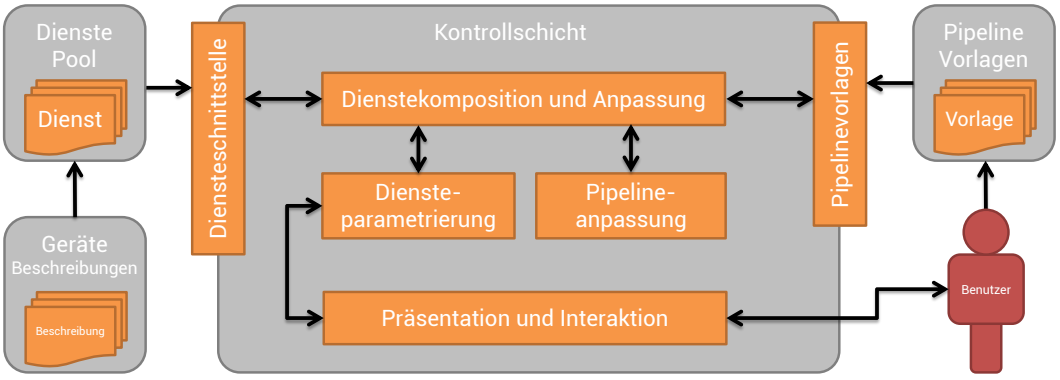



Abbildung 3.29: Dienste-orientierte Architektur zur Datenvisualisierung

Die Dienste selbst werden zur Laufzeit von einer Kontrollebene verwaltet, die folgende drei Aufgaben wahrnimmt: (1) Diensteregistrierung, (2) Dienstekomposition und Diensteparametrierung und (3) Diensteaufruf (siehe Abbildung 3.29). Die Registrierung der Dienste erfolgt dabei manuell; sobald der Dienst aktiv ist, meldet er sich periodisch bei der Kontrollschicht, wodurch es möglich wird nicht mehr aktive Dienste zu erkennen. Bei der

Dienstekomposition erstellt die Kontrollschicht aus mehreren Diensten eine Visualisierungspipeline, wobei vordefinierte Pipeline-Vorlagen genutzt werden. Eine solche Vorlage beschreibt eine abstrakte Visualisierung durch die Verkettung von Diensteschnittstellen. Zur Laufzeit werden dann diese Vorlagen genutzt, um die passenden konkreten Dienste auszuwählen und aufzurufen.

Die Implementierung der Dienste sieht fünf Diensteschnittstellenkategorien vor: (1) Datenimport (z.B. CSV/SQL Import), (2) Datananalyse (z.B. hierarchisches Clustering, Filter), (3) Abbildung (z.B. Farbkodierung, geographische Karten), (4) Rendern (z.B. Java2D oder PDF) und (5) zusätzliche Dienste. Die Dienstekomposition erfolgt mit Hilfe einer XML-Definition (siehe Quellcode 3.7) auf Basis der Java Reflection API, wobei die in der XML-Datei definierten Schnittstellen auf die jeweils passenden Dienste abgebildet werden und anschließend die Dienste verkettet und gestartet werden. Ist keine direkte Abbildung möglich, kann das System zusätzliche Konverter aktivieren, die vorhandene Daten in notwendige Daten konvertieren. Sind die Dienste aktiv, nutzt die Kontrollschicht zusätzliche Techniken, wie zum Beispiel Datenkompression zwischen den Diensten oder ein Cachingsystem zur Vermeidung von unnötigen Berechnungen, um das System möglichst effizient zu halten.

**Quellcode 3.7: SOA-Visualisierung: Beispiel Pipeline-Vorlage**

```
<template>
  <obj name="filtering" type="Data">
    <arg>count(diagnosis=influenza and year=1999)</arg>
  </obj>
  <obj name="colorParam" type="Color">
    <arg><readObjb objName="filtering" /></arg>
  </obj>
  <obj name="deviceParam" type="Device">
    <arg>SEARCH</arg>
  </obj>
  ...
  <obj name="mapping" type="Mapping">
    <arg><readObjb objName="filtering" /></arg>
  <arg><readObjb objName="colorParam" /></arg>
  <arg><readObjb objName="deviceParam" /></arg>
  </obj>
  <obj name="rendering" type="Renderer">
  <arg><readObjb objName="mapping" /></arg>
  <arg><readObjb objName="deviceParam" /></arg>
  </obj>
</template>
```

## 3.8 Bewertung und Schlussfolgerung

In diesem Kapitel werden nun die einzelnen verwandten Arbeiten miteinander verglichen und die jeweiligen Vor- und Nachteile dargelegt. Diese Diskussion verwendet die gleiche Gliederung wie die Darstellung der eigentlichen Arbeiten.

### 3.8.1 Architekturkonzepte

Die vorgestellte Auswahl an Arbeiten im Bereich der Architekturkonzepte umfasst ein breites Spektrum an Konzepten und Umsetzungsstrategien. Generell lassen sich alle Architekturen in zwei Klassen einteilen. Eine Teilmenge konzentriert sich auf die Ausrüstung von intelligenten Objekten und die Modellierung von Wissen direkt am Objekt (Prottoy, SmartProducts). Die andere Teilmenge fokussiert sich auf die Ablegung von Objektdaten abseits der physischen Objekte auf Servern, teilweise mit Techniken des Internets (SmaProN, ToTEM, UbisWorld). Einzig RAN und EPCglobal betrachten beide Fälle und kombiniert die Ablage von Daten sowohl am Produkt (mittels RFID-Tags) als auch auf Servern. Die reine Ablage von Daten am Objekt selbst schränkt somit die Nutzbarkeit dieser Lösung auf Produkte ein, deren Kostenrahmen derart gestaltet ist, dass eine Instrumentierung entweder keine signifikant höheren Produktionskosten generiert oder bei denen der Nutzwert so hoch ist, dass sich die Instrumentierung lohnt (im Sinne einer Veredelung). Für Alltagsgegenstände können solche Systeme daher nicht genutzt werden. Die Server- beziehungsweise Web-basierte Lösung stellt eine kostengünstige Alternative dar, die allerdings bei jedem Zugriff auf Objektdaten eine Netzwerkverbindung voraussetzt. Diese Verbindung ist allerdings in der Regel einfacher und günstiger zu realisieren als eine Instrumentierung solcher Objekte. Einen praxistauglichen Mischweg gehen das EPCglobal System und RAN, die beide Verfahren unterstützen und kombinieren. Die EPC-Datenstrukturen sind allerdings relativ stark durch Vokabularien reglementiert und fest auf Schlüssel-Wert-Paar-orientierte Ereignisdaten und sogenannte Masterdaten festgelegt. Es steht kein Ansatz bereit mit dessen Hilfe eine beliebige Information (zum Beispiel ein Bild oder PDF-Dokument) in das System aufgenommen werden kann und mit Hilfe von geeigneten Metadaten wiedergefunden werden kann. Unter dieser Schwäche leiden praktisch alle Lösungen, die entweder ein festen Vokabular verwenden, welches zwar oft semantisch Definiert ist, sich allerdings nicht um weitere Daten, die nicht in das jeweilige Schema passen, erweitern lässt (SmaProN, PML, UbisWorld). Andere Ansätze wie ToTEM verwenden zwar offenere Ansätze mit Freitext und Bilder/Videos, aber auch hier fehlen zum Beispiel passende Metadaten, die ein leichtes Wiederauffinden von abgelegten Daten ermöglichen. Interessante Ansätze gibt es darüber hinaus noch bei ECPglobal, mit dessen ONS-System anhand der ID eines Objekts automatisch dessen Speichersystem gefunden werden kann. Nachteilig ist allerdings die zusätzlich notwendige Infrastruktur, die zum Beispiel durch Verwendung von Speicheradressen als ID umgangen werden könnte. Diese Überlegung führt direkt zur Art der Kommunikation, die

bei allen Systemen über Web-basierte Schnittstellen erfolgt und somit mit einer Vielzahl an Endgeräten genutzt werden kann. Einige Systeme, in der Regel (aber nicht nur) solche die direkt auf dem Objekt ablaufen, bieten zusätzliche Möglichkeiten der Datenverarbeitung, wie zum Beispiel automatisches Schließen oder Kommunikation mit anderen Objekten (SmartProducts). Die Funktion der Erweiterung der Objektspeicherfunktionen durch weiterführende Aktivitätsmodule, die als eigenständige Code-Fragmente umgesetzt werden können, wird von keinem der Systeme unterstützt.

### 3.8.2 Ontologien und Metadaten

Ontologien und Metadaten stellen ein adäquates Mittel zur Wissensrepräsentation dar. Die hier betrachteten vier Ansätze fokussieren sich dabei auf unterschiedliche Domänen. Die *SUMO* stellt aufgrund ihres übergeordneten Charakters eine sehr generische Lösung dar, die versucht mit generischen und abstrakten Konzepten unterschiedliche spezielle Ontologien in einen gemeinsamen Namensraum zu vereinen. Die direkte Nutzung ist daher sehr eingeschränkt, wodurch SUMO praktisch für jeden Anwendungsfall durch Domänenontologien erweitert werden muss. *WorderWeb* verfolgt dabei genau diesen Ansatz, nämlich die Erstellung einer Bibliothek an unterschiedlichen Domänenontologien, die spezielle Informationen enthalten, die nur im entsprechenden Anwendungsfall von Nöten sind. Dadurch können die Ontologien sehr kompakt und wartungsfreundlich gehalten werden. Zur Harmonisierung unterschiedlicher Daten kann allerdings ein zusätzlicher Überbau notwendig sein. In diesem Falle würden sich WonderWeb und SUMO ergänzen. Die einzige konkrete Ausprägung einer WonderWeb Ontologie ist die *DOLCE*. Diese Ontologie fokussiert sich allerdings sehr stark auf den Bereich der Darstellung natürlicher Sprache und menschlichen Allgemeinverständnisses. Aus diesem Grund kann DOLCE keine fundierte Basis für ein Objektgedächtnismodell übernehmen, wohl aber interessant sein, wenn die spezifischen DOLCE-Inhalte für semantische Daten gefragt sind. *UbisWorld* stellt als vollständiges semantisches Framework auch Ontologien zur Nutzung bereit. Diese teilen sich zum einen in die benutzerorientierte GUMO und in die allgemeinen UbisWorld-Ontologien auf. Die GUMO modelliert dabei ein recht vollständiges Bild eines Benutzers. Diese Darstellung kann vor allem für Systeme genutzt werden, die auf Basis von Objektgedächtnissen Mehrwertdienste für einen Benutzer anbieten. Unter Umständen kann es notwendig sein nur einen Teilausschnitt der GUMO zu betrachten, um den Fähigkeiten der jeweils eingesetzten Systeme Rechnung zu tragen. Die Ubisworld-Ontologien decken ein breites Spektrum an Domänen ab (physische Objekte, Raum- und Zeitangaben, Aktivitäten). Aus diesem Grund ist eine partielle Verwendung im Bereich von Objektgedächtnissen vorstellbar. Eine umfassende Verwendung für alle Anwendungsfälle scheidet jedoch aus, da zum einen der industrielle und prozessorientierte Bereich weniger abgedeckt werden und zum anderen in Objektgedächtnissen auch Dokumente jenseits dieser Ontologien erfasst werden müssen. *Dublin Core* bietet keine vollständige Ontologie, sondern nur einen Satz an Metadaten, die

genutzt werden können um Dokumente mit einem übergreifenden und generischen Modell näher spezifizieren zu können. Das Modell kann in Form von RDF/XML-Dateien genutzt werden oder alternativ lassen sich Dublin Core Konzepte direkt über URIs referenzieren. Daher eignet sich Dublin Core sehr gut um Metadaten für Dokumente in Objektgedächtnissen zu annotieren. Die verfügbaren Dublin Core Attribute werden dabei zwar nicht genügen um eine vollständige Darstellung Objektgedächtnis-relevanter Daten abzubilden, bieten jedoch eine gute Grundlage, die um weitere Spezifika erweitert werden kann. Als Fazit kann festgehalten werden, dass keine der vorgestellten Ontologien und Metadaten als vollständige Lösung fungieren können. Es ist allerdings sinnvoll bestehende Ansätze in eine eigene Lösung zu integrieren, um vorhandene Arbeiten explizit einzubinden und den Austausch mit anderen Systemen, die auf diesen Konzepten aufbauen, zu verbessern.

### 3.8.3 Datenspeicherlösungen

Der von Kahn, Wilensky und Denning definierte Vorschlag zur Umsetzung von physischen Objekten in der digitalen Welt bietet eine sehr gut Grundlage für Systeme von digitalen Objektgedächtnissen. Die Nutzung von Dokumenten als Speicherfragment und die Aufteilung in Metaattribute und Nutzinhalt kann direkt in eine Objektgedächtnisarchitektur übernommen werden. Da die gegebenen Empfehlungen zur Umsetzung eines Zugangsprotokolls für solche Dokumente eingängig und leicht zu realisieren sind, sollten sich diese in einer Zielarchitektur wiederfinden. Das heute gebräuchlichste System zur Datenablage ist ein relationales Datenbanksystem (RDBMS). Diese stellen ein effizientes Modell dar und bieten eine sehr gute Performanz. Das stark strukturierte und fixierte Datenmodell solcher RDBMS ist sowohl ihre Stärke als auch ihre Schwäche. Da die Schemata in Vorhinein durch eine Analyse abzulegender Daten erstellt werden müssen, ist es zum einen notwendig den exakten Aufbau der Daten zu kennen. Darüber hinaus ist eine spätere Anpassung nur mit großem Aufwand und während der Laufzeit nicht sinnvoll möglich. Daher empfehlen sich solche Systeme in der Regel nur als ein reines Speicher-Backend, dem eine zusätzliche Verarbeitungsschicht vorgeschaltet ist. Dokumenten-basierte No-SQL Systeme wie CouchDB sind von ihrer prinzipiellen Konzeption deutlich näher an den Anforderungen der Flexibilität und Modularisierbarkeit angesiedelt. CouchDB bietet eine große Überdeckung an benötigter Funktionalität, kann jedoch nicht alle Ansprüche erfüllen. So bietet das System leider keine vollständige Versionsverwaltung, da nicht sichergestellt ist, dass alle älteren Versionen abgerufen werden können. Zusätzlich ist die Art der Umsetzung von Aktivitätsmodulen nicht optimal, da diese in der Regel im Webbrowser erfolgen, für Objektgedächtnisse aber eine Ausführung innerhalb des Gedächtnissystems erfolgen soll. Einen anderen Interessanten Ansatz verfolgen semantische Datenbaken wie zum Beispiel Sesame. Diese bieten eine effiziente Datenspeicherung und optimierte Anfrageprotokolle zur Ablage von RDF- und RDFS-Daten. Dadurch ist beispielsweise eine Anfrage auf semantischer Ebene möglich („Ist das Produkt noch in Ordnung?“). Als alleiniges Backend können solche Systeme allerdings

auch nicht genutzt werden, da dazu eine homogene Datenhaltung aller Informationen in Form von RDF(S)-Daten notwendig wäre. Objektgedächtnisse zeichnen sich aber gerade durch die Eigenschaft aus, eine sehr heterogene Mischung an Daten anzusammeln. Aus diesem Grund wird die Entwicklung eines spezialisierten Speicher-Backends empfohlen, welches allerdings keine Insellösung darstellen soll und daher möglichst viele Konzepte der hier vorgestellten Systeme verbinden soll. Insbesondere die Ideen aus CouchDB in Kombination mit der (teilweisen) Ablage von semantischen Daten wird dazu weiter verfolgt.

### 3.8.4 Datenvisualisierung

Auf der Seite der Datenvisualisierung wurden unterschiedliche Ansätze betrachtet, wie sich Daten, die mit Hilfe von intelligenten Objekten verteilt werden und innerhalb von instrumentierten Umgebungen abgerufen werden, für einen Benutzer visualisieren lassen. Die Universität Linz entwickelte ein Agenten-basiertes System, welches dem Model-View-Controller Modell genügt, wodurch eine strikte Trennung zwischen Daten und Visualisierung erfolgen kann. Dadurch ist eine Wiederverwendbarkeit von Teilaspekten möglich. Es finden sich jedoch keine leicht zu verwendeten Ansätze, um das System zur Laufzeit mit Funktionen zu erweitern oder die Darstellung auf dedizierte Produkte zu spezialisieren. Das AVAM System der TU Wien bietet die interessante Möglichkeit eine 3D-Visualisierung über das Internet verfügbar zu machen. Diese Nutzung steht damit im Einklang mit der Absicht Technologien des Internets im Sinne des Internet der Dinge zu nutzen. Das System lagert dabei Verarbeitungslogik in Agenten aus und kann somit flexibel auf Filter- und Abbildungsanfragen reagieren. Das System ist allerdings auf 3D-Daten festgelegt und architekturell auf die effiziente Übertragung der Visualisierung zum Client ausgelegt. Der ebenfalls Multiagenten-basierte Ansatz des DFKI bietet einen interessanten Ansatz ein System zu entwickeln, welches sich selbstständig zwischen Flexibilität und Geschwindigkeit adaptiert. Dabei wird eine funktionale Pipeline genutzt, um die Daten für die finale Darstellung aufzubereiten. Dies erlaubt es dynamisch auf den Renderingprozess einzuwirken und sich an neue Gegebenheiten anzupassen. Ein vollständiger Austausch einzelner Module der Pipeline zur Laufzeit ist allerdings nicht möglich. Weiterhin liegt auch hier der Fokus auf Daten mit hohem Renderingaufwand (zum Beispiel dreidimensionale Darstellung). Das flexible MUVA System legt den Fokus auf die Darstellung multimodaler Inhalte auf unterschiedlichen Endgeräten. Diese unterscheiden sich hinsichtlich Rechenleistung und Datenverbindung. Das Rendering erfolgt in einzelnen Softwaremodulen, welche je nach Client direkt auf diesem oder auf einem Server ausgeführt werden. Dieser interessante Ansatz zeigt die Möglichkeiten eines modularen Ansatzes. Leider sind auch hier keine Möglichkeiten gegeben, während der Laufzeit flexible den Renderingprozess an neue Objektdaten und somit auch an zusätzliche Module anzupassen. Der Dienste-orientierte Ansatz der Universität Rostock wurde explizit für Anwendungen in intelligenten Umgebungen

entwickelt. Die einzelnen Funktionseinheiten werden als Dienste realisiert und zur Laufzeit miteinander gekoppelt, im Gegensatz zu einer statischen Pipeline. Dadurch ist eine flexible Anpassung an neue Datengrundlagen gegeben. Auch kann die Ausgabe zur Laufzeit auf andere Geräte umgeleitet werden. Der Ansatz stellt bereits eine sehr gute Grundlage dar. Ihm fehlen allerdings noch Fähigkeiten zur Unterstützung von neuen Modulen durch Objektgedächtnisse. Zusätzlich wird die selektive Ausgabe in dieser Arbeit nicht benötigt. Zusammenfassend lässt sich festhalten, dass keines der Systeme eine umfassende Lösung zur Visualisierung von Daten aus Objektgedächtnissen darstellt. Ein Zielsystem wird allerdings unterschiedliche Konzepte und Ideen der hier vorgestellten Frameworks vereinen. Besonders der flexible Ansatz einer Visualisierungspipeline in Kombination mit einem modularen und zur Laufzeit adaptiven Ansatz sollte sich in der Zielarchitektur wiederfinden.

### 3.8.5 Ontologie-Editoren

Die drei vorgestellten Ontologie-Editoren (Protégé, Swoop und Neon-Toolkit) bieten einen umfassenden Funktionsumfang. Die Erstellung und Bearbeitung von Klassen, Relationen und Instanzen sowohl von lokalen als auch im Internet befindlichen OWL-Dateien beherrschen alle genanten Programme. *Protégé* biete darüber hinaus einen eingebauten Reasoner und diverse gefilterte Darstellungen, zum Beispiel fokussiert auf Klassen, Relationen oder Instanzen. Darüber hinaus kann mit einigen Versionen von Protégé die Benutzeroberfläche an die tatsächlichen Erfordernisse angepasst werden, um zum Beispiel nicht benötigte Relationen auszublenden. Diese Einstellungen sind jedoch direkt an die importierten Ontologien geknüpft, so dass dortige Änderungen die alle ausgeblendete Relationen wieder herstellen. Des Weiteren ist es nicht möglich einem Laienbenutzer zusätzliche Hilfe bei der Eingabe mit an die Hand zu geben, wodurch das Tool nur für Experten zu empfehlen ist. *Swoop* bietet einen ähnlichen Funktionsumfang und bringt durch die HTML-basierte Darstellung eine gewisse Flexibilität in die Interaktion. Leider ist die Oberfläche nicht konfigurierbar oder erweiterbar und es lassen sich keine nicht-benötigte Relationen ausblenden. Erschwerend kommt hinzu, dass das Anlegen und Verlinken von neuen Instanzen immer in zwei Schritten erfolgen muss. Somit ist der Editor ebenfalls nur für Experten bedienbar. *Neon-Toolkit* bietet auch keine Anpassbarkeit der Oberfläche oder ergänzende Hinweise zur Bearbeitung. Die rudimentäre Autovervollständigung von Namen von Relationen oder Klassen erschwert zusätzlich die Bearbeitung. Auch hier liegt der Schwerpunkt eindeutig auf Ontologieexperten. Aus diesen Gründen ist es sinnvoll zur Unterstützung von Leien einen spezialisierten Editor zu entwickeln, der die Ontologiebearbeitung soweit vereinfacht und unterstützt, dass auch Nicht-Experten Daten anlegen können.

## 3.9 Fazit

In diesem Kapitel wurden ein Auswahl verwandter Arbeiten aus den Bereichen Architekturen für intelligente Objekte, Ontologien und Metadaten, Datenspeicherlösungen, Datenvisualisierung und Editoren vorgestellt. Anschließend wurden die Arbeiten der jeweiligen Kategorien miteinander vergleichen und überprüft, inwieweit diese die angestrebten Anforderungen erfüllen (siehe Kapitel 2.9). Die Tabellen 3.3 bis 3.5 zeigen abschließend in einer Übersicht, die von den unterschiedlichen Systemen erfüllten Anforderungen in einem funktionalen Vergleich. Dazu wird jede Anforderung entweder vollständig erfüllt („Ja“), nur teilweise oder eingeschränkt erfüllt („Teilweise“) oder nicht erfüllt („Nein“). Die Übersicht wurde in drei Tabellen untergliedert, da nicht alle verwandten Arbeiten Beiträge zu allen Anforderungen liefern. Die Tabellen 3.3 und 3.4 beinhalten die Architektur-getriebenen Arbeiten und werden daher mit dem Anforderungen zur Objektgedächtnissen ($R_D$) und der Speicherinfrastruktur ($R_S$) verglichen. Die visuellen Anforderungen ($R_V$) werden ausschließlich bei den expliziten Arbeiten zur Datenvisualisierung angewendet (siehe Tabelle 3.5), da die Architektur-getriebenen Arbeiten in der Regel keine Module zur Visualisierung beinhalten und somit kein objektiver Vergleich aller Arbeiten möglich ist. Die Anforderungen in Bezug auf Werkzeuge und Migration ($R_T$) werden von den hier vorgestellten verwandten Arbeiten wenn überhaupt nur in Ansätzen erfüllt, beziehungsweise stellen keinen Schwerpunkt oder kein Ziel der jeweiligen Arbeit dar. Somit entfällt an dieser Stelle eine objektive Gegenüberstellung in Form einer Tabelle.

Zusammenfassend kann anhand der Tabellen leicht festgestellt werden, dass keine der Arbeiten alle definierten Anforderungen vollständig erfüllt. Aus diesem Grund wird die Entwicklung einer eigenen Architektur erwogen, die von den verwandten Arbeiten Ansätze und Ideen übernimmt und diese zu einem Gesamtsystem kombiniert, welches alle Anforderungen erfüllt.

Im folgenden Kapitel wird diese Architektur für digitale Objektgedächtnisse vorgestellt, die im Rahmen dieser Arbeit entwickelt wurde. Dabei wird die Architektur und ihre Bestandteile zuerst konzeptuell vorgestellt und im darauffolgenden Kapitel die Implementierung dargelegt.

| **Name** | $R_D1$ | $R_D2$ | $R_D3$ | $R_D4$ |
|---|---|---|---|---|
| SmaProN | Ja | Ja | Nein | Ja |
| Protoy/FedNet | Ja | Nein | Ja | Ja |
| SmartProducts | Ja | Ja | Teilweise | Ja |
| Tales Of Things | Ja | Nein | Ja | Teilweise |
| UbisWorld | Ja | Ja | Ja | Ja |
| RAN | Ja | Teilweise | Teilweise | Ja |
| PML | Ja | Teilweise | Teilweise | Nein |
| EPCglobal | Ja | Teilweise | Teilweise | Ja |
| *DOMeMan* | Ja | Ja | Ja | Ja |

Tabelle 3.3: Verwandte Arbeiten im funktionalen Vergleich (Digitale Produktinformation)

| **Name** | $R_S1$ | $R_S2$ | $R_S3$ | $R_S4$ | $R_S5$ | $R_S6$ |
|---|---|---|---|---|---|---|
| SmaProN | Ja | Ja | Teilweise | Ja | Nein | Nein |
| Protoy/FedNet | Ja | Ja | Ja | Ja | Nein | Nein |
| SmartProducts | Ja | Ja | Nein | Ja | Nein | Ja |
| Tales Of Things | Nein | Ja | Nein | Ja | Nein | Nein |
| UbisWorld | Ja | Ja | Nein | Ja | Nein | Nein |
| RAN | Ja | Ja | Teilweise | Ja | Nein | Nein |
| PML | Nein | Ja | Nein | Ja | Nein | Nein |
| EPCglobal | Ja | Ja | Teilweise | Ja | Teilweise | Nein |
| *DOMeMan* | Ja | Ja | Ja | Ja | Ja | Ja |

Tabelle 3.4: Verwandte Arbeiten im funktionalen Vergleich (Speicherinfrastruktur)

| **Name** | $R_V1$ | $R_V2$ | $R_V3$ | $R_V4$ |
|---|---|---|---|---|
| Intellig. Umgebungen | Ja | Nein | Ja | Nein |
| AVAM | Ja | Nein | Nein | Ja |
| Multi-Agenten | Ja | Nein | Nein | Ja |
| MUVA | Ja | Nein | Nein | Ja |
| TU Rostock | Ja | Nein | Ja | Ja |
| *DOMeMan* | Ja | Ja | Ja | Ja |

Tabelle 3.5: Verwandte Arbeiten im funktionalen Vergleich (Visualisierung)

KAPITEL 4

# Architekturmodell für Objektgedächtnisse

## 4.1 Einleitung

In diesem Kapitel wird ein Architekturmodell für Objektgedächtnisse eingeführt, welches die Grundlage für alle weiteren, im Rahmen dieser Arbeit entwickelten, Komponenten bildet. Hierzu werden zuerst die Bestandteile der Architektur eingeführt, die von Modulen zur Objektidentifizierung bis hin zur Integration von Aktivitätsmodulen reichen. Alle diese Bestandteile werden konzeptuell vorgestellt und münden anschließend in ein Bündel an Werkzeugen, die es ermöglichen Anwendungen basierend auf Objektgedächtnissen zu realisieren. Abschließend werden zwei unterschiedliche Anwendungsszenarien vorgestellt, die die Anwendung und Umsetzung der Architektur motivieren.

## 4.2 DOMeMan-Architektur

Basierend auf der in Kapitel 1.1 dargestellte Motivation und den gestellten Anforderungen im Zusammenspiel mit den betrachteten verwandten Arbeiten, wurde im Rahmen dieser Arbeit ein Architekturmodell und zugehörige Werkzeuge für digitale Objektgedächtnisse erarbeitet (*DOMeMan*: Representation, Management and Usage of Digital Object Memories). Das Modell beschreibt das grundsätzliche Zusammenspiel aller Komponenten, die benötigt werden um Objekt im Sinne des Internets der Dinge identifizierbar zu gestalten und um solche Objekt mit einem Gedächtnis auszustatten. Diese Architektur bildet somit eine Infrastruktur zur Implementierung von Anwendungen aus dem Bereich des *Product Lifecycle Management*. Die im Rahmen dieser Arbeit entwickelten Werkzeuge stellen eine konkrete Anwendung und Realisierung der Ideen dieser konzeptuellen Architektur dar. Das

Zusammenspiel der einzelnen Bestandteile zeigt die Abbildung 4.1. Im Folgenden werden diese Teile im Detail vorgestellt.

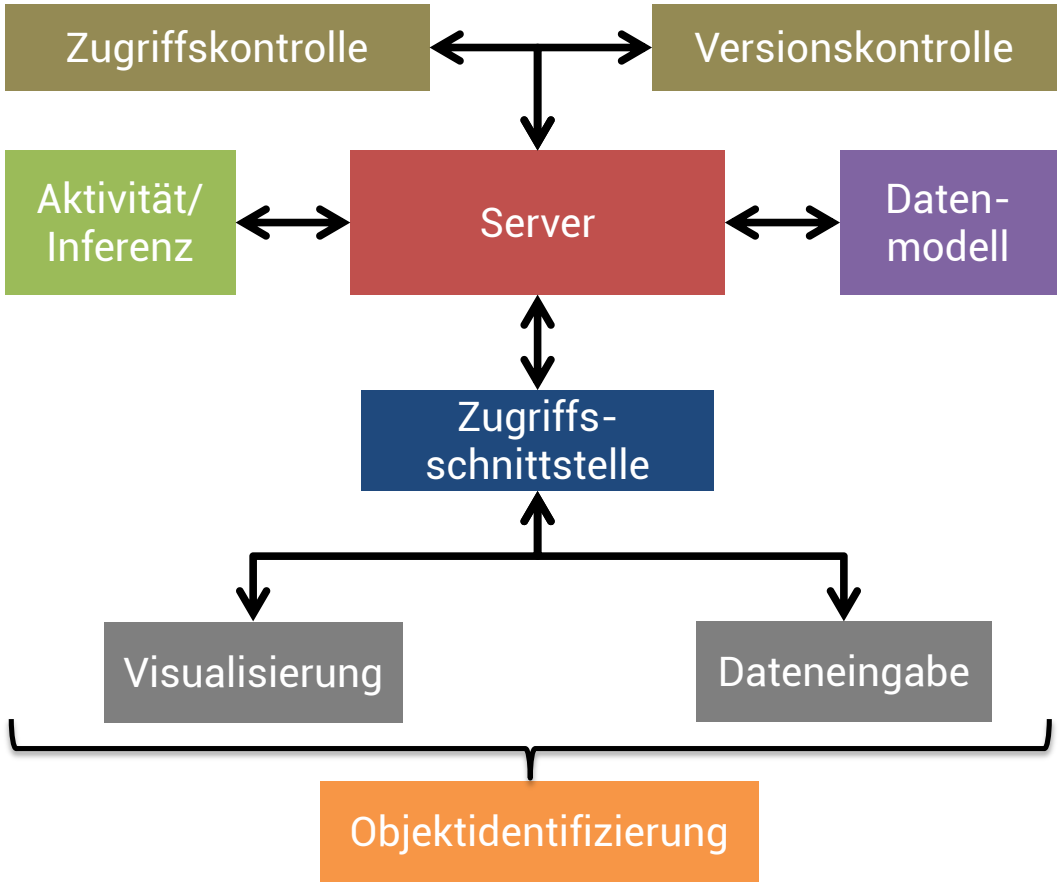



Abbildung 4.1: DOMeMan-Architektur für Objektgedächtnisse

### 4.2.1 Objektidentifizierung

Der erste Schritt zur Ausstattung von physischen Objekten mit digitalen Gedächtnissen ist die Erstellung einer eindeutigen Identifizierung (ID) im Sinne des Internets der Dinge (siehe Kapitel 2.2). Diese ID begleitet das Objekt entlang der gesamten Lebenszykluskette und stellt die Verbindung zwischen physischem Objekt und digitalem Gedächtnis her. Zu diesem Zweck ist es sinnvoll, die ID direkt am Objekt anzubringen. Damit wird jede Entität, die Zugriff auf das physische Objekt besitzt, in die Lage versetzt auch auf das Objektgedächtnis zuzugreifen.

Als eigentliche ID verwendet die DOMeMan-Architektur eine eindeutige URL. Dies hat den Vorteil, dass mit dieser URL nicht nur das Objekt identifiziert werden kann, sondern

direkt die Art und Weise des Zugriffs mit vorgegeben wird (zum Beispiel über eine HTTP-Schnittstelle). Die Architektur sieht konkret zwei unterschiedliche Verfahren vor, die sich darin unterscheiden, ob das Gedächtnis selbst direkt am Objekt angebracht ist oder nicht. Im „off-product"-Fall wird am Objekt nur die eigentliche ID angebracht und die gesamte Gedächtnisfunktionalität mit Hilfe der Umgebung realisiert (zum Beispiel via Webtechniken in die Cloud ausgelagert). Da die ID in diesem Falle nur gelesen wird, kann diese nun mit Barcodes realisiert und optisch ausgelesen werden (zum Beispiel 1D-Strichcode oder QR-Code) oder mit Funkchips umgesetzt werden, die mit speziellen Readern oder mit Smartphones gelesen werden können. Dieses Verfahren setzt allerdings dauerhaft das Vorhandensein einer Infrastruktur zur Kommunikation mit und zum Zugriff auf das Gedächtnis voraus. Im „on-product"-Fall befindet sich ein Teil oder die gesamte Gedächtnisfunktionalität direkt am Objekt, was zum Beispiel mit einem eingebetteten Controller umgesetzt werden kann. Dies hat den Vorteil, dass keine externe Infrastruktur notwendig ist, um mit dem Gedächtnis interagieren zu können; allerdings sind die Ausrüstungskosten für jedes Produkt deutlich höher als bei der kostengünstigen „on-product"-Lösung und somit nur zur Veredlung für höherpreisige Produkte geeignet. Unabhängig von der verwendeten Lösung ist eine Softwarekomponente notwendig, die auf dem jeweiligen System, welches mit dem Gedächtnis interagiert, abläuft, um die Kommunikation mit dem Gedächtnis steuern zu können.

### 4.2.2 Datenmodell

Kernelement der DOMeMan-Architektur ist das digitale Objektgedächtnis. Um dessen Eigenschaften der heterogenen Datenablage gerecht zu werden, ist ein passendes Datenmodell notwendig. Zu diesem Zweck wird eine Block-basierte Struktur genutzt, die in Anlehnung zu einem Computerdateisystem die Informationen in einzelne kleinere Fragmente kapselt. Es werden bewusst keine Aussagen über mögliche Inhalte getroffen, beziehungsweise Datentypen und Formate vorgegeben. Durch diesen offenen Ansatz der Objektgedächtnisse, tritt somit allerdings sehr häufig der Fall ein, dass ein Benutzer oder eine Anwendung nicht explizit einen exakten Block (dessen Existenz a priori bekannt ist) lesen oder schreiben möchte. Vielmehr findet bei jedem Zugriff auf ein noch unbekanntes Gedächtnis eine Suche nach Blöcken statt, die die gewünschten Daten in der gewünschten Art und Weise bereitstellen. Zu diesem Zweck ist jeder Block mit einer Menge einheitlicher Metadaten gekennzeichnet, die Auskunft darüber geben, welche Art und welches Format die im Block befindlichen Daten haben und welche Inhalte zu erwarten sind. Für einen vernünftigen Datenaustausch mit Hilfe dieses Modells ist eine Festlegung auf verwendete Datenformate und Inhalte unbedingt notwendig. Diese Aushandlung wird allerdings nicht durch das Modell vorgegeben, sondern erfolgt eigenständig zwischen den Partnern der Lebenszykluskette. Lediglich für Daten, die in allen Domänen gleichermaßen vorkommen (Strukturinformationen, Identifikationsbezeichner), bietet das Modell einige fest definierte Blöcke an, die als Einstieg in die Welt der Objektgedächtnisse verwendet werden können.

### 4.2.3 Speicherinfrastruktur

Das im vorherigen Kapitel beschriebene Speichermodell bietet nur die Möglichkeit der Strukturierung für Objektgedächtnisdaten. Zur Implementierung und Nutzung in konkreten Anwendungen ist allerdings eine auf dem Modell aufbauende Speicherinfrastruktur notwendig. Zu diesem Zweck wird ein zweistufiger Prozess verfolgt. In einer ersten Phase wird die Referenzimplementierung einer Softwarebibliothek zur Behandlung des Speichermodells erstellt. Diese erlaubt es über eine definierte Programmschnittstelle mit Objektgedächtnissen zu kommunizieren, welche mit Hilfe der Bibliothek dauerhaft gespeichert werden können. Diese Lösung eignet sich allerdings nur für Systeme, die die Gedächtnisdaten lokal ablegen und keine zusätzlichen Funktionen (wie zum Beispiel eine Versionsverwaltung) benötigen. Daher wird in einer zusätzlichen Phase eine komplexere Speicherlösung implementiert. Diese basiert auf dem beschriebenen Modell, kann aber zusätzlich mit den Daten mehreren Gedächtnisse umgehen. Daher eignet sich dieses System zusätzlich auch für den Einsatz auf dedizierten Servern, so dass am Objekt nur noch die Identifikation angebracht sein muss. Das System bietet nun die zusätzlichen Funktionen einer Versionsverwaltung, die den Benutzer in die Lage versetzt alle früheren Gedächtnisversionen abzurufen und einem Rechte- und Rollen-basierten Zugriffsschutz auf einzelne Blöcke oder vollständige Gedächtnisse. Dadurch wird sichergestellt, dass der Zugriff auf sensible Daten einer beschränkten Benutzergruppe vorbehalten bleibt.

### 4.2.4 Kommunikationsschnittstellen

Endbenutzern und Anwendungen stehen zwei unterschiedliche Schnittstellen zur Verfügung um mit Objektgedächtnissen, die auf der beschriebenen Speicherinfrastruktur abgelegt sind, zu kommunizieren. Für den Menschen bietet das System eine HTML5-basierte Weboberfläche, mit deren Hilfe direkt auf die Blockstruktur des Gedächtnisses zugegriffen werden kann und die mit der Oberfläche direkt manipuliert werden kann. Für eine Maschine-zu-Maschine-Kommunikation (M2M) bietet das System eine REST-basierte Schnittstelle an, die einen Zugriff auf alle Gedächtnisfunktionen erlaubt. Zu diesem Zweck ist die REST-Schnittstelle in einzelne Module aufgeteilt, die zum Beispiel die Speicherfunktionalität oder das Rechte-/Rollen-Modell direkt adressieren. Darüber hinaus kann das Gedächtnis mit REST-Funktionen auch Auskunft über seine Eigenschaften geben, zum Beispiel über den zur Verfügung stehenden Speicherplatz oder über die unterstützten Funktionen.

### 4.2.5 Dateneingabe und Visualisierung

Um die Erstellung von Daten für Objektgedächtnisse zu vereinfachen und zu erleichtern, bietet die DOMeMan-Architektur zwei unterschiedliche Verfahren an. Der erste Ansatz

richtete sich an Anwender, die bereits heute objektbezogenen Daten vorliegen haben, diese aber in Datenbanken abspeichern, die nur einem oder wenigen Partner vorliegen. Diese können mit einem Konvertierungswerkzeug eine Verarbeitungskette erstellen (in Sinne einer Vorlage), die es ihnen erlaubt automatisch bestehenden Daten aus Datenbanken in gedächtniskompatible Form zu bringen und in das Gedächtnismodell abzulegen. Somit ist eine schrittweise Migration möglich, da die bisherigen Systeme weiter verwendet werden können und trotzdem ab einem bestimmten Punkt automatisch ein Gedächtnis erstellt wird. Der zweite Ansatz bietet Unterstützung bei der manuellen Eingabe von Daten in semantische Strukturen (zum Beispiel Ontologien). Ein Editor erlaubt Benutzern Rohdaten in Ontologien zu instanziieren, ohne direkt mit der Struktur und Komplexität der semantischen Strukturen vertraut zu sein.

In der Gegenrichtung bietet die DOMeMan-Architektur auch ein eigenes Framework zur Visualisierung von Objektgedächtnisdaten für Benutzer. Da Gedächtnisse durch ihren offenen Ansatz keine verlässliche Datengrundlage mitbringen, ist das Framework in der Lage sich dynamisch an Objekte anpassen zu können, in dem alle Komponenten als Module realisiert sind, die bei Bedarf geladen werden und Daten lesen, konvertieren oder anzeigen können. Diese Module werden in Form eine Pipeline zusammengeschaltet. Zusätzliche Rückfallstrategien erlauben es zum Beispiel mit fehlenden Daten umzugehen oder spezialisierte Sichten für gewisse Produkte anbieten zu können. Darüber hinaus beinhaltete DOMeMan auch eine Android-App, mit deren Hilfe das Gedächtnis auf mobilen Geräten betrachtet und bearbeitet werden kann.

### 4.2.6 Aktivität

Im DOMeMan-Ökosystem fungieren Objektgedächtnisse nicht nur als Datenspeicher, sondern agieren aktiv in dem sie zum Beispiel eigenständige Aufgaben übernehmen. Zu diesem Zweck stehen drei unterschiedliche geartete Konzepte zur Verfügung um Gedächtnisse aktiv Handeln zu lassen. Allen dreien ist gemeinsam, dass die Aktivität ein modulares Konzept darstellt, indem sich Aktivitätskomponenten als Codefragmente implementieren lassen, welche direkt im Gedächtnis abgelegt werden. Zu diesem Zweck bringt die Infrastruktur eine Verarbeitungseinheit mit, die in der Lage ist diese Codefragmente auszuführen. Aktiviert werden die Komponenten entweder von außen, das heißt über die Kommunikationsschnittstelle erhält das Gedächtnis den Befehl ein bestimmtes Aktivitätsmodul auszuführen. Alternativ können Module auch an Ereignisse gebunden werden (zum Beispiel bei einer Änderung des Gedächtnisses) oder zeitgesteuert aufgerufen werden (zum Beispiel alle 60 Minuten). Zusätzlich zur Ablage der Module direkt im Gedächtnis, können diese auch einmalig von außen zugeführt und ausgeführt werden.

### 4.2.7 Werkzeuge

Im Rahmen der DOMeMan-Architektur werden alle bisher genannten Funktionseinheiten zu einem Baukastensystem kombiniert (siehe Abbildung 4.2). Grundlage bildet dabei immer der Object Memory Server (OMS), der mit seinen vier Funktionseinheiten das Grundgerüst bildet. Darauf aufbauend stehen sechs weitere Werkzeuge zur Verfügung, die je nach Bedarf genutzt werden können, um Anwendungen zu entwickeln oder zu erweitern, so dass diese Objektgedächtnisse nutzten können. Die Realisierung und Implementierung der einzelnen Werkzeuge ist im folgenden Kapitel ausführlich beschrieben.

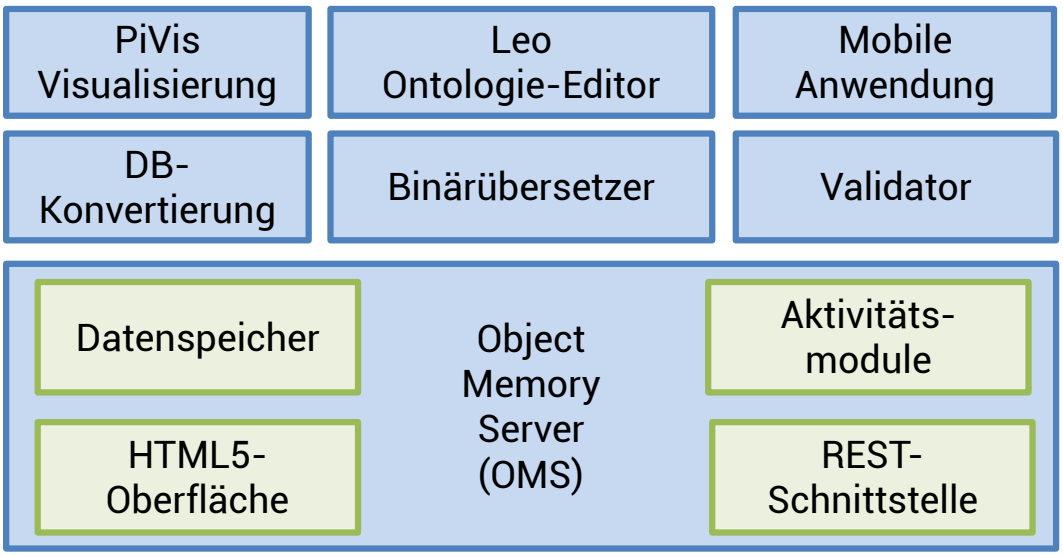



Abbildung 4.2: DOMeMan-Werkzeuge

## 4.3 Anwendungsszenarien

Zur Motivation der vorgestellten Architektur werden zwei Beispiele aus dem Bereich des Lifycycle-Management vorgestellt und verwendet, um die Anwendung der entwickelten Konzepte zu demonstrieren.

### 4.3.1 Szenario 1: Smart Pizza

Das erste Szenario zeigt den kompletten Produktlebenszyklus eines Tiefkühlprodukts, in diesem Fall eine Tiefkühlpizza. Dieses Objekt wird mit einem digitalen Objektgedächtnis basierend auf der *DOMeMan*-Architektur ausgestattet (siehe obere Hälfte der Abbildung

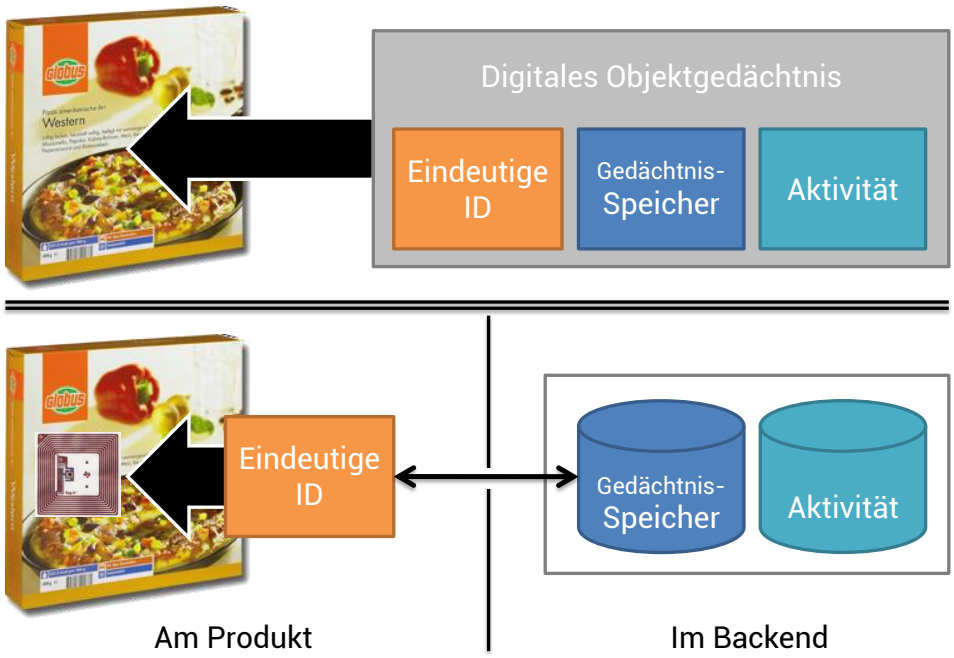



Abbildung 4.3: Szenario 1: Ausstattung einer Pizza mit einem Objektgedächtnis

4.3). Die Pizza wird dazu mit einem RFID-Tag versehen, da es bei der Pizza um ein Produkt in der Preisklasse unter 5 Euro handelt und somit nur eine Instrumentierung im Cent-Bereich wirtschaftlich möglich ist. Das Tag beinhaltet nur eine ID, die es Anwendungen ermöglicht über ein Backendsystem auf die Daten zuzugreifen. Die Funktionen des Datenspeichers und der Ausführung eigenständiger Berechnungen erfolgt dabei mit Hilfe einer Server-basierten Lösung (siehe untere Hälfte der Abbildung 4.3). Mit dieser Ausstattung können nun alle Partner im Produktlebenszyklus Daten auf diesem Gedächtnis ablegen (siehe Abbildung 4.4). Bereits bei der Herstellung werden Daten wie Inhaltsstoffe und Haltbarkeit abgelegt, die mit Hilfe des Konvertierungswerkzeuges aus bestehenden Datenbanken abgeleitet wurden. Anschließend verlässt das Produkt den Hersteller und wird mit verschiedenen Transporteuren zum Ziel bewegt. Diese überwachen während des Transports kontinuierlich den Zustand der Pizza und messen dafür Daten wie Temperatur und Luftfeuchtigkeit. Der Einzelhändler, der die Pizza für den Kunden zur Verfügung stellt, führt diese Sensorüberwachung fort. Zusätzlich fügt der Einzelhändler Informationen wie Lagerbedingungen, Preis und Verkaufsdatum hinzu. Für den Fall, dass der Kunde zum Beispiel eine instrumentierte Kühlbox besitzt, kann dieser die Überwachung der Kühlkette bis nach Hause fortsetzten (siehe [SK08]). Mit Hilfe der Visualisierungskomponente können alle Partner der Kette jederzeit auf die einzelnen Daten des Objekts zugreifen.

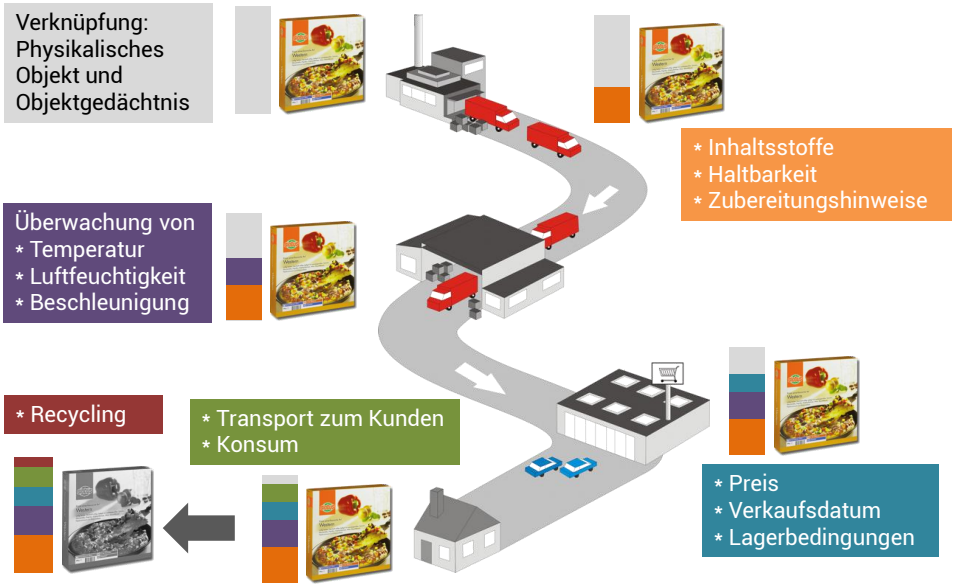



Abbildung 4.4: Datenaufbau im Objektgedächtnis mit Hilfe von Sensorik entlang der Produktlebenszykluskette

### 4.3.2 Szenario 2: Industrielle Wartung

Das zweite Szenario zeigt einen Anwendungsfall aus dem Bereich Industrie 4.0[1]. Dabei werden Komponenten einer industriellen Fertigungsanlage, wie zum Beispiel ein Motor, mit einem digitalen Objektgedächtnis basierend auf der *DOMeMan*-Architektur ausgestattet (siehe Abbildung 4.5). Im Unterschied zu Szenario 1 wird in diesem Szenario ein eingebetteter Controller am Objekt angebracht, der alle Aufgaben des Gedächtnisses übernimmt, wodurch keine Server-basierten Komponenten notwendig sind. Der Controller verwendet hierzu ebenfalls das Datenmodell und eine Teilmenge der Speicherinfrastruktur. Der Motor wird in den Lebensphasen der Herstellung bis zur Nutzung ähnliche Daten wie die Pizza im Gedächtnis ansammeln. Zusätzlich beinhaltet das Gedächtnis ein Modell des Motors, welche zum Beispiel Sollwerten und Richtgrößen enthält. Dieses Modell wird mit Hilfe eines Werkzeugs zur semantischen Datenverarbeitung mit Werten gefüllt. Während der Nutzung werden nun Daten, wie z.B. Verbrauch und Laufeigenschaften, überwacht und mit den Sollwerten verglichen. Diese Aufgaben werden teilweise vom Gedächtnis selbst durchgeführt, komplexere Schritte werden durch die nachgelagerte Infrastruktur verarbeitet, die auf Daten aus dem Gedächtnis basiert. Dadurch lassen sich zum Beispiel frühzeitig Ausfalle und Schäden erkennen und so die Wartung des Motors auf die tatsächlichen Gegebenheiten anpassen. Wird der Motor tatsächlich ausgetauscht, anschließend gelagert und schließlich

[1] http://www.hightech-strategie.de/de/2676.php [Letzter Zugriff: 28.06.2012]

wieder in einem anderen System genutzt, sind alle diese Daten jederzeit verfügbar und bieten auch späteren Nutzern ein vollständiges Bild über den Lebenszyklus des Motors.

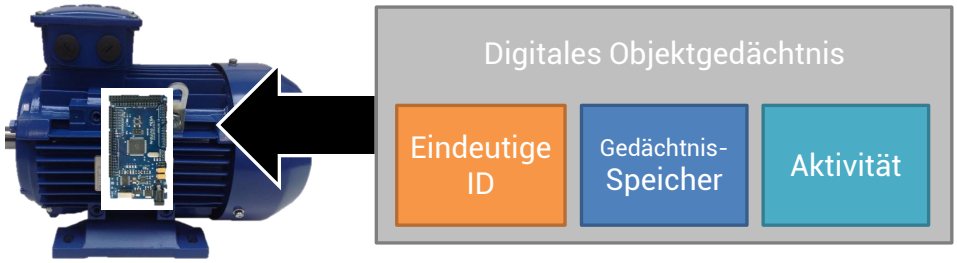



Abbildung 4.5: Szenario 2: Ausstattung eins Motors mit einem Objektgedächtnis

## 4.4 Fazit

In diesem Kapitel wurde das DOMeMan-Architekturmodell eingeführt, mit dessen Hilfe sich Anwendungen basierend auf digitalen Objektgedächtnissen realisieren lassen. Dieses Modell beinhaltet Konzepte und Lösungen für die Bereiche Objektidentifizierung, Datenmodellierung, Speicherinfrastrukturen, Kommunikationsschnittstellen, Dateneingabe, Visualisierung und Aktivitätsmodellierung, welche jeweils vorgestellt wurden. Daran anschließend wurden die im Rahmen dieser Arbeit entwickelte Werkzeugsammlung vorgestellt, welche in den folgenden Kapiteln umfassend beschrieben werden. Zwei Anwendungsszenarien runden die Darstellung ab und motivieren die Architektur an konkreten Umsetzungsbeispielen. Das folgende Kapitel 5 stellt das verwendete Datenmodell vor, während das darauffolgende Kapitel 6 die einzelnen Werkzeuge einführt.

KAPITEL 5

# Datenmodell

## 5.1 Einleitung

Im vorherigen Kapitel wurde die in dieser Arbeit entwickelte DOMeMan-Architektur in abstrakter Weise dargestellt und festgelegt, dass die Architektur aus einem Modulbaukasten besteht. Die Grundlage für alle Module, die in einem Lebenszyklus-übergreifenden Objektgedächtnis genutzt werden, stellt ein durchgängiges Datenformat dar. In diesem Kapitel werden daher die dazu notwendigen Arbeiten vorgestellt. Zuerst wird in Kapitel 5.2 das Object Memory Model präsentiert, welches eine Aufteilung von Objektgedächtnissen in einzelne Blöcke erlaubt. Dabei wird detailliert auf die verwendeten Metadaten eingegangen. Zusätzlich werden bereits definierte und direkt nutzbare Blöcke vorgestellt. Im Anschluss wird die Datenrepräsentation dieses Modells in Form von XML- und RDFa/Microdata-Repräsentationen gezeigt. Die Kapitel 5.3 und 5.4 zeigen darauf folgend zusätzliche ontologische Datenmodelle, mit deren Hilfe Produkt- und Benutzerdaten im Objektgedächtnis abgelegt werden können. Abschließend gibt das Kapitel 5.5 einen Exkurs zum Thema regelbasierte Daten, welche ebenfalls für die Verwendung in Objektgedächtnissen angepasst wurden.

## 5.2 Object Memory Model (OMM)

Um die in Kapitel 2.9 gestellten Anforderungen zu erfüllen wurde im Rahmen einer W3C[1] Incubator Group (XG) das *Object Memory Model* (OMM) entwickelt [KHS+11]. Das OMM partitioniert die zu speichernden Informationen in mehrere Datenblöcke. Jeder Block beinhaltet ein spezifisches Informationsfragment und stellt zusätzlich eine Menge an Metadaten bereit, um eine Suche nach passenden Informationen zu erleichtern. Diese Liste an Blöcken

[1] http://www.w3c.org

wird durch ein zusätzliches, optionales Inhaltsverzeichnis (Table of Contents, ToC) und einen dedizierten Header ergänzt (siehe Abbildung 5.1).


Abbildung 5.1: Beispielhaftes Objektgedächtnis basierend auf OMM-Format

### 5.2.1 Gedächtnis-Header

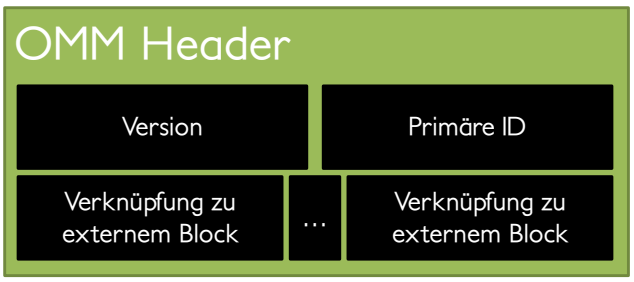



Abbildung 5.2: OMM-Header

Der OMM-Header findet sich zu Beginn jedes Gedächtnisses und gibt zum einen die verwendete *Version des OMM-Formats* an und definiert zusätzlich die *primäre ID* des Gedächtnisses (siehe Abbildung 5.2). Diese ID ist optional (aber unveränderlich sobald sie einmal festgelegt wurde) und wird dazu genutzt verschiedene Gedächtnisse und Blöcke von Gedächtnissen in Relation zu setzen. Solange keine dieser Relationen definiert sind, kann die primäre ID entfallen; werden Relationen genutzt ist diese zwingend zu definieren. Sollte die Notwendigkeit bestehen weitere IDs für ein Objekt definieren zu müssen, so kann dies in einem eigenständigen Datenblock erfolgen (siehe Abbildung 5.2.4).

Zusätzlich ist es möglich *Verweise zu externen Blöcken* zu definieren, die sich nicht innerhalb dieses Gedächtnisses befinden (siehe Abbildung 5.3). Dies erlaubt es z.B. Blöcke mit großem Nutzinhalt oder seltenem Bedarf an einen anderen Speicherort auszulagern; Blöcke von verschiedenen Gedächtnissen gemeinsam zu nutzen oder Blöcke aus anderen Gedächtnissen zu importieren. Es stehen dabei drei verschiedene Arten der Verlinkung zur Verfügung:

**Typ 1: Verweis auf ein anderes Gedächtnis** In diesem Fall werden alle Blöcke dieses externen Gedächtnisses integriert. Mit Hilfe dieses Verfahrens, lassen sich Gedächtnisse über mehrere Speicherorte aufteilen. Das kann insbesondere dann von Interesse sein, wenn Teile des Gedächtnisses jederzeit verfügbar sein müssen oder wenn mehrere Gedächtnisse

eine große gemeinsame Schnittmenge haben, so dass dieser gemeinsame Teil in ein eigenes Gedächtnis ausgelagert werden kann.

**Typ 2: Verweis auf einen dedizierten Block eines anderen Gedächtnisses** Mit diesem Verfahren können einzelne Blöcke aus anderen Gedächtnissen integriert werden, so dass auch bei einzelnen Blöcken die Redundanz reduziert und die Erreichbarkeit erhöht werden kann.

**Typ 3: Verweis auf einen externen Speicherplatz für OMM-Blöcke** Um größere oder selten benutzte Blöcke auszulagern, können diese auf einem externen Speicherplatz abgelegt werden. Als Speicher werden ein Webserver oder ein OMS-System (siehe Kapitel 6.5) unterstützt.

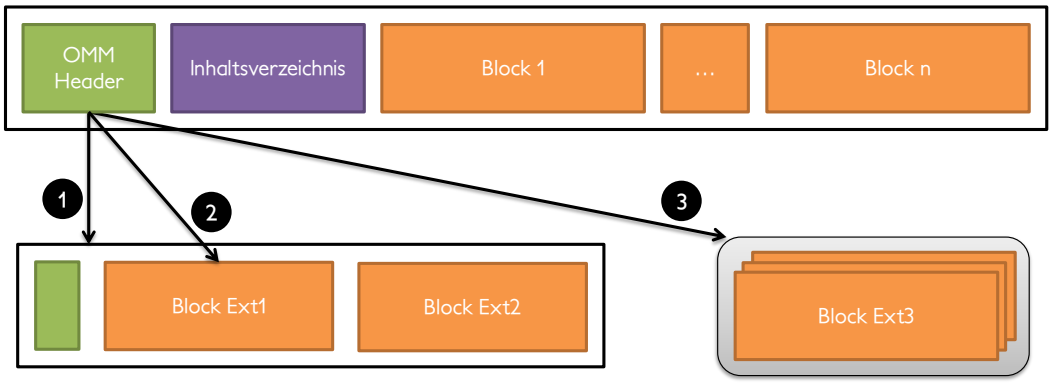



Abbildung 5.3: OMM-Block mit Links zu externen Blöcken

### 5.2.2 Metadaten

Jeder Block im Object Memory Modell beinhaltet Informationen zu einem speziellen Themenkomplex. Dabei sind alle Blöcke gleichbedeutend, d.h. es gibt keine Gewichtung oder Priorisierung bestimmter Blöcke, bzw. deren Anordnung und Reihenfolgen impliziert keine Ordnung. Weiterhin gibt es (in der Regel, Ausnahmen siehe Kapitel 6.5.4) keine Zugriffseinschränkungen und Randbedingung zum Aufbau oder Inhalt eines Gedächtnisses, daher hat jeder Anwender bzw. jede Anwendung die Möglichkeit Blöcke hinzuzufügen, zu verändern oder zu entfernen. Aus diesen Gründen kann eine Anwendung keine Annahmen treffen, welche Daten in einem Gedächtnis anzutreffen sind und wo diese abgelegt sind. Aus diesem Grund bietet jeder Block, im Sinne einer Selbstauskunft, einen Satz an Metadaten an, mit dessen Hilfe Anwender und Anwendungen in der Lage sind die Blöcke zu identifizieren und

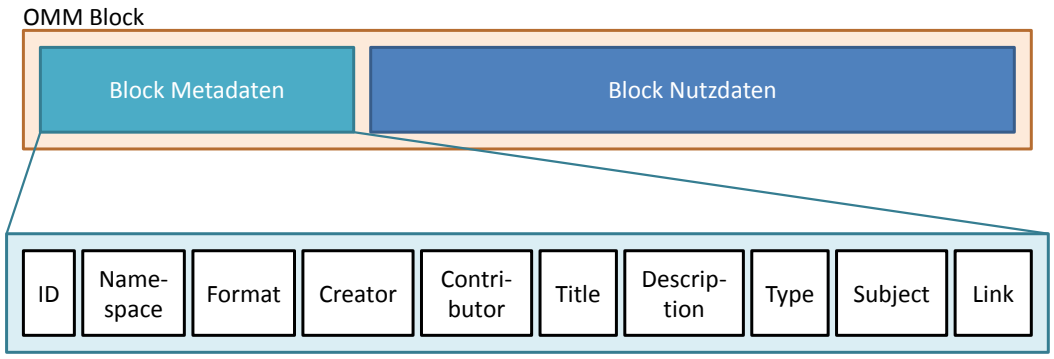



Abbildung 5.4: OMM-Block mit Metadaten und Nutzinhalt

zu lokalisieren, die relevante Daten als Nutzinhalt bereitstellen (siehe Abbildung 5.4). Die einzelnen Metadaten werden nun in der folgenden Aufstellung detailliert beschrieben:

### ID

- *Zweck*: Eindeutige Identifikation des Blocks innerhalb des Gedächtnisses
- *Datentyp*: String
- *Wichtigkeit*: Zwingend erforderlich (1x pro Block)
- *Häufigkeit*: 1x pro Block
- *Notwendigkeit*: Die Block-ID wird benutzt, um einen Block innerhalb eines Gedächtnisses eindeutig identifizieren zu können (z.B. vom Inhaltsverzeichnis aus, siehe auch Abbildung 5.2.3) oder einen Verweis von einer externen Relation zu ermöglichen. Der Block wird dabei durch die Verkettung von Gedächtnis-ID und Block-ID adressiert.
- *Beispiel*: „`block123`“

### Namespace

- *Zweck*: Der Namespace definiert sowohl den Inhalt als auch das Format des Blocks in eindeutiger Art und Weise.
- *Datentyp*: URI
- *Wichtigkeit*: Optional bzw. zwingend erforderlich (falls kein *Format* angegeben)
- *Häufigkeit*: 1x pro Block

- *Notwendigkeit*: Mit Hilfe dieses Attributes kann der Inhalt und das Datenformat von Blöcken direkt eindeutig spezifiziert werden.
- *Beispiel*: „urn:omm:block:indentifications“

### Format

- *Zweck*: Definiert das Format bzw. die Kodierung des Nutzinhalts eines Blocks in Form einer MIME-Typ Angabe. Der Wert dieses Attributes entspricht dem Dublin Core Attribut „Format“ (siehe http://purl.org/dc/elements/1.1/format).
- *Datentyp*: MIME-Typ als String
- *Attribute*:
  - *Schema*: Eine XML-Schema Definition für den MIME-Typ application/xml.
  - *Encryption*: Angabe des Verschlüsselungsalgorithmus, der auf den Nutzinhalt des Blocks angewendet wurde (z.B. AES256)
- *Wichtigkeit*: Optional bzw. zwingend erforderlich (falls kein *Namespace* angegeben)
- *Häufigkeit*: 1x pro Block
- *Notwendigkeit*: Das Attribut erlaubt es Parsern ausschließlich Blöcke mit bekannten Formaten zu betrachten und dient gleichzeitig als Rückfallebene falls kein Namespace angegeben wurde.
- *Beispiel*: „text/plain“

### Creator

- *Zweck*: Beinhaltet das Datum, an dem der Block erstellt wurde, sowie die Identität des Erstellers. Entspricht dem Dublin Core Eintrag „Creator“ (siehe http://purl.org/dc/elements/1.1/creator).
- *Datentyp*: Ein Schlüssel-Wert-Paar bestehend aus einem Zeitstempel und einer Entität (s. Beispiele). Der Zeitstempel wird im ISO8601-Format angegeben. Für den Wert der Entität sind folgende Möglichkeiten vorgesehen: DUNS (Data Universal Numbering System), GLN (Global Location Number), Email, OpenId, CertificateID.
- *Wichtigkeit*: Zwingend erforderlich
- *Häufigkeit*: 1x pro Block

- *Notwendigkeit*: Attribut wird für Zwecke wie Logging, Herkunftsangaben („Provenance“) oder rechtliche Nachweise verwendet.
- *Beispiel1*: „(2011-09-30T18:30:00+02:00, [195505177 *ofType* duns])“
- *Beispiel2*: „(2012-02-29T08:12:00+02:00, [jens.haupert@dfkid.de *ofType* email])“

### Contributor

- *Zweck*: Beinhaltet das Datum, an dem der Block verändert wurde, sowie die Identität des Editors. Entspricht dem Dublin Core Eintrag „Contributor“ (siehe http://purl.org/dc/elements/1.1/contributor).
- *Datentyp*: Ein Schlüssel-Wert-Paar bestehend aus einem Zeitstempel und einer Entität (s. Creator).
- *Wichtigkeit*: Optional
- *Häufigkeit*: Beliebige Anzahl pro Block
- *Notwendigkeit*: Attribut wird für Zwecke wie Logging, Herkunftsangaben („Provenance“) oder rechtliche Nachweise verwendet.
- *Beispiel1*: „(2011-09-30T18:30:00+02:00, [195505177 *ofType* duns])“
- *Beispiel2*: „(2012-02-29T08:12:00+02:00, [jens.haupert@dfkid.de *ofType* email])“

### Title

- *Zweck*: Gibt einen kurzen, mehrsprachigen und lesbaren Titel für diesen Block an (weniger als 255 Zeichen) und entspricht dem Dublin Core Attribut „Title“ (siehe http://purl.org/dc/elements/1.1/title).
- *Datentyp*: String
- *Wichtigkeit*: Zwingend erforderlich
- *Häufigkeit*: 1x pro Block und pro Sprache
- *Notwendigkeit*: Das Attribut wird verwendet um einem Benutzer einen lesbaren und verständlichen Titel dieses Blocks zu präsentieren.
- *Beispiel1*: „(en, structure information)“
- *Beispiel2*: „(de, Strukturdaten)“

### Description

- *Zweck*: Gibt eine mehrsprachige und lesbare Beschreibung für diesen Block an und entspricht dem Dublin Core Attribut „Description“ (siehe `http://purl.org/dc/elements/1.1/description`).
- *Datentyp*: String
- *Wichtigkeit*: Optional
- *Häufigkeit*: 1x pro Block und pro Sprache
- *Notwendigkeit*: Das Attribut wird verwendet um einem Benutzer eine lesbare und verständliche Beschreibung dieses Blocks zu präsentieren.
- *Beispiel1*: „`(en, This block contains structural information, e.g., connections to other object)`“
- *Beispiel2*: „`(de, Dieser Block beinhaltete Strukturinformationen, z.B. mit welchen anderen Objekten dieses Objekt verbunden ist.)`“

### Type

- *Zweck*: Entspricht dem Dublin Core Attribut „Type“. Mögliche Werte sind: Collection, Dataset, Event, Image, InteractiveResource, Service, Software, Sound, Text, PhysicalObject.
- *Datentyp*: String (Dublin Core „Type“)
- *Wichtigkeit*: Optional
- *Häufigkeit*: 1x pro Block
- *Notwendigkeit*: Zusätzlich Informationen für Anwendungen die das Dublic Core Format unterstützen.
- *Beispiel*: „`http://purl.org/dc/dcmitype/Dataset`“

### Subject

- *Zweck*: Als Subject können freie Klartext-Tags oder Ontologiekonzepte (RDF oder OWL) angegeben werden, die den Nutzinhalt des Blocks näher klassifizieren. Zusätzlich zu einfachen Strings, können Hierarchien auf einfache Art und Weise als String genutzt werden. Dazu werden die Hierarchieebenen durch Punkte getrennt (siehe Beispiel 3 für Tag „e12 *istUnterklasseVon* din *istUnterklasseVon* norms“). Einfache Klartext-Tags basieren auf dem Dublin Core „Subject“ Attribut (siehe `http://purl.org/dc/elements/1.1/subject`).

- *Datentyp*: Ein Schlüssel-Wert-Paar bestehend aus dem Typ und dem Wert des Subjects (s. Beispiele).

- *Wichtigkeit*: Optional

- *Häufigkeit*: Beliebige Anzahl pro Block

- *Notwendigkeit*: Erlaubt es Blöcke mit beliebigen Tags zu versehen, die als einfacher Text, als Text-Hierarchie oder als Ontologiekonzepte dargestellt sein können.

- *Beispiel1*: „`(ontology, http://o.org/def.owl#ManufacturerData)`“

- *Beispiel2*: „`(text, ingredients)`"

- *Beispiel3*: „`(hierachiy, norms.din.e12)`"

### Link

- *Zweck*: Sollte es notwendig sein, die Nutzdaten eines Blocks auf einem externen Server ablegen zu wollen, kann das Link-Attribut genutzt werden, mit dessen Hilfe die Position des externen Speicherorts definiert werden kann.

- *Datentyp*: Eine Adresse zu einer externen Quelle (typischerweise eine URL).

- *Wichtigkeit*: Optional / Zwingend erforderlich (falls kein „Payload“ vorhanden ist)

- *Häufigkeit*: 1x pro Block

- *Notwendigkeit*: Erlaubt es große Nutzdaten in externe Speicherquellen auszulagern.

- *Beispiel*: „`(url, ⟨hash⟩, http://www.w3.org/2005/Incubator/omm/samp/p1.xml)`“

#### Payload

- *Zweck*: Hierbei handelt es sich um kein Meta-Attribut im eigentlichen Sinne, sondern um die Struktur zur Aufnahme von Nutzinhalten die direkt im Gedächtnis abgelegt werden.
- *Datentyp*: Binär oder UTF-8 Text
- *Attribute*: *keine*
- *Wichtigkeit*: Optional bzw. zwingend erforderlich (falls kein „Link“ vorhanden ist)
- *Häufigkeit*: 1x pro Block
- *Notwendigkeit*: Für alle Daten, die direkt im Gedächtnis abgelegt werden sollen, kann der „Payload“ genutzt werden. Größere Daten können üben das Attribut „Link“ ausgelagert werden.

### 5.2.3 Inhaltsverzeichnis

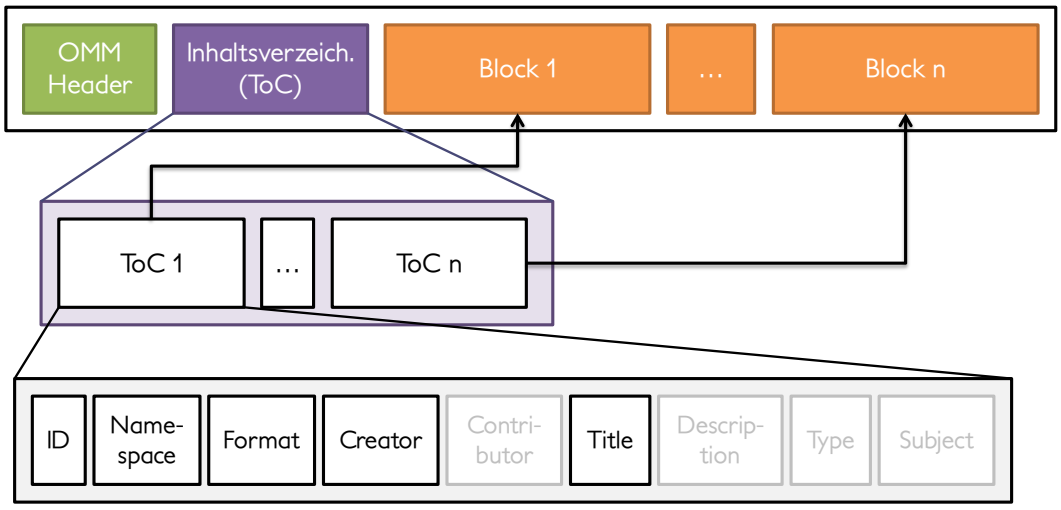



Abbildung 5.5: OMM-Inhaltsverzeichnis

Als zusätzliche dritte Datenstruktur in Ergänzung zum Header und zu den Blöcken, besitzt das OMM-Gedächtnis ein Inhaltsverzeichnis (Table of Contents, ToC), dessen Zweck es ist in Szenarios, in denen das Lesen des gesamten Gedächtnisses nicht gewünscht ist oder technische nicht machbar ist, trotzdem einen Überblick über alle vorhandenen Blöcke zu erhalten. Folgenden Beispielszenarien machen dabei Gebrauch vom Inhaltsverzeichnis, wodurch der Kommunikationsaufwand im Vergleich zum Transfer des gesamten Gedächtnisses deutlich reduziert werden kann:

- Klassifikation von Objekten mit Gedächtnissen, die eine sehr große Anzahl an Blöcken aufweisen oder Blöcke besitzen, die sehr große Nutzdaten enthalten

- Interaktion mit Gedächtnissen mit Blöcken, die in externen Speichern abgelegt sind

- Gedächtnisspeicher ohne die Fähigkeit des wahlfreien Speicherzugriffs

Das Inhaltsverzeichnis ist analog zu den Metadaten der einzelnen Blöcke aufgebaut und bildet eine Untermenge ebendieser (siehe Abbildung 5.5). Folgende Metadaten sind im Inhaltsverzeichnis zwingend vorgeschrieben:

- *ID* - Über die ID wird die Verknüpfung zwischen Einträgen im Inhaltsverzeichnis und den jeweiligen Blöcken hergestellt.

- *Namespace* oder *Format* - Für jeden Block muss entweder der Namespace oder als Rückfallebene zumindest eine Formatangabe vorliegen, damit Blöcke mit unbekanntem Inhalt oder Kodierung ignoriert werden können.

- *Creator* - Sollten mehrere Blöcke mit gleichem Inhalt vorliegen, kann eine Anwendung anhand des Erstellers entscheiden, welcher Block von Interesse ist.

- *Title* - Um auch Benutzern eine Blockübersicht mit Klartextnamen anbieten zu können, muss auch ein Titel für jeden Block angegeben werden.

Alle weiteren Attribute sind hingegen nicht verpflichtend, können jedoch auch im ToC angegeben werden. Die Entscheidung welche dieser optionalen Attribute gelistet werden, trifft dabei der Anwender oder die Anwendung selbst.

### 5.2.4 Standardisierte Blöcke

Das Object Memory Modell beinhaltet zusätzlich zur eigentlichen Blockarchitektur noch drei Vorlagen zur Darstellung von häufig auftretenden Nutzdaten. Werden diese Vorlagen in Blöcken genutzt spricht man von sogenannten „standardisierten Blöcken“, welche entweder eine bestimmte Funktion im Objektgedächtnis übernehmen (und somit einen Aspekt des OMM implementieren) oder als Basisstrukturen zur Verfügung stehen. In beiden Fällen dienen standardisierte Blöcke dazu die Kommunikation verschiedener Anwendungen über Domänen- und Applikationsgrenzen hinweg mit Hilfe des Objektgedächtnisses zu erleichtern.

**Struktur-Block** Der Strukturblock erlaubt es Benutzern und Anwendungen Relationen zwischen diesem Objekt und anderen Objekten zu definieren. In Produktions- und Herstellungsszenarios können mit dieser Struktur Verbindungs- und Integrationsinformationen abgelegt werden, wie z.B. „Objekt A ist verbunden mit Objekt B“ oder „Objekt C ist ein Teil von Objekt D“.

Zusätzlich wird zu jeder Relation entweder ein Zeitstempel abgespeichert, der angibt seit wann diese Relation Gültigkeit hat oder ein Zeitraum, der angibt wann die Relation gültig war. Der Strukturblock ist dabei eindeutig über das „Namespace“-Attribut `http://www.w3.org/2005/Incubator/omm/ns/structureBlock` zu erkennen (siehe Abbildung 5.7).

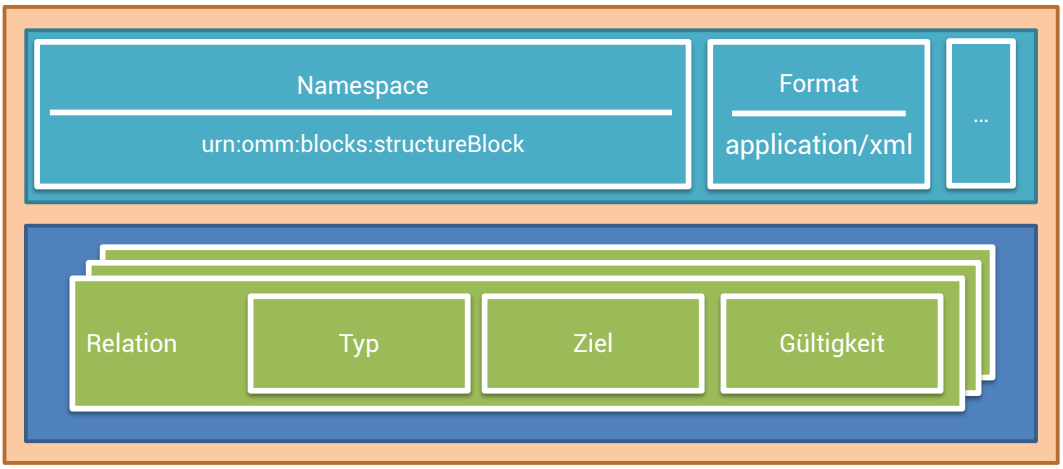



Abbildung 5.6: OMM-Strukturblock

Um eine gewisse Interoperabilität sicherzustellen, sind folgende Relationen bereits durch das OMM-Format definiert (es können aber jederzeit auch eigenständige Relationen verwendet werden):

- *isConnectedWith* - Dieses Objekt ist mit einem anderen Objekt verbunden; üblicherweise handelt es sich dabei um eine (z.B. kabelgebundene) Verbindung zum Datenaustausch.

- *isPartOf* - Dieses Objekt ist ein Teil eines anderen Objekts und fungiert als strukturelle oder funktionale Subkomponente; die inverse Relation ist *hasPart*.

- *hasPart* - Dieses Objekt besitzt als Teilkomponente ein anderes Objekt, welches eine strukturelle oder funktionale Subkomponente darstellt; die inverse Relation ist *isPartOf*.

- *isStoredIn* - Dieses Objekt ist vollständig in einem anderen Objekt eingebettet, ohne jedoch tatsächlich einen funktionalen Teil darzustellen (z.B. das Einladen von Produkten in ein Transportfahrzeug).

Beispielhaft könnte ein Motor einer Industrieanlage über eine Steuerleitung mit dem Anlagenmodul verbunden sein. Dieses Modul besitzt auch ein OMM-Gedächtnis mit der URL `http://omm.org/memory2`. Zusätzlich besteht diese Verbindung seit dem 10.02.2012 10:00 Uhr. In diesem Fall würde das Gedächtnis des Motors folgenden Eintrag besitzen:


Abbildung 5.7: Beispielhafter OMM-Strukturblock mit *isConnectedWith*-Relation

**IDs-Block** Der ID-Block hat die Aufgabe, verschiedene Identifikationskennungen (ID) eines Objekts an zentraler Stelle zu bündeln. Diese IDs stellen eine Ergänzung zur eventuell definierten primären ID im Header des Objektgedächtnisses dar (siehe Abbildung 5.2.1) und treten während den einzelnen Phasen des Produktlebenszyklus auf. Die IDs können dabei dauerhaft gespeichert werden oder nur temporäre Gültigkeit haben (wobei sie nach Gültigkeitsende wieder entfernt werden).

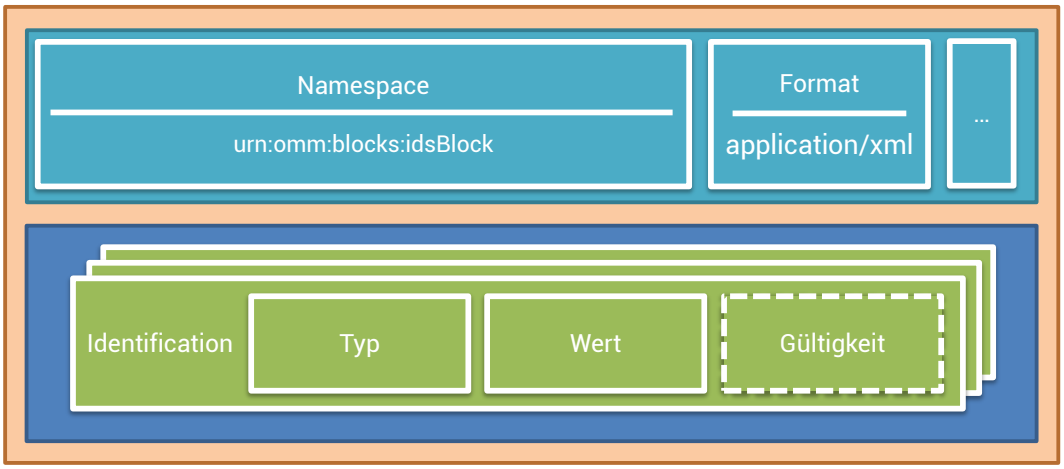



Abbildung 5.8: OMM-IDs-Block

Der IDs-Block wird eindeutig über den Namespace `http://www.w3.org/2005/Incubator/omm/ns/idsBlock` identifiziert. Der IDs-Block erlaubt es daher auch die Gültigkeit einer ID anzugeben. Diese Angabe ist nur optional, erlaubt es aber die ID-Historie eines Objekts

zu verfolgen. Wird keine Gültigkeit angegeben, gilt diese ID seit Beginn des Gedächtnisses und bleibt bis zum Ende bestehen.

Beispielhaft könnte ein Motor einer Industrieanlage von einem Hersteller konstruiert worden sein, der die DUNS ID *19-620-5025* besitzt. Da sich der Hersteller des Motors nicht mehr ändert, kann das Gültigkeits-Attribut entfallen und daher würde sich im Gedächtnis des Motors folgender Eintrag befinden:

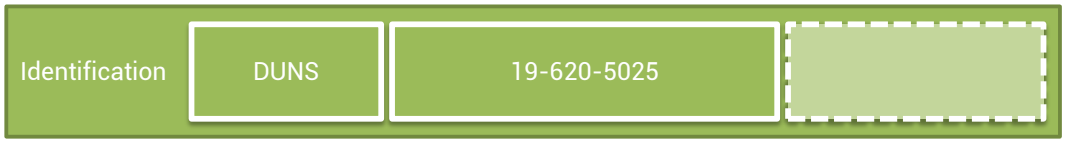



Abbildung 5.9: Beispielhafter OMM-IDs-Block mit Angabe einer *DUNS*-ID

**Schlüssel-Wert-Paar Blockvorlage** Hierbei handelt es sich um keinen Block mit fest definiertem Inhalt, sondern um eine Strukturvorlage zur Ablage beliebiger einfacher Daten, welche in diversen Domänen Verwendung finden. Die Daten werden dazu als Schlüssel-Wert-Paare (Key-Value-Pair) abgelegt. Abbildung 5.10 zeigt ein Beispiel, indem (innerhalb eines Abfüllprozesses) die Menge von abgefüllten Pillen getrennt nach ihrem Typ gespeichert werden.

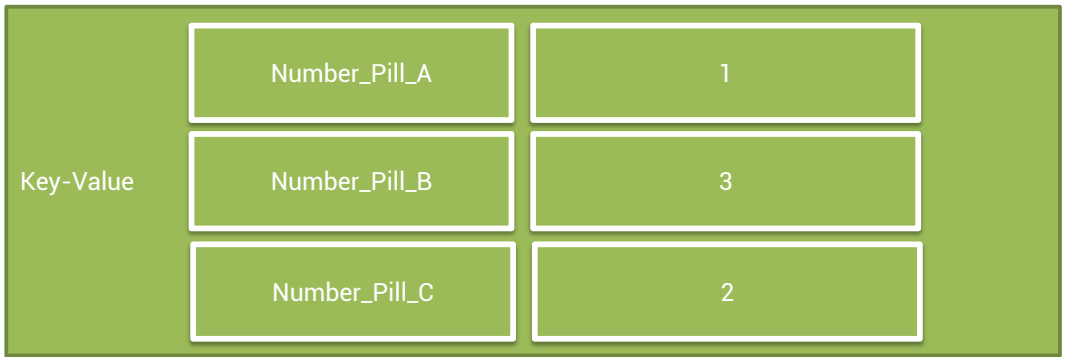



Abbildung 5.10: Beispielhafter OMM-Key-Value-Block

### 5.2.5 Zusätzliche fest-definierte Blöcke

Zusätzlich zu den in der OMM XG-Definition festgelegten Standardblöcken, sind weitere fest definierte Blöcke im Bereich des OMM-Ökosystems sinnvoll. Diese als *OMM+* bezeichneten Ergänzungen werden im Folgenden beschrieben.

### OMM+-Semantik Block

Die durch den OMM-Strukturblock möglichen semantischen Darstellungen sind durch die fixe Liste an Relationen stark begrenzt. Aus diesem Grund erweitert der OMM+-Semantik Block dieses Konzept um einen offenen und flexibleren Ansatz. Daher wird auch der strukturelle Aufbau des Strukturblocks übernommen und mit dem OMM+-Namespace `urn:ommplus:blocks:semanticBlock` versehen (siehe Abbildung 5.11). Der Typ der Relation ist nun aber frei wählbar und kann entweder explizit für dieses Gedächtnis definiert werden oder von einer bestehenden RDF- oder OWL-Ontologie importiert werden. Zusätzlich ist das Subjekt der Relation nicht automatisch das jeweilige Objekt (wie im Strukturblock), sondern auch frei wählbar. Alle drei Attribute (Subjekt, Prädikat und Objekt) werden als Typ URI (Uniform Resource Identifier [BLFM05]) definiert und können somit als URL (Uniform Resource Locators) oder URN (Uniform Resource Names) angegeben werden. Des Weiteren kann jede dieser Relationen mit einem Gültigkeitszeitraum belegt werden. Mit diesem Umfang können einfache Relationen analog zu RDF und OWL direkt im Gedächtnis abgelegt werden. Der Vorteil gegenüber der expliziten Verwendung von RDF oder OWL liegt in der Tatsache, dass zum einen die jeweilige OMM-Bibliothek direkt diese Angaben auswerten kann und zum anderen an einem fest definierten Vokabular, welches ohne externe Ontologie nutzbar ist.

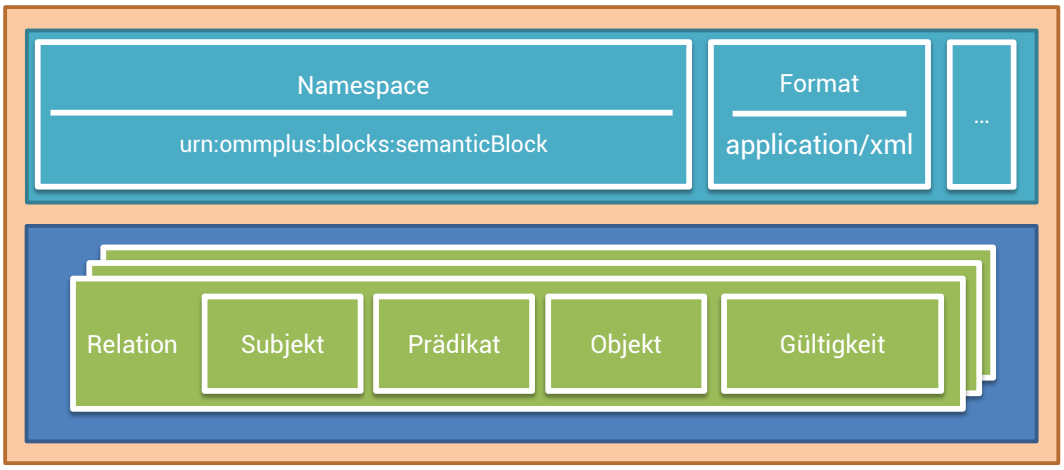



Abbildung 5.11: OMM+-Semantic-Block

Quellcode 5.1 zeigt an drei exemplarischen Beispielen die Möglichkeiten dieses Blocks. Beispiel 1 zeigt eine Integration des OMM-Strukturblocks mit Hilfe der Syntax des Semantikblocks. Als Subjekt dient das Objekt, welches durch dieses Gedächtnis beschrieben wird, (`urn:omm:this`) in Kombination mit der Struktur-Relation *isConnectedWith* (dargestellt als URN `urn:omm:structure:isConnectedWith`). Das Zielobjekt der Relation ist in diesem Fall das Objekt mit dem Gedächtnis `http://www.samplememories.org/p2`. Analog zum Strukturblock kann auch ein Gültigkeitszeitpunkt oder -zeitraum angegeben werden (in diesem

Beispiel ein Zeitraum). Im Falle der Benutzung des Zeitraums kann dieser auch nach oben offen sein und stellt damit die Tatsache dar, dass diese Aussage ab diesem Zeitpunkt gilt. Somit stellt der Semantikblock eine Obermenge des Strukturblocks dar. Beispiel 2 zeigt die Integration und Verwendung von externen OWL-Relationen mit Hilfe des Semantikblocks. Dabei zeigt die OWL-Relation *hasCondition* (`http://omm.org/cond.owl#hasCondition`) von diesem Gedächtnis (`urn:omm:this`) auf das Konzept *OK* (`http://omm.org/cond.owl#OK`). In diesem Fall wird auch ein Zeitpunkt angegeben, an dem diese Aussage eine Gültigkeit hatte (zum Beispiel, weil zu diesem Zeitpunkt eine Messung und Berechnung stattgefunden hat). Durch die Verwendung von OWL-Konzepten kann zusätzlich mit Hilfe eines nachgelagerten Reasonings auf die Erweiterung des Faktenwissens durch automatisches Schließen zurückgegriffen werden. Beispiel 3 zeigt abschließend die Benutzung einer eigens definierte Relation (hier `urn:myCompany:isManufacturer`), die von einer externen Ressource (`http://www.myCompany.com/`) auf das dem Gedächtnis zugeordnete Objekt zeigt (`urn:omm:this`), womit klar ist, dass *this* auch als Objekt fungieren kann. Im dritten Fall wird nur eine vereinfachte Semantik verwendet, die auf die Möglichkeiten des Reasonings verzichtet.

**Quellcode 5.1: OMM+-SemantikBlock Beispiel in XML-Darstellung**

```
<ommplus:semantic>
 <ommplus:semanticInformation> <!-- Beispiel 1 -->
   <omm:timeSpan>
     <omm:begin omm:encoding="ISO8601">2012-08-30T18:30:00+02:00</omm:begin>
     <omm:end omm:encoding="ISO8601">2012-08-17T18:30:00+02:00</omm:end>
   </omm:timeSpan>
   <ommplus:subject>urn:omm:this</ommplus:subject>
   <ommplus:predicate>urn:omm:structure:isConnectedWith</ommplus:predicate>
   <ommplus:object>http://www.samplememories.org/p2</ommplus:object>
 </ommplus:semanticInformation>

 <ommplus:semanticInformation> <!-- Beispiel 2 -->
   <omm:date omm:encoding="ISO8601">2012-08-03T00:00:00+02:00</omm:date>
   <ommplus:subject>urn:omm:this</ommplus:subject>
   <ommplus:predicate>http://omm.org/cond.owl#hasCondition</ommplus:predicate>
   <ommplus:object>http://omm.org/cond.owl#OK</ommplus:object>
 </ommplus:semanticInformation>

 <ommplus:semanticInformation> <!-- Beispiel 3 -->
   <ommplus:subject>http://www.myCompany.com/</ommplus:subject>
   <ommplus:predicate>urn:myCompany:isManufacturer</ommplus:predicate>
   <ommplus:object>urn:omm:this</ommplus:object>
 </ommplus:semanticInformation>
</ommplus:semantic>
```

### OMM+-Embedded Block

Da Objektgedächtnisse sowohl in externen Backendsystemen als auch am Objekt selbst gespeichert werden können, z.B. in RFID-Tags, kann der Fall eintreten, dass Objekte in andere Objekte so verbaut werden, dass der Zugriff auf ein RFID-Tag nicht mehr möglich ist, da das Tag nicht mehr erreicht werden kann oder abgeschirmt wird. Lag das Gedächtnis nun im Chip vor, ist somit kein Zugriff auf das Gedächtnis möglich. Als eine mögliche Lösung für das Problem, bietet OMM+ einen speziellen Block an, der es erlaubt komplette Gedächtnisse als Nutzdaten in einem Block abzulegen (siehe Abbildung 5.12).

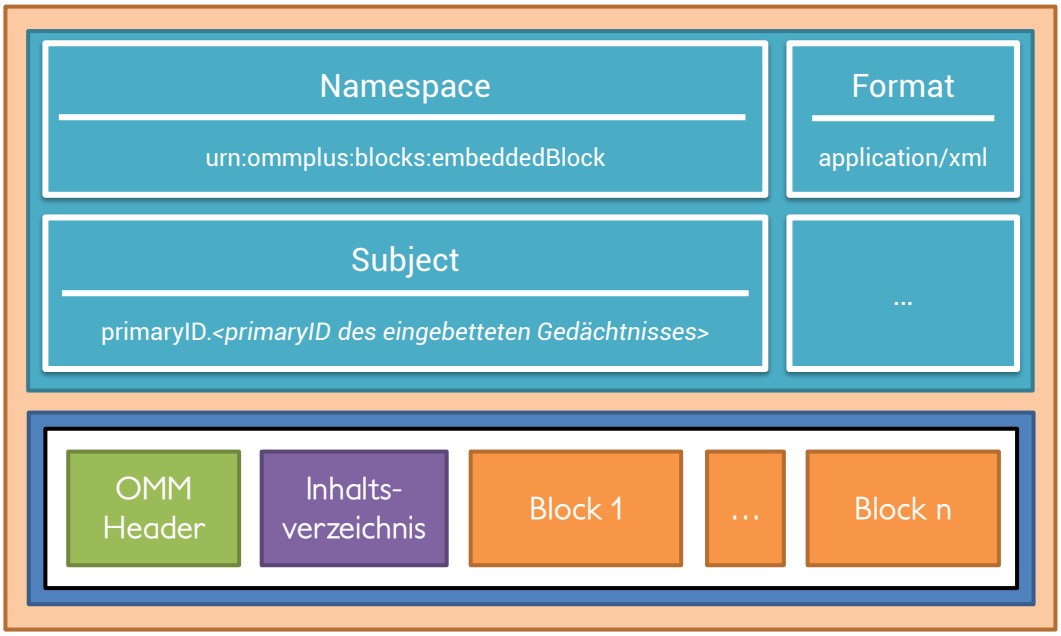



Abbildung 5.12: OMM+-Embedded-Block

Diese Vorgehensweise wird dadurch begünstigt, dass Gedächtnisse als XML-Datei repräsentiert werden können (siehe Kapitel 5.2.6) und somit ein Gedächtnis leicht in den Payload-Abschnitt eingebettet werden kann. Ein solcher Block kann durch den Namespace-Wert `urn:ommplus:blocks:embeddedBlock` erkannt werden. Zusätzlich wird die *primaryID* des eingebetteten Blocks als Schlagwort im Subject-Bereich des äußeren Blocks festgehalten. Dadurch können Anwendungen feststellen, zu welchem Objekt dieses Gedächtnis gehört. Siehe auch die weitere Abbildung 5.13 zum Verbau von Objekten. Ein XML-Beispiel eines solchen Blockes findet sich im Kapitel 5.2.6.

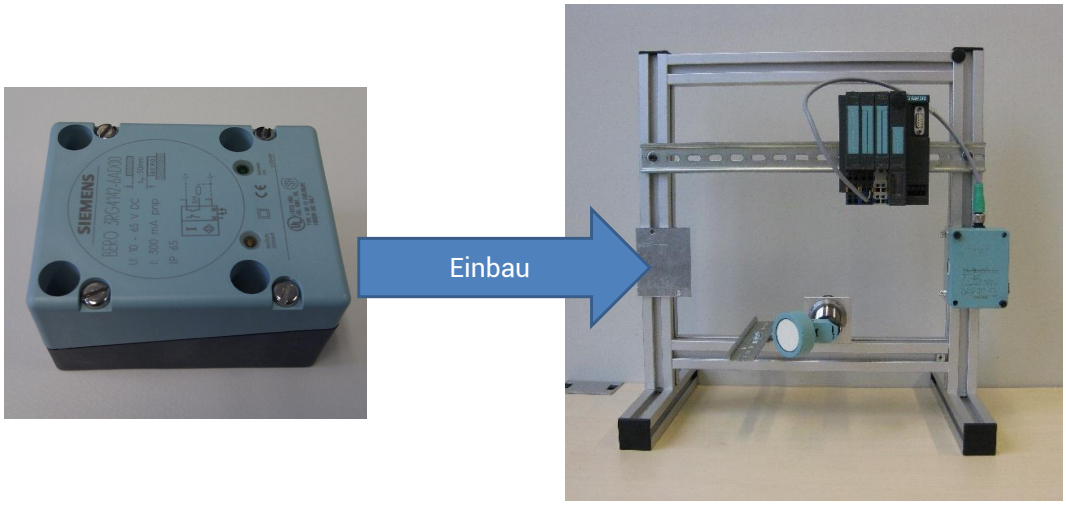



Abbildung 5.13: Verbau eines Objekts in einer Anlage

### PiVis-Plugin Block

Das Visualsierungsframework PiVis (siehe Kapitel 6.7.2) bietet die Möglichkeit, das Objekte selbst Plugins für die Visualisierung mitbringen. Zu diesem Zweck wurde ein eigener OMM+-Block definiert, so dass Plugins leicht im Gedächtnis abgelegt werden können. Da diese als Java-JAR-Datei vorliegen, kann die Datei direkt in den Nutzdatenabschnitt des Blocks eingelagert werden. Mit Hilfe eines Namespace wird der Block eindeutig gekennzeichnet und ein gesetztes Schlagwort vom Typ *datasource*, *filter*, oder *visualization* legt die Art des Plugins eindeutig fest (siehe Abbildung 5.14).

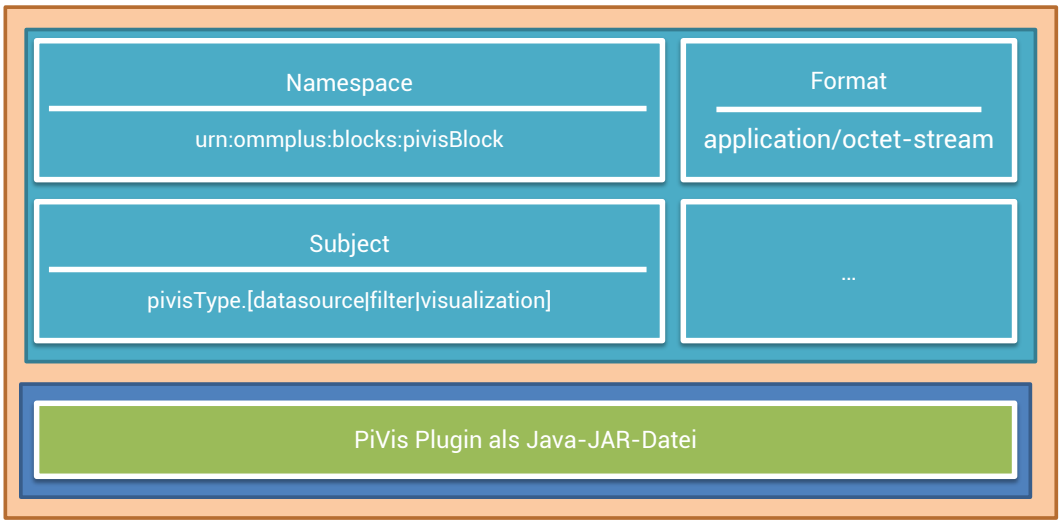



Abbildung 5.14: OMM+-PiVis-Block

### 5.2.6 XML-Repräsentation

Die bisher definierte abstrakte Darstellung des Object Memory Model (OMM) findet auch in konkreten Repräsentationen Anwendung. Die vom W3C bevorzugte und die im Web gebräuchlichste Form bildet XML[2] (Extensible Meta Language).

Dabei wird je ein Gedächtnis in einer XML-Datei gekapselt. Die dortige hierarchische Darstellung entspricht faktisch exakt dem abstrakten Modell. Auf oberster Ebene wird das Gedächtnis durch die XML-Tags `<omm:omm>...</omm:omm>` eingefasst. Das gesamte Dokument und alle Tags verwenden den OMM-Namensraum `http://www.w3.org/2005/Incubator/omm/elements/1.0/`. Auf der nächsten darunterliegenden Ebene finden sich die Einträge für den Omm-Header (`<omm:header>...</omm:header>`), sowie alle definierten OMM-Blöcke (je Block ein `<omm:block>...</omm:block>`-Abschnitt). Pro Block werden die Metadaten jeweils in eigenen Tags definiert. Die Nutzdaten (payload) werden ebenfalls als Wert eines eigenen Tags gespeichert. Dabei ist zu beachten, dass binäre Daten stets in ASCII-Zeichen kodiert werden müssen. Dazu bieten sich die Formate *Base64* [Jos06] oder UUEncode/MIME [BF93] an. Der jeweils genutzte Typ wird als Encoding-Attribut des Payload-Tags gesetzt. Quellcode 5.2 zeigt ein Beispielgedächtnis als XML-Datei.

**Quellcode 5.2: OMM-Darstellung als XML-Datei**

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <omm:omm xmlns:omm="http://www.w3.org/2005/Incubator/omm/elements/1.0/">
3    <omm:header>
4      <omm:version>1</omm:version>
5      <omm:primaryID omm:type="url">http://www.myOMM.org/samples/p1</omm:
           primaryID>
6      <omm:additionalBlocks omm:type="omm_http">http://www.myOMM.org/samples/p1/
           ext</omm:additionalBlocks>
7    </omm:header>
8
9    <omm:block omm:id="1">
10     <omm:namespace>http://www.myOMM.org/ns/sample</omm:namespace>
11     <omm:creation>
12       <omm:creator omm:type="duns">195505177</omm:creator>
13       <omm:date omm:encoding="ISO8601">2011-01-30T18:30:00+02:00</omm:date>
14     </omm:creation>
15     <omm:contribution>
16       <omm:contributor omm:type="email">user@dfki.de</omm:contributor>
17       <omm:date omm:encoding="ISO8601">2011-01-31T08:12:50+02:00</omm:date>
18     </omm:contribution>
19     <omm:title xml:lang="en">sample title</omm:title>
20     <omm:title xml:lang="de">Beispieltitel</omm:title>
21     <omm:description xml:lang="en">sample description</omm:description>
22     <omm:description xml:lang="de">Beispielbeschreibung</omm:description>
23     <omm:format>application/xml</omm:format>
```

[2] `http://www.w3.org/TR/REC-xml/` [Letzter Zugriff: 26.11.2012]

```
24      <omm:type>http://purl.org/dc/dcmitype/Dataset</omm:type>
25      <omm:link omm:type="url">http://www.myOMM.org/samples/p1/ext.xml</omm:link>
26      <omm:payload omm:encoding="base64">(...)</omm:payload>
27    </omm:block>
28
29  </omm:omm>
```

Für die direkt in OMM definierten Blöcke (IDs-Block, Struktur-Block, Schlüssel-Wert-Block) existiert ebenfalls eine XML-Repräsentation. Dazu werden die Daten dieser Blöcke ebenfalls als XML-Ausschnitt dargestellt und in den Playload der jeweiligen Blöcke eingefügt.

Aufgrund der Möglichkeit XML-repräsentierte Daten direkt als Wert eines Tags in einer anderen XML-Datei zu verwenden, kann der Fall des OMM+-Embedded-Block problemlos realisiert werden, in dem das gesamte Gedächtnis (<omm:omm>...</omm:omm>) zwischen die beiden Payload-Tags des entsprechenden Blocks eingefügt wird. Quellcode 5.3 zeigt ein entsprechendes Beispiel.

**Quellcode 5.3: Einbettung eines OMM-Gedächtnisses als Nutzdaten eines OMM-Blocks**

```
1  <omm:block omm:id="block_12346">
2    <!-- OTHER METADATA -->
3    <omm:namespace>urn:ommplus:block:embedded</omm:namespace>
4    <omm:format omm:encryption="none">application/xml</omm:format>
5    <omm:payload omm:encoding="none">
6      <omm:omm>
7        <omm:header>
8          <omm:version>1</omm:version>
9          <omm:primaryID omm:type="url">http://oms.sb.dfki.de/v2/oms/sample2</omm
               :primaryID>
10       </omm:header>
11       <omm:block omm:id="1">
12       (... sample block ...)
13       </omm:block>
14     </omm:omm>
15   </omm:payload>
16 </omm:block>
```

### 5.2.7 RDFa/Microdata-Repräsentation

Das bisher beschriebene XML-Repräsentationsformt für OMM-Gedächtnisse wird in der Regel ausschließlich zur internen Datenspeicherung beziehungsweise zum Datenaustausch verwendet. Um eine für den Benutzer leichter lesbare Darstellung zu erreichen und gleichzeitig auf die maschinenlesbaren semantischen Informationen nicht zu verzichten, wurde eine zusätzliche Darstellung in Form von RDFa [AHSB12] und Microdata [Mic12] erstellt. Beide Formate haben dabei die grundlegende Idee gemeinsam, semantische Informationen

in HTML-Seiten zu integrieren. Dies erlaubt es eine lesbare Darstellung für den Menschen und die Maschine in einer einzigen Datei zu generieren. Im Fall OMM wird dabei eine HTML-Darstellung des Gedächtnisses erstellt, welches die Inhalte in Form von Aufzählungen darstellt und gleichzeitig der Inhalt des Gedächtnisses in die jeweiligen RFDa- und Microdata-Tags verpackt. Somit ist eine verlustlose Übertragung des gesamten Gedächtnisses in Form von gewöhnlichen HTML-Dateien möglich. Abbildung 5.15 zeigt die HTML-Darstellung eines RFDa-Ausschnitts im Webbrowser. Eine Ausführliche Darstellung beider Formate findet sich im Anhang.

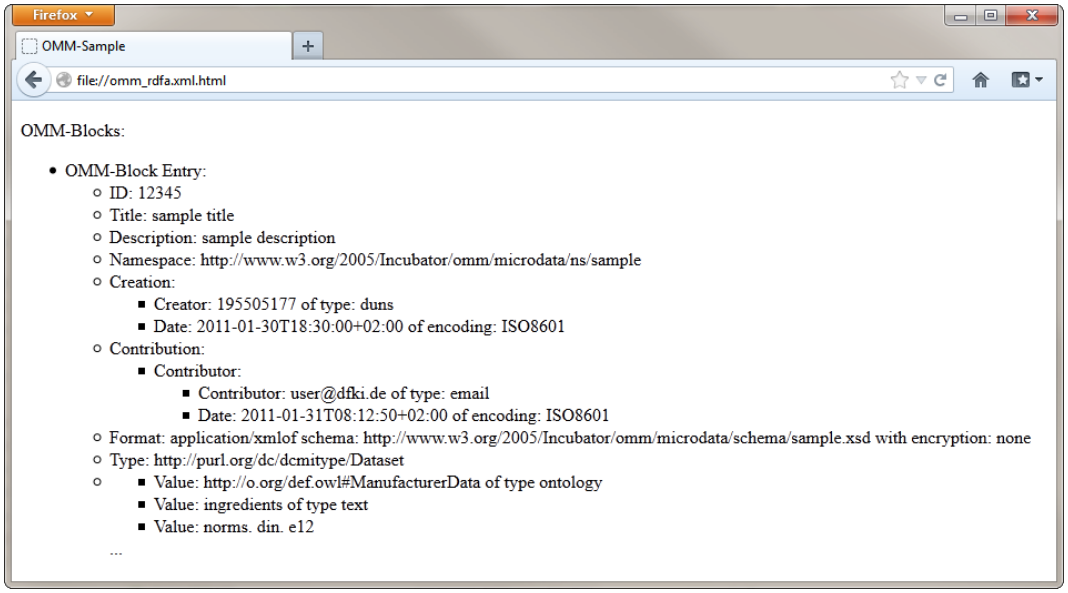



Abbildung 5.15: HTML-Darstellung von RDFa-OMM-Gedächtnissen

### 5.2.8 Implementierung (libOMM)

Basierend auf den Funktionalitäten und Möglichkeiten des Object Memory Models wurde im Rahmen dieser Arbeit eine Referenzimplementierung erstellt. Dabei handelt es sich um eine für die Java 1.6 Plattform bestimmte Bibliothek mit dem Namen *libOMM*. Diese Bibliothek bietet dem Benutzer eine API (Application Programming Interface), mit deren Hilfe sich OMM-Gedächtnisse verwalten lassen. Die einzelnen OMM-Konzepte wie Gedächtnisse, Blöcke, Meta- und Nutzdaten wurden im Sinne einer objektorientierten Programmierung als eigene Klassen umgesetzt. Jede dieser Klassen bietet Operationen, um die jeweiligen Objekte modifizieren zu können. Zusätzlich bietet die Bibliothek die Möglichkeit der Datenausgabe in Form von XML-Daten, welche mit Hilfe der Bibliothek *JDOM2*[3] ermöglicht wird. Des Weiteren lassen sich mit Hilfe zusätzlicher Module (in Form von weiteren Java Archiven)

[3] `http://www.jdom.org/` [Letzter Zugriff: 26.11.2012)]

die Ausgabe in den Formaten RDFa/Microdata und Binär ergänzen. Zur Integration in komplexere Systeme kann diese Bibliothek ein Ereignis bereitstellen, sobald Teile des Gedächtnis von der Anwendung verändert werden. Auf diese Ereignisse können externe Anwendungen reagieren und zusätzliche Funktionalität bereitstellen. Dies wird bereits im Object Memory Server (siehe folgendes Kapitel 6.5) genutzt.

## 5.3 Ontologie für Produktdaten

Um mit Hilfe des Ontologieeditors Leo (siehe Kapitel 6.3) erste OWL-Dateien mit Instanzen zu Produktdaten erstellen zu können wurde eine Basis-Ontologie mit Konzepten zur Modellierung von Lebensmittelprodukten erstellt. Die Konzepte wurden dabei so generisch gehalten, dass sich viele verschiedene Produkte aus dem Bereich von Endverbrauchern modellieren lassen. Beispielsweise wurde das Anwendungsfeld in einem zweiten Schritt auf Oberbekleidung ausgedehnt.

Um eine Integration und Datenabbildung mit anderen Ontolgien zu erleichtern, wurden alle definierten Konzepte von Konzepten der SUMO[4]-Ontologie abgeleitet. Zentraler Kern stellt die Klasse `ManufacturerData` dar, welches alle Informationen enthält, die bereits zur Entwicklungsphase des Produkts bekannt sind und spätestens bei der Produktion vom Hersteller in das Gedächtnis des Objekts geschrieben werden können. Ausgangspunkt bilden die Informationen, die bereits heute auf der Packung abgedruckt sind (bzw. je nach rechtlichen Vorgaben auch abgedruckt sein müssen). Folgende Liste zeigt einen kleinen Auszug aus den möglichen Relationen die mit Instanzen versehen und somit mit Daten belegt werden können:

- *hasManufacturerDescription*: beinhaltet Informationen über den Hersteller selbst (zum Beispiel Name, Adresse und Geschäftsform).
- *hasImageAttributeCollection*: eine einfache flache Liste, die Bilder aufnehmen kann, die sich in keiner andere Kategorie verorten lassen.
- *hasStringAttribute*: eine flache Liste an Text-Strings, die sich an keiner anderen Stelle zuordnen lassen.
- *hasProductCode*: eine Angabe zur Klasse des jeweiligen Produkts. In der Regel finden sich hier Daten, die auch mit Hilfe eines konventionellen 1D-Barcodes auf der Verpackung abgedruckt sind. Beispiele für solche IDs sind EAN-8 oder EAN-13 bzw. GS1.

[4] `http://www.ontologyportal.org/` [Letzter Zugriff: 26.11.2012]

- *hasPackage*: detailliertere Informationen zur Verpackung des Objekts. Diese Gruppe beinhaltet sowohl Daten zu den physischen Dimensionen (Länge, Breite, Höhe, Gewicht) als auch Angaben zu Texten und Bildern, die auf der Verpackung abgedruckt sind.

- *hasComponentFactCollection*: beschreibt Informationen über den Inhalt einer Packung. Da die Ontologie ursprünglich für Lebensmittel entwickelt wurde, sind hier die Inhaltsstoffe aufgelistet. Ebenso wird diese Angabe für die Bestandteile von Kleidungsstücken verwendet.

- *hasNutritionFactCollection*: umfasst Daten zu Nährwertangaben (zum Beispiel Kohlenhydrate, Fett, Proteine usw.) falls es sich um Lebensmittel handelt.

- *hasStorageHintCollection*: beinhaltet Hinweise zur Lagerung des Objekts. Dies ist besonders bei Produkten von Bedeutung, die zum Beispiel bestimmte Temperaturen oder Luftfeuchtigkeitswerte voraussetzen.

- *hasPreparationHintCollection*: fasst Hinweise zur Zubereitung von Lebensmitteln. Diese Information kann zum Beispiel direkt in einer instrumentierten Küche verwendet werden, um den Benutzer bei der Zubereitung zu unterstützen.

- *hasRelevantSensor*: dieses Feld kann Informationen aufnehmen, welche Sensorwerte in der Umgebung des Objekts gemessen werden sollten, da diese direkten Einfluss auf die Qualität oder die Integrität des Produktes haben. Teilweise kann die Information auch aus der StorageHintCollection abgeleitet werden. Dieses Feld bietet aber zusätzlich umfassendere Angaben.

Bei der Erstellung der Ontologien wurde ein hoher Grad an Modularisierung erreicht, so dass es möglich ist je nach Anwendungsdomäne unterschiedliche Teile des Gesamtmodells im Sinne eines Baukastensystems zu verwenden. Die Maschinenlesbarkeit der ontologisch modellierten Daten erlaubt einige zusätzliche Mehrwerte im Vergleich zur klassischen aufgedruckten Information, so dass zusätzlich zu den explizit definierten Daten weitere Informationen durch Reasoning automatisch ermittelt werden können (siehe auch [MK11]).

## 5.4 Ontologie für Personalisierung

Der zweite Bereich der Ontologiemodellierung befasst sich mit der Abbildung von Daten und Eigenschaften von Benutzern, die Systeme benötigen um Mehrwertdienste anbieten zu können, die auf einzelne Benutzer zugeschnitten sind. Dabei stehen hier, im Gegensatz zu anderen vergleichbaren Ansätzen zum Beispiel aus der Fahrzeugdomäne [FM11, Fel11], Anwendungen im Bereich der Benutzerinteraktion mit Objektgedächtnissen im Vordergrund.

Zu diesem Zweck stehen drei unterschiedliche Möglichkeiten zur Verfügung: (1) eine Erfassung von generischen Benutzerfakten, (2) Fakten, die über einfache Listen definiert werden und (3) gewichtete Präferenzen des Benutzers (siehe Abbildung 5.16).

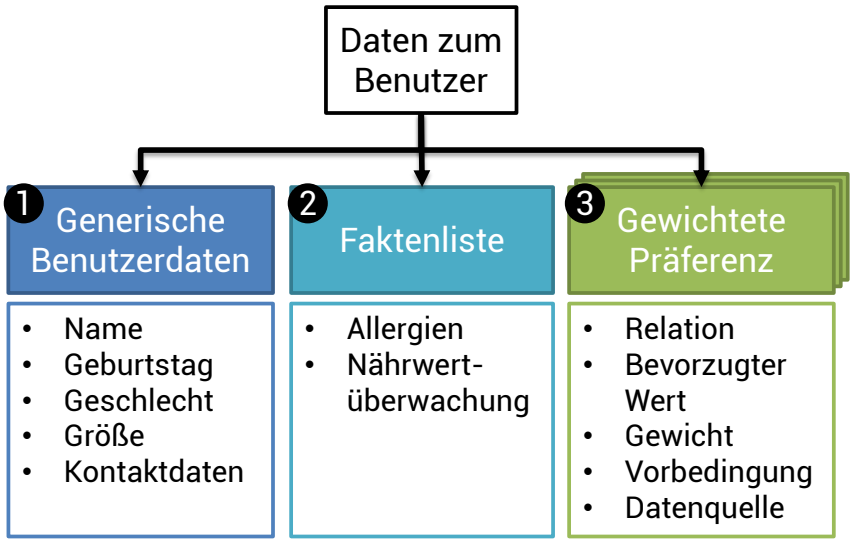



Abbildung 5.16: Ontologische Modellierung von Benutzerdaten

Generische Benutzerdaten enthalten die klassischen Benutzerinformationen wie Name, Geburtstag, Geschlecht, Kontaktdaten und Körpergröße. Die Faktenliste bietet eine einfache und flache Struktur um fixe Daten abzulegen. Dies wird exemplarisch genutzt um zum einen Allergien des Benutzers zu definieren, zum Beispiel:

$$Benutzer_A \overset{reagiertAllergischAuf}{\rightarrow} Stoff_{Xanthan}$$

Zum anderen wurde exemplarisch eine Datenhaltung zur Nährwertüberwachung realisiert, mit dessen Hilfe der Benutzer festlegen kann, welche Menge an Nährwerten er in ein einem Zeitraum konsumieren möchte; zum Beispiel eine maximale Energieaufnahme von 10500 Kilojoule pro Tag.

Die größte funktionale Mächtigkeit hat das dritte Konzept der gewichteten Präferenzen. Ausgangspunkt dieses Ansatzes ist die Idee, dass der Benutzer bezüglich einer Relation einen bevorzugten Wert definieren kann. Eine Relation kann dabei jede OWL-Relation darstellen. Der Ausgangspunkt der Relation (Subjekt) ist immer der Benutzer selbst. Das Ziel der Relation (Objekt) ist dabei der bevorzugte Wert, das heißt der Benutzer macht zu dieser Relation mit dem definierten Objekt eine positive oder negative Aussage. Möchte der Benutzer beispielsweise keine Pizza, die mit Salami belegt ist, wird er eine Präferenz erstellen, die aus der OWL-Relation `hasIngredient` und dem OWL-Konzept `#Salami` als Subjekt besteht. In dieser Ausprägung sagt diese Präferenz aus, dass der Benutzer nur Pizzen mit Salami möchte. Um diese Aussage abzuschwächen oder zu negieren, kommt

das Gewicht-Attribut ins Spiel. Diese besteht aus einer Ganzzahl aus dem Intervall $[-1;1]$. Die Werte 1 und $-1$ definieren absolute Aussagen, das heißt die Präferenz muss bei 1 immer erfüllt sein und darf bei $-1$ niemals erfüllt sein. Werte dazwischen definieren entsprechend abgeschwächte Präferenzen, die dann entscheidend sind, wenn mehrere Präferenzen gegeneinander abgewogen werden müssen. Die Angabe eines Gewichtes der Größenordnung 0 setzt diese Präferenz außer Kraft, ist also nur formal möglich und hat praktisch keine Relevanz.

Optional ist es für jede Präferenz zusätzlich möglich eine sogenannte Datenquelle anzugeben. Diese wird ebenfalls als ontologisches Konzept dargestellt und hat die Funktion einer Anwendung zu ermöglichen festzustellen, wer diese Präferenz definiert hat. Diese Information kann dem Benutzer weiterhelfen, Präferenzen, die von automatischen Systemen angelegt werden, von den eigenen selbst definierten Aussagen zu unterscheiden.

In Analogie zur ähnlichen Umsetzung bei Planungssystemen kann für jede Präferenz eine Menge an Vorbedingungen definiert werden. Diese müssen zuerst vollständig erfüllt sein, bevor die Präferenz ausgewertet wird. Dies erlaubt es auf der einen Seite gewisse Regeln kontextabhängig zu definieren und wirken zu lassen. Auf der anderen Seite können mit Vorbedingungen zusätzliche Relationen in die eigentliche Aussage einer Präferenz mitaufgenommen werden. Formal betrachtet stellt jede Vorbedingung selbst wieder eine gewichtete Präferenz dar. Dadurch können Präferenzen auch rekursiv formuliert werden, da die Vorbedingung selbst wieder Vorbedingungen benutzen kann. Der Algorithmus, der solche Präferenzen auswertet, muss dabei sicherstellen, dass zum einen keine Endlosschleifen definiert werden und zum anderen Präferenzen durch ihre Vorbedingungen keinen inkonsistenten Zustand erreichen können, in dem zum Beispiel die Vorbedingung die eigentliche Präferenz negiert.

Durch die konsistente semantische Repräsentation von Präferenzen mit der zusätzlichen Angabe von Gewicht und Quelle der jeweiligen Aussage, werden Anwendungen in die Lage versetzt zum einen auf die Wünsche des Benutzers einzugehen und zum anderen dem Benutzer eine Erklärung anbieten zu können, auf welchen Regeln und Präferenzen die Entscheidungen des Systems beruhen, wodurch solche System transparenter agieren und das Vertrauen des Benutzers in solche Systeme gestärkt werden kann.

### Beispielszenario

Abbildung 5.17 zeigt ein beispielhaftes Szenario, bei dem ein Benutzer eine Tiefkühlpizza aus dem Kühlschrank nimmt und eine Warnung erhält, dass er eine Unverträglichkeit zu diesem Produkt hat und den Konsum vermeiden sollte. Sowohl die Pizza als auch der Benutzer besitzen ontologisch modellierte Daten und der Kühlschrank ist im Sinne einer instrumentierten Umgebung mit Sensorik und Display ausgestattet. In diesem Szenario werden vier unterschiedliche Ontologie-Module verwendet: Die Beschreibung der Tiefkühlpizza samt

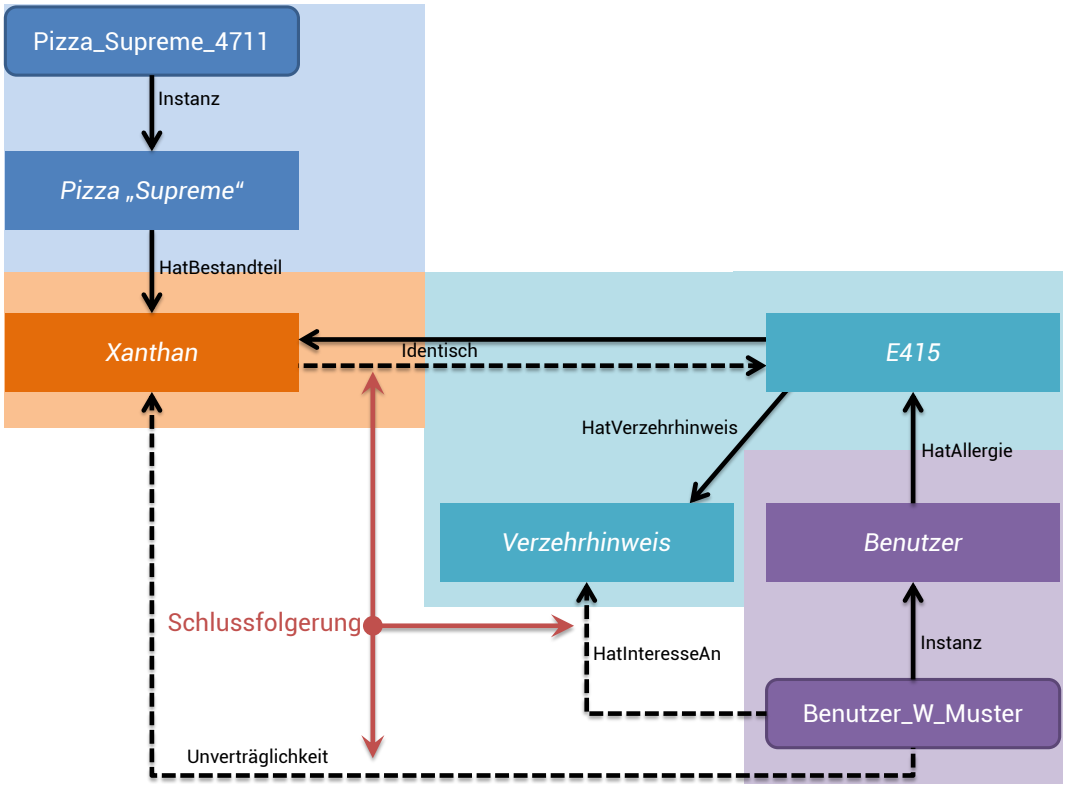



Abbildung 5.17: Atomatisches Schließen mit Hilfe der Produktontologie

ihrer konkreten Instanz (blau). Eine Darstellung von möglichen Inhaltsstoffen (orange). Die Daten und Eigenschaften des Benutzers (violett) und eine Abbildung von Inhaltsstoffen zu E-Nummern inklusive zusätzlicher Hinweise zu einzelnen Stoffen (cyan). Obwohl es aus Sicht der Pizzadaten (inklusive Inhaltsstoffen) keine Verbindung zu den Benutzerdaten gibt, kann durch die Bijektivität der Abbildung von E415 zu Xanthan automatisch geschlossen werden, dass Xanthan auch identisch zu E415 ist. Dadurch kann als nächster Schritt eine Unverträglichkeit des Benutzers zu Xanthan abgeleitet werden, welches als Inhaltsstoff der Pizza vermerkt ist. Darüber hinausgehend kann durch Transitivität und der Aussage, dass der Benutzer allergisch auf E415 reagiert, geschlossen werden, dass der Benutzer an den Verzehrhinweisen bezüglich E415 interessiert ist.

## 5.5 Regelbasierte Daten

Neben den bereits erwähnten statischen oder dynamisch ergänzten und teilweise semantisch annotierten Objektinformationen, bieten sich Objektgedächtnisse auch an um Verarbeitungslogik mitzuliefern, die direkt in Bezug zum jeweiligen Objekt stehen. Diese ermöglicht

es Anwendungen auf die Anforderungen gewisser Produkte einzugehen, ohne direkt an diese angepasst werden zu müssen, wodurch der offene Ansatz der Objektgedächtnisse unterstützt wird.

Der hier vorgestellte Ansatz wird als *Objektregeln* bezeichnet und nutzt das Regelsystem Drools[5], um solche regelbasierte Daten in der Anwendungsschicht auszuwerten [SVH10]. Eine konkrete Objektregel wird durch das Tupel $(C, R, F, D)$ beschrieben, wobei $C$ den aktuellen Kontext definiert, $R$ eine Übergangsregel darstellt, $F$ die aktuellen Fakten repräsentiert und $D$ die bereits erwähnte Regel- und Inferenzengine symbolisiert. Zusätzlich existieren zwei Funktionen $r(c)$ und $f(c)$, die Daten (Funktion $r$) beziehungsweise die Regeln (Funktion $f$) aus dem Gedächtnis lesen können. Dadurch lassen sich zwei unterschiedliche Übergänge definieren.

Im ersten Fall ändert sich der Kontext $C$ dergestalt, dass eine neue Informationseinheit $c \in C$ auftritt. Dadurch wird ein neues Tupel $(C', R', F', D')$ mit folgender Logik generiert: $C' = C \cup \{c\}$, $R' = R \cup r(c)$ und $F' = F \cup f(c)$. Schließlich werden $(R' \setminus R)$ und $(F' \setminus F)$ zur Regelbasis $D$ hinzugefügt.

Im zweiten Fall wird eine Informationseinheit $c$ aus $C$ entfernt. In diesem Fall wird ebenfalls ein neues Tupel $(C', R', F', D')$ generiert, allerdings nun mit folgender Logik: $C' = C \setminus \{c\}$, $R' = \bigcup_{c' \in C} r(c')$ und $F' = \bigcup_{c' \in C} f(c')$. Anschließend werden $(R' \setminus R)$ und $(F' \setminus F)$ aus der Regelbasis $D$ entfernt.

Eine beispielhafte Implementierung wurde anhand eines Medikamentenwechselwirkungsszenarios realisiert (siehe auch Kapitel 7.2.2). Hierbei sind sowohl Medikamente als auch sonstige Lebensmittel mit digitalen Objektgedächtnissen ausgestattet. Beide Typen an Produkten besitzen Informationen über ihre Inhaltsstoffe; die Medikamente zusätzlich auch Objektregeln, die Unverträglichkeiten mit anderen Medikamenten oder Lebensmitteln definieren. Beispielsweise enthält ein Medikament die Regel `contains(X, substance-118)` → `incompatible(drug-47, X)`, welche folgende Aussage ausdrückt: falls ein Objekt `X` aus `substance-118` besteht, dann ist es inkompatibel mit dem Medikament `drug-47`. Die Funktion `incompatible` löst dabei zum Beispiel eine festgelegte Reaktion aus, die in der zusätzlichen Regel `incompatible(x,y)` → `showWarningToUser(Nehmen Sie 'X' und 'Y' getrennt ein!")` festgelegt ist; in diesem Fall eine Warnung an den Benutzer bezüglich der Wechselwirkung.

## 5.6 Fazit

In diesem Kapitel wurde das Object Memory Model (OMM) vorgestellt, welches im Rahmen dieser Arbeit verwendet wird, um eine Strukturierung der Objektgedächtnisdaten

[5] `https://www.jboss.org/drools/` [Letzter Zugriff: 26.11.2012]

zu ermöglichen. Zu diesem Zweck wurden die Bestandteile des OMM, insbesondere die einzelnen Metadaten und die Ergänzung von Nutzdaten, sowohl mit direkter Ablage im Block als auch in Form von verlinkten und extern abgelegten Daten erläutert. Des Weiteren wurden die bereits fest definierten Blöcke und deren Anwendungsgebiete vorgestellt sowie die möglichen Repräsentationen des Modells in Form von XML und RFDa/Microdata-Daten dargelegt. Zur Nutzung dieses Modells wurde eine Referenzimplementierung erstellt (libOMM), welche ebenfalls vorgestellt wurde. Als Ergänzung zum Modell wurden zwei entwickelte Ontologien zur Ablage von Produkt- und Benutzerdaten, inklusive der dazugehörigen Anwendungsszenarien, präsentiert. Schließlich wurde gezeigt, in welcher Art und Weise regelbasierte Daten in Objektgedächtnisse eingebunden werden können.

Mit Hilfe dieses Modells können nun die im Kapitel 2.9.1 aufgestellten Anforderungen erfüllt werden. Mit Hilfe von eindeutig definierten Blöcken mit Namespaces und durch die Angabe von semantischen Konzepten (Subjects) in den Metadaten ist das maschinelle Auffinden von Daten sehr leicht möglich. Zusätzlich können zum Beispiel durch den OMM+-Semantik Block beliebige semantische Relationen in Gedächtnisse integriert werden. Somit ist für kleine Anwendungsfälle eine leichtgewichtige Lösung möglich, während die beschriebenen Produktdaten- und Benutzerontologien für komplexere Szenarien genutzt werden können, um semantische Strukturen zu nutzen ($R_D1$).

Alle im OMM-Strukturmodell verwendeten Konzepte und Formate (zum Beispiel XML, RDFa/Microdata, OWL) basieren auf Techniken des World Wide Web und sind durch Empfehlungen des W3C standardisiert ($R_D2$). Des Weiteren sind alle Strukturen auf eine offene und flexible Nutzung ausgerichtet, so dass beliebige Daten in das Strukturmodell abgelegt werden können. Durch die Vergabemöglichkeit von beliebigen Annotationen in Form von Metadaten kann das Domänen- und Anwendungs-übergreifend genutzt werden ($R_D3$). Die Metadaten erlauben es jedoch auch in solchen heterogenen Datensammlungen sowohl durch den Menschen als auch durch die Maschine eine effiziente Suche nach den gewünschten Daten durchführen und sicher erkennen zu können ($R_D4$).

KAPITEL 6

# OMM-Werkzeuge

## 6.1 Einleitung

Im vorangegangenen Kapitel wurde das OMM-Strukturmodell zur Ablage von Daten in Objektgedächtnissen vorgestellt. Dieses Modell bildet die Grundlage für alle weiteren Entwicklungen. In diesem Kapitel werden nun zusätzliche Werkzeuge vorgestellt, die es Entwicklern ermöglichen in Form eines Baukastens eigene Infrastrukturen und Anwendungen für Objektgedächtnisse aufsetzen zu können.

Diese Werkzeuge umfassen dabei den Bereich der Datengenerierung. Kapitel 6.2 zeigt dabei eine Methode zur Konvertierung von bestehenden Datenbank-basierten Daten zur Objektgedächtnissen. Mit Hilfe des leichtgewichtigen Ontologieeditors Leo lassen sich auch von Nicht-Experten semantische Daten in Form von Ontologien mit Daten füllen (Kapitel 6.3). Schließlich bietet der in Kapitel 6.4 vorgestellte Datenvalidator die Möglichkeit alle Objektgedächtnisdaten, die mit den hier vorgestellten Werkzeugen als auch solche die von externen Quellen vorliegen, auf syntaktische und semantische Wohlgeformtheit zu überprüfen.

Der zweite große Bereich bildet die Server-basierte Speicherung von Objektgedächtnisdaten. Kapitel 6.5 führt den Objekt Memory Server (OMS) ein, der diese Aufgabe in der DOMeMan-Architektur übernimmt. Dabei wird zunächst eine funktionale Übersicht aller Bestandteile des OMS präsentiert. Anschließend werden Schnittstellen zur Kommunikation von externen Anwendungen mit dem OMS gezeigt. Anschließend werden die Funktionen der Versionsverwaltung von Gedächtnissen und der Rechte- und Rolle-basierte Zugriff erläutert und die Implementierungsarbeiten belegt. Abschließend werden die Möglichkeiten des OMS zur Bereitstellung von aktiven Komponenten für Objektgedächtnisse eingeführt.

Der Bereich der Datenvisualisierung (Kapitel 6.7) bildet den Abschluss dieses Kapitels. Darin wird das modulare und Plugin-basierte Visualiserungsframework PiVis und eine Android-basierte App zum mobilen Zugriff vorgestellt.

## 6.2 Konvertierung von DB-basierten Informationen

Betrachtet man den Stand der Technik und die heute genutzten Systeme im Bereich der Datenhaltung bei einzelnen Partnern der Produktlebenszykluskette, so stellt man fest, dass diese in der Regel Datenbanken (DB) nutzen, um ihre Daten effizient ablegen zu können. Werden die Daten derart organisiert, ist eine direkt Nutzung im Sinne der digitalen Objektgedächtnisse nicht direkt möglich. Sollen allerdings Objektgedächtnisse erzeugt werden, stellt sich die Frage wie eine einfache und nutzbringende Migration stattfinden kann. Da in der Regel in einem ersten Schritt weiterhin die bestehende Infrastruktur genutzt werden soll, ist ein Prozess notwendig, der die Daten aus dem bestehenden System in eine Gedächtnisstruktur überführt. Aus Sicht des Betreibers ist es darüber hinaus wünschenswert, dass diese Konvertierung einmalig definiert wird und im Folgenden für jedes Produkt automatisch abläuft. Zu diesem Zweck wird im Folgenden ein Konzept vorgestellt, welches in der Lage ist bestehenden Daten aus unterschiedlichen Quellen mit Hilfe eines fixen Prozesses in Objektgedächtnisse zu überführen. Abbildung 6.1 zeigt die verwendete konzeptuelle Pipeline der Generierung von Objektgedächtnisdaten. Dazu werden die Daten aus den ursprünglichen Quellen abgerufen, semantisch harmonisiert, in Module separiert und anschließend in einzelne Blöcke des Gedächtnisses abgelegt (zum Aufbau von Objektgedächtnissen siehe Kapitel 5). Im Folgenden werden nun die einzelnen Bestandteile dieser Datenverarbeitung im Detail beschrieben.

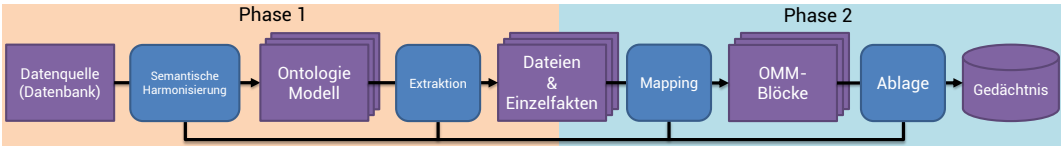



Abbildung 6.1: Werkzeugkette zur Konvertierung von Datenbank-basierten Informationen in Objektgedächtnisse

**Semantische Harmonisierung und Datenextraktion** Grundlage jeder weiteren Verarbeitung ist die semantische Harmonisierung. Diese dient dem Zweck, die bisher vorliegenden Daten (zum Beispiel in Form von relationalen Datenbanken) in ein semantisches Modell (zum Beispiel OWL [MvH03]) zu überführen. Durch die Verwendung eines solchen Modells steht eine klar strukturierte semantische Darstellung der abzulegenden Daten bereit, welches es auch erlaubt, Daten aus unterschiedlichen Quellen mit unterschiedlichem Umfang und Qualität an einem Ort zusammenfassen zu können. In der ersten Phase der Konvertierung erfolgt nun die eigentliche Harmonisierung mit Hilfe von sogenannten Diensten, welche die Aufgabe haben, einzelne Fakten aus unterschiedlichen Datenquellen auszulesen, wenn notwendig zu konvertieren und diese dann an die passende Stelle im Modell abzulegen (siehe auch [BBH+09]). Jeder Dienst besteht dabei aus einem Client, der

die Daten aus der ursprünglichen Datenquelle abruft. Dies kann bei relationalen Datenbanken zum Beispiel mit Hilfe einer SQL-Anfrage [ISO11] oder bei semantischen Datenbanken mit Hilfe einer SPARQL-Anfrage [ACD+12] erfolgen. Anschließend wird mit Hilfe von Abbildungsregeln ein Mapping durchgeführt und die Daten aus dem Client mit Hilfe von Instanzen im semantischen Modell abgelegt. Dieses Modell kann entweder monolithisch aufgebaut sein, das heißt alle Informationen sind in einem großem Modell erfasst. Alternativ kann auch eine Menge von kleineren semantisch abgeschlossenen Modellen genutzt werden womit erst die Fusion aller Modelle eine Gesamtdarstellung erzeugt. In beiden Fällen erfolgt in einem zweiten Schritt die Extraktion der nun harmonisiert vorliegenden Daten in einzelne Dateien (zum Beispiel OWL-Dateien) und einzelne Fakten (zum Beispiel ein einzelner Messwert).

**OMM-Mapping und -Ablage** In der zweiten Phase werden nun die einzelnen Dateien und Fakten in ein Objektgedächtnis überführt. Zu diesem Zweck legt ein „Mapper“ die Dateien und Fakten in einzelne OMM-Blöcke ab. Zusätzlich werden nach fest definierten Regeln die Metadaten der einzelnen Blöcke ergänzt. Dabei können die Metadaten sowohl statisch (in Form einer Regel) festgelegt werden oder alternativ mit Hilfe von Variablen (je nach abgelegter Information) berechnet werden. Zusätzlich ist es je nach Wert der einzelnen Daten möglich, diese gar nicht im Gedächtnis abzulegen oder zusätzliche Daten zu generieren, die in der ursprünglichen Quelle nicht oder nicht in dieser Form vorhanden war. Somit lassen sich die Daten auch während der Konvertierung mit zusätzlichen Informationen anreichern. Als letzter Schritt wird nun das passende Gedächtnis instanziiert und mit den erzeugten Blöcken samt Metadaten und Nutzinhalt befüllt, so dass die Daten aus einer oder mehreren bestehenden Datenquellen mit Hilfe eines automatischen Prozesses direkt in Objektgedächtnisse überführt werden können, die dann von den folgenden Partnern in der Produktlebenszykluskette genutzt werden können.

**Umsetzung** Für den Bereich der Phase 1 können bereits existierende Frameworks genutzt werden, die es erlauben die semantische Abbildung durchführen zu können. Arbeiten in diesem Bereich wurden beispielsweise forschungsorientiert im Projekt Theseus[1] umgesetzt oder stehen alternativ kommerziell zum Beispiel mit dem Produkt *OntoBroker* der Firma *semafora systems*[2] (früher Ontoprise) zur Verfügung. Je nach verwendetem Framework, gestaltet sich die Umsetzung und Realisierung dieser Pipeline unterschiedlich. Prinzipiell sind zwei unterschiedliche Verfahrensweisen üblich. Zum einen lassen sich die Dienste, welche die semantische Harmonisierung und Extraktion durchführen, direkt als Softwaremodule realisieren. Dieses Vorgehen bietet sich insbesondere bei vielen speziellen Konvertierungsschritten mit hoher Komplexität an. Zum anderen können auch generische Softwaremodule genutzt werden, die mit Hilfe von Konfigurationsdateien auf die einzelnen

[1] http://www.theseus-programm.de/ [Letzter Zugriff: 26.11.2012]
[2] http://www.semafora-systems.com [Letzter Zugriff: 26.11.2012]

Vorgänge zugeschnitten werden, wodurch ein geringerer Umsetzungsaufwand entsteht. Für die Phase 2 kann ein einfaches konfigurierbares Skript genutzt werden, welches mit Hilfe einer Konfigurationsdatei die einzelnen Daten in OMM-Strukturen abbilden kann. Quellcode 6.1 zeigt eine Beispielkonfiguration, die für eine Eingabedatei einen OMM-Block anlegt. Der Datei wird im <input>-Abschnitt einen Variablennamen zugewiesen, welches anschließend im <payload>-Bereich verwendet werden kann. Des Weiteren wird im Beispiel gezeigt, wie mit Hilfe von Variablen (@@NOW_ISO8601@@ für das aktuelle Datum im ISO8601-Format) dynamische Daten ergänzt werden können. Quellecode 6.2 zeigt eine Variante, die keine Datei sondern nur einen einfachen Wert im OMM-Block ablegt.

**Quellcode 6.1: OMM-Mapper Konfiguration mit Datei-basiertem Block**

```
<rule>
  <input type="file">_MANUFACTURER_DATA_</input>
  <output>
    <metadata>
      <omm:namespace>http://sample.org/MD</omm:namespace>
        <omm:creation>
          <omm:creator omm:type="duns">@@DUNS@@</omm:creator>
          <omm:date omm:encoding="ISO8601">@@NOW_ISO8601@@</omm:date>
        </omm:creation>
     <omm:format>@@MIME_TYPE@@</omm:format>
     <omm:type>http://purl.org/dc/dcmitype/Dataset</omm:type>
     <!-- ... -->
    </metadata>
    <payload>_MANUFACTURER_DATA_</payload>
  </output>
</rule>
```

**Quellcode 6.2: OMM-Mapper Konfiguration mit Werte-Block**

```
<rule>
  <input type="string">_BEST_BEFORE_DATE_</input>
  <output>
    <metadata>
      <!-- ... -->
    </metadata>
    <payload>_BEST_BEFORE_DATE_</payload>
  </output>
</rule>
```

## 6.3 Ontologie-Editor (Leo)

In einer frühen Phase der Integration von digitalen Objektgedächtnissen beziehungsweise der Migration zum Einsatz von Objektgedächtnissen liegen in der Regel noch keine

semantisch annotierten Daten vor oder es ist nicht möglich solche semantischen Daten automatisch aus bestehenden Systemen abzuleiten. Daher ist es für diesen Zweck vorteilhaft ein Werkzeug nutzen zu können, dass einen Benutzer bei der Aufgabe unterstützt vorhandene Daten in ein semantisches Modell zu überführen. Die Anforderungen liegen bei einem solchen Tool verstärkt auf der Methoden der Dateneingaben und daher bei der der Nutzung von zum Beispiel OWL-modellierten semantischen Daten auf dem Prozess des Instanziierens von Klassen und Relationen. Es wird aus diesem Grund davon ausgegangen, dass die Definition einer Hierarchie von Konzepten und Relationen bereits einmalig von einem Experten mit entsprechenden Werkzeugen generiert wurde (siehe Kapitel 3.5) und sich nun der Benutzer ausschließlich auf die Übertragung existierender Daten in diese Modelle konzentriert.

Zu diesem Zweck wurde ein eigenes Werkzeug entwickelt (siehe [Sch11]). Der *Lightweight Ontology Editor* (LEO) wurde darauf ausgelegt den Benutzer die Dateneingabe so einfach wie möglich zu gestalten und ihn bei der Eingabe mit Hinweisen zu unterstützen. Um die Bearbeitung auch durch Nicht-Experten schnell und effizient durchführen zu können, sollen die Komplexität und die Mächtigkeit des als Basis dienenden ontologischen Modells weitgehend vor dem Benutzer verborgen werden, ergänzt durch die Fokussierung auf die eigentliche Aufgabe. Ziel ist es, den Benutzer einen festen Startpunkt innerhalb der Ontologie vorzugeben, von dessen Lage sich der Benutzer mit Hilfe von Relationen eigenständig im Ontologiegraph bewegt, ohne direkt auf die Struktur Einfluss zu nehmen.

### 6.3.1 Eingabemodus

Die Abbildung 6.2 zeigt die Oberfläche mit deren Hilfe auch Nicht-Experten Daten in ontologischer Form eingeben können. Zu diesem Zweck ist die Oberfläche in drei horizontale Bereiche gegliedert. Auf der linken Seite befinden sich eine Baumdarstellung der Klassenhierarchie und eine Liste aller Instanzen, der im Baum selektierten Klasse sowie ein Eingabefeld zur Suche von Daten innerhalb der Hierarchie. Dieser Bereich kann zur weiteren Steigerung der Übersichtlichkeit auch eingeklappt und damit verborgen werden. In der Mitte befindet sich der zentrale Eingabebereich für Daten. Mit Hilfe der Navigationsleiste im oberen Bereich kann der Benutzer wie in einem Webbrowser zurück und wieder vorwärts navigieren. Darunter befindet sich der große Bereich zur Eingabe von Daten für Relationen. Datentypen-Relationen können dabei direkt mit Werten gefüllt werden. Während der Eingabe wird überprüft, ob die eingebenden Daten auch dem geforderten Datentyp entsprechen. Beim Klick auf eine Objekt-Relation wird automatisch eine Instanz der Zielklasse erstellt und diese zur Bearbeitung geöffnet. Weiterhin ist es möglich, Relationen, die im jeweiligen Szenario nicht mit Daten gefüllt werden sollen (da diese zum Beispiel durch Ableitung von höheren Klassen durchgereicht werden), optisch auszublenden. Auf der rechten Seite befindet sich der Assistenzbereich, der dem Benutzer Hilfetexte zur Eingabe anzeigen kann. Dabei werden sowohl Daten, die bereits in der Ontologie vorgegeben sind, angezeigt (zum

Beispiel <rdfs:label> und <rdfs:comment>) als auch zusätzliche Daten sowohl für die aktuelle Klasse als auch für jede einzelne Relation, die extra für diesen Anwendungsfall festgelegt wurden.

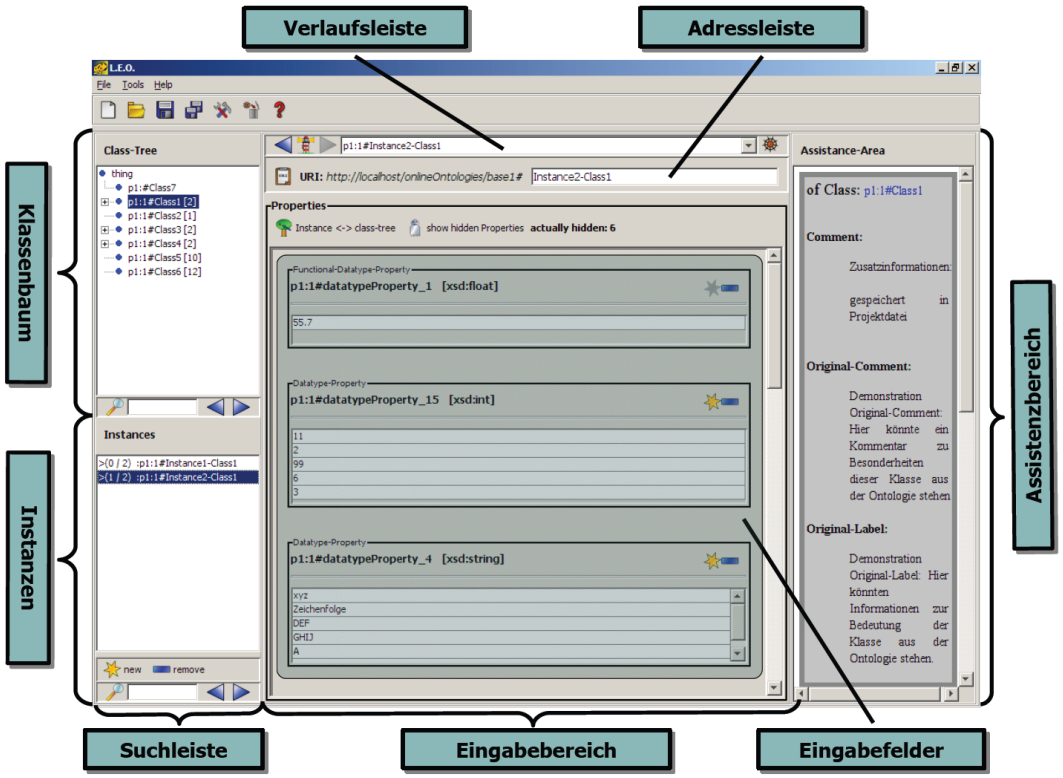



Abbildung 6.2: Lightweight Ontology Editor (LEO) - Grafische Oberfläche

### 6.3.2 Administrationsmodus

Zur Konfiguration der Oberfläche zur Eingabe von Daten, besitzt die Anwendung den sogenannten Administrationsmodus (siehe Abbildung 6.3). Dieser bietet eine ähnliche Sicht auf die Ontologie wie der Eingabemodus. Zusätzlich ist es allerdings möglich die Hilfetexte für die jeweiligen Klassen und Relationen einzugeben und festzulegen, welche Klasse beim Start automatisch aufgerufen werden soll. Des Weiteren lässt sich hier für jede Klasse festlegen, welche Relationen versteckt werden sollen.

### 6.3.3 Datenhaltung

Zur Datenhaltung verwendet der Editor zwei getrennte Dateien. In der ersten Datei werden analog zu anderen OWL-Editoren die Instanzen und Relationen im OWL-Format in

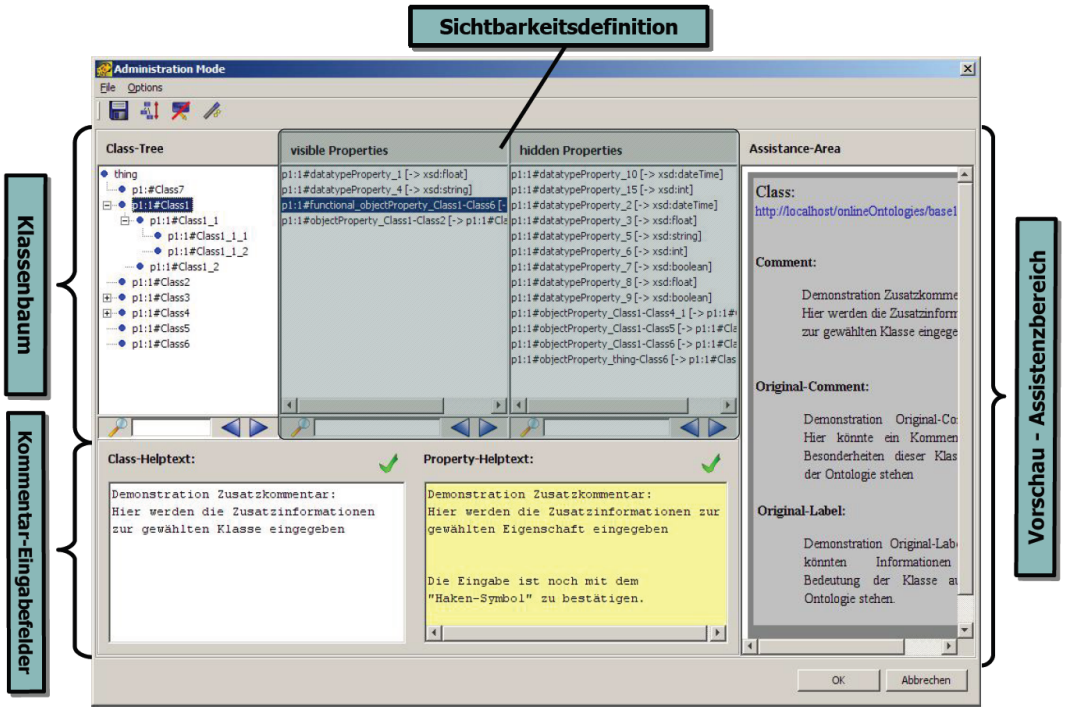



Abbildung 6.3: Lightweight Ontology Editor (LEO) - Administrationsoberfläche

Form einer RDF/XML-Syntaxkodierung verwaltet. Die zweite Datei dient der Konfiguration der Oberfläche und umfasst alle Informationen, die mit Hilfe des Administrationsmodus eingegeben wurden. Hierzu wird ein eigenes XML-Format erstellt (siehe Quellcode 6.3). Es umfasst eine Liste von Klassen, für die Eigenschaften definiert sind. Dies können zum Beispiel Hilfstexte sein oder die Festlegung, dass gewisse Relationen ausgeblendet werden. Zusätzlich werden im XML-Wurzelelement die Startklasse angegeben (`startInstance`) sowie die Eigenschaften festgelegt, ob die Klasse „Thing“ angezeigt werden soll (`domThingVis`), beziehungsweise wie viele Ebenen die Suche umfasst (`domainTreeSearchDepth`).

**Quellcode 6.3: Lightweight Ontology Editor (LEO) Projektdatei**

```
<?xml version="1.0" encoding="UTF-8"?>
<project owlFilePath="test.owl" version="1.0"
         startInstance="http://sample.org/text.owl#Instance1-Class1"
         domThingVis="true" domainTreeSearchDepth="-1">
  <class id="http://sample.org/text.owl#Class1">
    <property id="http://sample.org/text.owl#datatypeProperty_1">
      <non-visible />
    </property>
    <property id="http://sample.org/text.owl#datatypeProperty_2">
      <non-visible />
    </property>
    <property id="http://sample.org/text.owl#datatypeProperty_3">
```

```
13        <non-visible />
14      </property>
15      <property id="http://sample.org/text.owl#datatypeProperty_4">
16        <non-visible />
17      </property>
18      <helptext>Hilfstext zu Class1</helptext>
19      <property id="http://sample.org/text.owl#objectProperty_Class1-Class2">
20        <helptext>Hilfstext zu objectProperty_Class1-Class2 von Class1</helptext>
21      </property>
22    </class>
23    <class id="http://sample.org/text.owl#Class3">
24      <helptext>Hilfstext zu Class3</helptext>
25    </class>
26    <class id="http://sample.org/text.owl#Class4">
27      <helptext>Hilfstext zu Class4</helptext>
28    </class>
29  </project>
```

## 6.4 Datenvalidierung

Um sicherzustellen, dass OMM-basierte Gedächtnisse stets korrekt sind, kann mit Hilfe eines Datenvalidierungsmoduls dieser Zustand überprüft werden. Das Modul besteht aus unterschiedlichen Bestandteilen, die jeweils eigene Aspekte des Modells auf Korrektheit überprüfen. Die Phasen eins und zwei führen zuerst eine syntaktische Prüfung durch, während in der dritten Phase eine semantische Analyse der Metadaten erfolgt.

1. Die erste Phase der Überprüfung widmet sich der **Syntax der OMM-Darstellung**. Diese Phase überprüft dabei die OMM-XML- und die RDFa/Microdata-Darstellung mit Hilfe des fest definierten OMM-Schemas auf syntaktische Korrektheit.

2. Die zweite Phase überprüft die **Nutzdaten der OMM-Blöcke**, sofern diese Daten im XML-Format vorliegen und ein XML-Schema angegeben wurde.

3. Die dritte Phase überprüft die **Metadaten der einzelnen Blöcke** auf semantische Korrektheit. Dabei werden zum Beispiel geprüft, ob der angebenden MIME-Typ zu den Nutzdaten passt, ob zum angegebenen Namespace passende Werte für Type, Format und Payload angegeben sind und ob die Einträge für Creator und Contributor zeitlich korrekt angegeben sind.

4. Darüber hinausgehend kann diese Komponente durch zusätzliche Module erweitert werden, die eigene Prüfungen für bestimmte Blockinhalte integrieren. Zu diesem Zweck muss eine Java-Klasse erstellt werden, die vom vorgegebenen Interface `OMMValidator` abgeleitet ist (siehe Quellcode 6.4).

**Quellcode 6.4: Schnittstelle zur Integration eigener Prüfmethoden in den OMM-Validator**

```
1  public enum OMMValidationResultType { OK, WARNING, ERROR }
2
3  public class OMMValidationResult
4  {
5    public OMMValidationResultType result;
6    public String errorMessage;
7  }
8
9  public interface OMMValidator
10 {
11   public OMMValidationResult validate(OMMBlock block);
12 }
```

Die Nutzung der Validierungskomponente erfolgt in zweierlei Hinsicht (siehe Abbildung 6.4). Zum einen ist diese direkt in die libOMM-Implementierung integriert. Dadurch erfolgt bei jedem Zugriff auf OMM-basierte Gedächtnisse eine syntaktische und semantische Überprüfung der eingelesenen und der geschriebenen Daten. Zum anderen kann die Komponente auch eigenständig genutzt und in eigene Anwendungen integriert werden, um zum Beispiel Anwendereingaben oder noch unbekannte Gedächtnisse zusätzlich zu überprüfen und dem Ersteller direkt eine Rückmeldung geben zu können.

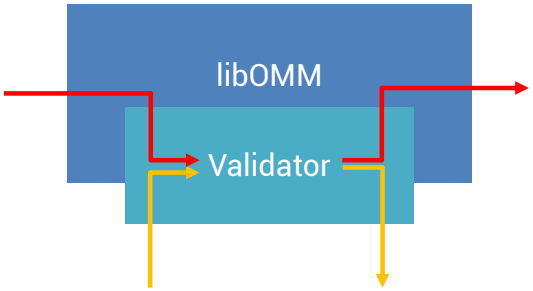



Abbildung 6.4: OMM-Validator mit Integration in libOMM (roter Pfad) bzw. zur eigenständigen Nutzung (gelber Pfad)

## 6.5 Object Memory Server (OMS)

Wie in den vorangegangenen Kapiteln beschrieben, können OMM-basierte Objektgedächtnisse entweder als XML-Datei oder als Binärblock implementiert werden. Für Anwendungsfälle, in denen nur eine kleine Datenmenge anfällt und die Daten ausschließlich auf einem physischen Speicher abgelegt werden oder nur ein einziges Gedächtnis verwaltet werden muss, ist diese Lösung kostengünstig und praktikabel. In vielen Szenarien skalieren diese

Lösungen jedoch nicht in ausreichendem Maße. Häufig möchte man Produkte ohne eingebettete Controller mit Gedächtnissen ausstatten. In diesem Fall eignet sich nur die Lösung des Anbringens einer Referenz am Produkt und die Speicherung des Gedächtnisses in einem Server-basierten System. Des Weiteren kann es sinnvoll sein, Gedächtnisse vollständig oder teilweise an externe Speicherorte auszulagern, um den Zugriff zu vereinfachen oder die Redundanz zu erhöhen. Schließlich kann es Anwendungsfälle geben, in denen zusätzliche Funktionen, wie eine Versionsverwaltung oder ein Rechte- und Rollen-basierter Zugriff auf Gedächtnisse, notwendig sind. Auch in diesem Fall kann der OMS das Hosting von Gedächtnissen übernehmen.

Aus diesem Grund wird das OMM-Format mit Hilfe einer dedizierten Anwendung einem breiteren Einsatzspektrum zur Verfügung gestellt. Der sogenannte Object Memory Server (OMS) fungiert hierbei als Multi-Gedächtnis-Server, der in der Lage ist einen OMM-kompatiblen Speicher für eine große Anzahl an Gedächtnissen zur Verfügung zu stellen [Sch07, HS13]. Die Funktionalität, die Schnittstellen und die Arbeitsweise des OMS werden im Folgenden beschrieben.

### 6.5.1 Funktionalität

Die folgende Auflistung zeigt die Kernfunktionalität des Object Memory Server (siehe auch Abbildung 6.5):

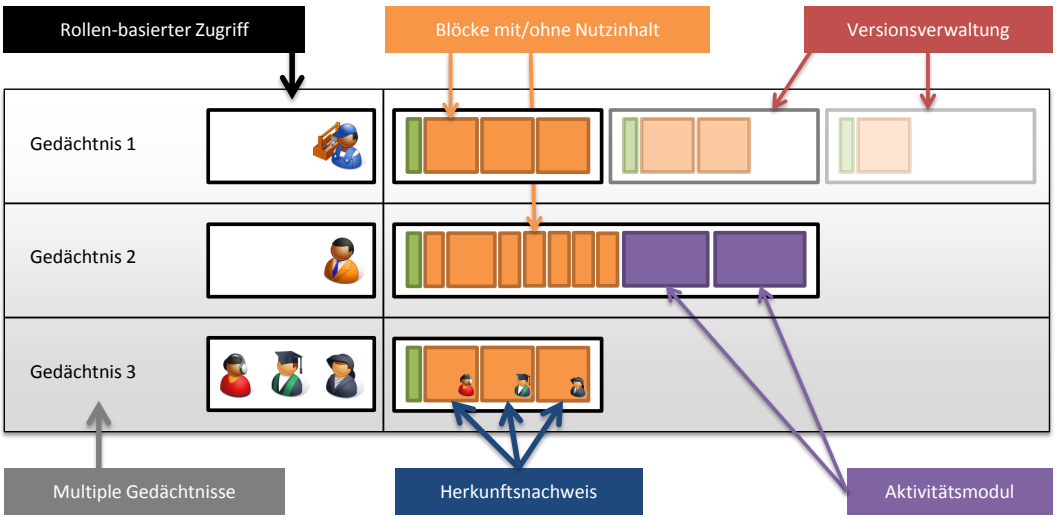



Abbildung 6.5: Wichtige funktionale Merkmale des Object Memory Server (OMS)

- **Ablage von Gedächtnisblöcken mit und ohne Nutzdateninhalt** - Die Hauptaufgabe des OMS ist die Ablage und Bereitstellung von Objektgedächtnissen. Der OMS nutzt dazu das OMM-Format, welches sowohl als Austauschformat mit Anwendungen und

externen Diensten als auch zur internen Datenspeicherung verwendet wird. Dadurch besteht für Entitäten, welche Daten im Gedächtnis ablegen möchten, die Möglichkeit diese Daten entweder direkt im jeweiligen Block abzulegen oder den OMS nur als einen Index zu verwenden und die Daten auf einen externen Datenspeicher (z.B. Webserver oder in der Cloud) auszulagern.

- **Server für multiple Gedächtnisse** - Der OMS erlaubt es an der gleichen zentralen Stelle eine große Anzahl an Gedächtnissen mit einem System zu verwalten. Dadurch lassen sich mit nur einer OMS-Instanz viele Gedächtnisse an gleicher Stelle zur Verfügung stellen. Da der Zugriff durch spezielle Kommandos auf die Interkation mit dem Gedächtnis optimiert ist, skaliert der OMS bei einer großen Zahl an Gedächtnissen besser, zum Beispiel die Verwaltung von OMM-XML-Gedächtnissen auf einem Webserver.

- **Versionsverwaltung** - Im Gegensatz zur bisher vorgestellten Lösung jedes Gedächtnis in einer XML-Datei oder in einem Binärblock zu persitieren, bietet der OMS darüber hinausgehende eine integrierte Versionsverwaltung. Jede Änderung an einem Gedächtnis führt dabei zu einer neuen Version des Gedächtnisses. Über diverse Metadaten (z.B. Datum) lassen sich gezielt ältere Gedächtniszustände abfragen.

- **Datenintegrität & Herkunftsnachweis** - Um sicherzustellen, dass die Daten in einem Gedächtnis nicht verändert werden, bietet das OMM-Format bereits die Möglichkeit Hashwerte als Metadaten abzulegen. Der OMS führt dieses Konzept konsequent weiter, in dem die Datenübertragung von und zum OMS über XML-Zertifikate signiert wird, um Man-in-the-Middle-Zugriffe auszuschließen. Zusätzlich ist es möglich, dass sich Entitäten, die das Gedächtnis verändert, ebenfalls mit Zertifikaten ausweisen müssen. Mit der Versionsverwaltung kann eine lückenlose Kette alle Datenlieferanten an das Gedächtnis hergestellt werden.

- **Rollen-basierter Zugriff** - Der OMS erlaubt mit den erwähnten Zertifikaten auch die Steuerung des Zugriffs auf Gedächtnisse. Zum einen lässt sich über sogenannte Black- und White-Listen der Zugriff strikt regeln, sowohl für das gesamte Gedächtnis, als auch für einzelne Blöcke. Darüber hinausgehend kann der Zugriff auch durch gültige Zertifikatsketten nachgewiesen werden.

- **Integriertes Aktivitätsmodul** - Über eine eigenständige Komponente lassen sich sogenannte Aktivitätsmodule in das System laden, die es einem externen Anwender erlauben bestimmte Gedächtnis-relevante Operationen direkt im Gedächtnis ablaufen zu lassen.

### 6.5.2 Schnittstellen

Der Object Memory Server besitzt zur Kommunikation mit der Außenwelt zwei grundlegend unterschiedliche Schnittstellen. Um Anwendungen und Dienste anzubinden besteht die Möglichkeit über eine HTTP-Schnittstelle Daten zu übertragen. Zusätzlich können die Inhalte eines Gedächtnisses einem Endanwender direkt über eine HTML5-Weboberfläche präsentiert werden.

Zur Kommunikation mit dem OMS stellt dieser eine auf dem Hypertext Transport Protocol (HTTP) basierende Schnittstelle bereit. Um Daten vom OMS abzurufen oder Daten zum OMS zu schicken, wird eine neue zustandslose Verbindung aufgebaut. Die Übermittlung von Befehlen zum OMS wird über die URL der HTTP-Anfrage codiert. Die Übertragung von Daten als auch Antworten vom OMS werden stets als XML-Dokument übermittelt. Der Zugriff auf diese einzelnen Funktionen des OMS erfolgt dabei über die Codierung innerhalb der URL:

```
http://<DNS-Name des OMS>/<Schnittstelle>/<Gedächtnisname>
```

Der *DNS-Name des OMS* (oder alternativ die IP-Adresse) wird durch die Infrastruktur vorgegeben, auf der der OMS betrieben wird. Die *Schnittstelle* definiert auf welches Subsystem des OMS zugegriffen werden soll. Der *Gedächtnisname* bezeichnet das anzusprechende Gedächtnis und muss dabei aus ASCII-Zeichen bestehen und wird von der Länge her dahingehend begrenzt, dass die URL maximal 255 Zeichen lang sein darf. Jeder Gedächtnisname darf zusätzlich nur einmal pro OMS-Instanz verwendet werden. Als Namen können auch Konzepte aus anderen Arbeiten genutzt werden, zum Beispiel die UbisIdentifier aus UbisWorld (siehe Kapitel 3.2.5). Dadurch ergibt sich aus dem DNS-Namen des OMS, der Schnittstelle und dem Gedächtnisnamen eine eindeutige ID des Gedächtnisses. Diese muss daher als primäre OMM-ID definiert werden (siehe Abbildung 5.2.1), wenn der OMS als Datenspeicher verwendet wird. Als Schnittstelle sind folgende Werte definiert:

- **.../web/...**: Ermöglicht den Zugriff auf die HTML5-Benutzeroberfläche des OMS. Über diese Schnittstelle ist auch eine direkte Bearbeitung der Gedächtnisse möglich, da diese Schnittstelle auch von der HTML5-Oberfläche selbst genutzt wird, um Benutzeranweisungen umzusetzen. Für einen externen Zugriff wird allerdings empfohlen die mächtigere REST-Schnittstelle zu verwenden.
- **.../rest/...**: Eine REST-basierte Schnittstelle zur offenen Interaktion mit dem Gedächtnis. Alle Funktionalitäten der *libOMM* können mit Hilfe der REST-Schnittstelle an den OMS delegiert werden.
- **.../query/...**: Die dritte Schnittstelle erlaubt es OMS-+übergreifend auf allen Gedächtnissen eines OMS Anfrageoperationen durchzuführen.

Alle drei Schnittstellen werden in den folgenden Abschnitten detailliert beschrieben.

### WEB-Schnittstelle

Die WEB-Schnittstelle wird zur Anzeige der HTML5-Benutzeroberfläche verwendet und kann darüber hinaus von externen Anwendungen aus angesprochen werden. Die Zugriffsverwaltung muss dabei über die Weboberfläche abgewickelt werden, da angemeldete Benutzer über Cockies identifiziert werden, welche auch bei einer direkten Nutzung der WEB-Schnittstelle verwendet werden.

Die folgende Übersicht zeigt nun eine Aufstellung aller Befehle, die vom OMS verarbeitet werden können; alle Beispiele werden anhand des fiktiven OMS-Server `omm.org` und dem Gedächtnis `dome1` gezeigt:

#### Gesamtes Gedächtnis abrufen

- **Befehl:** `?cmd=download&output=[xml|html_rw|html_rdfa|html_microdata]`
- **Argument von `output`:** Definiert den Typ der Rückgabe
  - **Als OMM-XML-Datei:** `xml`
  - **Als HTML5-Webseite mit OMM-Daten als RDFa:** `html_rdfa`
  - **Als HTML5-Webseite mit OMM-Daten als Microdata:** `html_microdata`
  - **Als HTML5-Webanwendung (Defaultwert):** `html_rw`
- **Rückgabe:** Das Gedächtnis im angegebenen Format.
- **Beispiel:** `http://omm.org/m/dome1?cmd=download&output=xml`

#### Inhaltsverzeichnis eines Gedächtnises abrufen

- **Befehl:** `?cmd=toc`
- **Rückgabe:** Das Inhaltsverzeichnis des Gedächtnisses als OMM-XML-Datei.
- **Beispiel:** `http://omm.org/m/dome1?cmd=toc&output=xml`

### Gesamtes Gedächtnis zum OMS hochladen

- **Befehl:** `?cmd=updload`
- **Beispiel:** `http://omm.org/m1?cmd=upload`
  + OMM-XML-Gedächtnis im HTTP-POST-Block
- **Rückgabe:** *keine*
- **Auswirkung:** Ist dieses Gedächtnis noch nicht im OMS vorhanden, wird es neu angelegt und mit den Daten aus der OMM-XML-Datei befüllt. War das Gedächtnis bereits vorhanden, erstellt der OMS ein Delta zwischen dem gespeicherten Gedächtnis und der Datei und aktualisiert das OMS-Gedächtnis mit allen Änderungen.

### Neuen Block im Gedächtnis anlegen

- **Befehl:** `?cmd=add&part=block`
- **Beispiel:** `http://omm.org/m1?cmd=add&part=block`
  + Block als OMM-XML-Datei im HTTP-POST-Block
- **Rückgabe:** *keine*
- **Auswirkung:** Der OMS überprüft ob die Block-ID des übermittelte Blocks bereits im Gedächtnis verwendet wird und meldet in diesem Fall einen Fehler; ansonsten wird der Block übernommen. Wird der übermittelte Block ohne ID transferiert, ermittelt der OMS selbstständig eine ID und weist diese dem Block zu.

### Bestehenden Block im Gedächtnis löschen

- **Befehl:** `?cmd=remove&part=block&block_id=⟨BLOCK-ID⟩`
- **Beispiel:** `http://omm.org/m1?cmd=add&part=block&block_id=block123`
- **Rückgabe:** *keine*
- **Auswirkung:** Der OMS überprüft, ob die übermittelte Block-ID im Gedächtnis verwendet wird und löscht den Block, falls keine Zugriffsregeln dies verbieten. Ist die Block-ID nicht im Gedächtnis vorhanden meldet der OMS einen Fehler.

### Gedächtnis-Block bearbeiten

- **Befehl:** ?cmd=edit&part=[namespace|format|creator|contributor|↩ title|description|type|subject|link|payload]&block_id=⟨BLOCK-ID⟩
- **Beispiel:** http://omm.org/m1?cmd=edit&part=format&block_id=block123 + OMM-XML-Format-Wert im HTTP-POST-Block
- **Rückgabe:** *keine*
- **Auswirkung:** Der OMS überprüft, ob die übermittelte Block-ID im Gedächtnis verwendet wird und führt die gewünschte Aktion an den Metadaten des Blocks durch, falls keine Zugriffsregeln dies verbieten. Ist die Block-ID nicht im Gedächtnis vorhanden meldet der OMS einen Fehler.

### REST-Schnittstelle

Die REST-Schnittstelle wurde speziell darauf ausgelegt externen Anwendungen und Diensten einen stabilen und einfachen Zugriff auf Objektgedächtnisse, welche auf einem OMS abgelegt sind, zu ermöglichen. Die Schnittstelle reflektiert dabei die typischen OMM-Operationen mit Hilfe einer zustandslosen REST-Kommunikation. Im Gegensatz zur vereinfachten WEB-Schnittstelle sind über REST auch zusätzliche OMS-Funktionen, wie zum Beispiel die Aktivitätsmodule erreichbar. Die Schnittstelle selbst verwendet ein modulares Konzept, in dem einzelne Funktionsbausteine in Module ausgelagert wurden. Somit ist es möglich, dass nicht alle Gedächtnisse den gleichen oder vollständigen Funktionsumfang bereitstellen müssen. Alle Befehle werden über die vier in REST verwendeten HTTP-Kommandos GET, PUT, POST und DELETE abgewickelt. Das Objekt, auf dem diese Operation durchgeführt werden soll, wird durch den jeweiligen Pfad (in diesem Fall über die URL) festgelegt. Für jeden Pfad haben die vier REST-Kommandos eine eindeutige Funktion, welche in den folgenden Abschnitten beschrieben ist. Zusätzlich kann eine Anwendung über eine sogenannte Funktionsaushandlung („feature negotiation“) in Erfahrung bringen, welche vom Modul des jeweiligen Gedächtnissess bereitgestellt werden.

**Funktionsaushandlung (feature negotiation)** Da die Funktionsaushandlung beim Zugriff auf ein unbekanntes Objektgedächtnis als erstes aufgerufen werden sollte, um festzustellen welche funktionalen Module dieses Gedächtnis bereitstellen kann, wird dieser Dienst direkt mit der URL der REST-Schnittstelle angesprochen; für ein beispielhaftes Gedächtnis mit dem Namen „dome1“ lautet die URL wie folgt:

http://sample-oms.org/rest/dome1/

Eine HTTP-GET-Anfrage an diese URL liefert nun eine JSON-Datei zurück, die die verfügbaren Funktionsmodule angibt. Folgendes Codebeispiel (Quellcode 6.5) zeigt eine beispielhafte Antwort des OMS, wobei die beiden Funktionsmodule *storage* und *management* verfügbar sind und zu beiden zusätzliche Attribute definiert wurden.

**Quellcode 6.5: Beispielrückgabewert: REST-Funktionsaushandlung im JSON-Format**

```
1  {
2    "VERSION": 1,
3    "STORAGE":
4      {
5        "LINK": "http://sample-oms.org/rest/dome1/st/",
6        "CAPACITY": "20G",
7        "FREE_SPACE": "19G",
8        "DISTRIBUTED": true
9      }
10   "MANAGEMENT":
11     {
12       "LINK": "http://sample-oms.org/rest/dome1/mgmt/",
13       "FLUSH": true
14     }
15   "ACTIVITY":
16     {
17       "HEARTBEAT": "100",
18       "TRIGGER": ["MEMORY", "BLOCK"],
19       "IO_CAPABILITIES": ["EMAIL"]
20     }
21 }
```

Da jedes Objektgedächtnis (wie der Name bereits vorgibt) immer mindestens die Funktion *storage* anbieten muss, zeigt der folgende Codeausschnitt (Quellcode 6.6) eine minimale Antwortdatei. Die Angabe `null` als Attribute des *storage*-Moduls bedeutet hierbei nicht, dass dieses Modul nicht verfügbar ist, sondern dass keine weiteren Attribute zu diesem Modul definiert sind.

**Quellcode 6.6: Minimale REST-Funktionsdefintion im JSON-Format**

```
1 {
2   "VERSION": 1,
3   "STORAGE":  null
4 }
```

Die Rückgabedatei listet nur diejenigen Module auf, die auch verfügbar sind. Auf der ersten Ebene finden sich dabei immer zuerst die Angabe der Version (in den Beispielen immer 1) und die Namen der Funktionsmodule. Alle Attribute werden per Definition immer in Großbuchstaben gelistet. Mögliche Module sind `STORAGE`, `MANAGEMENT` und `ACTIVITY`,

welche später in diesem Abschnitt ausführlicher beschrieben werden. Die Module können dabei folgende Optionen mit angeben:

- **Alle Module**:
  - `LINK`: Gibt die URL an, unter der dieses Funktionsmodul zu erreichen ist. Wird dieser Wert nicht definiert, kann eine Anwendung davon ausgehen, dass die Module unter ihrer standardmäßigen URL zu erreichen sind. Diese lautet für `STORAGE` → /st/, für `MANAGEMENT` → /mgmt/ und für `ACTIVITY` → /fm/.
- `STORAGE`:
  - `CAPACITY`: Dieser Wert gibt an, auf welche Größe das Gedächtnis maximal wachsen darf. Ist der Speicherplatz unbeschränkt, wird der Wert `max` (default) verwendet.
  - `FREE_SPACE`: Dieser Wert gibt den noch freien Speicherplatz für dieses Gedächtnis an. Ist der Speicherplatz unbeschränkt, wird der Wert `max` (default) verwendet.
  - `DISTRIBUTED`: Zeigt an, ob dieses Gedächtnis vollständig lokal gespeichert wird (`false`, default) oder ob Teile davon auf externen Server ausgelagert wurde (`true`). Dadurch lassen sich Rückschlüsse auf die Abrufgeschwindigkeit ziehen. Zusätzlich ist es möglich, dass ein Abruf des Gedächtnisses im verteilten Fall nicht vollständig ist.
- `MANAGEMENT`:
  - `FLUSH`: Ein Gedächtnis gibt über dieses Attribut an, ob Änderungen am Gedächtnis automatisch dauerhaft gespeichert werden (`false`, default) oder ob ein gesondertes „Flush“-Kommando erforderlich ist, um Änderungen am Gedächtnis permanent zu persistieren (`true`). Dies ist insbesondere auf mobilen oder eingebetteten Geräten der Fall.
- `ACTIVITY`:
  - `HEARTBEAT`: Definiert den Heartbeat-Grundtakt (siehe Kapitel 6.5.6) in Millisekunden, das heißt die Frequenz mit der Heartbeat-Skripte maximal aufgerufen werden können. Ein Wert von 100 gibt beispielsweise an, dass der Heartbeat-Timer alle 100 ms aufgerufen wird. Mit Hilfe dieser Angabe können externe Anwendungen feststellen, welchen Taktteiler sie einstellen müssen, um ein Skript mit der gewünschten Frequenz zu betreiben.

- TRIGGER: Dieser Wert beinhaltet eine Liste aller Möglichkeiten, um ein ereignisgesteuertes Skript zu starten. Es sind dabei die Werte MEMORY und BLOCK möglich, die angeben, dass ein Skript bei einer Änderung des gesamten Gedächtnisses oder bei Änderungen an einzelnen Blöcken gestartet werden kann.
- IO_CAPABILITIES: Mit Hilfe dieses Wertes kann kenntlich gemacht werden, welche zusätzlichen Schnittstellen zur Kommunikation nach außen von Skripten genutzt werden können, beziehungsweise vom Gedächtnis bereitgestellt werden. Zurzeit kann nur der Wert EMAIL angegeben werden, der angibt, dass ein Skript eine E-Mail versenden kann.

**Speicherzugriffsmodul (storage)** Mit Hilfe des Speicherzugriffs wird es externen Anwendungen ermöglicht Operationen auf dem Objektgedächtnis durchzuführen. Dies umfasst z.B. Funktionen zum Anlegen neuer Blöcke, zum Bearbeiten von Metadaten oder zum Ändern von Nutzdaten. Als Datenformat wird für OMM-Bestandteile die OMM-XML-Darstellung verwendet, alle weiteren Daten werden im JSON-Format übertragen. Durch die Verwendung von standardisierten Web-Techniken, gibt es auf Anwendungsseite keine Einschränkungen bezüglich der verwendeten Formate und Plattformen. Auf oberster Pfadebene stehen hierzu drei unterschiedliche Zweige zur Verfügung: .../block_ids/ ermöglicht es mit dem GET-Kommando eine Liste aller verfügbarer Blöcke des Gedächtnisses abzurufen, welches als JSON-Liste zurückgegeben wird (siehe Quellcode 6.7).

**Quellcode 6.7: Bespielrückgabewert der REST-GET-Aufruf auf .../block_ids/**

```
1 {"IDs": [ "block_1", "block_2", "block_3" ]}
```

Zusätzlich bietet ein GET-Kommando auf der Adresse .../toc/ einen Zugriff auf das Inhaltsverzeichnis (ToC) des Gedächtnisses, welches eine vereinfachte Sicht auf alle Metadaten darstellt. Das ToC wird dabei als OMM-XML-Datei zurückgegeben.

Schließlich definiert .../block/ alle Operationen, die auf einem einzelnen Block durchgeführt werden, z.B. das Erstellen oder Löschen von Blöcken, das Ändern von Metadaten oder das Hinzufügen von Nutzinhalten. Die möglichen Operationen werden im Folgenden detaillierter beschrieben.

- .../block/
  - POST: Erlaubt das Anlegen eines neuen Blocks. Als Argument wird ein OMM-XML-Ausschnitt des zu erstellenden Blocks erwartet und anschließend die ID des neuen Blocks als String zurückgeliefert.
- .../block/<block_id>/

  - DELETE: Löscht den Block mit der ID „<block_id>“, falls dieser existiert.

- .../block/<block_id>/meta/

  - GET: Ruft alle Metadaten des Blocks mit der ID „<block_id>“ ab, falls dieser existiert und gibt diese Information als OMM-XML-Datei zurück.

- .../block/<block_id>/meta/[id | namespace | creator | contributor | title | description | format | subject | type | link]/

  - GET: Ruft das jeweilige Metadatenattribut des Blocks mit der ID „<block_id>“ ab, falls dieser existiert und gibt diese Information als OMM-XML-Ausschnitt zurück.

  - PUT: Ersetzt das jeweilige Metadatenattribut des Blocks mit der ID „<block_id>“ durch den übermittelten Wert in Form eines OMM-XML-Ausschnitts.

  - DELETE: Löscht das jeweilige Metadatenattribut des Blocks mit der ID „<block_id>“.

- .../block/<block_id>/payload/

  - GET: Ruft die Nutzdaten des Blocks mit der ID „<block_id>“ ab und gibt diese als Base64-kodiertes Byte-Array zurück.

  - PUT: Ersetzt die Nutzdaten des Blocks mit der ID „<block_id>“ durch die mitgelieferten Daten in Form eines Base64-kodiertes Byte-Array.

  - DELETE: Löscht die Nutzdaten des Blocks mit der ID „<block_id>“.

**Verwaltungsmodul (management)** Das Verwaltungsmodul erlaubt den Zugriff und den Austausch von vollständigen Gedächtnisdaten. Diese können dabei als eine Einheit abgerufen werden, zum Beispiel zur Datensicherung und können auch von außen durch einen neuen Datensatz überschrieben werden, zum Beispiel zur Initialisierung von neuen Gedächtnissen. Der Zugriff auf die Schnittstelle des Verwaltungsmoduls ist dabei nur für den Administrator des OMS möglich.

- .../sync/

  - GET: Ruft das vollständige Gedächtnis als OMM-XML-Datei ab.

  - PUT: Überschreibt das aktuelle Gedächtnis mit dem als OMM-XML-Datei übermittelten neuen Daten.

  - DELETE: Löscht das aktuelle Gedächtnis.

**Aktivitätsmodul (activity)** Mit Hilfe der Schnittstelle zum Aktivitätsmodul können Skripte, die im Gedächtnis abgelegt sind oder von extern zugeführt werden, zur Ausführung gebracht werden. Die Beschreibung dieser Schnittstelle erfolgt gemeinsam mit weiteren Informationen zur Aktivitätskomponente in Kapitel 6.5.6.

### QUERY-Schnittstelle

Der OMS bietet als dritte Möglichkeit eine Schnittstelle, mit deren Hilfe Anfrageoperationen über alle Gedächtnisse eines OMS durchgeführt werden können. Dies ermöglicht es ein oder mehrere Gedächtnisse mit unbekannter ID aufzufinden, falls Daten bekannt sind, die in solchen Gedächtnissen abgelegt sind. Beispielsweise ist es mit diesem Ansatz möglich anhand einer weiteren ID eines Objekts herauszufinden, unter welcher ID dieses Gedächtnis über den OMS zu erreichen ist. Die eigentlichen Anfragen werden dazu in einer eigenen sehr einfachen Query-Syntax im JSON-Format definiert.

Diese Schnittstelle bringt nur eine Operation mit, mit deren Hilfe die Anfrage im JSON-Format an den OMS übertragen wird. Sobald die Operation durchgeführt wurde, wird das Ergebnis der Anfrage ebenfalls als JSON-Datei zurückgegeben. Soll zum Beispiel ein oder mehrere Gedächtnisse gefunden werden, welche einen Block vom Type *Dataset* enthalten und gleichzeitig einen OMM-ID-Block besitzen, der eine ID (http://mydomain.org/mem1/) definiert, die am 31.08.2007 gültig war, wird folgende URL des Beispielgedächtnisses *dome1* aufgerufen: `http://sample-oms.org/query/dome1/` und folgende JSON-Anfrage der HTTP-POST übertragen:

**Quellcode 6.8: OMM-Query im JSON-Format**

```
{
   "select" : "omm"
   "where"  :
   [
      {
         "omm.block.type" : "http://purl.org/dc/dcmitype/Dataset"
      },
      {
       "omm.id" :
         {
            "type"     : "url"
            "id"       : "http://mydomain.org/mem1/"
            "valid_at" : "2007-08-31T16:47+00:00"
         }
      }
   ]
}
```

Die Anfrage würde dann beispielsweise folgende Antwort im JSON-Format zurückliefern und somit angeben, dass zwei Gedächtnisse (*dome1* und *dome2*) den geforderten Bedingungen entsprechen:

**Quellcode 6.9: Rückgabewert der OMM-Query ebenfalls im JSON-Format**

```
1 {
2    "result" : ["http://omm.org/query/dome1/", "http://omm.org/query/dome2/"]
3 }
```

Die Syntax einer OMS-Query-Anfrage besteht generell aus einer einfachen SQL-typischen Klausel der Form:

```
select <TYP> where <LISTE AN BEDINGUNGEN>
```

Der Typ als erstes Argument legt hierbei fest, welche Art der Rückgabe gewünscht ist. Zum Zeitpunkt der Erstellung dieser Ausarbeitung sind folgende Werte möglich:

- **omm**: Gibt die primäre ID des Gedächtnisses als string zurück. Diese ID entspricht automatisch der URL unter der das Gedächtnis auf dem OMS zu erreichen ist.
- **omm.blockcount**: Gibt die Anzahl der Blöcke an, die das Gedächtnis enthält.
- **omm.lastchange**: Gibt die letzte Änderung an dem jeweiligen Gedächtnis an. Der Wert setzt sich hierbei aus dem Paar <Entität, Datum> zusammen, welches sowohl die Entität als auch das Datum der letzten Änderung angibt.

Die Liste an Bedingungen als zweites Argument definiert einen Filter, der auf alle Gedächtnisse angewendet wird, so dass nur noch die Gedächtnisse über bleiben, die alle Bedingungen erfüllen. Die einzelnen Bedingungen werden hierbei mit einer UND-Konjunktion verknüpft, was bedeutet, dass alle Bedingungen erfüllt sein müssen.

- **omm.block.[id | namespace | format | type | link]**: Das jeweilige Attribut der Blockmetadaten kann hier vordefiniert werden. Werden mehrere Attribute in eine Bedingung kombiniert, muss ein Block existieren, der alle Attribute vorweisen kann. Werden die Attribute jeweils als getrennte Bedingung gelistet, genügt es wenn für jedes Attribut mindestens ein Block vorhanden ist. Der Wert der jeweiligen Attribute ist ein String.
- **omm.block.[creator | contributor]**: Die Daten zum Ersteller und dem oder den Beitragenden eines Blocks werden durch eine Entitätsangabe definiert und setzen sich analog der OMM-Definition wie folgt zusammen:
  - **type [obligatorisch]**: Typ der Entität als String

  - **value [obligatorisch]**: Wert der Entität als String
  - **valid_at [optional]**: Eintrag war/ist gültig zu diesem Zeitpunkt als ISO8601-String
  - **valid_before [optional]**: Eintrag war gültig vor diesem Zeitpunkt als ISO8601-String
  - **valid_after [optional]**: Eintrag war/ist gültig nach diesem Zeitpunkt als ISO8601-String

- **omm.block.[title | description]**: Der Titel und die Beschreibung eines Blocks können mehrsprachig definiert sein. Aus diesem Grund wird diese Option auch als komplexes Objekt definiert. Der Wert ist hierbei optional. Dies ermöglicht es einen Test zu formulieren, der nur prüft ob ein Titel oder eine Beschreibung in einer bestimmten Sprache vorhanden sind.
  - **language [obligatorisch]**: OMM-konforme Sprache des Titels oder der Beschreibung als String
  - **value [optional]**: Wert des Titels oder der Beschreibung als String

- **omm.block.subject**: Schlagwortangaben im OMM-Subject-Tag können in Form von einfachen Textfeldern oder als Ontologiekonzepte definiert werden. Daher wird dieser Umstand in getrennte Attribute aufgeschlüsselt. Sollen hierarchische Schlagworte getestet werden, kann die „Punktnotation“ (siehe Kapitel 5.2.2) verwendet werden (z.B. „norms.din.e12“).
  - **type [obligatorisch]**: Typ des Schlagworts als String („text“ oder „ontology“)
  - **value [obligatorisch]**: Wert des Schlagworts als String

- **omm.id**: Definiert die Eigenschaft, dass ein Gedächtnis einen OMM-IDs-Block besitzt, der die gegeben ID beinhaltet. Der Wert dieses Attributs ist ein komplexeres Objekt, wobei die Attribute *type* und *id* immer angegeben werden müssen, während hingegen die Attribute *valid_at*, *valid_at* und *valid_at* optional sind.
  - **type [obligatorisch]**: Typ der ID als String
  - **id [obligatorisch]**: Wert der ID als String
  - **valid_at [optional]**: ID war/ist gültig zu diesem Zeitpunkt als ISO8601-String
  - **valid_before [optional]**: ID war gültig vor diesem Zeitpunkt als ISO8601-String
  - **valid_after [optional]**: ID war/ist gültig nach diesem Zeitpunkt als ISO8601-String

- **omm.structure**: Definiert die Eigenschaft, dass ein Gedächtnis einen OMM-Struktur-Block besitzt, der die gegebene Relation beinhaltet. Der Wert dieses Attributs ist ein komplexeres Objekt, wobei die Attribute *Relation*, *type* und *id* immer angegeben werden müssen, während hingegen die Attribute *valid_at*, *valid_at* und *valid_at* optional sind.

  - **relation [obligatorisch]**: Typ der Relation als String (z.B. „partOf“)
  - **type [obligatorisch]**: Typ des Ziels der Relation als String
  - **id [obligatorisch]**: Wert des Ziels der Relation als String
  - **valid_at [optional]**: Relation war/ist gültig zu diesem Zeitpunkt als ISO8601-String
  - **valid_before [optional]**: Relation war gültig vor diesem Zeitpunkt als ISO8601-String
  - **valid_after [optional]**: Relation war/ist gültig nach diesem Zeitpunkt als ISO8601-String

### HTML5-Oberfläche

Als Ergänzung zur eigentlichen Datenschnittstelle via HTTP, kann der OMS Daten eines Gedächtnisses auch direkt für Endanwender aufbereiten. Zu diesem Zweck wird das Gedächtnis als HTML5-Webanwendung dargestellt, die mit jedem HTML5-kompatiblen Browser betrachtet werden kann. Dies ermöglicht es auch, Gedächtnisse auf Mobilgeräten (z.B. Smartphones oder Tablets) einzusehen, ohne eine eigenständige Anwendung installieren zu müssen.

Die Oberfläche lehnt sich dazu an die in Abbildung 5.2 beschriebene OMM-Blockstruktur an. Mit Hilfe der Oberfläche können Benutzer Blöcke anlegen, löschen und bearbeiten. Dazu können Block-Metadaten angelegt, gelöscht und bearbeitet werden (siehe Abbildung 6.6). Zusätzlich können für jeden Block die Nutzdaten bearbeitet oder ausgetauscht beziehungsweise aktualisiert werden. Für einige bekannte Daten bietet die Weboberfläche eine direkte Bearbeitungsmöglichkeit der Nutzdaten, zum Beispiel für OMM- und OMM+-Blöcke sowie für Aktivitätsskripte. Diese können darüber hinaus auch mit Hilfe der Weboberfläche konfiguriert werden.


Abbildung 6.6: OMS HTML5-Weboberfläche im Bearbeitungsmodus eines Blocks

### 6.5.3 Versionsverwaltung

Als weitere Funktionalität bietet der OMS ein Modell zur Versionsverwaltung. Dies erlaubt es auch auf ältere Gedächtnisstände zugreifen zu können. Dabei werden die Konzepte der *Archivierung*, *Protokollierung*, *Wiederherstellung* und *Atomare Operation* bereitgestellt. Explizit nicht unterstützt wird ein Modell zur Arbeit mit *Entwicklungszweigen*, welches eine Datenaufsplittung und Wiederzusammenführung beinhaltet.

**Archivierung** - Zu diesem Zweck wird bei jeder Änderung an einem Gedächtnis eine Kopie des Originalgedächtnisses erstellt. Alle älteren Stände sind jederzeit verfügbar. Es werden keine Verfahren unterstützt um Gedächtnishistorien zu entsorgen. Daher ist jederzeit eine lückenlose Überwachung der Herkunft aller Daten im Gedächtnis möglich. In Zusammenarbeit mit einem Rollen-basierten Zugriff kann darüber hinaus eine sichere Änderungskette verwaltet werden.

**Protokollierung** - Jeder Änderungsschritt beinhaltet Informationen über Datum und Uhrzeit der Änderung, die Identität desjenigen, der die Veränderung durchgeführt hat, den betreffenden Block sowie die durchgeführte Änderungsoperation. Durch Nutzung eines externen Werkzeugs zur Darstellung von Unterschiedenen in zwei Dateien ist auch eine exakte Kenntlichmachung aller pro Schritt durchgeführten Änderungen möglich.

**Wiederherstellung** - Die Wiederherstellung einer älteren Version ist mit zwei unterschiedlichen Lösungen möglich. Im ersten Fall ist der Zugriff auf alle älteren Versionen möglich, in dem beim Gedächtniszugriff die Revisionsnummer mit angegeben wird. Ältere Versionen von Gedächtnissen können allerdings nur ausgelesen und nicht mehr verändert werden. Für Testzwecke ist es darüber hinaus möglich, alle Änderungen an einem Gedächtnis, die nach einer gegebenen Revision erfolgt sind, zu verwerfen. Somit kann ein älterer Stand leicht wiederhergestellt werden. Diese Funktionalität ist allerdings standardmäßig abgeschaltet und kann für jedes neu angelegte Gedächtnis explizit aktiviert werden; eine nachträgliche Aktivierung ist aus Sicherheitsgründen nicht möglich.

**Atomare Operation** - Alle schreibenden Zugriffe auf eine Gedächtnis werden als atomare Operation behandelt. Dies verhindert, dass mehrere gleichzeitige Zugriffe auf identische Gedächtnisbestandteile dieses in einen inkonsistenten Zustand überführen. Zu diesem Zweck wird jede Änderungsoperation erst dann ausgeführt, wenn alle vorherigen Änderungsoperationen beendet sind. Bei Nur-Lese-Zugriffen ist dieser Mechanismus nicht notwendig, da immer die letzte vollständige Version ausgeliefert wird.

**Entwicklungszweige** - Die Möglichkeit das Gedächtnis in mehrere Zweige (Branch) aufzuteilen wird nicht unterstützt, da dies zum Betrieb von Objektgedächtnissen nicht notwendig ist. Jedes Gedächtnis stellt immer eine lineare Struktur dar. Aus diesem Grund ist hier auch der Fall der Datenzusammenführung (Merge) nicht betrachtet, da keine Zusammenführung notwendig ist. Weil der Zugriff auf OMS-Daten immer nur über einen dedizierten OMS erfolgt, kann auch hier kein inkonsistenter Zustand entstehen.

Der Zugriff auf die Versionsverwaltung ist entweder über die HTML5-Weboberfläche (siehe Abbildung 6.7) oder über die Android Anwendung möglich.

| Revision | Date | Entity | Block | Changes | Actions |
|---|---|---|---|---|---|
| [0] | 3/13/12 9:00 AM (2 months and 10 days ago) | jens.haupert@dfki.de (email) | N/A | INITIAL_REVISION | |
| [1] | 5/22/12 2:06 PM (about 58 seconds ago) | 134.[redacted].199 (ipv4) | (7) Misc Block | BLOCK_ADDED | Withdraw this version |
| [2] | 5/22/12 2:07 PM (about 45 seconds ago) | 134.[redacted].199 (ipv4) | (7) Misc Block | TITLE_CHANGED | Withdraw this version |
| [3] | 5/22/12 2:07 PM (about 31 seconds ago) | 134.[redacted].199 (ipv4) | (2) sample text block | PAYLOAD_CHANGED | Withdraw this version |
| CURRENT (4) | 5/22/12 2:07 PM (about a second ago) | 134.[redacted].199 (ipv4) | (3) RES-COM logo | DESCRIPTION_CHANGED | Withdraw this version |

Abbildung 6.7: Historie in Form von Änderungen an einem OMS-Gedächtnis

### 6.5.4 Rechte- und Rollenverwaltung

Um einen sicheren Zugriff auf Gedächtnisdaten zu ermöglichen, bietet der OMS eine integrierte Rechte- und Rollenverwaltung. Es können für jedes Gedächtnis unterschiedliche Verfahren festgelegt werden, wie und auf welche Art und Weise Daten geschützt werden können. Der Zugriff erfolgt in der Regel nach dem Modell der *benutzerbestimmbaren Zugriffskontrolle* (Discretionary Access Control, DAC). Hierbei wird die Entscheidung, ob auf

eine Informationseinheit im Gedächtnis zugegriffen werden darf, auf der Basis der Identität des Zugreifenden getroffen. Darüber hinaus kann auch das Modell der *Rollen-basierten Zugriffskontrolle* (Role Based Access Control, RBAC) genutzt werden [FK92, ans04].

Zentrales Element der gesamten Sicherheitsarchitektur ist die eindeutige Identifizierung der zugreifenden Entität. Zu diesem Zweck bietet der OMS zwei unterschiedliche Verfahren. Die einfachere, aber auch weniger sichere Variante, setzt sich aus einer Kombination aus Benutzeridentität und Passphrase zusammen, die jeweils durch eine Zeichenkette dargestellt ist. Dieser Modus entspricht somit dem klassischen Benutzernamen und Passwort. Die zweite Variante verwendet Zertifikate zur Identifikation der Entität. Der OMS verwendet hierzu das im Web verbreitete Zertifikat-Format X.509 [ITU97]. Als dritte Möglichkeit kann der *neue elektronische Personalausweis*[3] genutzt werden. Dieser stellt mit Hilfe eines externen eID-Servers [BSI12] eine elektronische Identität zur Verfügung, die als Entitätsnachweis verwendet werden kann. Eine detailliertere Beschreibung dieses Verfahrens findet sich in Kapitel 7.6.1 wieder. Bei allen Verfahren kommt immer die Kombination aus Besitz und Wissen zum Einsatz. Aus diesem Grund generiert der OMS nur Zertifikate, die mit einer Passphrase geschützt sind.

Ist die zugreifende Entität dem OMS bekannt, erfolgt die Prüfung der durchzuführenden Aktion. Die Entität kann dabei eine der folgenden Aktionen durchführen:

- $O_R$: Lesend auf das Gedächtnis zugreifen
- $O_C$: Neue Blöcke anlegen
- $O_W$: Bestehende Blöcke verändern bzw. löschen

Die Operationen $O_C$ und $O_W$ beinhalten dabei automatisch das Recht das Gedächtnis auch lesen zu können ($O_R$). Zusätzlich kann die Operation $O_W$ für das gesamte Gedächtnis erlaubt werden oder nur für einzelne Blöcke. Zur Absicherung des Zugriffs werden intern die nun folgenden Funktionen verwendet. Zur Darstellung der einzelnen Bestandteile werden diese Symbole benutzt: Benutzer $U$, Benutzername $N_U$, Passwort $P_U$, Benutzerzertifikat $C_U$, neuer Personalausweis $E_U$, Gedächtnis $M$, Block $B_M \in M$, Freigabetabelle $A_M$ und Operation $O \in \{O_R, O_C, O_W\}$

Je nach verwendeter Art der Benutzeridentifizierung erfolgt die Ermittlung der Zugriffsentität gemäß der jeweiligen drei folgenden Funktionen:

$$F_{Name+Passwort} : N_U \times P_U \rightarrow U$$

$$F_{Zertifikat} : C_U \rightarrow U$$

[3] Gesetz über Personalausweise und den elektronischen Identitätsnachweis sowie zur Änderung weiterer Vorschriften, vom 18. Juni 2009, http://dipbt.bundestag.de/extrakt/ba/WP16/154/15407.html [Letzter Zugriff: 26.11.2012]

$$F_{eID}: E_U \rightarrow U$$

Die Freigabe der Daten wird anschließend über folgende Funktion ermittelt:

$$F_{Zugriff}: U \times M \times B_M \times O \times A_M \rightarrow \{Ja, Nein\}$$

Die Freigabe ($A$) für die oben beschriebenen Operationen ($O_R$, $O_C$, $O_W$), kann über vier unterschiedliche Modi erfolgen. Jede dieser Freigabetypen kann entweder für das gesamte Gedächtnis oder für jeden einzelnen Block getrennt festgelegt werden.

- $A_N$: Kein Schutz definiert, das bedeutet diese Operation kann von jeder Entität durchgeführt werden.
- $A_O$: Nur der Ersteller, der den Block angelegt hat, kann die Operation durchführen.
- $A_W$: Eine Liste kann definiert werden, die alle Entitäten enthält, die Operation durchführen können (Whitelist).
- $A_{C1}$: Das Zertifikat der zugreifenden Entität muss direkt vom Zertifikat des Erstellers nachweisen signiert worden sein.
- $A_{C2}$: Das Zertifikat der zugreifenden Entität muss eine gültige Zertifikatskette zum Zertifikat des Erstellers nachweisen können.

Der Modus $A_{C2}$ erlaubt es jedem, der ein signiertes Zertifikat mit gültiger Zertifikatskette besitzt, weitere Zertifikate zu erstellen, die ebenfalls genutzt werden können. Somit können zum Beispiel Subunternehmer ihr eigenes Zertifikat nutzen, um eine eindeutige Änderungshistorie zu ermöglichen und müssen nicht alle das Zertifikat des Auftraggebers nutzen. Ist dieses Verhalten nicht gewünscht, kann der Modus $A_{C1}$ genutzt werden, bei dem keine weiteren Glieder in der Zertifikatskette erlaubt werden.

### 6.5.5 Implementierung

Basierend auf den Funktionalitäten und Möglichkeiten des Object Memory Models wurde im Rahmen dieser Arbeit eine Referenzimplementierung einer Server-basierten Gedächtnislösung erstellt. Dabei handelt es sich um eine für die Java7-Plattform[4] bestimmte Anwendung mit dem Namen *Object Memory Server* (*OMS*). Der Server ist dabei funktional in vier unterschiedliche Module aufgeteilt: *Hauptapplikation*, *HTML5-Schnittstelle*, *REST-Schnittstelle*, *OMS-Query-Schnittstelle*. Der OMS ist als Servelet-Anwendung konzipiert, so dass diese vier Module jeweils als eigner Servelet-Container realisiert wurden. Somit kann der OMS mit

[4] `http://www.oracle.com/us/technologies/java/standard-edition/overview/index.html`, [Letzter Zugriff: 26.11.2012]

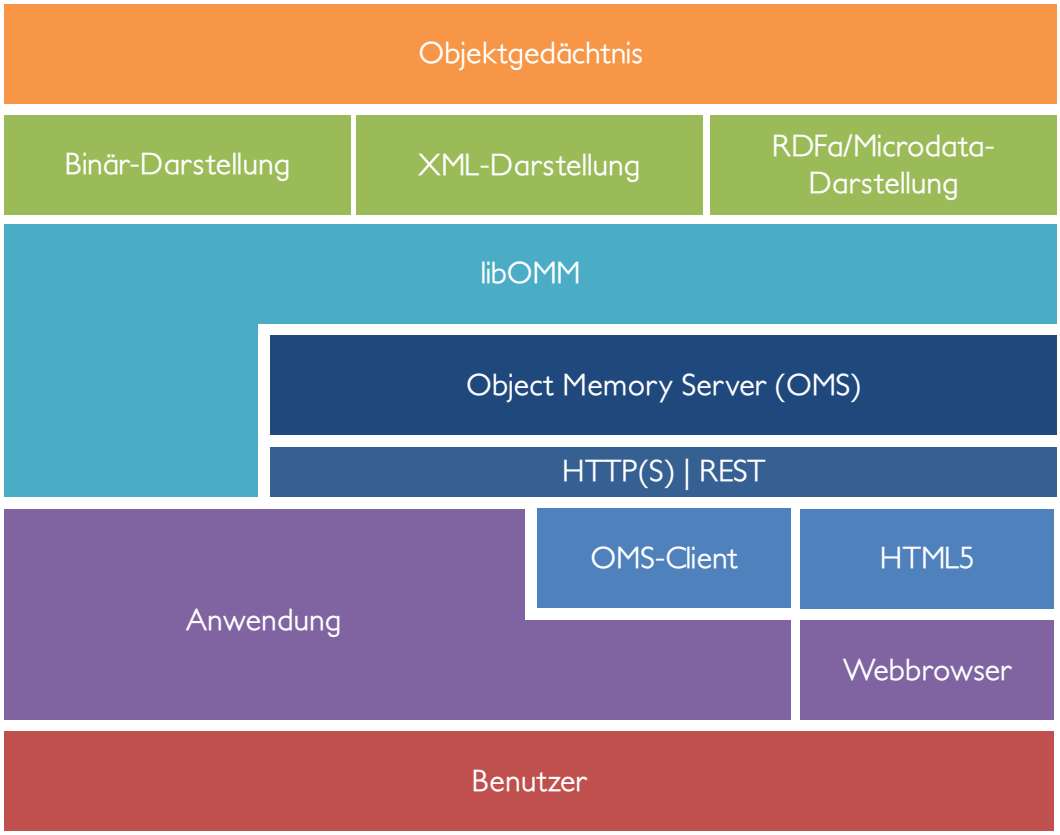



Abbildung 6.8: Architektur für den Zugriff auf OMM/OMS-Gedächtnisse

Hilfe eines beliebigen Servlet-Servers betrieben werden, zum Beispiel Apache Tomcat[5]. Die kompakte Grundausstattung des OMS benutzt allerdings den kleineren Jetty8-Server[6]. Die Hauptapplikation nutzt als Datengrundlage für die Verwaltung der einzelnen Gedächtnisse die Bibliothek libOMM (siehe Kapitel 5.2.8). Mit Hilfe des Ereignismechnismus der libOMM wurden die funktionalen Ergänzungen für die Versionsverwaltung und die Rechte/Rollen-verwaltung realisiert. Die Gedächtnisse werden zur Speicherung als XML-Dateien abgelegt. Da der OMS mehrere Gedächtnisse verwalten kann, wird jedes Gedächtnis in einem eigenen Verzeichnis abgelegt. Im Zuge der Versionsverwaltung hält der OMS jede Gedächtnisversion als eigene Datei vor, um mit geringer Verarbeitungsleistung eine alte Version bereitstellen zu können. Zu jedem Gedächtnis wird eine Konfigurationsdatei angelegt, die zum einen Informationen zu allen Versionen enthält und zum anderen die Einstellungen im Bereich der Rechte- und Rollenverwaltung umfasst, die der Besitzer des Gedächtnisses festgelegt hat. Der Zugriff auf die Gedächtnisse erfolgt entweder über die REST-Schnittstelle oder über die HTML5-Weboberfläche. Beide Module können unabhängig voneinander genutzt werden.

[5] `https://tomcat.apache.org/` [Letzter Zugriff: 26.11.2012]
[6] `http://www.eclipse.org/jetty/` [Letzter Zugriff: 26.11.2012]

Es ist auch möglich eine OMS-Konfiguration zu erstellen, die nur je eine der beiden Zugriffsmodule umfasst. Die REST-Schnittstelle verwendet die Bibliothek *restlet*[7] in Kombination mit dem OMS-Webserver Jetty. Die HTML5-Oberfläche wird ohne externe Abhängigkeiten direkt vom OMS erstellt. Abbildung 6.8 zeigt eine erweiterte Schichtendarstellung der typischen Nutzung des OMS.

### 6.5.6 Aktives Objektgedächtnis

Die bisher präsentierte Architektur stellt dem Benutzer nur eine „passive" Infrastruktur zur Verfügung, die genutzt werden kann um Objektgedächtnisdaten abzulegen. In der Benutzung von solchen Gedächtnissen gibt es allerdings Szenarien, die davon profitieren, dass das Gedächtnis über die Möglichkeit der Datenablage hinausgehende Eigenschaften anbietet. Im Folgenden sind nun vier solcher Anwendungsfälle dargestellt, die zusätzliche Anforderungen an die Infrastruktur stellen. Alle Fälle basieren auf der gleichen Ausgangssituation. Hierin ist ein physisches Objekt mit einem Objektgedächtnis ausgestattet, welches mit einem mobilen Gerät von einer Arbeitskraft ausgelesen werden kann.

**Szenario „Test bei Bedarf"** Ein Techniker möchte, basierend auf den Daten des Objektgedächtnisses, einen Integritätstest des physischen Objekts durchführen. Da das Gedächtnis im Laufe der Zeit eine große Menge an Daten gespeichert und die Datenübertragung zum mobilen Gerät des Technikers eine geringe Bandbreite hat, ist ein Download aller Daten und eine Überprüfung dieser Daten auf dem mobilen Gerät nicht sinnvoll. Aus diesem Grund benutzt der Techniker die Fähigkeiten des Gedächtnisses diesen Test selbständig ausführen zu können. Er ruft dazu nur die ebenfalls im Gedächtnis abgelegt Logik zum Test auf und erhält das Ergebnis sobald der Test beendet ist. Somit besteht auch keine Notwendigkeit die Logik zum Test in die Applikation des Technikers zu integrieren, wodurch es möglich ist, diese Anwendung für beliebige Objekte zu nutzen, die den Test selbständig durchführen können (siehe Abbildung 6.9).

**Szenario „Autonomes Testen"** In diesem Anwendungsfall führt das Objekt den Integritätstest eigenständig durch. Das heißt, das Objekt startet die Testlogik eigenständig in einem festen Zeitintervall und vermerkt das Resultat des Tests im Objektgedächtnis. Somit ist es nicht notwendig den Test erst bei Bedarf zu starten, sondern es genügt beim Zugriff auf das Gedächtnis das Auslesen des Ergebnisblocks, um das Resultat des Integritätstests zu erhalten. Zusätzlich wird automatisch eine E-Mail versendet, sobald festgestellt wurde, dass gewisse Kriterien verletzt wurden und die Integrität des Objekts nicht mehr sichergestellt ist (siehe Abbildung 6.10).

[7] http://www.restlet.org/ [Letzter Zugriff: 26.11.2012]

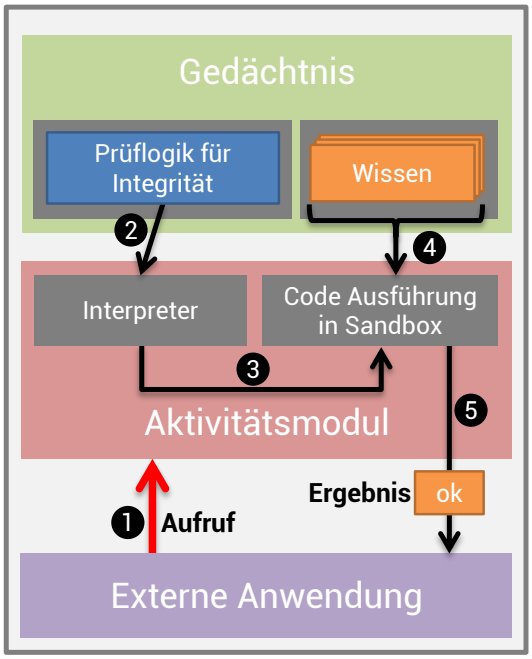



Abbildung 6.9: OMS-Aktivitätsmodul mit im Objektgedächtnis abgelegter Logik und externem Aufruf

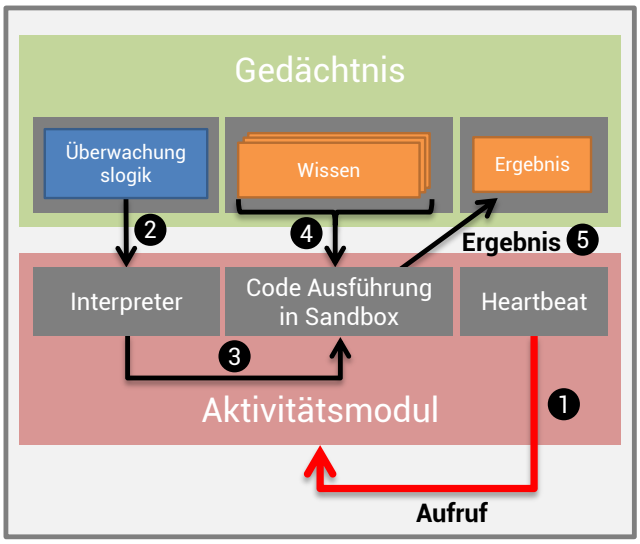



Abbildung 6.10: OMS-Aktivitätsmodul mit im Objektgedächtnis abgelegter Logik und zeitgesteuertem Aufruf

**Szenario „Angepasstes Testen“** Im dritten Fall wurde das Objekt an einen weiteren Kunden verkauft. Dieser hat zusätzliche und veränderte Anforderungen an einen Integritätstest. Diese Logik soll und kann allerdings nicht im Gedächtnis abgelegt werden, da der Test zum einen nur einmalig ausgeführt wird und zum anderen unter Umständen nicht öffentlich einsehbar ist. Daher erlaubt es das Objektgedächtnis Logik von Extern aufzunehmen, auszuführen und das Ergebnis zurückzuliefern. In diesem Fall übermittelt der Techniker nicht den Befehl einen vorhandenen Logikblock zu starten, sondern liefert den Code direkt beim Aufruf an das Objektgedächtnis mit (siehe Abbildung 6.11).

**Szenario „Intelligenter Werkstückträger“** In diesem zusätzlichen Szenario werden die Werkstückträger in einer Fabrikanlage mit aktiven Objektgedächtnissen ausgestattet (siehe Abbildung 6.13). Mit Hilfe der im Gedächtnis abgelegten Logikmodule, kann der Werkstückträger aufgrund von Daten in seinem Gedächtnis selbstständig Entscheidungen treffen und diese über das Gedächtnis nach außen kommunizieren. Dazu werden die Daten in seinem Gedächtnis im Sinne eines *Produktionssystems* genutzt [KLN87]. Dabei dienen im Gedächtnis neu abgelegte Informationen als Fakten für die Logikmodule. Diese berechnen mit den Fakten neue Ergebnisse und legen diese ebenfalls im Gedächtnis aus, worauf dann externe Anwendungen in den folgenden Verarbeitungsschritten wieder Zugriff erhalten und das Objekt dadurch diese Anwendungen steuern und parametrieren kann (siehe Abbildung 6.12).

Diese vier Szenarien bilden nun die Grundlage für eine Aktualisierung und Erweiterung der Infrastruktur.

### Aktivitätsmodule

Ausgehend von den im letzten Abschnitt definierten Anforderungen wurde der OMS um eine Aktivitätskomponente ergänzt [HHK12]. Diese besteht aus mehreren Bestandteilen, die die einzelnen Verarbeitungsschritte repräsentieren. Die Logik selbst ist nicht hartverdrahtet, sondern kann dynamisch geladen werden. Wie bereits in Kapitel 6.5.2 angesprochen, besteht die Möglichkeit diese Komponente über die REST-Schnittstelle anzusprechen (analog zur Speicher- und Verwaltungskomponente).

Im Folgenden werden nun die Bestandteile und Komponenten des Aktivitätsmoduls genauer erläutert (siehe auch Abbildung 6.14) und gezeigt wie dieser realisiert wurde.

**Interpreter und Sandbox** Die eigentliche Aktivitätslogik wird in einer Skriptsprache definiert, welche eine plattformunabhängig Formulierung der Logik erlaubt. In der

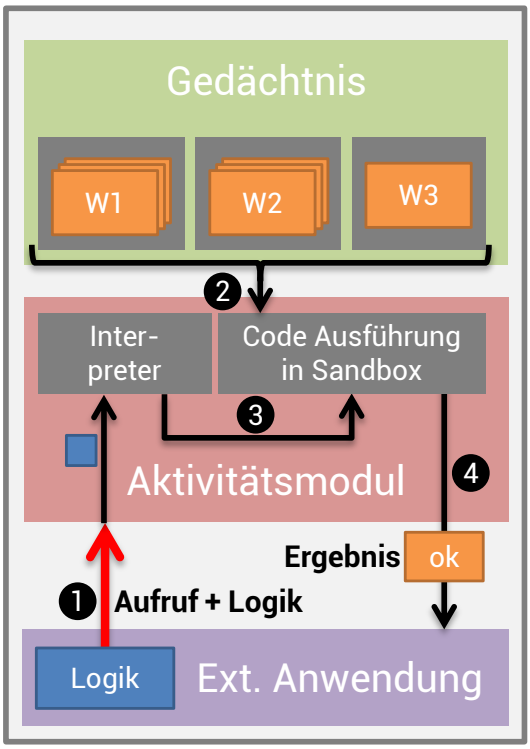



Abbildung 6.11: OMS-Aktivitätsmodul angesteuert durch die Übergabe einer externen Logik

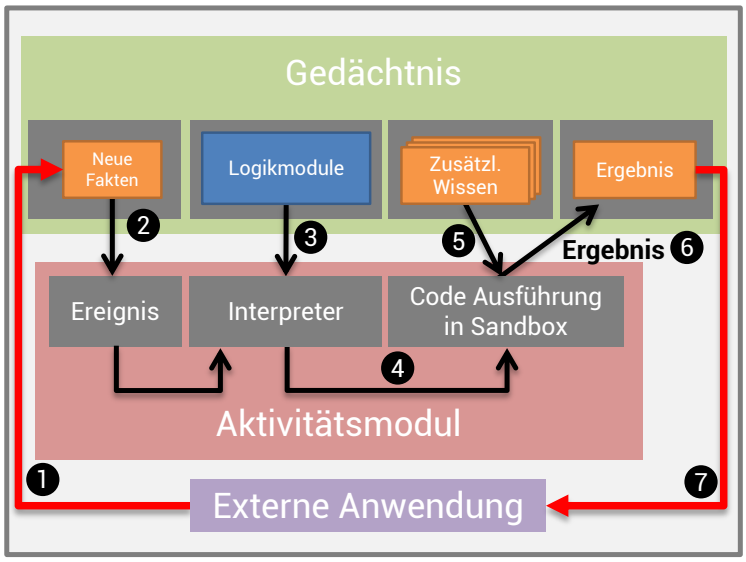



Abbildung 6.12: OMS-Aktivitätsmodul angesteuert durch neue Fakten generiert von einer externen Logik

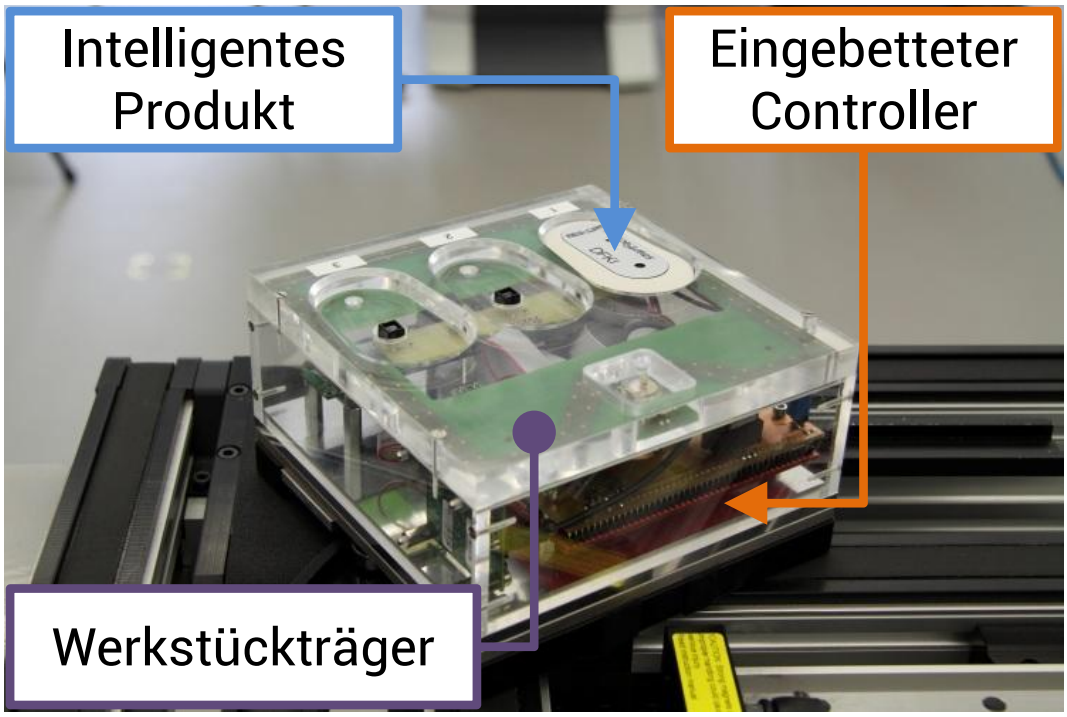



Abbildung 6.13: Instrumentierter Werkstückträger in der intelligenten Fabrik

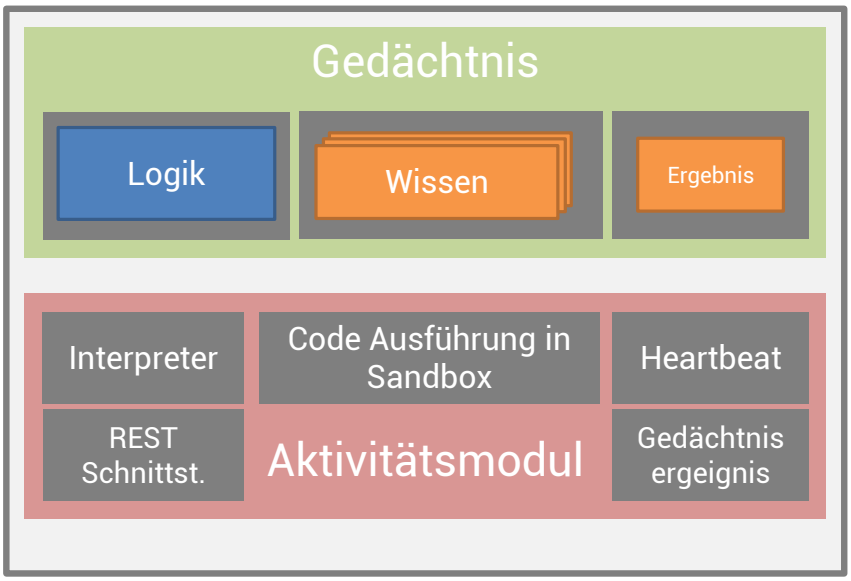



Abbildung 6.14: Komponenten eines OMS-Aktivitätsmoduls

ersten Implementierung verwendet der OMS die Sprache LUA[8], deren Entwicklungsschwerpunkt darin lag, bestehende Frameworks mit einer Skriptsprachen für Erweiterungen oder Automatisierungen zu ergänzen [IdFC06]. Die Sprache bietet mit Hilfe von sogenannten Benutzermodulen die Möglichkeit auf unterschiedliche Funktionalitäten (zum Beispiel Dateisystem, Netzwerk, SQL-Datenbanken, XML-Dateien) zuzugreifen. Diese werden in der OMS-Implementierung allerdings nicht zur Verfügung gestellt, damit keine unerlaubten Zugriffe auf das Host-System erfolgen können. Während der Ausführung des Codes stehen lediglich Module zum Zugriff auf das jeweilige Gedächtnis und zur Nutzung von fest definierten Kommunikationsstrukturen (zum Beispiel das Versenden einer Mail) zur Verfügung. Auf Implementierungsseite verwendet der OMS zwei Komponenten: der *Interpreter* verarbeitet den auszuführenden Lua-Code und die *Sandbox* bringt diesen zur Ausführung. Quellcode 6.10 zeigt eine Beispiellogik in Lua.

**Quellcode 6.10: Lua-Beispielcode für einen simplen Integritätstest**

```
1  -- This run-function is supposed for the trigger task
2  -- that gets called each time the watched block is updated.
3  -- It will react to a change in the block by reading its payload
4  -- and testing if the temperature exceeds the fixed value of 50.
5  -- If so, an EMail is sent warning the user of the overheating.
6  function run()
7    allBlocks = dfki.getBlockIDs()
8    thresholdBlocks = dfki.nameSpaceFilter(allBlocks, "urn:adome:example:
         threshold");
9    threshold = tonumber(dfki.getPayload(thresholdBlocks[next(thresholdBlocks)]))
10
11   -- Retrieve the current payload of the watched block as a number
12   current = tonumber(dfki.getPayload("watchedBlock"))
13   print("Current temperature is: "..tostring(current))
14
15   -- Check if the temperature is greather than 50 and no EMail was sent before.
16   if (current > threshold and tonumber(dfki.getPayload("emailindicator")) == 0)
17     then sendMail()
18   end
19 end
20
21 function sendMail()
22   allBlocks = dfki.getBlockIDs()
23   print("Sending an EMail about the system overheating.")
24   messageBlocks = dfki.nameSpaceFilter(allBlocks, "urn:adome:example:warning")
25   message = dfki.getPayload(messageBlocks[next(messageBlocks)])
26
27   recipients = dfki.nameSpaceFilter(allBlocks, "urn:adome:example:recipients")
28   for i, v in ipairs(recipients) do
29     dfki.mail(dfki.getPayload(v), "ADOMe warning.", message)
30   end
31
32   dfki.putPayload("emailindicator", 1)
```

[8] http://www.lua.org/, [Letzter Zugriff: 26.11.2012]

```
33 end
```

Wie bereits in den Beispielen beschrieben, werden zwei unterschiedliche Verfahren angeboten, um den zu verwendeten Code zu aktivieren. Im ersten Fall kann der Code in einen Block des Gedächtnisses abgelegt werden. Dieser Block kann mit Hilfe des *Namespace* `urn:adome:block:featuremodule:lua` eindeutig identifiziert werden. Zusätzlich wird dem Skript eine ID vergeben, mit deren Hilfe es aufgerufen werden kann. Diese ID wird im *Subjekt*-Attribut abgelegt und folgt der Form `lua.<ID>` (zum Beispiel „lua.testScript“). Liegt der Code im Gedächtnis vor, kann dieser direkt wieder zur Ausführung gebracht werden, auch wenn das OMS-System zeitweilig nicht aktiv war (zum Beispiel nach Stromausfall oder auf einem eingebetteten System). Die zweite Möglichkeit erlaubt es Code von extern zusammen mit einer Anfrage zu übermitteln. Dieses Verfahren wird im folgenden Abschnitt beschrieben. Zusätzlich legt die Aktivitätskomponente einen eigenen Verwaltungsblock an, in dem zum Beispiel definiert wird, welche Logik Zeit- oder Ereignisgesteuert ausgeführt werden soll (siehe Quellcode 6.11). Dieser Block kann anhand des *Namespace* `urn:adome:block:configuration` eindeutig identifiziert werden. In den Nutzdaten ist für jedes aktive Skript (mit Heartbeat oder Ereignisgesteuert) ein Eintrag vorhanden. Dieser Eintrag legt bei einem Heartbeat-Skript den Namen des Skripts fest und in welchem Intervall dieses ausgeführt wird. Bei einem ereignisgesteuerten Skript (Triggerscript) wird hingegen festgelegt, ob das gesamte Gedächtnis oder nur eine spezieller Block überwacht wird (dann wird dieser angegeben).

**Quellcode 6.11: Lua-Konfigurationsdaten in OMM-Block**

```
1 <omm:namespace>urn:adome:block:configuration</omm:namespace>
2 <omm:format>text/xml</omm:format>
3 <!-- Other OMM-Metadata -->
4 <omm:payload omm:encoding="none">
5   <omm:heartbeattask omm:interval="2" omm:enabled="true">
6     <omm:luaexecution>counterScript</omm:luaexecution>
7   </omm:heartbeattask>
8   <omm:triggertask omm:enabled="true" omm:blockid="watchedBlock" />
9 </omm:payload>
```

**REST-Schnittstelle** Um die Aktivitätsmodule für externe Anwendungen erreichbar zu machen, können diese über die gemeinsame OMS-REST-Schnittstelle angesprochen werden. Die Schnittstelle bietet die Möglichkeit einzelne Modelle, die im Gedächtnis abgelegt sind, zu aktivieren oder zusammen mit dem Aufruf einen Code zu übermitteln, der dann einmalig ausgeführt wird. Je nach Eigenschaften der Logik ist es auch möglich beim Aufruf zusätzliche Parameter zu übergeben und einen Rückgabewert zu erhalten.

- `.../execute/<script_id>/`

  - GET: Führt das Skript mit der ID `script_id` aus. Liefert das Skript einen Rückgabewert, so wird dieser an den Aufrufer zurückgeliefert.
  - PUT: Führt das Skript mit der ID `script_id` aus und übergibt die mit dem PUT-Kommando mitgelieferten Daten als Argumente an das Skript.

**Quellcode 6.12: Parameterübergabe an Aktivitätsskript als JSON-Datei**

```
1 {
2   "key": "sampleKey",
3   "value": "sampleValue",
4   "amount": 1
5 }
```

- `.../executeExternal/`
  - PUT: Führt das Skript, welches mit dem PUT-Kommando mitgeliefert wurde, aus und liefert (falls vorhanden) den Rückgabewert zurück.

**Quellcode 6.13: Übergabe eines externen Skripts an das Aktivitätsmodul**

```
1 {
2   "script": "<LUA-CODE>",
3   "parameters":
4    {
5      "key": "sampleKey",
6      "value": "sampleValue",
7      "amount": 1
8    }
9 }
```

- `.../config/<script_id>/`
  - GET: Ruft die Konfigurationsdatei für das Skript mit der ID `script_id` ab.
  - PUT: Setzt die Konfiguration für das Skript mit der ID `script_id` auf die Daten, die mit Hilfe des PUT-Kommandos übermittelt wurden.

**Quellcode 6.14: Konfigurationsmöglichkeiten eines Aktivitätsskripts**

```
1 // Skript alle 2 Heartbeat-Takte ausführen
2 {
3   "HEARTBEAT": 2
4 }
5
6 // Skript bei jeder Gedächtnisänderung ausführen
7 {
8   "TRIGGER": null
```

```
9 }
10
11 // Skript bei jeder Änderung des Blocks "block123" ausführen
12 {
13   "TRIGGER": "block123"
14 }
15
16 // Skript deaktivieren
17 {
18   "DISABLE": null
19 }
```

**Heartbeat und Ereignisse** Alternativ zur Möglichkeit Aktivitätslogik von extern zu aktivieren, kann diese auch durch einen Timer oder ein Ereignis gestartet werden. Der Timer wird auch als *Hearbeat* bezeichnet und hat je nach Plattform einen unterschiedlichen (aber immer festen) Grundtakt. Die Frequenz beträgt beim OMS in der Regel 1 Takt/Sekunde. Aktivitätsmodule können darauf aufbauend so konfiguriert werden, dass eine Ausführung bei einem beliebigen vielfachen dieses Grundtaktes erfolgt. Das bedeutet, wenn ein Modul ein Mal pro Stunde gestartet werden soll, wird bei einer Frequenz von 1 Takt/Sekunde das Modul auf einen Heartbeat-Wert von 3.600 gesetzt. Auf eingebetteten Plattformen ist die Frequenz üblicherweise aufgrund geringerer Verarbeitungsgeschwindigkeit und aus Stromspargründen deutlich niedriger (z.B. nur 1 Takt/Minute). Damit eine externe Anwendung trotzdem die gewünschte Ausführungshäufigkeit erreichen kann, wird die Frequenz des Heartbeats während der Funktionsaushandlung zusätzlich über die Beschreibungsdatei kenntlich gemacht, so dass ein passendes Vielfaches gewählt werden kann. Als weitere Möglichkeit kann die Ausführung an ein fest definiertes OMS-Ereignis gekoppelt werden, so dass das Modul jedes Mal gestartet wird, wenn das Ereignis eintritt. Folgende Ereignisse sind möglich:

- Schreibender Zugriff auf ein Gedächtnis
- Schreibender Zugriff auf einen bestimmten Block eines Gedächtnisses
- Neustart des OMS

Das Neustart-Ereignis kann zum Beispiel genutzt werden um eine Aktivität zu starten, die Notwendig ist, wenn das Gedächtnis längere Zeit nicht erreichbar war.

## 6.6 Konvertierung von Gedächtnisdaten in Binärstrukturen

Als Ergänzung und Alternative zu den bisher beschriebenen OMM-Darstellungen, die mit XML, RDFa und Microdata eine sehr ausführliche und menschenlesbare Darstellungsform verwenden, gibt es Anwendungsfälle, bei denen eine kompaktere und effizientere Kodierung erforderlich ist, zum Beispiel im Bereich der eingebetteten Systeme, welche in der Regel eine geringe Leistungsfähigkeit besitzen. Im Rahmen dieser Arbeit wurden vier Unterschiedliche Ansätze untersucht, um eine kompaktere Darstellung der OMM-Informationen erzielen zu können. Diese Verfahren werden nun im Folgenden beschrieben.

### 6.6.1 Externe Konvertierung

Die einfachste Art eine kompaktere Darstellung der Daten zu erreichen ist eine nachgelagerte Konvertierung. Dabei wird das von der OMM-Implementierung generierte Klartextformat (zum Beispiel XML) durch einen weitere Konvertierungsschritt in eine lokale Binärdarstellung überführt und anschließend abgespeichert (siehe Abbildung 6.15). Dieser Ansatz hat den Vorteil, dass keine Anpassungen an der eigentlichen OMM-Implementierung notwendig sind und je nach Anwendungsfall unterschiedliche Binärkodierungen verwendet werden können. Nachteilig ist allerdings der Umstand, dass mit diesem Verfahren ein zweiter Verarbeitungsschritt notwendig wird und die (XML-)Klartextdarstellung unter Umständen erneut in ein internes Modell überführt werden muss, um eine effiziente Binärdarstellung zu erreichen. Aus diesem Grund wurde dieser Ansatz nicht weiter verfolgt.

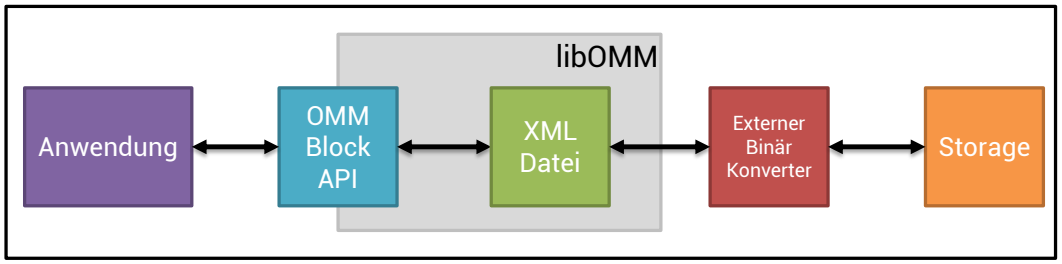



Abbildung 6.15: OMM-Binärdarstellung mit Hilfe einer externen Konvertierung

### 6.6.2 Binär XML (EXI)

Ein zweiter Ansatz sieht die Verwendung einer Bibliothek vor, die anstelle einer XML (oder RDFa/Microdata) Datei eine binäre XML-Datei erstellt. Eine mögliche Kodierung kann im

Format *Efficient XML Interchange* (EXI) erfolgen [PPG09] (siehe Abbildung 6.16). Dieses Vorgehen erlaubt eine leichte Integration in die OMM-Implementierung, da bestehende EXI-Bibliotheken für unterschiedliche Plattformen bereitstehen. Auch lassen sich aus dem gleichen Quellcode leicht unterschiedliche Varianten generieren, zum Beispiel eine Ausprägung mit XML und eine mit EXI als Ausgabeformat. Nachteilig wirkt sich allerdings die Tatsache aus, dass mit einem solchen Format die gesamte Mächtigkeit von XML in eine Binärdarstellung überführt wird. Somit müssen beispielsweise für Datentypen mit variabler Länge stets Längenangaben mitgeführt werden. Daher kann mit EXI kein maximal effizientes Format erstellt werden, es bietet sich aber als gute Kompromisslösung an.

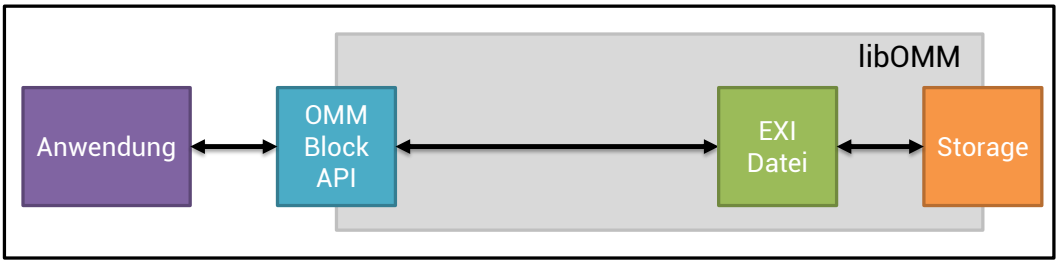



Abbildung 6.16: OMM-Binärdarstellung als binäre XML Datei (z.B. EXI)

### 6.6.3 Schema-Konverter

Eine Alternative zur binären XML-Darstellung zum Beispiel mit EXI stellt die Datenkonvertierung per angepasstem Schema da, basierend auf den Ideen aus [SMK+10, SHSS10]. Zu diesem Zweck wird für das OMM-Modell eine Schema-Datei erstellt, die eine Abbildung des Modells in eine Bit-genaue Binärdarstellung überführt (siehe Abbildung 6.17). Dieses Verfahren bietet dabei zwei Besonderheiten. Zum einen ist es möglich auch Datentypen mit variabler Länge (im OMM-Format) eine feste Größe zu geben. Dies erlaubt es auf Längenangaben zu verzichten und erzeugt gleichzeitig eine Binärdatei mit Informationen, die sich an festen Positionen wiederfinden (z.B. Block 23 beginnt immer an Byte 2048). Dies erleichtert das Parsen der Binärdaten in erheblichem Maße und stellt daher auf eingebetteten Systemen einen Vorteil dar. Zusätzlich ist es mit Hilfe des Schemas möglich, gewisse OMM-Metadaten bei der Konvertierung zu entfernen, um den Platzbedarf der Daten zu minimieren. So sind zum Beispiel die Metadaten „Format“, „Datatype“ und „Namespace“ beim Block „123“ fest definiert und können daher in der Binärdarstellung entfallen. Beim Auslesen der Daten werden die Metainformationen wieder ergänzt und es kann ein vollständiges OMM-Format in Richtung Anwendung angeboten werden.

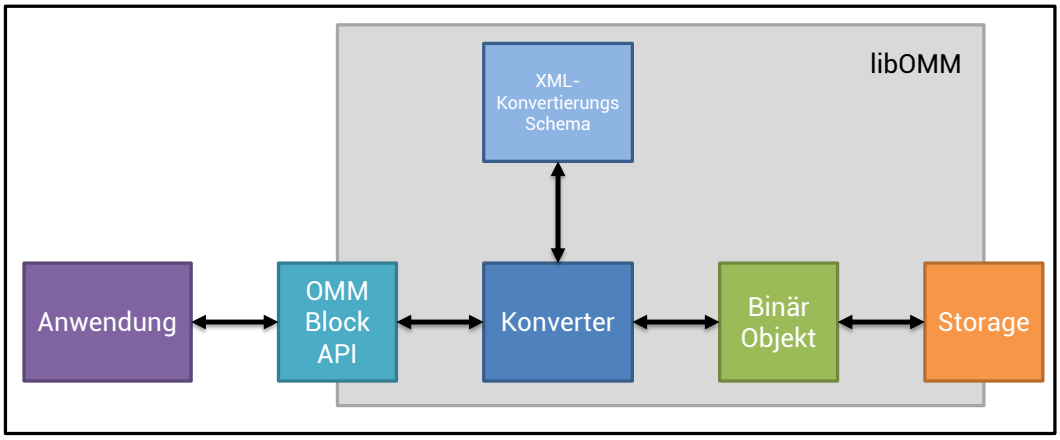



Abbildung 6.17: OMM-Binärdarstellung mit Hilfe einer Konvertierung basierend auf externem XML-Schema

### 6.6.4 Feste Binärausgabe

Eine Abwandlung des Schema-Konverters ist die Integration des Schemas in die OMM-Implementierung, so dass beide eine feste Einheit bilden (siehe Abbildung 6.18). Dies hat zwar den Nachteil, dass bei einer Änderung am Schema eine neue Implementierung kompiliert werden muss, allerdings lässt sich somit auch eine sehr kompakte Bibliothek erstellen, die ohne äußere Abhängigkeiten funktioniert, was auf extrem kleinen und leistungsschwachen Systemen von Vorteil sein kann.

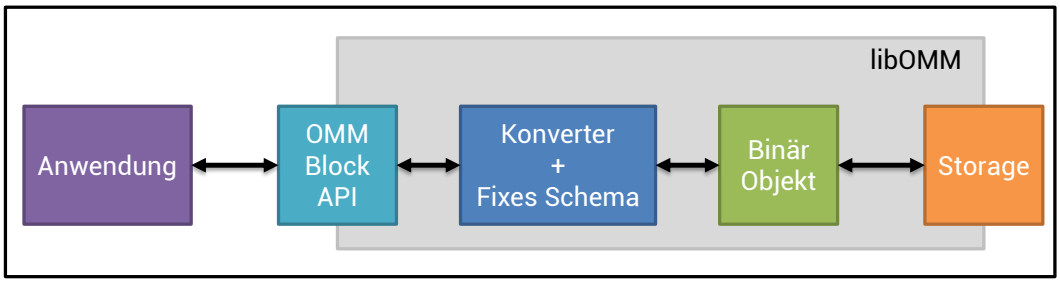



Abbildung 6.18: OMM-Binärdarstellung mit Hilfe einer fixen und hardcodierten Konvertierung

## 6.7 Visualisierung

Die vorangegangenen Kapitel beschreiben ausführlich, wie Objekte mit Gedächtnissen ausgestattet werden und wie Daten dort abgelegt und wieder entnommen werden können.

Diese Daten möchte man nun (zumindest teilweise) auch einem Benutzer, sei es eine technische Fachkraft oder ein Endverbraucher, zugänglich machen. Zu diesem Zweck wurde ein Visualisierungsframework erstellt, welches den Anforderungen von digitalen Objektgedächtnissen genügt und dessen vielfältige Möglichkeiten ausnutzen. Der folgende Abschnitt nennt nochmal die Anforderungen an ein solches Framework und erläutert anschließend die erfolgte Umsetzung.

### 6.7.1 Anforderungen an eine Visualisierung

Im Bereich von open-loop Anwendungen muss ein allgemeingültig nutzbares Framework, welches möglichst unabhängig von der jeweiligen Anwendung genutzt werden soll, einige Anforderungen erfüllen. Der folgende Abschnitt erläutert die drei wichtigsten Anforderungen:

**Erweiterbarkeit:** Eine wichtige Eigenschaft von Objektgedächtnissen ist, dass diese beliebige Daten enthalten können. In der Regel sind die verwendeten Daten, Formate und Kodierungen nicht im Voraus bekannt und können sich während der Lebenszeit eines Objekts weiter verändern. Ein Framework muss daher in der Lage sein mit unterschiedlichsten Daten umzugehen und so flexibel gestaltet sein, dass auch im Nachhinein eine Erweiterung zur Unterstützung neuer Daten und Datenformate integriert werden kann.
**Wiederverwendbarkeit:** Weiterhin soll das Framework Entwickler in der Lage versetzen Komponenten zur Datenverarbeitung und zur Visualisierung in den verschiedensten Anwendungsfällen wiederverwenden zu können, so dass sowohl der Entwicklungsaufwand minimiert wird und gleichzeitig eine konsistente Darstellung aller Anwendungen zur Verfügung steht.
**Anpassbarkeit:** Es gibt allerdings Einzelfälle, in denen man von konsistenter Standarddarstellung abweichen möchte. Dies kann zum Beispiel die explizitere Darstellung von bestimmten Objekteigenschaften sein, ein sogenanntes Hersteller-„Branding“ oder auch die Anpassung an spezielle Bedürfnisse oder Eigenschaften von bestimmten Benutzern oder Benutzergruppen.

### 6.7.2 PiVis Framework

Ein System zur Visualisierung von Objektgedächtnissen, welches die definierten Anforderungen erfüllt, ist das *PiVis*-Framework [Hau11]. Die zentrale Idee des Frameworks ist eine sogenannte *Pipeline zur Datenverarbeitung und Visualisierung*. Jede Pipeline besteht aus einer Liste von Plugins, die miteinander verkettet sind. Eine solche Kette beginnt immer mit einem sogenannten *Datenquellen*-Plugin und endet mit einem *Visualsierungs*-Plugin. Dazwischen lassen sich optional mehrere *Datenkonvertierungs*-Plugins einfügen. Der Datenfluss entsteht dabei durch die Verkettung von Datenausgabekanäle mit Dateneingabekanälen der

jeweiligen Plugins. Datenquellen liefern dabei nur Daten und besitzen daher nur Ausgabekanäle. Nur Datenkonvertierungs-Plugins bieten sowohl Eingabe- als auch Ausgabekanäle, die ankommende Daten verarbeiten und verändern und anschließend zum nächsten Plugin weiterreichen. Visualisierungen konsumieren nur Daten und bieten daher nur Eingabekanäle an. Zusätzlich lässt sich bei Visualisierungen mit einer Art Sonderkanal die Art oder der Typ der Visualisierung definieren. Jeder Datenkanal spezifiziert einen Datentyp der die Daten beschreibt, die von diesem Kanal gesendet oder empfangen werden, so dass nur passende Daten übermittelt werden. Sobald die Pipeline erstellt wird, werden die Daten automatisch, basierend auf einem „Push“-Paradigma, zum nächsten Plugin weitergeleitet. Abbildung 6.19 zeigt eine Beispielkette mit mehreren Datenkonvertierungs-Plugins.

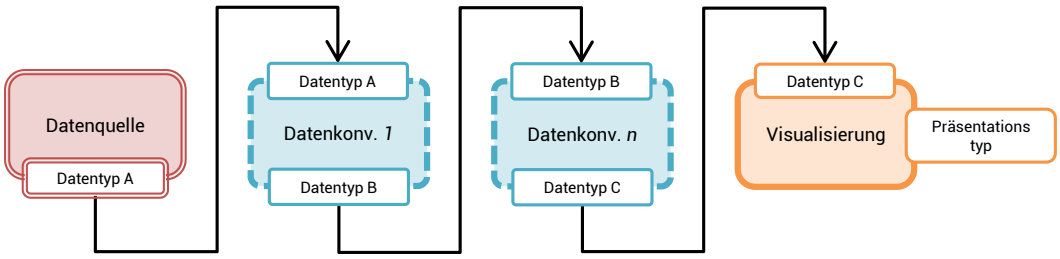



Abbildung 6.19: Beispielhafte PiVis-Datenverarbeitungspipeline

Die Plugins werden zur Erstellung einer Pipeline aus einem Repository ausgewählt, welches initial nur Plugins für grundlegende Datentypen und anwendungsabhängige Visualisierungen enthält. Während der Laufzeit des Frameworks kann das Repository durch zusätzliche Plugins erweitert werden, um zum Beispiel neue Datenquellen zu unterstützen oder spezialisierte Visualisierungen bereitzustellen. Diese können entweder aus dem Web heruntergeladen werden oder Objekte bringen diese in ihrem Speicher mit.

Für jedes zu visualisierende Objekt samt Gedächtnis erstellt das Framework immer zuerst ein Datenquellen-Plugin, welches den Inhalt des Gedächtnisses bereitstellt. Je nach Anwendung kann über eine anwendungsspezifische Konfigurationsdatei der initiale Visualisierungstyp festgelegt werden und das Framework erstellt die dazu passende Pipeline und verknüpft somit die Datenquelle mit der Visualisierung, welche wiederum weitere Visualisierungs-Plugins einbinden kann, um die gewünschte Darstellung zu erzeugen. Anstelle von festkodierten Namen von konkreten Visualisierungs-Plugins, werden diese über einen Typ aufgerufen. Daraus ergibt sich für jede Darstellung immer einen Typ der Datenquelle und einen Typ der gewünschten Präsentation. Beide sind jeweils in einer hierarchischen Struktur abgelegt (siehe Abbildungen 6.20 und 6.21), die einige Vorteile bietet (siehe nächster Abschnitt). Durch die Verwendung dieser abstrakten Daten- und Präsentationstypen lassen sich einige Spezialfälle leichter behandeln, wie zum Beispiel der Fehlerfall, dass kein Plugin für den gewünschten Fall zur Verfügung steht oder die Erweiterung zur Laufzeit durch verfeinerte oder generalisierte Typen. Sobald nun die Pipeline erstellt wurde, können alle aktiven Plugins eine Anfrage an das Framework schicken, um die Erstellung

einer neuen Pipeline (basierend auf den übergebenen Daten- und Präsentationstyp) auszulösen, wodurch eine neue Ansicht generiert wird.

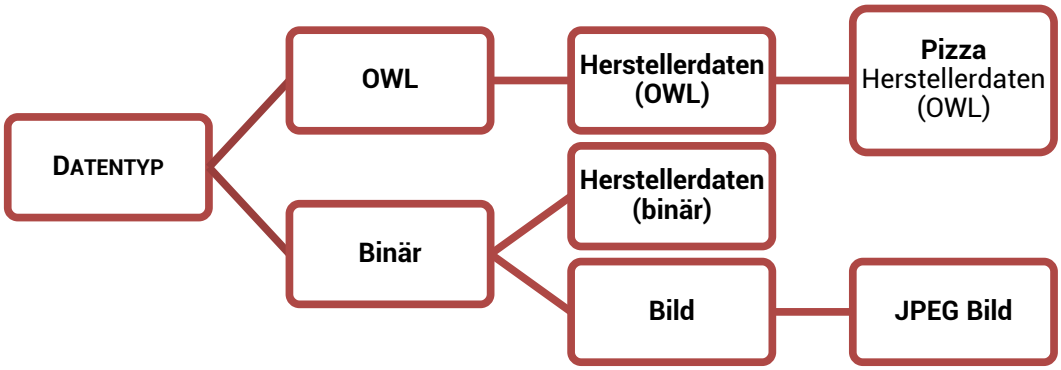



Abbildung 6.20: Beispielhierarchie von PiVis-Datentypen

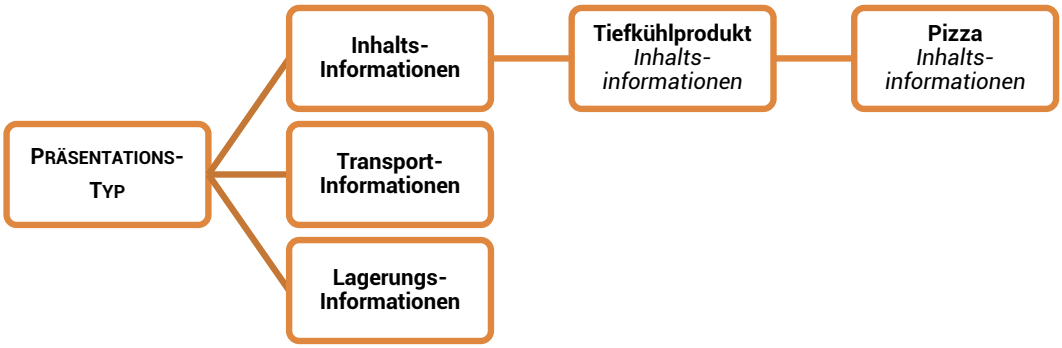



Abbildung 6.21: Beispielhierarchie von PiVis-Präsentationstypen

### Pipelineerstellung

Aufbauend auf den bereits erwähnten hierarchischen Datenstrukturen, wählt das Framework dynamisch die Plugins aus dem Repository, welche am besten auf die gestellte Anfrage passt. Die Verkettung erfolgt mit Hilfe eines einfachen *Rückwärtsverkettungs*-Algorithmus [Hau11]. Wie oben erläutert besteht jede Anfrage aus dem Typ der gewünschten Präsentation und dem Typ der Datenquelle. Das System durchsucht nun den Pluginpool nach dem am besten passenden Visualisierungs-Plugin. Falls ein Plugin gefunden wird, welches vom Typ exakt zur Präsentation und zum Datentyp passt, wird eine Pipeline aus der Datenquelle und dem gefundenen Visualisierungs-Plugin erstellt. Sollte keine exakte Übereinstimmung gefunden werden, bietet das Framework einige Rückfallstrategien um trotzdem eine Pipeline erstellen zu können.

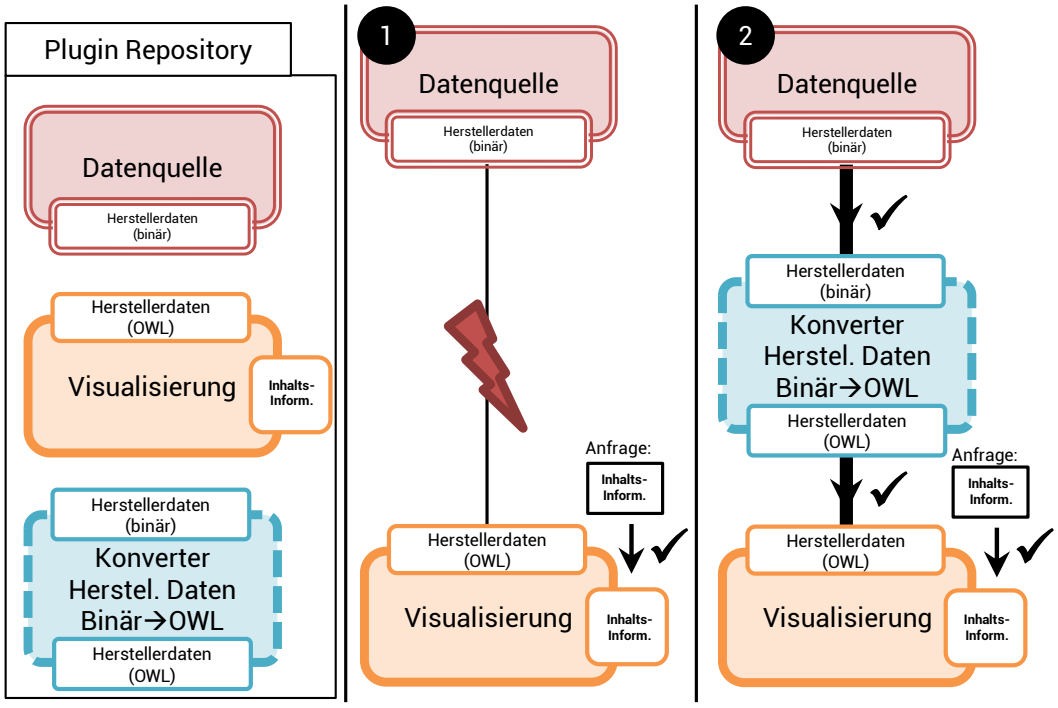



Abbildung 6.22: PiVis-Pipelineerstellung mit Hilfe von Konvertierungs-Plugins

Eine Diskrepanz zwischen Anfrage und verfügbaren Plugins wird entweder durch eine nicht passende Visualisierung oder einen nicht passenden Datentyp verursacht. Im Falle eines Problems durch einen nicht passenden Datentyp, versucht das Framework zuerst im Repository ein passendes Konvertierungs-Plugins zu finden, um die Lücke zu schließen. Dieses Vorgehen erlaubt es Objekten mit Hilfe ihrer Gedächtnisse die passenden Konvertierungsplugins mitzuliefern und somit eine Visualisierung auch mit Daten sicherzustellen, die zur Entwicklungszeit der Anwendung noch nicht betrachtet wurden. Ein mögliches Beispiel könnte wie folgt aussehen: Es besteht eine Anfrage zum Typ „Inhaltsinformationen“ und es steht eine Datenquelle vom Typ „Herstellerdaten (binär)“ zur Verfügung. Die zur Anfrage passende Visualisierung akzeptiert allerdings nur Daten vom Typ „Herstellerdaten (OWL)“. In diesem Fall kann das Framework einfach das vorhandene Konvertierungsplugin einfügen, um die Herstellerdaten von der Binär- in eine OWL-Darstellung zu überführen und die Pipeline kann erstellt werden (siehe Abbildungen 6.20 und 6.22).

Falls das Framework mit Hilfe der Konvertierungs-Plugins keine gültige Pipeline erstellen kann, wird die Hierarchie der Datentypen nutzbringend eingesetzt. Falls das Visualisierungs-Plugin einen spezielleren Datentyp (der also in der Datentypenhierarchie tiefer angesiedelt ist) unterstützt als gefordert, erstellt das Framework trotzdem mit diesem Plugin eine Pipeline, mit dem Wissen, dass das Plugin mit den allgemeineren Daten umgehen kann (siehe linke Hälfte der Abbildung 6.23). Aus diesem Grund sollen Einträge in tiefer liegenden

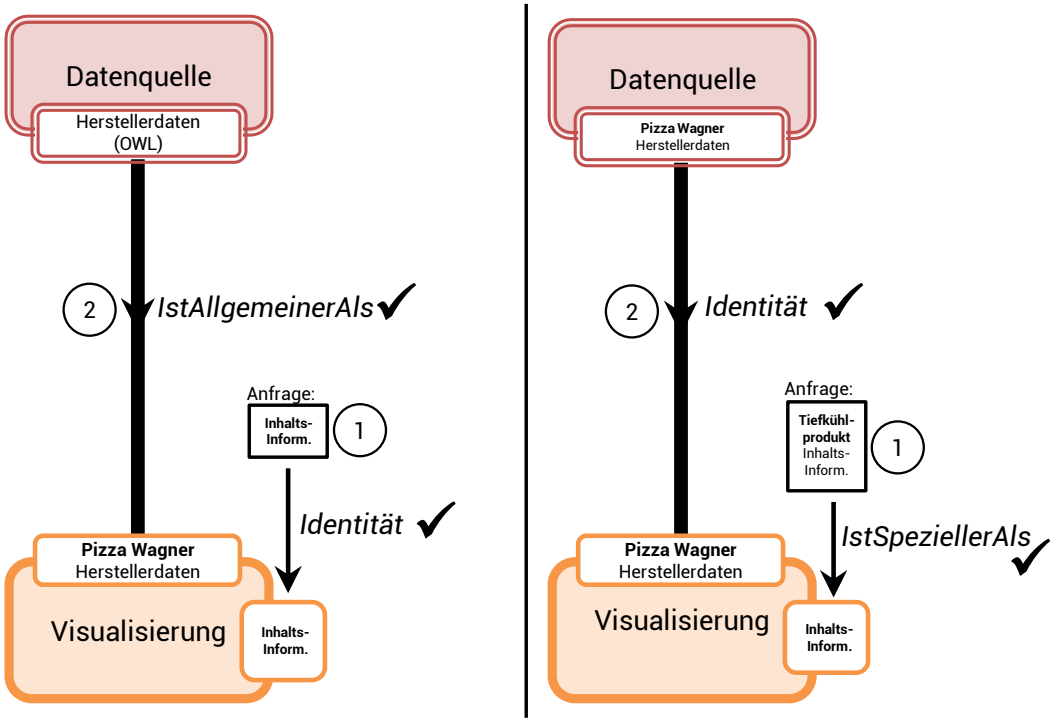



Abbildung 6.23: Rückfallstrategien der PiVis-Pipelineerstellung basierend auf der Typhierarchie für Daten- und Präsentationstypen

Teilbäumen der Datentypenhierarchie immer nur zusätzliche Daten bereitstellen. Daher beinhaltet der Datentyp „Pizza Wagner Herstellerdaten“ auch die Daten des Typs „Herstellerdaten (OWL)“ und ergänzt diese nur um zusätzliche Informationen (siehe Abbildung 6.20). Ist der Datentyp der Datenquelle hingegen spezieller als die Typen aller möglichen Visualisierungen, kann keine Pipeline hergestellt werden, da nicht sichergestellt werden kann, dass das Visualsierungs-Plugin mit den spezielleren Daten umgehen kann.

Eine Abweichung im Bereich der Präsentationstypen wird auf eine ganz ähnliche Art und Weise gehandhabt. Falls kein Plugin gefunden wird, welches dem geforderten Typ exakt entspricht, sucht das Framework nach Plugins, die eine generischere Ansicht anbieten können. Beispielsweise könnte die Ansicht „Inhaltsinformationen für Tiefkühlprodukte“ gefordert werden, wobei aber kein passendes Plugin zur Verfügung. Dann könnte das Framework auf ein Plugin vom Typ „Inhaltsinformationen“ zurückgreifen, da dieses eine generischere Ansicht bereitstellt (siehe rechte Hälfte der Abbildung 6.23). Die führt zwar zu einer Ansicht, die nicht die gewünschte Spezialisierung aufweist, aber doch zumindest einige allgemeine Informationen bereitstellen kann. Analog zu den Datentypen kommt auch im umgekehrten Fall eine Pipeline zu Stande, da ein spezielleres Plugin keine sinnvolle Ansicht zur gewünschten allgemeinen Ansicht präsentieren kann. Das Framework kann in all diesen Fehlerfällen einen Bericht über verschiedene Quellen versenden. Dazu können

im Framework selbst Empfänger definiert werden (z.B. der Anwendungsentwickler), der Bericht kann im Gedächtnis des jeweiligen Produkts abgelegt werden oder die Meldung wird als letzte Möglichkeit dem Benutzer präsentiert, der allerdings in der Regel keinen Abhilfe leisten kann, so dass die Methode wenn möglich vermieden werden sollte.

### Implementierung

Die Visualisierung mit Hilfe einer Plugin-basierten Datenpipeline in Kombination mit Fehlerbehandlungsstrategien erlaubt es Entwicklern flexible und anpassbare Anwendungen zu erstellen. Im Folgenden werden nun einige beispielhafte Anwendungen angeführt, die zeigen, wie mit Hilfe des Systems die in Kapitel 6.7.1 definierten Anforderungen umgesetzt werden können [Hau11].

Ein *Kundeninformationskiosk* ist ein Terminal, dass einen Benutzer in die Lage versetzt zusätzliche Informationen zu Produkten, die ein digitales Objektgedächtnis besitzen, abrufen können. Um einen möglichst großen Nutzwert für Kunden zu ermöglichen, muss ein solches System eine große Anzahl an unterschiedlichen Produkten unterstützen. Dies kann mit sehr allgemeinen Visualisierungs-Plugins erreicht werden, die viele Produkte unterstützen, aber nur eine sehr generische Sicht auf die vorhandenen Daten anbieten. Zusätzlich kann das System jederzeit mit zusätzlichen Plugins ausgestattet werden, die weitergehende, spezialisierte oder produktabhängige Daten darstellen können. Diese Plugins können entweder über den Betreiber des System eingespielt werden oder durch die einzelnen Produkte selbst mit eingebracht werden (→ *Erweiterbarkeit*).

*Wenige Plugins - Viele Anwendungen*: Mit dem flexiblen Plugin-basierten Ansatz lassen sich viele verschiedene Anwendungen mit einem kleinen Satz an Basisplugins erstellen. Es werden lediglich einige anwendungs- oder domänenabhängige Portalansichten ergänzt, die als Einstiegspunkte in die Anwendung dienen oder als Rahmen für eine eignen Ansicht fungieren, die sich aus vielen bestehenden Plugins zusammensetzt. Mit einigen generischen Plugins lassen sich für eine Vielzahl von Produkten eine allgemeine Ansicht bereitstellen, die die gleiche Darstellung für alle Produkte bietet (→ *Wiederverwendbarkeit*).

*Premium Produktinformationen*: Mit Hilfe des Frameworks ist es möglich Herstellern ein System bereitzustellen, mit dem sie ihrer eigenen Visualisierung für ihr eigenes Warenangebot erstellen können. Somit lassen sich detaillierte Ansichten genieren (im Sinne eines Alleinstellungsmerkmals), tiefergehende Informationen in den Fokus rücken oder einzigartige Charakteristika in den Vordergrund stellen. Diese speziellen Plugins werden über das Gedächtnis der Produkte zu den jeweiligen Systemen verteilt, wo sie automatisch in den Pluginpool integriert werden (→ *Anpassbarkeit*).

### Architektur

Implementiert wurde das *PiVis*-Framwork in Java 6 basierend auf dem Java Plugin Framework JPF[9], welches bereits die Funktionen wie eine Pluginabhängigkeitsprüfung und eine späte Plugin-Aktivierung (late binding) bietet. Die Beschreibung und Annotation der Daten- bzw. Präsentationstypen der jeweiligen Plugins erfolgt direkt innerhalb der JPF-Konfigurationsdatei des Plugins, wobei die Hierarchie selbst mit Hilfe einer OWL-Ontolgie modelliert wird. Zusätzlich zu den bisher erläuterten Plugintypen (Datenquelle, Konverter, Präsentation) bringt das Framework ein zusätzliches „Verwaltungs-Plugin“ mit, welches immer aktiv ist und nicht direkt in den Datenfluss der Pipeline eingebunden ist. Es beinhaltet einige häufig genutzte Basisfunktionalitäten und kümmert sich um die Generierung einer Pipeline aus den bestehenden Plugins. Um es Plugins zu ermöglichen abseits der Pipeline miteinander zu kommunizieren, steht ein sogenannter „Plugin-Kommunikationsbus“ zur Verfügung, der alle Plugins einer Pipeline miteinander und mit dem Verwaltungs-Plugin verbindet (siehe Abbildung 6.24). Mit Hilfe dieses Busses kann das aktive Visualsierungs-Plugin auch die Erstellung einer neuen Pipeline anstoßen, um die Ansicht zu ändern oder Daten eines anderen Produkts anzuzeigen.

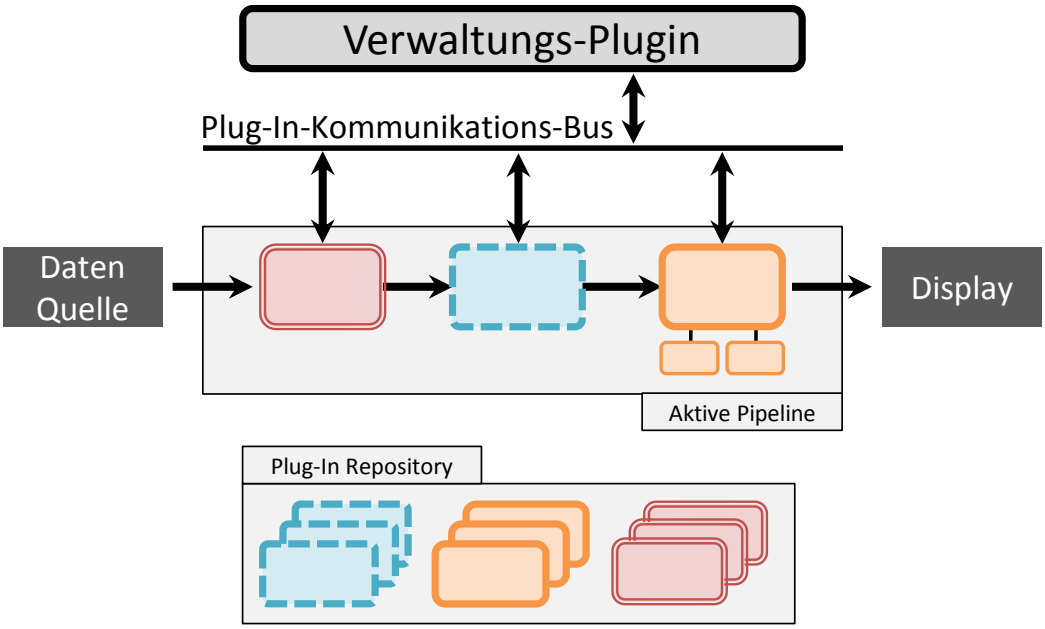



Abbildung 6.24: PiVis-Architektur mit Pluginrepository, aktiver Pipeline mit Datenquelle und Display, Plug-In-Kommunikationsbus und Verwaltungs-Plugin

[9] http://jpf.sourceforge.net/ [Letzter Zugriff: 16.05.2012]

Um die Datenweitergabe der einzelnen Plugins über die Pipeline so einfach wie möglich zu gestalten, werden die Daten in sogenannte Datenkanäle gruppiert. Jeder Kanal besteht aus einer Anzahl an Slots, wobei jeder einen bestimmten Datentyp überträgt (basierend auf der Typhierarchie aus Abbildung 6.20). Zur Übertragung von nicht-komplexen Daten bietet die Hierarchie auch einfache Datentypen wie Integer oder String. Komplexere Daten können damit auch mit Hilfe von Listen zusammengesetzt werden, wobei sowohl Listen von einfachen Typen als auch Listen von Listen möglich werden. Während der Erstellung der Pipeline werden einfach die jeweiligen Datenkanäle (basierend aus mehreren Slots) miteinander verbunden. Über Java Listener-Strukturen werden die nachfolgenden Plugins über Datenänderungen informiert und somit die Daten weiter propagiert.


Abbildung 6.25

Die Visualisierung selbst wird mit Hilfe eines vektorbasierten Szenengraph, basierend auf dem JavaFX-Framework, umgesetzt. Mit diesem Ansatz können Visualsierungs-Plugins ohne Probleme die Ansichten von anderen Plugins in ihre eigene Darstellung integrieren, in dem sie den Graph des andere Plugins in ihren eigenen Graphen an die entsprechende Stelle einhängen (siehe Abbildung 7.11 mit einer kombinierten Darstellung von Produktdetails aufbauend aus vier einzelnen Modulen). Um es Entwicklern zu ermöglichen einen konsistenten Look&Feel bereitzustellen, bietet das Verwaltungs-Plugin eine Anzahl an vordefinierten grafischen Komponenten, Schriften und Farben, die von allen Plugins genutzt werden können.

Weitere Details zu allen auf dem PiVis-Framework basierten Anwendungen finden sich in Kapitel 7.

### 6.7.3 Mobiler Gedächtniszugriff

Zusätzlich zu den in Kapitel 7 beschriebenen spezialisierten Anwendungen, wurde auch eine Android-basierte mobile Anwendung erstellt, die es erlaubt auf die Rohdaten eines Gedächtnisses zuzugreifen. Die Darstellung und der Funktionsumfang umfassen dabei den gleichen Gehalt wie die Web-basierte Darstellung des OMS. Es können die Blöcke des Gedächtnisses dargestellt werden und Metadaten und Nutzdaten können bearbeitet werden. Zusätzlich ist auch eine einfache Kontrolle der Aktivitätskomponenten möglich.

Zusätzlich zur Browser-basierten Lösung, bei deren Einsatz stets die URL des jeweiligen Gedächtnisses bekannt sein muss, ist es möglich bei der mobilen Lösung zusätzlich optische Marker (wie zum Beispiel QR-Codes) und Funklabels (wie zum Beispiel NFC/RFID-Tags) zu nutzen, um das zum Objekt passend Gedächtnis zu laden.

Die Abbildungen 6.26 und 6.27 zeigen die Anwendung auf eine 10.1“-Tablet und einem 4.5“-Mobilgerät.


Abbildung 6.26: Android OMM-Browser auf einem 10.1“-Tablet


Abbildung 6.27: Android OMM-Browser auf einem 4.5“-Mobilgerät

## 6.8 Fazit

In diesem Kapitel werden nun zusätzliche Werkzeuge vorgestellt, die es Entwicklern ermöglichen in Form eines Baukastens eigene Infrastrukturen und Anwendungen für Objektgedächtnisse aufsetzen zu können ($R_T 3$).

Diese Werkzeuge umfassen dabei den Bereich der Datengenerierung. Kapitel 6.2 zeigt dabei eine Methode zur Konvertierung von bestehenden Datenbank-basierten Daten zu Objektgedächtnissen ($R_T 2$). Mit Hilfe des leichtgewichtigen Ontologieeditors Leo lassen sich auch von Nicht-Experten semantische Daten in Form von Ontologien mit Daten füllen (Kapitel 6.3, $R_T 1$). Schließlich bietet der in Kapitel 6.4 vorgestellte Datenvalidator die Möglichkeit alle Objektgedächtnisdaten, die mit den hier vorgestellten Werkzeugen als auch solche die von externen Quellen vorliegen, auf syntaktische und semantische Wohlgeformtheit zu überprüfen.

Der zweite große Bereich bildet die Server-basierte Speicherung von Objektgedächtnisdaten. Kapitel 6.5 führt den Objekt Memory Server (OMS) ein, der diese Aufgabe in der DOMeMan-Architektur übernimmt ($R_S1$, $R_S2$, $R_S3$). Dabei wird zunächst eine funktionale Übersicht aller Bestandteile des OMS präsentiert. Anschließend werden Schnittstellen zur Kommunikation von externen Anwendungen mit dem OMS gezeigt ($R_S4$). Danach werden die Funktionen der Versionsverwaltung von Gedächtnissen und der Rechte- und Rolle-basierte Zugriff erläutert und die Implementierungsarbeiten belegt ($R_S5$). Abschließend werden die Möglichkeiten des OMS zur Bereitstellung von aktiven Komponenten für Objektgedächtnisse eingeführt ($R_S6$).

Der Bereich der Datenvisualisierung (Kapitel 6.7) bildet den Abschluss dieses Kapitels. Darin wird das modulare und Plugin-basierte Visualisierungsframework PiVis und eine Android-basierte App zum mobilen Zugriff vorgestellt. Das PiVis-Framework bietet dabei eine anwendungsorientiere Sicht auf Objektgedächtnisdaten. Mit Hilfe der Plugin-basierten Module ist es dabei möglich, das System jederzeit durch Hinzufügen oder durch Austausch von Plugins zu erweitern und zu verändern ($R_V1$, $R_V4$). Zusätzlich kann mit Hilfe dieses Konzepts adaptiv auf heterogene und inkonsistente Datensätze flexibel reagiert werden ($R_V2$). Gleichzeitig kann dem Benutzer einen anwendungsübergreifenden und konsistenten Zugang zu Objektgedächtnissen ermöglicht werden, in dem große Teile des System auch für andere Szenarien mit gleichen oder ähnlichen Produkten genutzt werden ($R_V3$).

Im nun folgenden Kapitel werden konkrete Anwendungen vorgestellt, die auf der gezeigten Architektur basieren und Teile oder alle der vorgestellten Werkzeuge nutzen.

KAPITEL 7

# Anwendungen

## 7.1 Einleitung

Um die Qualitäten einer Architektur bewerten zu können, ist es entscheidend, dass eine konkrete Implementierung einer Anwendung erfolgt. Dies ermöglicht auf der einen Seite, das System an reale Gegebenheiten anzupassen und auf der anderen Seite zeigt eine existierende Umsetzung den praktischen Nutzen eines solchen Frameworks. Aus diesem Grund wurden unterschiedlichste Prototypen erstellt, die einzelne Bausteine der Architektur zu einer konkreten Anwendung formen. Grundlage aller Demonstratoren bildet das digitale Objektgedächtnis, welches je nach Anwendungsfall mit mehreren zusätzlichen Modulen und Diensten kombiniert wird. In diesem Kapitel werden nun einige System vorgestellt, die entweder als Anwendung des sogenannten SemProM-Browsers ablaufen oder im Rahmen des Forschungslabors IRL getestet und genutzt werden. Die Nutzung der Demonstratoren in diesen beiden Umgebungen bildet somit die Grundlage für die Auswahl der Demonstratoren.

## 7.2 Demonstratoren aus dem Projekt SemProM

Im Folgenden werden nun zwei Demonstratoren aus dem Projekt SemProM[1] detaillierter beschrieben, die auf der in dieser Arbeit entwickelten Infrastruktur aufbauen [Wah13a]. Beide Szenarien haben die Tatsache gemein, dass physische Objekte mit RFID-Tags ausgestattet werden und ein digitales Objektgedächtnis erhalten. Die Datenhaltung erfolgt dabei sowohl außerhalb des Produktes im sogenannten OMS als auch direkt am Produkt. Die so gespeicherten Daten werden über die PiVis-Oberfläche dem Benutzer präsentiert (siehe auch [Hau11]).

[1] Semantic Product Memory - Produkte führen Tagebuch, `http://www.semprom.org` [Letzter Zugriff: 26.11.2012]

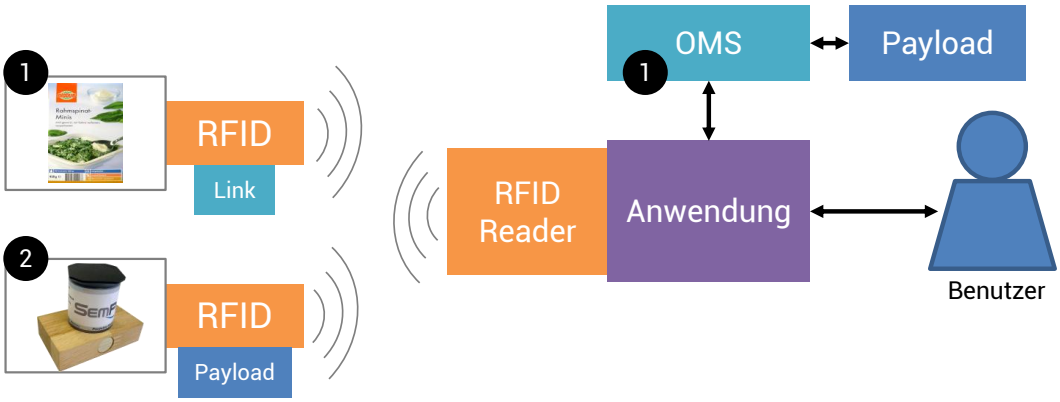



Abbildung 7.1: SemProM Demonstratorsysteme mit Link am Objekt und Gedächtnisdaten im OMS (1) oder Daten direkt am Produkt (2)

Als Zugangssystem für den Benutzer wird ein Terminal in Form eines Kiosks verwendet (siehe Abbildung 7.2). Dieses System bietet dem Benutzer zum einen eine Ablagefläche mit integrierter RFID-Antenne für ISO15693-Tags [Int00b] und zum anderen eine NFC-Antenne für ISO14443-Tags [Int00a]. Zusätzlich ist ein Touchscreen installiert, mit dessen Hilfe Daten aus dem Objektgedächtnis angezeigt werden und mit dem der Benutzer interagieren kann [BKH12a]. Als interne Softwarekomponenten wird das PiVis-Framework verwendet. Als Zusatzkomponente kann ein Lesegerät für den neuen Personalausweis (nPA) installiert werden, um mit Hilfe des nPA den Benutzer authentifizieren zu können.

### 7.2.1 SemProM-Objektdatenvisualisierung

Der erste Demonstrator beschäftigt sich mit der Frage, wie eine Umsetzung aussehen kann, die es einem Benutzer erlaubt auf Daten aus einem Objektgedächtnis zuzugreifen. Mit dem gleichen Aufbau werden dabei zwei unterschiedliche Aspekte auf das Gedächtnis abgebildet. Im ersten Fall fungiert als Objekt ein Holzklötzchen als Träger, der im Rahmen des Szenarios mit einem Thermometer, einem Spitzer oder einer Pillendose bestückt werden kann, welche noch mit unterschiedlichen Tabletten befüllt werden kann. Der Benutzer kann zu Beginn des Szenarios die Art der Bestückung/Befüllung festlegen und das Objekt durchläuft anschließend den realen Produktions- und einen simulierten Transportprozess. Schließlich werden die Daten, die bis zur Auslieferung an den Kunden gesammelt wurden, mit Hilfe des Kiosksystems dem Benutzer präsentiert [NHFB11].

**Involvierte Werkzeuge (Szenario 1)** Die Datenhaltung erfolgt in diesem Szenario in der ersten Phase der Produktion am Produkt selbst. Zu diesem Zweck sind die Objekte mit RFID-Chips von Siemens ausgestattet, die 112 Bytes Nutzdaten aufnehmen können.

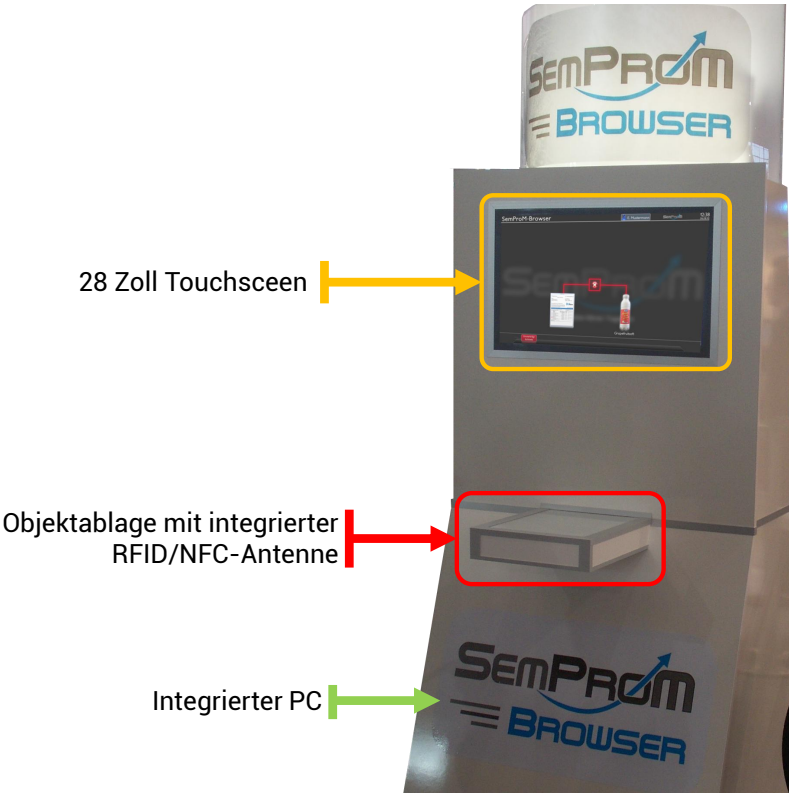



Abbildung 7.2: SemProM Demonstratorkiosk

Diese werden mit Hilfe der *Binärabbildung* und des *Validators* im Rahmen der Simulation des Transportvorgangs zu auf einen *OMS*-Server übertragen und dort mit weiteren Daten des Transports befüllt. Anschließend kann der Benutzer die Daten mit Hilfe des *PiVis*-Frameworks betrachten. Zu diesem Zweck werden folgende Plugins genutzt: „Dateneingabe via OMS (Datenquelle)“, „Datenaufbereitung (Filter)“ und „Visualisierung von Lebenszyklusdaten (Präsentation)“. Diese Darstellung zeigt im unteren Bildbereich den Lebenszyklus des Objekts im Form einer Liste von diskreten Ereignissen an, durch die der Benutzer scrollen kann. Jedes Ereignis kann angeklickt werden, wodurch im oberen Bildschirmteil Details zu diesem Ereignis als Text und Bild gezeigt werden (siehe Abbildung 7.3).

Im zweiten Fall fungiert ein Bauteil einer Industrieanlage als Objekt, das im Rahmen des Szenarios im Sinne eines Wartungsvorgangs aus der Anlage ausgebaut und durch ein anderes Bauteil ersetzt wurde. Dabei wird der gesamte Wartungsvorgang protokolliert und diese Daten sowohl in den Objektgedächtnissen der ein- und ausgebauten Bauteile, als auch im Gedächtnis der Anlage selbst vermerkt.

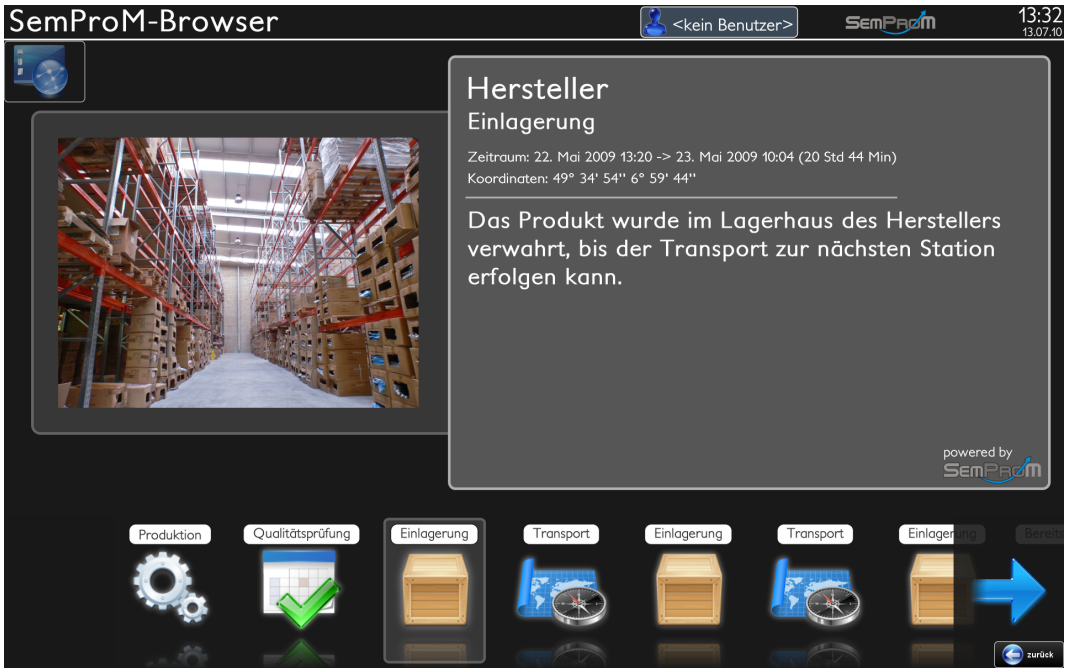



Abbildung 7.3: PiVis: Ereignisdatensicht auf den Produktlebenszyklus

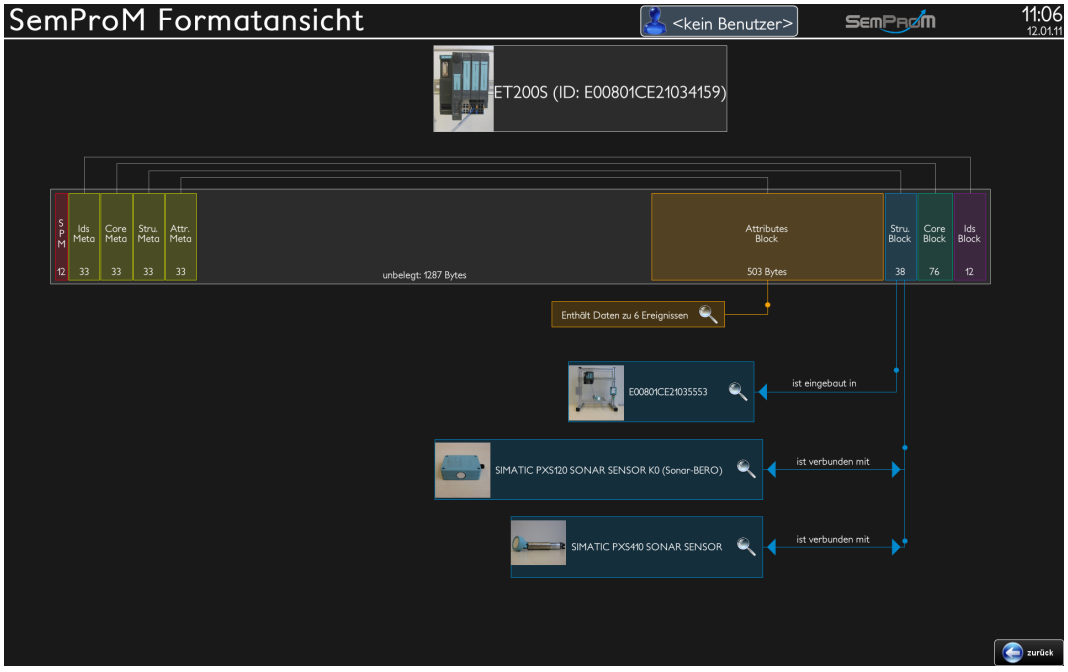



Abbildung 7.4: PiVis: Strukturansicht auf das Objektgedächtnis

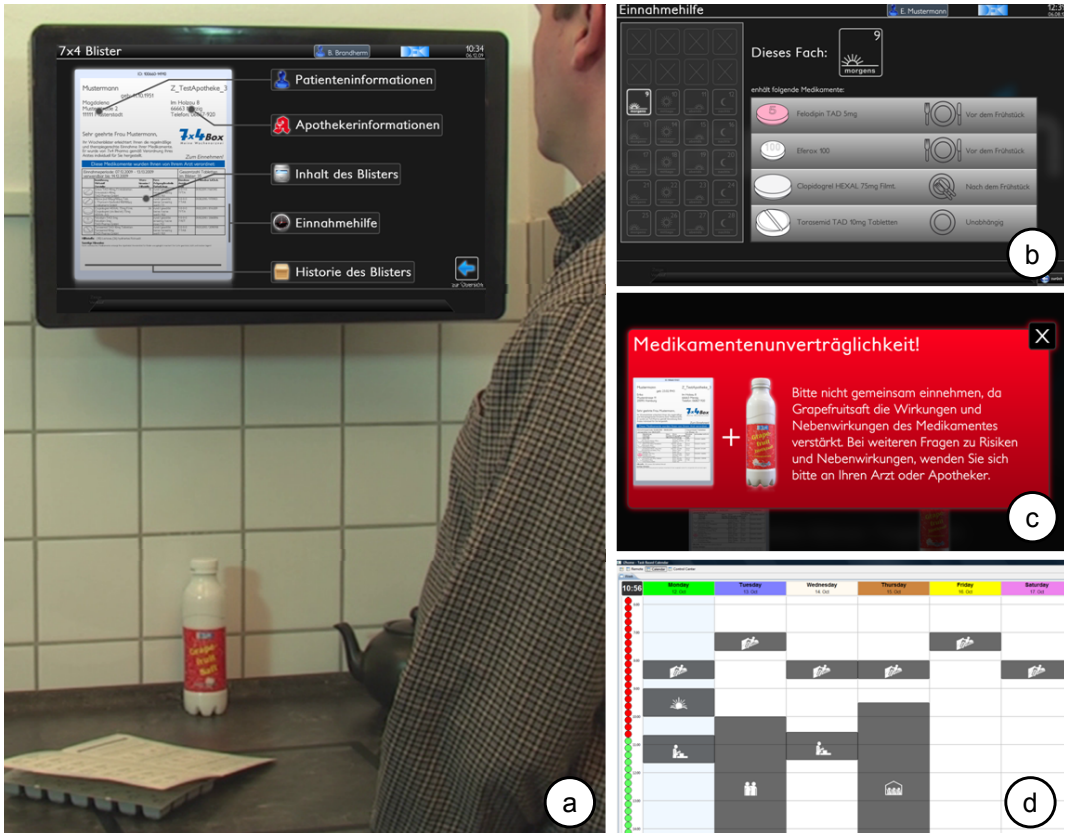



Abbildung 7.5: Medikamentenwechselwirkungsszenario

**Involvierte Werkzeuge (Szenario 2)** Die Datenhaltung erfolgt in einem zweiten Szenario ausschließlich am Produkt selbst. Zu diesem Zweck sind die Objekte mit RFID-Chips von Siemens ausgestattet die 2.000 Bytes Nutzdaten aufnehmen können. Diese werden mit Hilfe der *Binärabbildung* und des *Validators* beim Auslesen in eine OMM-kompatible Darstellung überführt und dem Benutzer mit Hilfe des *PiVis*-Frameworks angezeigt. Zu diesem Zweck werden folgende Plugins genutzt: „Dateneingabe via RFID-Chip (Datenquelle)“, „Datenaufbereitung (Filter)“, „Visualisierung von Lebenszyklusdaten (Präsentation)“ und „Strukturansicht des Objektgedächtnisses (Präsentation)“. Die Präsentation beginnt wie im ersten Fall mit der identischen Darstellung der Lebenszyklusdaten. Hierbei kann allerdings bei jedem Ereignis in die sogenannte Strukturansicht umgeschaltet werden, die den Aufbau des OMM-Gedächtnisses auf dem RFID-Chip anzeigt. Dazu wird im oberen Teil des Bildschirms eine graphische Darstellung der OMM-Blöcke angezeigt. Darunter befindet sich eine abstrakte Übersicht von Daten, die sich in den jeweiligen Blöcken befinden. Beide Komponenten sind mit Linien miteinander verbunden. Ein Klick auf die Blockinhalte erlaubt es dem Benutzer wieder in die Ereignisdarstellung zu wechseln (siehe Abbildung 7.4).

### 7.2.2 Medikamentenwechselwirkung

Der zweite Demonstrator zeigt das Konzept der Verbesserung der Einnahmetreue von Medikamenten durch die automatische Erkennung von Wechselwirkungen mit anderen Medikamenten oder Lebensmitteln [BHK+10a, BHK+10b, BKH11, KHB+11, BKS+11, BKS+12]. Zu diesem Zweck wird der Medikamentenblister von 7x4 Pharma[2] verwendet. Dieser Blister beinhaltet die Wochenration an Medikamenten für einen Patienten, die abgetrennt in kleine Gefächer aufgeteilt sind, mit den jeweiligen Tabletten für die Einnahmen am Morgen, am Mittag, am Abend und für die Nacht für den jeweiligen Tag. Zusätzlich sind die Inhaltsstoffe und die Einnahmehinweise aufgedruckt. Dieser Blister wurde nun mit einem Objektgedächtnis ausgestattet und alle bereits aufgedruckten Daten in semantischer Form im Gedächtnis abgelegt. Der Benutzer hat nun die Möglichkeit mit Hilfe eines Kiosksystems oder in einer entsprechend instrumentierten Küche oder eines Medikamentenschranks auf diese Blister-Informationen zuzugreifen (siehe Abbildung 7.5a). Um die Daten des Blisters auslesen zu können, muss sich der Benutzer zuerst am System anmelden, da die Gedächtnisdaten geschützt sind. Ist dieser Schritt erfolgt, kann das System Hilfe bei der korrekten Einnahme der Medikamente bieten, in dem angezeigt wird welches Fach genau zu diesem Zeitpunkt geöffnet werden muss (siehe Abbildung 7.5b). Des Weiteren kann das System Wechselwirkungen erkennen, die mit anderen Objekten auftreten können. Dazu wurden weitere Medikamente und Lebensmittel mit Objektgedächtnissen ausgestattet, die die Inhaltsstoffe dieser Objekte beinhalten. Mit im Gedächtnis des Blisters hinterlegten Regel kann nun die Wechselwirkung erkannt werden und dem Benutzer ein entsprechender Hinweis gegeben werden [SVH10] (siehe Abbildung 7.5c). Der Benutzer wird allerdings nicht über die Ursache der Wechselwirkung informiert, da dies durch erfahrenes Personal erfolgen soll. Zu diesem Zweck kann sich auch ein Arzt oder Apotheker in dessen Rolle am System anmelden. Dieser erhält dann zu der Wechselwirkung weitere Informationen zur Ursache, die aus einem Fachportal aus dem Internet nachgeladen werden.

**Involvierte Werkzeuge** Die Datenhaltung erfolgt beim Wechselwirkungsszenario ausschließlich mit Hilfe des OMS. Die Objekte einschließlich des Blisters sind nur mit einem einfachen RFID-Aufkleber mit 112 Byte Nutzdaten versehen. Dieser Tag beinhaltet die URL des jeweiligen OMS-Servers, der die Daten aufnimmt. Die Authentifizierung erfolgt mit Hilfe des neuen Personalausweises [Bun09, Hau12]. Weitere Details über dieses System findet sich in Kapitel 7.6.1. Die Präsentation der Daten erfolgt über das *PiVis*-Framework. Zu diesem Zweck werden folgende Plugins genutzt: „Dateneingabe via OMS (Datenquelle)“, „Datenaufbereitung (Filter)“, „Visualisierung von Blisterdaten (Präsentation)“, „Einnahmehilfe (Präsentation)“, „Wechselwirkungsanzeige (Präsentation)“ und „Visualisierung von Lebenszyklusdaten (Präsentation)“. Das System präsentiert dem Benutzer als Grunddarstellung alle erkannten Objekte. In dieser Ansicht werden auch Wechselwirkungen angezeigt,

[2] http://www.7x4pharma.de [Letzter Zugriff: 26.11.2012]

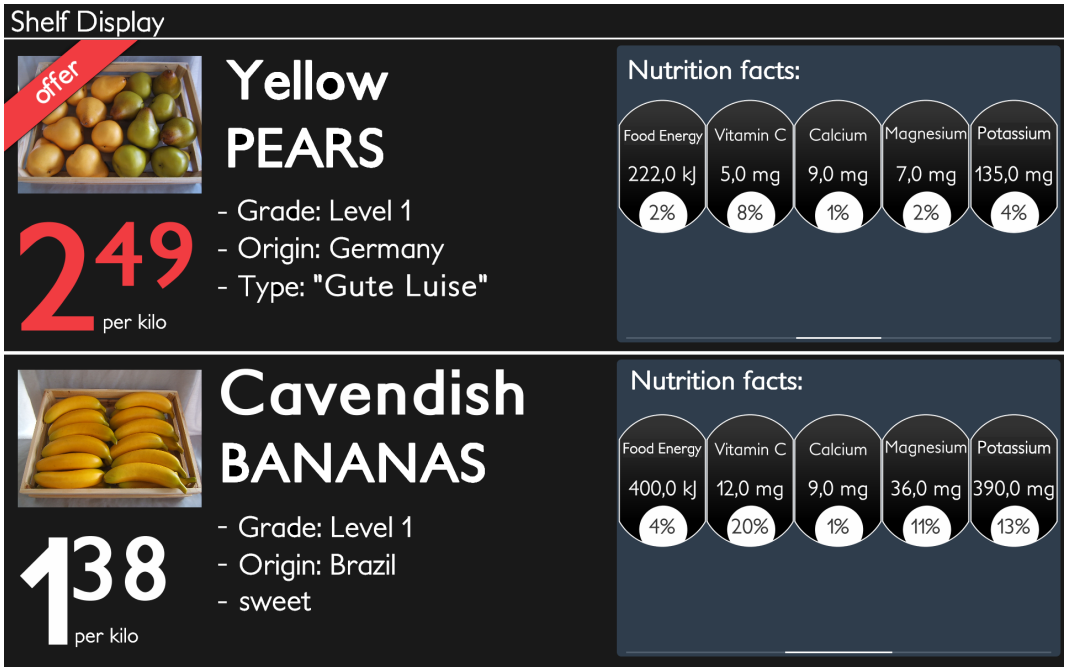



Abbildung 7.6: Produktdatenanzeige der Gemüseschräge

falls zwei Objekte mit Wechselwirkungen gleichzeitig erkannt werden. Wird ein Blister ausgewählt, können zu diesem weitere Informationen betrachtet werden, zum Beispiel die Inhaltsstoffen, Einnahmehinweise und der Produktlebenszyklus.

## 7.3 Anwendungen des Innovative Retail Laboratory

Der zweite große Demonstratorbereich beinhaltet Exponate, die im Rahmen eines Handels- und Heimszenarios erstellt wurden und auf die Wünsche des Endkunden ausgerichtet sind. Die technische Umsetzung erfolgt in enger Anlehnung an die bereits dargestellten SemProM-Exponate. Die Objekte sind durchgängig mit RFID-Tags ausgestattet, die Gedächtnisse werden ebenfalls über den OMS verwaltet und die Präsentation der Daten erfolgt mit Hilfe des PiVis-Frameworks.

### 7.3.1 Obst-/Gemüseschräge

Dieser Demonstrator beinhaltet einen mechanischen Aufbau, der es erlaubt, zur Präsentation der Waren für den Kunden, Körbe mit Obst oder Gemüse auf eine Schräge zu platzieren. Zusätzlich ist jeder Aufbau mit zwei Displays ausgestattet, die Produktdaten der

darunter platzierten Waren anzeigen. Diese Daten umfassen unter anderem eine textuelle Beschreibung, ein Bild, Nährwertangaben und der aktuelle Preis. Damit die korrekten Daten präsentiert werden, ist jeder Korb mit einem RFID-Tag ausgestattet und mit einem Objektgedächtnis verknüpft, das die Fakten zu den in diesem Korb befindlichen Waren beinhaltet. Der Benutzer hat hierbei, im Gegensatz zu allen anderen Aufbauten, keine direkte Interaktionsmöglichkeit mit dem System. Die Präsentation wird nur durch eine Veränderung der auf der Schräge platzierten Körbe angepasst. Somit erleichtert das System die Arbeit des Personals eines Supermarktes, da keine manuelle Anpassung der Beschilderung mehr erfolgen muss.

**Involvierte Werkzeuge** Die Datenhaltung erfolgt auch bei der Obst-/ Gemüseschräge ausschließlich mit Hilfe des OMS. Die Warenkörbe sind mit einem einfachen RFID-Aufkleber mit 112 Byte Nutzdaten versehen. Dieser Tag beinhaltet die URL des jeweiligen OMS-Servers, der die Daten aufnimmt. Diese Produktdaten werden mit Hilfe des Datenbank-Konverters automatisch aus der Datenbank des Marktes extrahiert. Die Präsentation der Daten erfolgt über das *PiVis*-Framework. Zu diesem Zweck werden folgende Plugins genutzt: „Dateneingabe via OMS (Datenquelle)“, „Datenaufbereitung (Filter)“, „Visualisierung von Produktdaten im Markt (Präsentation)“. Dabei wird dem Benutzer im linken Bildbereich dauerhaft der Name, ein Bild und der Preis des Produkts angezeigt. Im rechten Bereich werden abwechselnd, ein ausführlicherer Text, ein Detailbild (in diesem Fall ein Querschnittbild des Obstes bzw. des Gemüses) und eine Angabe der Nährwerte.

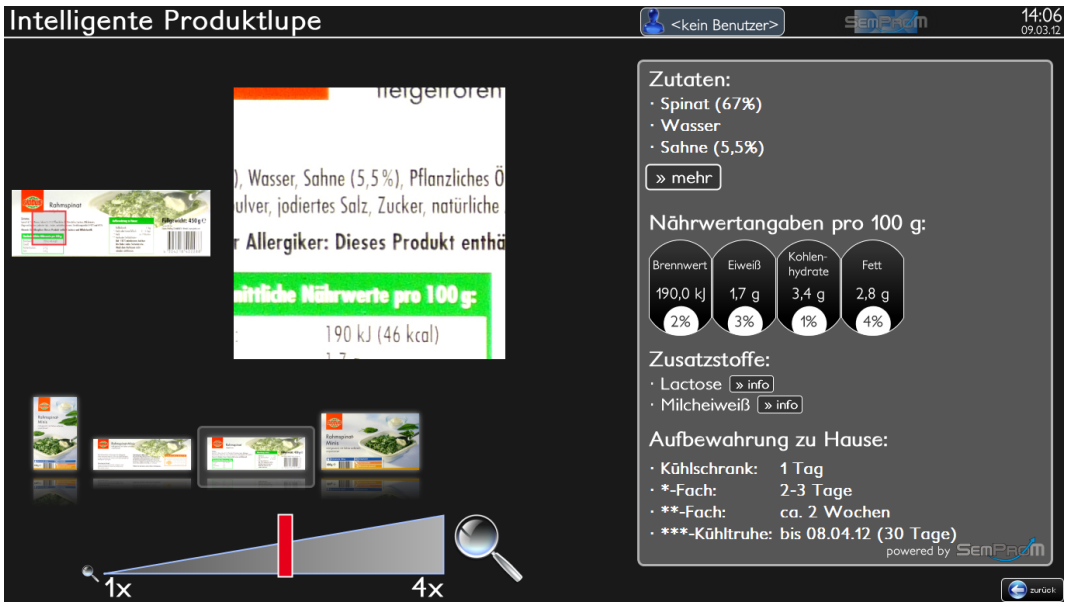



Abbildung 7.7: Produktlupe mit Anzeige weiterer semantischer Daten

### 7.3.2 Produktlupe

Die Produktlupe verwendet ein Kiosksystem analog zum bereits vorgestellten SemProM-Browser. Auch hier sind alle Objekte mit einem RFID-Tag ausgestattet, welches vom System ausgelesen werden kann. Die Daten liegen auch hier im OMS und werden abgerufen, sobald ein Objekt auf der Ablage des Kiosks platziert wird. Der Benutzer kann mit Hilfe des Systems Inhaltsstoffe und Nährwerte von Lebensmitteln abrufen, die oft nur in sehr kleiner Schrift und teilweise auch unvollständig abgedruckt sind. Zu diesem Zweck liegt ein hochauflösender Scan der Produktverpackung im Gedächtnis, so dass diese elektronisch betrachtet und gezoomt werden kann. Zusätzlich liegen alle Daten auch in semantisch modellierter Form vor, so dass die Daten auch vom System in übersichtlicher Weise präsentiert werden können.

**Involvierte Werkzeuge** Die Datenhaltung erfolgt beim auch bei der Produktlupe mit Hilfe des OMS. Die Lebensmittel sind mit einem einfachen RFID-Aufkleber mit 112 Byte Nutzdaten versehen. Dieser Tag beinhaltet die URL des jeweiligen OMS-Servers, der die Daten aufnimmt. Diese Produktdaten werden mit Hilfe des Datenbank-Konverters automatisch aus der Datenbank des Marktes extrahiert. Die Präsentation der Daten erfolgt über das *PiVis*-Framework. Zu diesem Zweck werden folgende Plugins genutzt: „Dateneingabe via OMS (Datenquelle)“, „Datenaufbereitung (Filter)“, „Lupenfunktion (Präsentation)“ und „Visualisierung von Produktdaten (Präsentation)“. Dabei wird dem Benutzer im linken Bildbereich die Produktverpackung gezeigt, wobei der Benutzer die Betrachtungsseite und die Zoomstufe frei wählen kann. Auf der rechten Seite werden die Produktdaten nochmals aufbereitet präsentiert. Dazu gehören die Inhaltsstoffe und Zusatzstoffe, die Nährwerte und Aufbewahrungshinweise.

### 7.3.3 Intelligente Kleiderkabine

Der Kleiderberater ist in eine übliche Kleiderkabine integriert und soll dem Benutzer eine Hilfestellung geben, um Kleidungsstücke zu finden die dem Farbtyp des Benutzers entsprechen [Lü77]. Dazu wird ein Display und ein Kleiderhaken mit integrierter RFID-Antenne genutzt. Zusätzlich wurden unterschiedlichste Oberbekleidungen mit RFID-Tags und OMS-Gedächtnissen ausgestattet. Der Benutzer kann sich zusätzlich mit einer ID-Karte, die ebenfalls auf RFID-Technik basiert, gegenüber dem System identifizieren. Zur Interaktion werden ein oder mehrere Kleidungsstücke am Haken aufgehängt. Das System erkennt die Kleidungsstücke und gleicht deren Farben und Größen mit den im Benutzerprofil abgelegten Daten des Kunden ab und zeigt an, ob diese Auswahl passend ist und gibt im Fehlerfall auch Alternativen an.


Abbildung 7.8: Benutzerschnittstelle des Kleiderberaters und Kabinenaufbau

**Involvierte Werkzeuge** Sowohl die Daten der Produkte als auch die Informationen zum Kunden werden mit Hilfe von OMS-Gedächtnissen semantisch abgelegt. Dazu wurden beide Datensätze mit Hilfe des Leo-Editors in eine OWL-Ontologie überführt und im Gedächtnis gespeichert. Die Textilien sind mit einem einfachen RFID-Aufkleber mit 112 Byte Nutzdaten versehen. Zur Benutzeridentifikation wird ebenfalls eine RFID-Karte genutzt, die mit Hilfe eines Bandes ebenfalls an den Kleiderhaken der Kabine angebracht werden kann. Diese Tags beinhalten die URL des jeweiligen OMS-Servers, der die Daten aufnimmt. Die Präsentation der Daten erfolgt über das *PiVis*-Framework. Zu diesem Zweck werden folgende Plugins genutzt: „Dateneingabe via OMS (Datenquelle)“, „Datenaufbereitung (Filter)“, „Visualisierung und Empfehlung von Textilien (Präsentation)“. Die Darstellung ist auch hier zweigeteilt: Auf der linken Seite werden im oberen Bereich alle mit in die Kabine gebrachten Kleidungsstücke angezeigt und der untere Bereich zeigt alternative Textilien, die auch zum Farbtyp des Benutzers passen. Sobald der Benutzer angemeldet ist, werden in der Liste der mitgebrachten Textilien automatisch alle bezüglich der Größe oder der Farbe nicht passenden Kleidungsstücke ausgegraut. Die rechte Seite ermöglicht die Wahl zwischen Informationen zu dem gerade ausgewählten Kleidungsstück und zu allgemeinen Hinweisen zum Farbtyp des Benutzers.

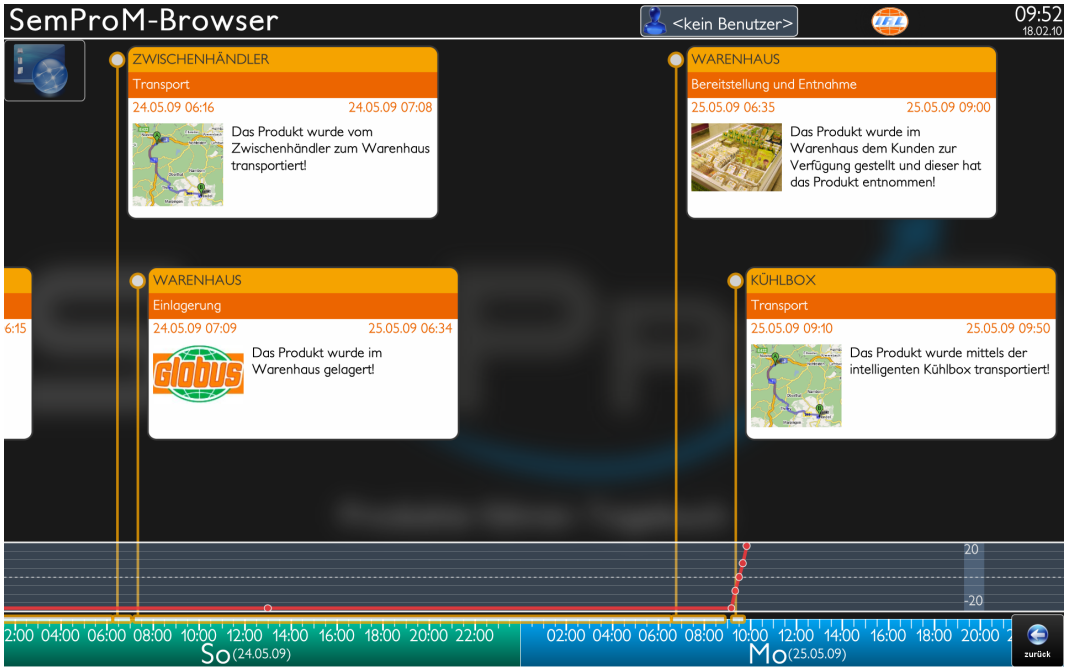



Abbildung 7.9: PiVis: Zeitstrahlsicht auf den Produktlebenszyklus inklusive Anzeige der Temperaturkurve

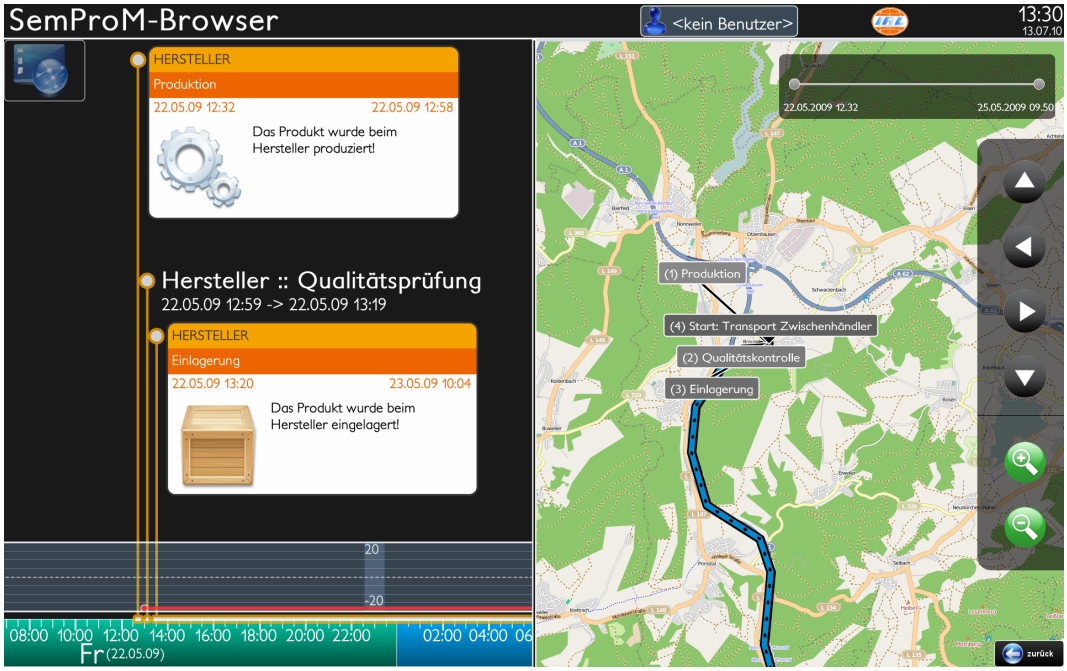



Abbildung 7.10: PiVis: Kombinierte Sicht mit Zeitstrahl und Geographischen Daten auf den Produktlebenszyklus


Abbildung 7.11: PiVis: Detaillierte Produkdatenansicht des SmartFridge

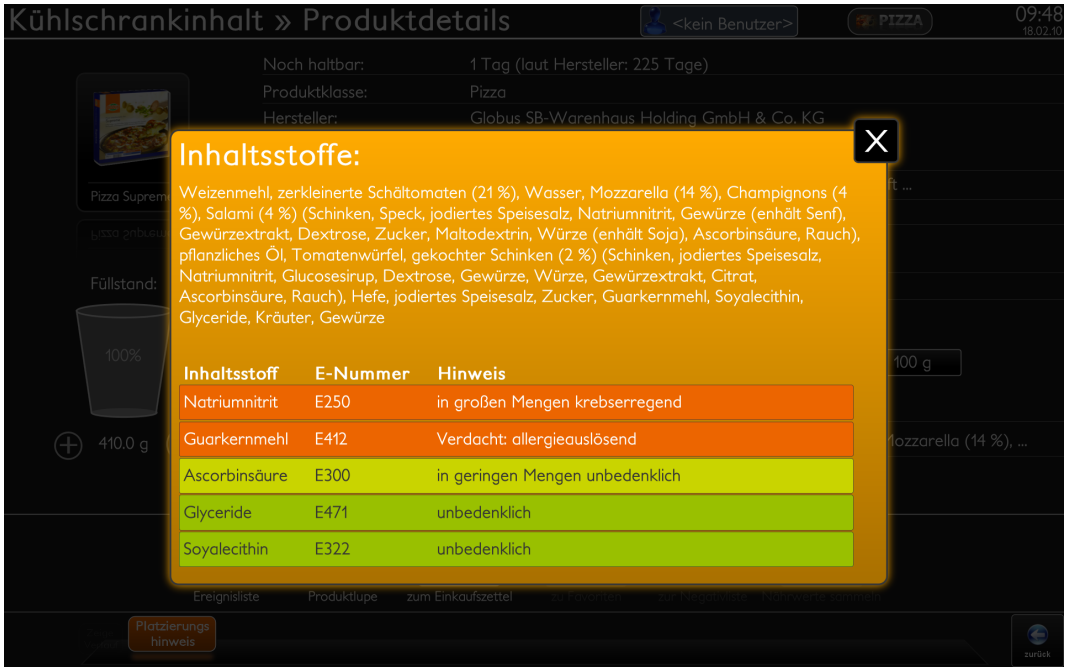



Abbildung 7.12: PiVis: Inhalststoffe einer Pizza mit Angabe und Bewertung aller E-Nummern

### 7.3.4 SmartFridge

Der intelligente Kühlschrank kombiniert viele Funktionen der bereits beschrieben Demonstratoren in einem System. Dazu wurde ein handelsüblicher Kühlschrank mit einem Rechner inklusive einem außen zugänglichen Display und einer Reihe von RFID-Antennen im Innenraum ausgestattet. Der Benutzer kann so Objekte mit RFID-Tags in den Kühlschrank ablegen der weitere Informationen und Hilfestellungen zu diesen Produkten anbietet. So werden von außen sichtbar alle Produkte angezeigt, die sich im inneren des Kühlschranks befinden und mit einen Objektgedächtnis ausgestattet sind, wobei jedes Objekt direkt die Resthaltbarkeit anzeigt. Während sich die Objekte im Kühlschrank befinden, wird deren Temperatur überwacht und Abweichungen im Gedächtnis vermerkt. Zu jedem dieser Produkte lassen sich Detailinformationen wir Inhaltsstoffe, Nährwertangaben, Zertifikate, Recycling und Haltbarkeitsdatum betrachten. Die Liste der Inhaltsstoffe wird dabei automatisch von sogenannten E-Nummern[3] befreit und diese Stoffe werden darunter (zusammen mit einer Bewertung der Bedenklichkeit der Stoffe) in einer eigenen Liste dargestellt. Des Weiteren kann manuell festgelegt werden welche Menge eines Produkts bereits verbraucht wurde und jedes Objekt kann direkt auf den virtuellen Einkaufszettel gelegt werden. Es besteht darüber hinaus ein direkter Zugriff auf die Ereignisliste und die Produktlupe, die bereits bei den beschriebenen Exponaten erläutert wurde. Die Ereignisliste wurde allerdings erweitert, so dass diese auch eine geographische Darstellung der Ereignisse anbietet, sofern solche Daten vorliegen. Das System überwacht darüber hinaus alle Objekte, die in den Kühlschrank hinein gelegt und entnommen werden und gibt somit Hinweise auf die korrekte Platzierung von Produkten und bei der Entnahme Hinweise zur Zubereitung. Als Besonderheit kann sich auch ein Benutzer am Kühlschrank anmelden. In diesem Fall kann das System erkennen welche Produkte für den Benutzer nicht geeignet sind und gibt entsprechende Warnungen aus.

**Involvierte Werkzeuge** Bei diesem Exponat werden alle Werkzeuge dieser Arbeit genutzt. Die Datenhaltung erfolgt mit Hilfe des OMS. Die Lebensmittel sind mit einem einfachen RFID-Aufkleber mit 112 Byte Nutzdaten versehen, welcher wieder die URL des jeweiligen OMS-Servers beinhaltet. Diese Produktdaten werden mit Hilfe des Datenbank-Konverters automatisch aus der Datenbank des Marktes extrahiert. Zusätzlich wurden die Daten mit Hilfe des Editors Leo semantisch aufbereitet und verfeinert. Die Präsentation der Daten erfolgt über das *PiVis*-Framework. Zu diesem Zweck werden fast alle verfügbaren Plugins genutzt, so dass mit Hilfe dieser Anwendung alle Ansichten genutzt werden können. Nicht verwendet werden lediglich die Präsentation der Daten im Modus der Obst-/Gemüseschräge und der Kleiderkabine.

[3] `http://www.aid.de/downloads/liste_zusatzstoffe.pdf` [Letzter Zugriff: 26.11.2012]

## 7.4 Demonstrator aus dem Projekt RES-COM

Im Rahmen der Initiative *Industrie 4.0*[4] wurde im Projekt *RES-COM*[5] ein Demonstratorsystem entwickelt, welches die Aufgabe hat die Idee der Cyber-Physischen-Systeme zu verdeutlichen (siehe Abbildung 7.13). Cyber-Physische Systeme sind verteilte, intelligente Objekte, die miteinander über Internettechnologien vernetzt sind. Im Bereich der Produktionstechnik kann dies z.B. einzelne Prozessmodule bis hin zu Anlagen und Betriebsmitteln aber auch individuelle intelligente Produkte umfassen. Die Anlage hat dabei die Aufgabe ein intelligentes Produkt herzustellen (Schlüsselfinder), welches mit einem Objektgedächtnis ausgestattet ist, welches alle Produktionsdaten enthält. Zusätzlich sind auch mehrere Komponenten der Anlage mit Objektgedächtnissen ausgestattet, beispielsweise die Anlage als Einheit, der Werkstückträger und die einzelnen Pressen.


Abbildung 7.13: Integrierte Demonstratoranlage des Projekts RES-COM

**Involvierte Werkzeuge** Bei diesem Exponat werden die OMM-Daten in der Binärdarstellung auf das Produkt abgelegt. Zusätzlich werden die Gedächtnisdaten der Anlage in einem OMS-Server gespeichert. Diese Gedächtnisse werden mit zusätzlichen aktiven

[4] http://www.hightech-strategie.de/de/2676.php [Letzter Zugriff: 26.11.2012]
[5] http://www.res-com-projekt.de/ [Letzter Zugriff: 26.11.2012]

Komponenten eigenständig überwacht. Zur Identifikation werden sowohl NFC-Tags als auch QR-Codes verwendet, die je nach genutztem Eingabegerät den Zugriff zu den Gedächtnissen ermöglichen. Die Gedächtnisse der Anlagenmodule verwenden einen aktiven Logikcode zur Berechnung lokaler Operationen. Mit Hilfe der Gedächtnisse werden die Eingangsdaten und Ergebnisse dieser Berechnungen zwischen den Bestandteilen im Sinne einer M2M-Kommunikation ausgetauscht. Die Darstellung der Daten erfolgt anschließend in Form einer speziellen Android-Anwendung, die mit den entwickelten Modulen zum Gedächtniszugriff und zur Gedächtnisvisualisierung ausgestattet wurde (siehe Kapitel 6.7.3).

## 7.5 Nutzung der Werkzeuge in den einzelnen Demonstratoren

Folgende Tabelle 7.1 zeigt abschließend die Nutzung der einzelnen Werkzeuge in den jeweiligen Demonstratoren. Einige Kernkomponenten wie der OMS und die Validierung werden dabei in allen Systemen verwendet. Ebenso wird die PiVis-Darstellung in allen Exponenten genutzt um dem Benutzer die Interaktion mit den Gedächtnissen zu ermöglichen.

| ↓ Demonstrator \| Werkzeug → | OMS | DB | Binär | Valid. | Leo | PiVis |
|---|---|---|---|---|---|---|
| Objektdatenvisualisierung | ✓ | – | ✓ | ✓ | – | ✓ |
| Medikamentenwechselwirkung | ✓ | – | – | ✓ | – | ✓ |
| Obst-/Gemüseschräge | ✓ | ✓ | – | ✓ | ✓ | ✓ |
| Produktlupe | ✓ | ✓ | – | ✓ | ✓ | ✓ |
| Intelligente Kleiderkabine | ✓ | – | – | ✓ | ✓ | ✓ |
| SmartFridge | ✓ | ✓ | – | ✓ | ✓ | ✓ |
| RES-COM Produktionsanlage | ✓ | – | ✓ | ✓ | – | ✓ |

Tabelle 7.1: Nutzung der Werkzeuge in den einzelnen Demonstratoren

## 7.6 Anbindung und Integration externer Infrastruktur

Im Rahmen der Umsetzung der beschriebenen Anwendungen wurden zusätzlich zu den in dieser Arbeit vorgestellten Werkzeugen weitere externe Module verwendet, um zusätzliche Technologien zu integrieren und weitere Anwendungen anzubinden. Somit konnten zum einen bereits bestehende Infrastrukturen für OMM-basierte Anwendungen genutzt werden

und zum anderen konnte die in dieser Arbeit entwickelte Architektur als Basis für weitere Anwendungen genutzt werden.

### 7.6.1 Neuer Personalausweis

Wie bereits in Kapitel 6.5.4 beschrieben, bietet der OMS die Möglichkeit mit Hilfe eines Rechte- und Rollen-basierten Konzepts den Zugriff auf Gedächtnisse teilweise oder vollständig einzuschränken. Die Authentifikation des Benutzers erfolgt in der ersten Phase mit Hilfe von Benutzername/Passwort-Paaren oder X.509-Zertifikaten. Als weitere Möglichkeit wurde in einer zweiten Phase der neue elektronische Personalausweis der Bundesrepublik Deutschland genutzt [Bun09].

**Neuer Personalausweis (nPA)** Dieser nutzt das Scheckkartenformat und enthält einen RFID-Chip, der verschiedene Formen der elektronischen Authentisierung ermöglichen soll. Folgende Informationen werden digital auf dem Personalausweis gespeichert: Familienname, Vornamen, Geburtsdatum, Geburtsort, Anschrift, Lichtbild, Seriennummer und optional zwei Fingerabdrücke. Nicht gespeichert werden die eigenhändige Unterschrift, die Körpergröße und die Augenfarbe. Dabei wird zwischen hoheitlichen und nicht-hoheitlichen Funktionen unterschieden. Auf die hoheitlichen Funktionen erhalten nur Behörden und die Polizei (bzw. der Zoll) Zugriff und können somit auf biometrische Merkmale zur Authentifizierung des Benutzers zurückgreifen, ähnlich eines elektronischen Reisepasses. Zum Zugriff auf diese Daten ist allerdings ein hoheitliches Berechtigungszertifikat und zusätzlich die Eingabe einer auf dem Personalausweis angezeigten Information (Zugangsnummer oder MRZ) Voraussetzung. Die zusätzlich implementierten nicht hoheitlichen Funktionen können von Dritten genutzt werden, um eine elektronischen Authentisierung zum Beispiel über das Internet zur realisieren. Diese Funktionen bestehen aus den zwei Bereichen *eID* (Elektronische Identifikation) und *QES* (Qualifizierte Elektronische Signatur).

**eID** Die eID-Funktion erlaubt dem Ausweisinhaber sich über das Internet gegenüber Dritten eindeutig auszuweisen. Zu diesem Zweck ist ein nPA-kompatibles und zertifiziertes Lesegerät[6] und die Software „AusweisApp“[7] notwendig. Das Protokoll dieser Funktionen wurde vom Bundesamt für Sicherheit in der Informationstechnik (BSI) entwickelt und ist in der Version 2.03 der technischen Richtlinie BSI TR-03110[42] beschrieben. Auf Seiten des Diensteanbieters können folgende Informationen aus dem nPA ausgelesen werden: Vor- und Familienname, ggf. Ordens- und Künstlername oder Doktorgrad, „D“ für Bundesrepublik

[6] `http://www.ausweisapp.bund.de/pweb/cms/kartenleser.jsp` [Letzter Zugriff: 26.11.2012]
[7] `http://www.ausweisapp.bund.de` [Letzter Zugriff: 05.10.2012]

Deutschland, Angaben zur Über- oder Unterschreitung eines bestimmten Alters, Geburtstag und Geburtsort, Anschrift und Dokumententyp. Des Weiteren beherrscht der nPA die eigenständige Prüfung ob das Alter des Inhabers ein gegebenes Alter unter- oder überschreitet (Altersprüfung) und ob der Wohnort des Inhabers einem abgefragten Wohnort entspricht. Zusätzlich kann der Ausweis eine pseudonyme Kennung generieren, mit deren Hilfe keine Informationen über den Inhaber selbst herausgegeben werden, der Diensteanbieter aber trotzdem den Inhaber wiedererkennen kann. Da sich der Diensteanbieter selbst ebenfalls mit einen signierten Zertifikat gegenüber dem Ausweisinhaber ausweisen muss, ist es möglich für jeden Diensteanbieter eine eigene pseudonyme Kennung zu generieren, wodurch der Austausch der Kennungen zwischen unterschiedlichen Anbietern zwecklos ist.

Zur Nutzung der eID-Funktion ruft der Benutzer in der Regel die passende Webseite auf uns bringt seinen Ausweis mit dem Kartenleser in Kontakt. Alle weiteren Schritte werden nun mit Hilfe der AusweisApp in Form eines geführten Assistenten durchgeführt. Im ersten Schritt werden dem Benutzer die Daten des Diensteanbieterzertifikats präsentiert, damit dieser die Korrektheit überprüfen kann (siehe Abbildung 7.14). Im zweiten Schritt sieht der Benutzer eine Übersicht aller Daten, die der Diensteanbieter im Rahmen dieses Vorgangs auslesen möchte. Der Benutzer kann in diesem Schritt der Auswahl zustimmen oder einzelne Datenfelder sperren, so dass diese während dieses Vorgangs nicht ausgelesen werden können (siehe Abbildung 7.15). Im letzten Schritt muss der Benutzer noch seine 5-stellig PIN eingeben und anschließend werden die gewünschten Daten ausgetauscht.

**QES** Über die Funktion QES kann der Benutzer eine qualifizierte elektronische Signatur auf den Pass laden, um mit Hilfe dieser Signatur elektronische Dokumente rechtsgültig zu unterschreiben. Diese Fähigkeit kann ebenfalls zusammen mit der OMS-Funktion, einen Zugriffsschutz über Signaturen zur realisieren, genutzt werden. Da zum Zeitpunkt der Implementierung der Anbindung des nPA diese Funktion noch nicht zur Verfügung stand, wurde dieser Ansatz nicht weiter verfolgt.

**Integration** Zur Integration des nPA wurde die AusweisApp über einen Dummy-Server direkt mit dem PiVis-Framework verbunden. Möchte sich eine Benutzer anmelden, löst er den Vorgang durch einen Klick auf das Benutzersymbol in der PiVis-Oberfläche aus. Darauf hin wird er aufgefordert den nPA in den Kartenleser einzuführen und anschließend startet die AusweisApp. Das PiVis-Framwork fordert beim nPA nur das Datenfeld „pseudonyme Identifikation“ an, so dass eine Verknüpfung der Objekte mit der ID des Benutzers bereits erfolgt sein muss. Im Falle des Medikamentenblisters wird dieser Schritt bereits bei der Herstellung durchgeführt. Ist der Authentifizierungsvorgang mit Hilfe der AusweisApp abgeschlossen, wird die pseudonyme ID als Authentifizierungsschlüssel an den OMS geschickt, worauf hin dieser den Zugang zu gesperrten Daten freigibt.

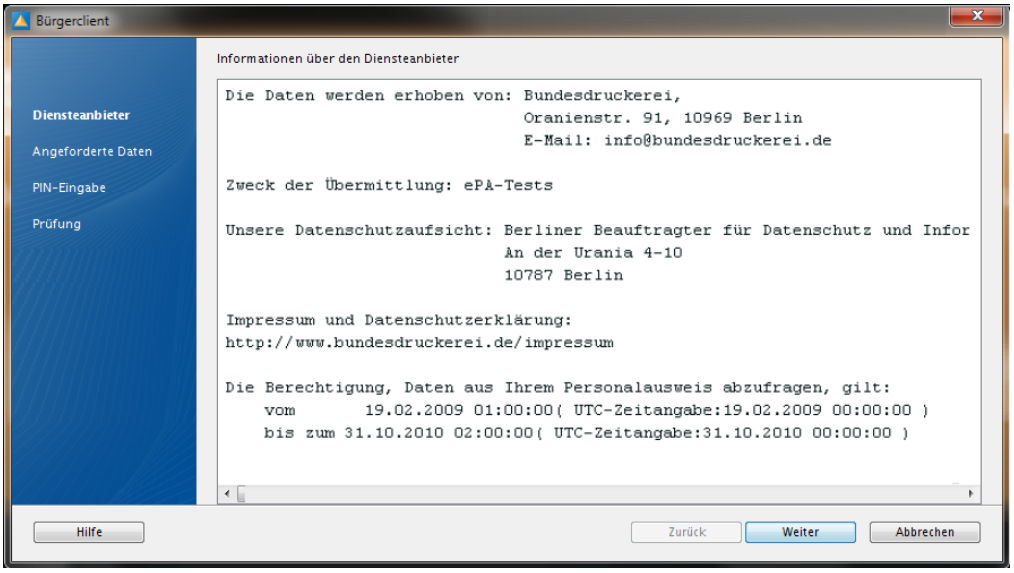



Abbildung 7.14: AusweisApp: Anzeige des Diensteanbieterzertifikats

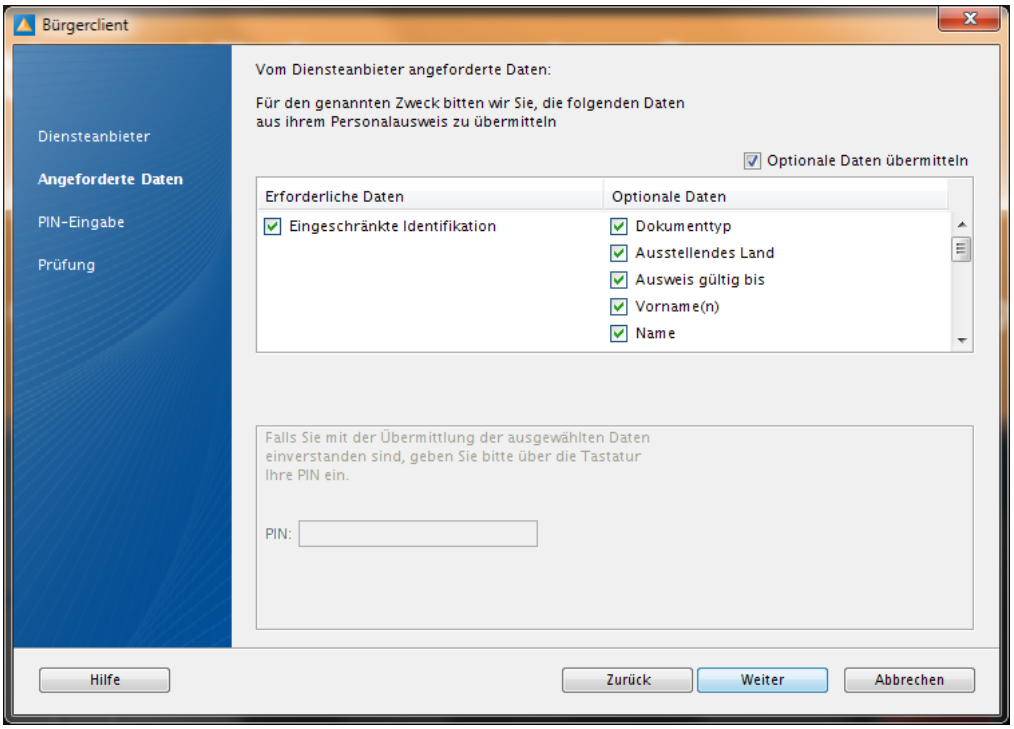



Abbildung 7.15: AusweisApp: Auswahl der zu übermittelnden Daten

### 7.6.2 TutDroid - Mobile Unterstützung bei der Durchführung und Protokollierung komplexer Aufgaben

Im Rahmen einer Bachelorarbeit wurde eine weitere Anwendung entwickelt, die sich die Möglichkeiten digitaler Objektgedächtnisse zu nutzte macht [Wei12]. Die Grundlage der Arbeit stellt ein Wartungsszenario dar, bei der ein Techniker im Außendienst Reparaturen an technischen Geräten durchführen muss. Diese Wartung erfolgt mit Hilfe von Informationen (zum Beispiel Bauplänen oder Handbüchern), Werkzeugen und eventuell Ersatzteilen. Die Geschwindigkeit, mit der der Techniker solche Wartungsarbeiten durchführt, hängt auch in großem Maße von seinem Erfahrungsschatz und seiner Expertise bezüglich der einzelnen Geräte ab. Die Arbeit hat sich zum Ziel gesetzt, den Techniker mit geringem Aufwand mit allen verfügbaren Informationen zu unterstützen und gleichzeitig das Wissen, dass der Techniker bei der Durchführung hinzugewinnt, für alle Kollegen nutzbar zu machen. Zu diesem Zweck nutzt der Techniker eine mobile Anwendung, mit deren Hilfe er verschiedene Wartungsprozesse für unterschiedliche Geräte abrufen kann. Er wird mit Hilfe des Modells durch die Wartung geführt, erhält an passender Stelle die benötigten Informationen und kann neue Erkenntnisse oder Verbesserungsvorschläge direkt in den Prozess einarbeiten.

**TutDroid-Komponenten** Für die Geräte wurde ein eigenes Modell entwickelt, welches alle Daten dieses Geräts beinhaltet. Diese Daten wurden mit Hilfe des OMM-Formats partitioniert und an zentraler Stelle im Object Memory Server abgelegt, so dass der Zugriff von unterschiedlichen Orten aus möglich ist. Dabei handelt es sich um bereits erwähnte und typische Gedächtnisdaten wie Handbücher, Datenblätter, Bilder, Schaltzeichnungen und historische Daten. Des Weiteren werden diese TutDroid-Wartungsinformationen ergänzt. Dies umfasst die Strukturmodell des Geräts mit Hilfe des OMM-Strukturblocks und dem neu erstellten Prozessmodell, mit dessen Hilfe der Wartungsvorgang modelliert werden kann. Dazu wird das Datenmodell der Anwendung *SceneMaker* [GKKR03, GMK12] genutzt, mit deren Hilfe sich der Prozess als Zustandsautomat darstellen lässt. Dieses Modell wird ebenfalls im Gedächtnis abgelegt. Die einzelnen Bauteile des zu wartenden Geräts werden mit Hilfe von RFID-Tags identifiziert. Dieses Verfahren wurde bereits bei den zuvor genannten Demonstratoren genutzt. In diesem Fall wird das Objekt als Einheit über ein Tag identifiziert, welches direkt auf das Objektgedächtnis verweist. Alle untergeordneten Bestandteile, die auch alle „getagt“ sind, werden anhand der RFID-Tag-ID erkannt, welche im Objektgedächtnis des Geräts aufgeschlüsselt sind.

### 7.6.3 Verknüpfung von Objektgedächtnissen mit URC/UCH

Ziel des Konzepts *Universal Remote Console* (URC) ist die Schaffung einer einheitlichen und persönlichen Fernbedienung, die es Benutzern erlaubt Produkte zu Hause, auf der

Arbeit und an anderen öffentlichen Plätzen zu kontrollieren [Int09, SFAB11]. Mit Hilfe von Netzwerken und teilweise drahtlosen Verbindungen ist die Konnektivität sichergestellt. Auf der Anwendungsseite muss allerdings für jedes Gerät eine eigenständige Schnittstelle geschaffen werden, so dass viele unterschiedliche Ansätze verfolgt wurden und daher die Bedienung unterschiedlicher Geräte uneinheitlich und unintuitiv erfolgt. Mit URC wurde eine Middelware (Universal Controller Hub, UCH) geschaffen, mit deren Hilfe es möglich ist verschiedene Geräte mit unterschiedlichsten Controllern zu verbinden, ohne für jede dieser Verbindung eine eigene Anwendung entwickeln zu müssen (siehe Abbildung 7.16). Mit Hilfe dieses Hubs ist es möglich verfügbare Geräte zu finden und auf diese zuzugreifen. Über Beschreibungsmodelle werden die Eigenschaften und der Funktionsumfang der Geräte sichtbar gemacht. Als Steuercontroller sind unterschiedlichste Eingabemodalitäten möglich, so werden zum Beispiel mobile Geräte, Spracheingabe und Gestensteuerung unterstützt.

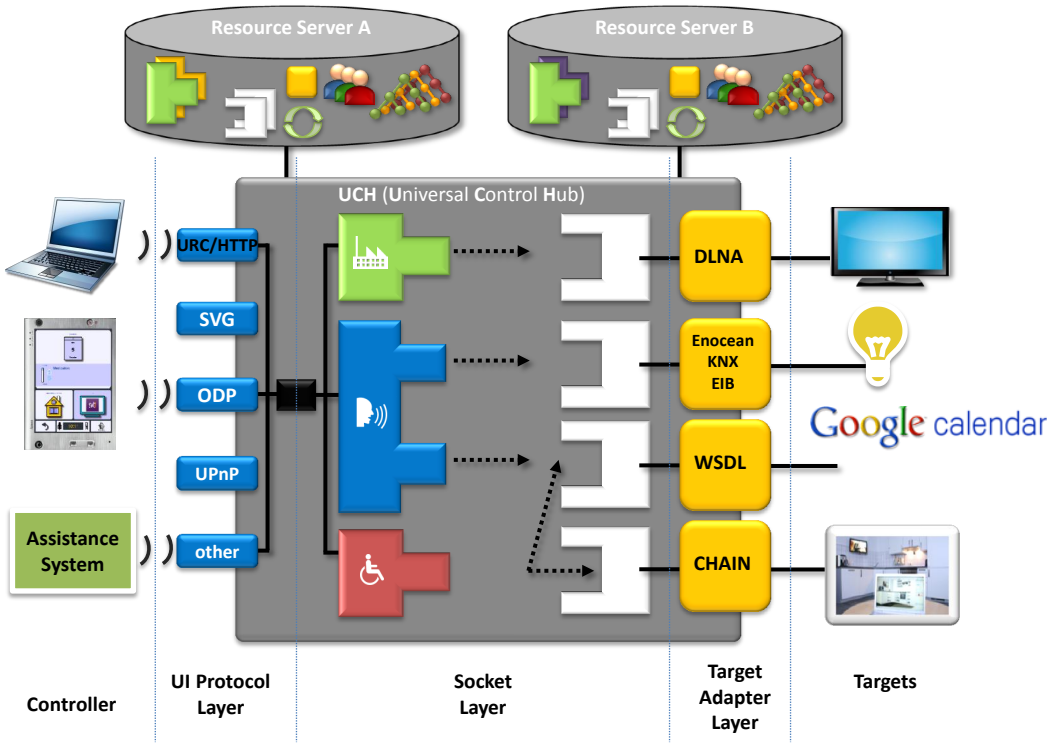



Abbildung 7.16: Universal Remote Console

**Verknüpfung von URC und Objektgedächtnisse** Im Rahmen eines Demonstrators wurden die Universal Remote Console erweitert um den Zugriff auf Objektgedächtnisse zu erhalten. Dazu wurde ein eigener Target-Adapter erstellt, der es erlaubt von jedem Controller aus auf Objektgedächtnisse zuzugreifen. Der Target-Adapter bietet eine Schnittstelle, die funktional zur REST-Schnittstelle des OMS identisch ist und somit den Zugriff auf die wich-

tigsten Funktionen des Gedächtnisses erlaubt. Somit können von allen URC-kompatiblen Anwendungen Gedächtnisfunktionalitäten genutzt werden. Konkret erlaubt es diese Anbindung, dass im Medikamentenwechselwirkungsszenario Einnahmehinweise und Daten aus dem Gedächtnis des Blisters gelesen werden und diese automatisch in den Kalender des Patienten eingetragen werden [NHFB11].

Im gleichen Vorhaben wurde auch der umgekehrte weg realisiert. Dazu wurde das PiVis-Framework ergänzt, so dass dieses als Controller fungieren kann. Dies ermöglicht beispielsweise im Medikamentenwechselwirkungsszenario auf Funktionen einer intelligenten Küche zuzugreifen, um den Benutzer weitere Hilfestellungen anzubieten. Konkret wurden dabei die RFID-Reader über eine URC-angebundenen Küche vom PiVis-Framework aus genutzt. Des Weiteren können beispielsweise bei Entnahme einer Tiefkühlpizza direkt aus den Zubereitungshinweise im Objektgedächtnis Temperatur und Timer des Backofens vorprogrammiert werden.

## 7.7 Fazit

In diesem Kapitel wurden unterschiedlichste Anwendungen vorgestellt, die mit Hilfe der im Rahmen dieser Arbeit entwickelten Werkzeuge realisiert oder erweitert werden konnten. Die SemProM-Objektdatenvisualisierung erlaubt es Endverbrauchern einen Blick auf alle Daten, die im Lebenszyklus eines Produkts erstellt werden, zu werfen. Die Demo zur Medikamentenwechselwirkung zeigt als weitere Funktionen den Rollen-basierten Zugang zu geschützten Gedächtnissen. Weitere Demonstratoren im IRL belegen die Wiederverwendbarkeit vielen Komponenten dieser Arbeit zum Aufbau unterschiedlichster Anwendungen. Des Weiteren wurden externe Dienste angebunden, um zum Beispiel den neuen Personalausweis zum Zugriff auf private Gedächtnisse zu nutzen oder mit weiteren Frameworks auf Objektgedächtnisse zuzugreifen. Alle diese Anwendungen zeigen die einfache und flexible Nutzbarkeit der Werkzeuge und belegen die praxisnahe Ausrichtung, die Art der Implementierung und die Notwendigkeit aller entwickelten Komponenten.

Das folgende und letzte Kapitel dieser Arbeit fasst abschließend die wissenschaftlichen und praktischen Beiträge dieser Arbeit zusammen, zeigt einen Überblick über die mit dieser Arbeit verbundenen Veröffentlichungen und schließt mit einem Ausblick über zukünftige Arbeiten, die die Nutzbarkeit und Effektivität der Werkzeuge weiter steigern können.

KAPITEL 8

# Fazit & Zukünftige Arbeiten

In dieser Arbeit wurde ein Strukturmodell für digitale Objektgedächtnisse vorgestellt, darauf aufbauende semantische Datenstrukturen, eine Infrastruktur zur Speicherung und Bereitstellung von Gedächtnissen und eine Sammlung weiterer Werkzeuge zur Implementierung von neuen Anwendungen und zur Integration von Objektgedächtnissen in bestehende Anwendungen dargelegt. Darauf aufbauend wurden realisierte Anwendungsprototypen vorgestellt, die die praktische Nutzbarkeit dieser Gesamtarchitektur belegen.

## 8.1 Beiträge

Die Beiträge dieser Arbeit umfassen die wissenschaftlichen Resultate im Bereich der digitalen Objektgedächtnisse, praktische Umsetzungen in Form einer Werkzeugsammlung und drauf aufbauenden Anwendungen und mehrere wissenschaftliche Veröffentlichungen, die auf dieser Arbeit basieren. Der folgende Abschnitt zeigt eine Übersicht über diese Arbeiten.

### 8.1.1 Wissenschaftliche Beiträge

Die wichtigsten wissenschaftlichen Beiträge dieser Arbeit sind folgende:

- **Entwicklung eines Strukturmodells zur Partitionierung und Annotation von Objektgedächtnisdaten**
  Basierend auf der Idee eine Speichermöglichkeit für alle Daten, die entlang des gesamten Produktlebenszyklus auftreten können, anbieten zu können, ermöglicht ein fest definiertes Modell einen einheitlichen Zugriff auf solche Daten (Object Memory

Model, OMM). Darüber hinaus muss dieses Modell der Heterogenität von Produktdaten Rechnung tragen, in dem ein offener Ansatz genutzt wird, der mit Hilfe von Metadaten ein Wiederauffinden von gespeicherten Daten erleichtert.

- **Erstellung eines Modells zur Speicherung von Gedächtnis-bezogenen und semantisch modellierten Produktdaten und Benutzerpräferenzen**
  Ausgehend von der grundsätzlichen Idee der Nutzung von Objektgedächtnissen als Datenaustauschmedium für Objekt-bezogene Daten, sollten diese Daten nach Möglichkeit über eine Struktur verfügen, die eine Kommunikation mit anderen Partnern erleichtert. In diesem Bereich haben sich daher semantisch modellierte Daten als gewinnbringend etabliert, so dass dieser Ansatz auch für Objektgedächtnisdaten übernommen wurde. Das daraus resultierende Modell unterstützt sowohl typische Produktdaten aus dem Bereich der Lebensmittel und Textilien und kann durch die Verwendung von gemeinsamen Konzepten auch zur Darstellung von Benutzerpräferenzen genutzt werden.

- **Bereitstellung von Schutz- und Zugriffsfunktionalität für Objektgedächtnisse basierend auf Rechten und Rollen**
  Der prinzipiell gewünschte, offene und unbeschränkte Zugang zu Objektgedächtnissen wird mit Hilfe der einheitlichen Schnittstelle und des durchgängigen Datenmodells erreicht. Im Hinblick auf eventuell sensible Daten in solchen Gedächtnissen kann jedoch eine Einschränkung des Zugriffs notwendig sein. Zu diesem Zweck bietet die Architektur ein Rechte- und Rollenmodell, das es erlaubt eine solche Beschränkung zu realisieren. Dabei wird nicht nur eine simple binäre Unterscheidung ermöglicht, dass heißt ob eine Zugriff stattfinden kann oder nicht, sondern ein Modell zur Abbildung von Zugriffsrollen ermöglicht eine differenziertere Steuerung beim Zugriff auf Objektgedächtnisdaten.

- **Zusammenführung der entwickelten Module zu einer gesamtheitlichen Architektur für Objektgedächtnisse**
  Die im Rahmen dieser Arbeit entwickelten Werkzeuge ermöglichen die Nutzung von Objektgedächtnissen durch Unterstützung bei unterschiedlichen Aufgaben an diversen Stellen entlang der Lebenszykluskette. Zusätzlich sind diese konsistent aufeinander abgestimmt, bieten kompatible Schnittstellen und basieren auf durchgängigen Datenmodellen. Dadurch ist eine nahtlose Integration möglich, wodurch Verluste durch unter Umständen mehrere Datenkonvertierungsschritte vermieden werden und somit der Migrations- und Integrationsaufwand minimiert werden kann.

### 8.1.2 Praktische Beiträge

- **Erstellung einer Referenzimplementierung zur Nutzung von Objektgedächtnissen**

Basierend auf dem Object Memory Model wurde eine zweistufige Referenzimplementierung umgesetzt. Die unterste Schicht bildet dabei eine Bibliothek, mit deren Hilfe Datei-basierte Objektgedächtnisse in Anwendungen integriert werden können. Darauf aufbauend wurde eine Server-basierte Lösung erstellt, die sich die Bibliothek zunutze macht und mit Hilfe einer definierten Schnittstelle die Gedächtnisse über das Web zur Verfügung stellt.

- **Entwicklung einer Werkzeugsammlung als Grundlage für die Anwendungsentwicklung**
  Zur Umsetzung von Objektgedächtnisanwendungen basierend auf dem genannten Modell wurde eine Werkzeugsammlung implementiert, die es ermöglicht bestehenden Anwendungen zu migrieren und neue Anwendungen zu realisieren. Die einzelnen Werkzeuge sind dabei aufeinander abgestimmt und bilden zusammen eine durchgehende Kette, die aber auch alle einzeln und punktuell genutzt werden können.

- **Bewertung und Weiterentwicklung durch die Erstellung von prototypischen Anwendungen**
  Aufbauend auf der genannten Werkzeugsammlung, wurden prototypische Anwendungen aus dem Bereich des Produktlebenszyklus und des Handels bzw. des Endverbrauchers realisiert. Die bei der Umsetzung gewonnen Erkenntnisse flossen dabei direkt in eine Weiterentwicklung der Werkzeuge ein.

### 8.1.3 Veröffentlichungen

Dieser Abschnitt zeigt eine Liste aller Publikationen, die auf Basis dieser Arbeit entstanden sind.

- **Journal-Artikel und Buchkapitel**
  Teile dieser Arbeit wurden im Buch *SemProM - The Semantic Product Memory: An Interactive Black Box for Smart Objects* und im Pervasive and Mobile Computing Journal: *Mobile Interactions with Digital Object Memories* veröffentlicht.

- **Konferenzen**
  Teile dieser Arbeit wurden auf folgenden internationalen Konferenzen veröffentlicht:

  - 8th Annual IEEE International Conference on Pervasive Computing and Communications 2010: *Demo: Authorized Access on and Interaction With Digital Product Memories* und *Roles and Rights Management Concept With Identification by Electronic Identity Card*

  - 6th International Conference on Intelligent Environments 2010: *The ObjectRules Framework*

  - 7th International Conference on Intelligent Environments 2011: *The PiVis Framework for Visualization of Digital Object Memories* und *Supporting Persons with Special Needs in Their Daily Life in a Smart Home*

  - 10th Annual IEEE International Conference on Pervasive Computing and Communications 2012: *Towards a Digital Object Memory Architecture* und *Sustainable Instrumentation of Everyday Commodities - Concepts and Tools*

- **Workshops**
  Teile dieser Arbeit wurden auf folgenden internationalen Workshops veröffentlicht:

  - International workshop on networking and object memories for the internet of things (NOMe-IoT 2011): *Incycling – Sustainable Concept for Instrumenting Everyday Commodities* und *Towards a Model of Object Memory Links*

  - International workshop on digital object memories for the internet of things (DOMe-IoT 2012): *How to Instill Activity into Digital Object Memories*

## 8.2 Zukünftige Arbeiten

- **Wie kann eine Domänen-übergreifende Kommunikation mit Objektgedächtnissen auf Basis wohl definierter Datenstrukturen erleichtert und gefördert werden?**
  Der offene OMM-Ansatz erlaubt es beliebige Datenformate und -strukturen in Gedächtnisse abzulegen. Daher müssen sich Partner für einen übergreifenden Austausch von Daten eigenständig um eine Harmonisierung der genutzten Formate kümmern. Zur Erreichung einer größeren Akzeptanz und zur Reduktion von Harmonisierungsaufwänden soll ein Satz von standardisierten Blöcken geschaffen werden, welche entlang der gesamten Produktlebenszykluskette oder über einzelne Domänengrenzen hinweg genutzt werden. Diese Blöcke basieren auf der Idee des kleinsten gemeinsamen Nenners, bieten also einen eingeschränkten Umfang, der allerdings wohl definiert ist und von möglichst vielen Partnern dieser Domäne unterstützt wird.

- **Wie lassen sich die bisher sprechenden und deklarativen OMM-Formate in eine kompakte und effiziente Binärdarstellung überführen, die unterschiedlichen Ansprüchen gerecht wird?**
  Die bisherige deklarative (= menschenlesbare) Darstellung von OMM-Daten (z.B. in Form von XML-Dateien) ist für viele industrienahe Anwendungen (z.B. im Bereich der eingebetteten Systeme) nicht praktikabel, da zum einen der Speicherplatz nicht effizient genutzt wird (hohe Redundanz in der Darstellung) und zum anderen nicht bei allen Systemen genügend Rechenkapazität zur Verfügung steht (hohe strukturelle Komplexität). Zu diesem Zweck soll ein generischer Ansatz entwickelt werden,

wie Gedächtnisdaten in eine Binärdarstellung und zurück überführt werden können. Die Konvertierung erfolgt dabei je nach Anwendungsfall mit unterschiedlichen Schwerpunkten. Die Extrema werden durch die Notwendigkeit der Beibehaltung aller Informationen oder der Notwendigkeit einer maximal kompakten Darstellung definiert. Daher soll die Konvertierung einen flexiblen Ansatz verfolgen, der sich an das jeweilige Szenario anpassen lässt.

- **Wie kann man den bisherigen OMM-Gedächtnisansatz nutzen um „Provenance"-Informationen sicher und vertrauenswürdig ablegen zu können?**
  Eine der Kernfunktionen eines Objektgedächtnisses ist die Haltung aller relevanten Informationen, die entlang der Lebenszykluskette anfallen. In einigen Szenarien gibt es für solche Daten gesteigerte Anforderungen in Form der Notwendigkeit einer lückenlosen Datenhaltung, der Forderung nach einer Erkennbarkeit jeglicher externer und ungewollter Veränderung von Daten und somit der Schaffung eines erhöhten Vertrauens in die Echtheit der Daten. Im Bereich der medizinischen Anwendungen gibt es beispielsweise Medikamenten-relevante Informationen, auf deren Herkunft und Integrität sich sowohl Arzt und Apotheker als auch Patient bedingungslos verlassen können müssen.

# Abbildungsverzeichnis

# Tabellenverzeichnis

# Quellcodeverzeichnis

# Index

# Literaturverzeichnis


[Aac07] WZL RWTH Aachen. Product lifecycle managements, 2007. Letzter Zugriff: 26.11.2012. URL: http://www.plm-info.de/de/09064b1689c2acd4c12573b5005390ba.html.

[ACD+12] Carlos Buil Aranda, Olivier Corby, Souripriya Das, Lee Feigenbaum, Paul Gearon, Birte Glimm, Steve Harris, Sandro Hawke, Ivan Herman, Nicholas Humfrey, Nico Michaelis, Chimezie Ogbuji, Matthew Perry, , Alexandre Passant, Axel Polleres, Eric Prud'hommeaux, Andy Seaborne, and Gregory Todd Williams. Sparql 1.1 overview, 2012. Letzter Zugriff: 26.11.2012. URL: ttp://www.w3.org/TR/2012/WD-sparql11-overview-20120501/.

[AHSB12] Ben Adida, Ivan Herman, Manu Sporny, and Mark Birbeck. RDFa Primer (W3C Recommendation), 2012. Letzter Zugriff: 26.11.2012. URL: http://www.w3.org/TR/xhtml-rdfa-primer/.

[AIM10] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The Internet of Things: A survey. *Computer Networks*, 54(15):2787–2805, October 2010.

[AKM07] Erwin Aitenbichler, Jussi Kangasharju, and Max Mühlhäuser. Mundocore: A light-weight infrastructure for pervasive computing. *Pervasive Mob. Comput.*, 3(4):332–361, August 2007.

[ALS10] J. Chris Anderson, Jan Lehnardt, and Noah Slater. *CouchDB: The Definitive Guide Time to Relax*. O'Reilly Media, Inc., 1st edition, 2010.

[ans04] Role based access control, 2004. ANSI/INCITS 359-2004.

[Ash09] Kevin Ashton. That internet of things thing. *RFID Journal*, 2009. URL: http://www.rfidjournal.com/article/print/4986.

[BBF+02] M. Barthel, J. Boyer, B. Fox, B. LaMacchia, and E. Simon. W3c xml-signature syntax processing, 2002.

[BBH+09] Grigori Babitski, Simon Bergweiler, Jörg Hoffmann, Daniel Schön, Christoph Stasch, and Alexander C. Walkowski. Ontology-based integration of sensor web services in disaster management. In *GeoSpatial Semantics. International Conference on GeoSpatial Semantics (GeoS-09), December 3-4, Mexico City,*

*Mexico*, volume 5892/2009 of *Lecture Notes in Computer Science*, *LNCS*, Seiten 103–121. Springer Berlin / Heidelberg, 11 2009.

[Ber04] Regina Bernhaupt. User experience as missing link in multimodal interfaces for ambient intelligence environments. In *Workshop paper for CHI 2004 Workshop on Ambient Intelligence*. ACM, 2004.

[BF93] N. Borenstein and N. Freed. Rfc 1521: Mime (multipurpose internet mail extensions) part one: Mechanisms for specifying and describing the format of internet message bodies, 1993.

[BGS01] Michael Beigl, Hans-Werner Gellersen, and Albrecht Schmidt. Mediacups: experience with design and use of computer-augmented everyday artefacts. *Computer Networks*, 35(4):401–409, 2001.

[BHK+10a] Boris Brandherm, Jens Haupert, Alexander Kröner, Michael Schmitz, and Frank Lehmann. Demo: Authorized access on and interaction with digital product memories. In *8th Annual IEEE International Conference on Pervasive Computing and Communications. IEEE International Conference on Pervasive Computing and Communications (PerCom-2010), March 29 - April 2, Mannheim, Germany*, Seiten 838–840. IEEE Computer Society, 2010.

[BHK+10b] Boris Brandherm, Jens Haupert, Alexander Kröner, Michael Schmitz, and Frank Lehmann. Roles and rights management concept with identification by electronic identity card. In *8th Annual IEEE International Conference on Pervasive Computing and Communications. IEEE International Conference on Pervasive Computing and Communications (PerCom-2010), March 29 - April 2, Mannheim, Germany*, Seiten 768–771. IEEE Computer Society, 2010.

[BHSdB10] Ralph Barthel, Andrew Hudson-Smith, Martin de Jode, and Benjamin Blundell. Tales of things – the internet of 'old' things: Collecting stories of objects, places and spaces. In *Proceedings of the Second International Conference on the Internet of Things 2010 (IoT 2010)*. IEEE Digital Library, 2010.

[BIP06] The international system of units (si). Technical report, Paris, 2006.

[BKH11] Boris Brandherm, Alexander Kröner, and Jens Haupert. Incycling - sustainable concept for instrumenting everyday commodities. In Harold Liu, Alexander Kröner, Chris Speed, Pan Hui, Fahim Kawsar, Wenjie Wang, Dan Wang, Boris Brandherm, Thomas Ploetz, Michael Schneider, Jens Haupert, and Peter Stephan, editors, *Proceedings of the International Workshop on Networking and Object Memories for the Internet of Things. Workshop on Digital Object Memories (DOME-11), located at UbiComp 2011, September 17-21, Peking, China*, Seiten 27–28. ACM, 9 2011.

[BKH12a] Ralph Barthel, Alexander Kröner, and Jens Haupert. Mobile interactions with digital object memories. *Pervasive and Mobile Computing*, 8, Issue 4:o.A., 2012. URL: http://dx.doi.org/10.1016/j.pmcj.2012.05.005.

[BKH+12b] Boris Brandherm, Alexander Kröner, Jens Haupert, Michael Schmitz, Frank Lehmann, and Ralf Gampfer. Sustainable instrumentation of everyday commodities - concepts and tools. In *10th Annual IEEE International Conference on Pervasive Computing and Communications. IEEE International Conference on Pervasive Computing and Communications (PerCom-12), 10th, March 19-23, Lugano, Switzerland*, Seiten 467 – 470. IEEE Computer Society, 2012.

[BKS+11] Boris Brandherm, Alexander Kröner, Michael Schneider, Jens Haupert, and Michael Schmitz. Patientenindividuelle förderung der therapietreue durch intelligente medikamentenverpackungen. In *Demographischer Wandel - Assistenzsysteme aus der Forschung in den Markt. Deutscher AAL-Kongress (AAL), 4. Deutscher AAL-Kongress mit Ausstellung, January 25-26, Berlin, Germany*. VDE VERLAG GMBH, Berlin und Offenbach, 1 2011.

[BKS+12] Boris Brandherm, Alexander Kröner, Michael Schmitz, Jens Haupert, Frank Lehmann, and Ralf Gampfer. Nachhaltiges konzept zur förderung der therapietreue [sustainable concept for increasing compliance]. In *Demographischer Wandel - Assistenzsysteme aus der Forschung in den Markt. Deutscher AAL-Kongress (AAL-12), 5. Deutscher AAL-Kongress mit Ausstellung, January 23-25, Berlin, Germany*. VDE VERLAG GMBH, Berlin und Offenbach, 2 2012.

[BKvH02] Jeen Broekstra, Arjohn Kampman, and Frank van Harmelen. Sesame: A generic architecture for storing and querying RDF and RDF Schema. In *The Semantic Web – ISWC 2002: First International Semantic Web Conference, Sardinia, Italy*, Seiten 54–68. Springer Berlin / Heidelberg, 2002.

[BLFM98] T. Berners-Lee, R. Fielding, and L. Masinter. Rfc 2396: Uniform resource identifiers (uri): Generic syntax, 1998.

[BLFM05] T. Berners-Lee, R. Fielding, and L. Masinter. Rfc 3986: Uniform resource identifier (uri): Generic syntax, 2005.

[BLHL01] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific American*, 284(5):34–43, May 2001.

[BMKL01] D. Brock, T. Milne, Y. Kang, and B. Lewis. The Physical Markup Language. Technical report, Auto-ID Center, Cambridge, MA, 2001.

[BS05] Boris Brandherm and Tim Schwartz. Geo referenced dynamic bayesian networks for user positioning on mobile systems. In Thomas Strang and

Claudia Linnhoff-Popien, editors, *LoCA*, volume 3479 of *Lecture Notes in Computer Science*, Seiten 223–234. Springer Berlin / Heidelberg, 2005.

[BSI12] BSI: Bundesamt für Sicherheit in der Informationstechnik. Tr-03130: Technische richtlinie eid-server 1.6, 2012.

[Bun09] Bundesministerium der Justiz, Bundesrepublik Deutschland. Bundesgesetzblatt Teil I - Nr. 33 - Gesetz über Personalausweise und den elektronischen Identitätsnachweis sowie zur Änderung weiterer Vorschriften, 2009.

[CB00] R. G. G. Cattell and Douglas K. Barry. *The Object Data Standard: ODMG 3.0*. Morgan Kaufmann, 2000.

[CD99] James Clark and Steve DeRose. Xml path language (xpath) 1.0, 1999. Letzter Zugriff: 26.11.2012. URL: `http://www.w3.org/TR/xpath/`.

[Cha10] Hakima Chaouchi, editor. *The Internet of Things: Connecting Objects*. Wiley-ISTE, Hoboken, NJ, London, 2010.

[Chi00] Ed H. Chi. A taxonomy of visualization techniques using the data state reference model. In *Proceedings of the IEEE Symposium on Information Vizualization 2000*, INFOVIS '00, Seiten 69–75, Washington, DC, USA, 2000. IEEE Computer Society.

[DCM12] Dcmi specifications, 2012. Letzter Zugriff: 26.11.2012. URL: `http://dublincore.org/specifications/`.

[Dey00] Anind K. Dey. *Providing architectural support for building context-aware applications*. PhD thesis, Georgia Institute of Technology, 2000.

[DH98] S. Deering and R. Hinden. Rfc 2460: Internet protocol, version 6 (ipv6), 1998.

[dJBHS11] Martin L. de Jode, Ralph Barthel, and Andrew Hudson-Smith. Tales of things: the story so far. In *Proceedings of the 2011 international workshop on Networking and object memories for the internet of things*, NoME-IoT '11, Seiten 19–20, New York, NY, USA, 2011. ACM.

[DK10] Peter J. Denning and Robert E. Kahn. The long quest for universal information access. *Communications of the ACM*, 53(12):34–36, December 2010.

[DVDNGP11] Roberto De Virgilio, Pierluigi Del Nostro, Giorgio Gianforme, and Stefano Paolozzi. A scalable and extensible framework for query answering over RDF. *World Wide Web*, 14(5-6):599–622, October 2011.

[Ebe03] Andreas Eberhart. *Ontology-based infrastructure for intelligent applications*. PhD thesis, Saarländische Universitäts- und Landesbibliothek, Postfach 151141, 66041 Saarbrücken, 2003. URL: http://scidok.sulb.uni-saarland.de/volltexte/2004/260.

[EFG07] Daniel Engovatov, Daniela Florescu, and Giorgio Ghelli. Xquery scripting extension 1.0 requirements, 2007. Letzter Zugriff: 26.11.2012. URL: http://www.w3.org/TR/xquery-sx-10-requirements/.

[Eig09] Ralph Eigner, Martin ; Stelzer. *Product Lifecycle Management : ein Leitfaden für Product Development und Life Cycle Management*. Springer Berlin / Heidelberg, Berlin, 2., neu bearb. aufl. edition, 2009.

[EN06] Ramez Elmasri and Shamkant B. Navathe. *Fundamentals of Database Systems (5th Edition)*. Addison Wesley, March 2006. URL: http://www.amazon.ca/exec/obidos/redirect?tag=citeulike04-20{&}path=ASIN/0321369572.

[EPC07a] EPCglobal. EPC Information Services (EPCIS) Version 1.0.1, 2007. URL: http://www.epcglobalinc.org.

[EPC07b] EPCglobal. The Pedigree Ratified Standard Version 1.0, 2007. URL: http://www.epcglobalinc.org.

[EPC08] EPCglobal. Epc radio-frequency identity protocols class-1 generation-2 uhf rfid protocol for communication at 860 mhz - 960 mhz version 1.2.0, 2008. URL: http://www.gs1.org/gsmp/kc/epcglobal/uhfc1g2/.

[EPC11] EPCglobal. The GS1 EPC Tag Data Standard Version 1.6, 2011. URL: http://www.epcglobalinc.org.

[FCD05] Elgar Fleisch, Oliver Christ, and Markus Dierkes. Die betriebswirtschaftliche Vision des Internets der Dinge. In *Das Internet der Dinge: Ubiquitous Computing und RFID in der Praxis*, Seiten 3–37. Springer Berlin / Heidelberg, 2005.

[Fel11] Michael Feld. *A Speaker Classification Framework for Non-intrusive User Modeling: Speech-based Personalization of In-Car Services*. Phd-thesis, Computer Science Institute, Saarland University, Saarbrücken, Germany, 12 2011.

[FHLW05] Dieter Fensel, James A. Hendler, Henry Lieberman, and Wolfgang Wahlster, editors. *Spinning the Semantic Web*. MIT Press, Cambridge, MA, new edition, 2005.

[FK92] David Ferraiolo and Richard Kuhn. Role-based access control. In *In 15th NIST-NCSC National Computer Security Conference, Baltimore, MD*, Seiten 554–563. United States Government Printing Office, 1992.

[FM11] Michael Feld and Christian Müller. The automotive ontology: Managing knowledge inside the vehicle and sharing it between cars. In *Proceedings of the 3rd International Conference on Automotive User Interfaces and Interactive Vehicular Applications. International Conference on Automotive User Interfaces and Interactive Vehicular Applications (AutomotiveUI-11), November 29 - December 2, Salzburg, Austria*, Seiten 79–86. ACM, 11 2011.

[FSJ99] Mohamed E. Fayad, Douglas C. Schmidt, and Ralph E. Johnson. *Building application frameworks: object-oriented foundations of framework design*. John Wiley & Sons, Inc., New York, NY, USA, 1999.

[Gen91] Michael R. Genesereth. Knowledge interchange format. In James F. Allen, Richard Fikes, and Erik Sandewall, editors, *KR*, Seiten 599–600. Morgan Kaufmann, 1991.

[GIK04] Thomas Grill, Ismail Khalil Ibrahim, and Gabriele Kotsis. Agents visualization in intelligent environments. In *The Second International Conference on Advances in Mobile Multimedia (MoMM), Bali, Indonesia*, Seiten 361–369. Austrian Computer Society, September 2004.

[GKKR03] Patrick Gebhard, Michael Kipp, Martin Klesen, and Thomas Rist. Authoring scenes for adaptive, interactive performances. In *Proc. of the Second International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'03).*, Seiten 725–732. ACM, 2003.

[GMK12] Patrick Gebhard, Gregor Mehlmann, and Michael Kipp. Visual SceneMaker—a tool for authoring interactive virtual characters. *Journal on Multimodal User Interfaces*, 6(1-2):3–11, 7 2012.

[Hau11] Jens Haupert. The pivis framework for visualization of digital object memories. In *Proceedings 2011 Seventh International Conference on Intelligent Environments (IE 2011). International Conference on Intelligent Environments (IE-11), July 25-28, Nottingham, United Kingdom*, Seiten 179–185. IEEE Computer Society, 7 2011.

[Hau12] Jens Haupert. Towards a digital object memory architecture. In *Fifth Annual PhD Forum on Pervasive Computing and Communications (PerCom 2012 PhD Forum). IEEE International Conference on Pervasive Computing and Communications (PerCom-12), 5th, March 19-23, Lugano, Switzerland*, Seiten 568 – 569. IEEE Computer Society, 3 2012.

[HBE+00] Hans Hagen, Henning Barthel, Achim Ebert, Andreas Divivier, and Michael Bender. A component- and multi agent-based visualization system architecture. In *Proceedings of the International Workshop on Mobile Computing (IMC), Rostock*. Fraunhofer IGD, Darmstadt, 2000.

[Hec02] Dominik Heckmann. Proposal for a user modeling markup language (UserML). In Nicola Henze, editor, *ABIS-Workshop: Personalization for the mobile World, Hannover*, Seiten 17–21, Hannover, Germany, October 2002. GI.

[Hec05] Dominikus Heckmann. *Ubiquitous User Modeling*. Phd-thesis, Department of Computer Science, Saarland University, 11 2005.

[HHK12] Jens Haupert, Christian Hauck, and Alexander Kröner. How to instill activity into digital object memories. In Alexander Kröner, Jens Haupert, Chris Speed, Fahim Kawsar, Thomas Ploetz, and Daniel Schreiber, editors, *International Workshop on Digital Object Memories in the Internet of Things 2012. Workshop on Digital Object Memories (DOME-12), located at International Conference on Ubiquitous Computing 2012, September 5-8, Pittsburgh„ PA, United States*. ACM, 2012.

[HLM+09] Dominikus Heckmann, Matthias Loskyll, Rafael Math, Pascal Recktenwald, and Christoph Stahl. Ubisworld 3.0: a semantic tool set for ubiquitous user modeling. In *User Modeling, Adaptation, and Personalization. International Conference on User Modeling, Adaptation, and Personalization (UMAP-2009), June 22-26, Trento, Italy*. Self, 2009.

[HS06] Sungchul Hong and Yeong-Tae Song. Incorporating heterogeneous xml formats using relational database. In *ACIS-ICIS*, Seiten 142–147. IEEE Computer Society, 2006. URL: `http://dblp.uni-trier.de/db/conf/ACISicis/ACISicis2006.html#HongS06`.

[HS13] Jens Haupert and Michael Schneider. *Object Memory Server*, volume SemProM - Foundations of Semantic Product Memories for the Internet of Things, Seiten 179 – 194. Springer Berlin / Heidelberg, 2013. im Erscheinen.

[HSB+05] Dominik Heckmann, Tim Schwartz, Boris Brandherm, Michael Schmitz, and Margeritta von Wilamowitz-Moellendorff. Gumo - the general user model ontology. In *User Modeling*, Seiten 428–432. Springer Berlin / Heidelberg, 2005.

[IdFC06] R. Ierusalimschy, L. H. de Figueiredo, and W. Celes. *Lua 5.1 Reference Manual*. 2006.

[Int00a] International Organization for Standardization. Identification cards - contactless integrated circuit(s) cards - proximity cards. ISO/IEC 14443:2000, 2000.

[Int00b] International Organization for Standardization. Identification cards — contactless integrated circuit(s) cards — vicinity cards. ISO/IEC 15693:2000, 2000.

[Int06a] International Organization for Standardization. Information technology — automatic identification and data capture techniques — qr code 2005 bar code symbology specification. ISO/IEC 18004:2006, 2006.

[Int06b] International Organization for Standardization. Information technology — automatic identification and data capture techniques — data matrix bar code symbology specification, 2006.

[Int09] International Organization for Standardization. Information technology – framework for specifying a common access profile (cap) of needs and capabilities of users, systems, and their environments. ISO/IEC 24756:2009, 2009.

[Int10] International Organization for Standardization. Near field communication interface and protocol-2. ISO/IEC 21481 / ECMA-352, 2010.

[ISO04] ndustrial automation systems and integration – Product data representation and exchange, 2004.

[ISO07] ISO/IEC 13249. Information technology - database languages - sql multimedia and application packages, edition 3, 2007.

[ISO11] ISO/IEC 9075. Information technology - database languages - sql, edition 4, 2011.

[ITU97] ITU-T Recommendation X.509 Version 3. Information technology - open systems interconnection - the directory: Authentication framework, 1997.

[JM08] Sabine Janzen and Wolfgang Maass. Smart product description object (spdo). *Poster Proceedings of the 5th International Conference on Formal Ontology in Information Systems (FOIS2008), Saarbrücken*, Seiten 25–30, 2008.

[Jos06] S. Josefsson. Rfc 4648: The base16, base32, and base64 data encodings, 2006.

[KAB+07] Gerd Kortuem, David Alford, Linden Ball, Jerry Busby, Nigel Davies, Christos Efstratiou, Joe Finney, Marian White, and Katharina Kinder. Sensor Networks or Smart Artifacts? An Exploration of Organizational Issues of an Industrial Health and Safety Monitoring System. In John Krumm, Gregory D. Abowd, Aruna Seneviratne, and Thomas Strang, editors, *UbiComp 2007: Ubiquitous*

*Computing*, volume 4717 of *Lecture Notes in Computer Science*, chapter 27, Seiten 465–482. Springer Berlin / Heidelberg, Berlin, Heidelberg, 2007.

[KAC+02] G. Karvounarakis, S. Alexaki, V. Christophides, D. Plexousakis, and M. Schol. RQL: A Declarative Query Language for RDF. In *Proceedings of the Eleventh International World Wide Web Conference (WWW'02)*, Seiten 592–603, Honolulu, Hawaii, USA, May7-11 2002. ACM.

[Kam99] U. Kampffmeyer. *Dokumenten-Management*. Project Consult GmbH, 1999. URL: http://books.google.de/books?id=QuwakDA7xeAC.

[Kaw09] Fahim Kawsar. *A Document-Based Framework for User Centric Smart Object Systems*. PhD thesis, Dept. Computer Science, Waseda Univ, 2009.

[KFN05] Fahim Kawsar, Kaori Fujinami, and Tatsuo Nakajima. Prottoy: A middleware for sentient environment. In Laurence Tianruo Yang, Makoto Amamiya, Zhen Liu, Minyi Guo, and Franz J. Rammig, editors, *EUC*, volume 3824 of *Lecture Notes in Computer Science*, Seiten 1165–1176. Springer Berlin / Heidelberg, 2005.

[KGB+11] Alexander Kröner, Patrick Gebhard, Boris Brandherm, Benjamin Weyl, Jörg Preißinger, Carsten Magerkurth, and Selcuk Anilmis. Personal shopping support from digital product memories. In *Proceedings of the 1st International Conference on Pervasive and Embedded Computing and Communication Systems (PECCS 2011), Vilamoura, Algarve, Portugal*, Seiten 64–73. SciTePress, 2011.

[KHB+11] Alexander Kröner, Jens Haupert, Boris Brandherm, Markus Miche, and Ralph Barthel. Towards a model of object memory links. In Harold Liu, Alexander Kröner, Chris Speed, Pan Hui, Fahim Kawsar, Wenjie Wang, Dan Wang, Boris Brandherm, Thomas Ploetz, Michael Schneider, Jens Haupert, and Peter Stephan, editors, *Proceedings of the International Workshop on Networking and Object Memories for the Internet of Things. Workshop on Digital Object Memories (DOME-11), located at UbiComp 2011, September 17-21, Peking, China*, Seiten 17–18. ACM, 9 2011.

[KHS+11] Alexander Kröner, Jens Haupert, Marc Seißler, Bruno Kiesel, Barbara Schennerlein, Sven Horn, Daniel Schreiber, and Ralph Barthel. Object memory modeling - w3c incubator group report, 2011. Letzter Zugriff: 26.11.2012. URL: http://www.w3.org/2005/Incubator/omm/XGR-omm-20111026/.

[KHW06] Alexander Kröner, Dominik Heckmann, and Wolfgang Wahlster. Specter: Building, exploiting, and sharing augmented memories. In *Workshop on Knowledge Sharing for Everyday Life 2006, Kyoto*, Seiten 9–16. ATR Media Information Science Laboratories, 2006.

[Kin02] Tim Kindberg. Implementing physical hyperlinks using ubiquitous identifier resolution. In *Proceedings of the 11th international conference on World Wide Web*, WWW '02, Seiten 191–199, New York, NY, USA, 2002. ACM.

[KKM06] Yoshinobu Kitamura, Yusuke Koji, and Riichiro Mizoguchi. An ontological model of device function: industrial deployment and lessons learned. *Appl. Ontol.*, 1(3-4):237–262, August 2006.

[KKS+10] Alexander Kröner, Gerrit Kahl, Lübomira Spassova, Thomas Feld, Dirk Mayer, Carsten Magerkurth, and Ali Dada. Demonstrating the application of digital product memories in a carbon footprint scenario. In *Proceedings of the Sixth International Conference on Intelligent Environments (IE 2010), Kuala Lumpur, Malaysia*, Seiten 164–169. IEEE Computer Society, 2010.

[KLN87] David Klahr, P. Langley, and R. Neches, editors. *Production system models of learning and development*. MIT Press, Cambridge, MA, 1987.

[KNPJ08] Fahim Kawsar, Tatsuo Nakajima, Jong Hyuk Park, and Young-Sik Jeong. A document based framework for smart object systems. In *FGCN (1)*, Seiten 178–183. IEEE Computer Society, 2008.

[KW06] Robert Kahn and Robert Wilensky. A framework for distributed digital object services. *Int. J. Digit. Libr.*, 6(2):115–123, April 2006.

[Lü77] Max Lüscher. *Der Lüscher-Test - Persönlichkeitsbeurteilung durch Farbwahl*. Rowohlt, Reinbek, 8 edition, 1977.

[Len00] B. Lenk. *Handbuch der automatischen Identifikation: ID-Techniken, 1D-Codes, 2D-Codes, 3D-Codes*. Handbuch der automatischen Identifikation. "M." Lenk, 2000.

[MB06] Wolfgang Maass and Wernher Behrendt. Trading semantically enhanced digital products in electronic markets. In Mareike Schoop, Aldo de Moor, and Jan L. G. Dietz, editors, *PragWeb*, volume 89 of *LNI*, Seiten 97–109. GI, 2006. URL: `http://dblp.uni-trier.de/db/conf/pragweb/pragweb2006.html#MaassB06`.

[MBG+01] Claudio Masolo, Stefano Borgo, Aldo Gangemi, Nicola Guarino, and Alessandro Oltramari. Wonderweb deliverable d18 - ontology library (final). 2001.

[MBG07] Wolfgang Maass, Wernher Behrendt, and Aldo Gangemi. Trading digital information goods based on semantic technologies. *J. Theor. Appl. Electron. Commer. Res.*, 2(3):18–35, December 2007.

[Mei07] Andreas Meier. *Relationale und postrelationale Datenbanken*. eXamen.press. Springer Berlin / Heidelberg, Berlin [u.a.], 6., überarb. und erw. aufl. edition, 2007. URL: http://gso.gbv.de/DB=2.1/CMD?ACT=SRCHA&SRT=YOP&IKT=1016&TRM=ppn+523539320&sourceid=fbw_bibsonomy.

[MF06] Wolfgang Maass and Andreas Filler. Towards an infrastructure for semantically annotated physical products. In Christian Hochberger and Rüdiger Liskowsky, editors, *GI Jahrestagung (2)*, volume 94 of *LNI*, Seiten 544–549. GI, 2006.

[MF07] Wolfgang Maass and Andreas Filler. Tip 'n tell: Product-centered mobile reasoning support for tangible shopping. In Lyndon J B Nixon, Roberta Cuel, David de Francisco, Elena Simperl, and Christoph Tempich, editors, *Proceedings of Workshop on Making Semantics Work For Business (MSWFB 2007), part of 1st European Semantic Technology Conference*, Seiten 12–17, Vienna, Austria, 2007.

[Mic12] Whatwg: Microdata — html5 draft standard, 2012. Letzter Zugriff: 26.11.2012. URL: http://www.whatwg.org/specs/web-apps/current-work/multipage/microdata.html.

[Mil01] Libby Miller. RDF Squish query language and Java implementation. Public draft, Institute for Learning and Research Technology, 2001. Letzter Zugriff: 26.11.2012. URL: http://ilrt.org/discovery/2001/02/squish/.

[MJ07] Wolfgang Maass and Sabine Janzen. Dynamic product interfaces: A key element for ambient shopping environments. In *eMergence: Merging and Emerging Technologies, Processes, and Institutions*, volume 20 of *Bled eConference*, Seiten 457 – 470. 20th Bled eCommerce Conference, June 2007.

[MK11] R. B. Mishra and Sandeep Kumar. Semantic web reasoners and languages. *Artificial Intelligence Review*, 35(4):339–368, April 2011.

[MM04] Frank Manola and Eric Miller. Rdf primer (w3c recommendation), 2004. Letzter Zugriff: 26.11.2012. URL: http://www.w3.org/TR/rdf-primer/.

[Moc87a] P. Mockapetris. Rfc 1034: Domain names - concepts and facilities, 1987.

[Moc87b] P. Mockapetris. Rfc 1035: Domain names - implementation and specification, 1987.

[Moo65] Gordon E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8), April 1965.

[MSM09] Markus Miche, Daniel Schreiber, and Hartmann Melanie. Core services for smart products. In *AMI-Blocks Workshop at 3rd European Conference on Ambient Intelligence, Salzburg, Austria*, Smart Products: Building Blocks of Ambient Intelligence, Seiten 1–4. Springer Berlin / Heidelberg, 2009.

[Müh07] Max Mühlhäuser. Smart Products: An introduction. In *Constructing Ambient Intelligence: AmI 2007 Workshops*, Seiten 158–164. Springer Berlin / Heidelberg, 2007.

[MvH03] D. L. McGuinness and F. van Harmelen. Owl web ontology language overview, 2003. Letzter Zugriff: 26.11.2012. URL: `http://www.w3.org/TR/owl-features/`.

[NdGV11] Andriy Nikolov, Mathieu d'Aquin, Marina Giordanino, and Elena Vildjiounaite. D.2.4.2: Final management framework and integration with the smart products infrastructure. *SmartProducts WP2 - Semantic Modelling and Management of Proactive Knowledge*, 2011. URL: `http://www.smartproducts-project.eu/mainpage/publications`.

[NdT11] Andriy Nikolov, Mathieu d'Aquin, and Keerthi Thomas. D.2.1.3: Final version of the conceptual framework. *SmartProducts WP2 - Semantic Modelling and Management of Proactive Knowledge*, 2011. URL: `http://www.smartproducts-project.eu/mainpage/publications`.

[NGK+05] Alassane Ndiaye, Patrick Gebhard, Michael Kipp, Martin Klesen, Michael Schneider, and Wolfgang Wahlster. Ambient intelligence in edutainment: Tangible interaction with life-like exhibit guides. In *Intelligent Technologies for Interactive Entertainment: First International Conference, INTETAIN 2005, Madonna di Campiglio, Italy*, Seiten 104–113. Springer Berlin / Heidelberg, 2005.

[NHFB11] Robert Neßelrath, Jens Haupert, Jochen Frey, and Boris Brandherm. Supporting persons with special needs in their daily life in a smart home. In *Proceedings 2011 Seventh International Conference on Intelligent Environments (IE 2011). International Conference on Intelligent Environments (IE-11), July 25-28, Nottingham, United Kingdom*, Seiten 370–373. IEEE Computer Society, 7 2011.

[Nor99] D. A. Norman. *The invisible computer : why good products can fail, the personal computer is so complex, and information appliances are the solution*. MIT Press, 1. mit press paperback ed edition, 1999.

[NP01] Ian Niles and Adam Pease. Towards a standard upper ontology. In *Proceedings of the international conference on Formal Ontology in Information Systems - Volume 2001*, FOIS '01, Seiten 2–9, New York, NY, USA, 2001. ACM.

[Ope12] Oasis open document format for office applications (opendocument) tc, 2012. Letzter Zugriff: 26.11.2012. URL: http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=office#odf11.

[Pas01] Jason Pascoe. *Context-Aware Software*. PhD thesis, University of Kent at Canterbury, 2001.

[PBdWdD89] Wolfgang. Pfeifer, Wilhelm. Braun, and Akademie der Wissenschaften der DDR. *Etymologisches Wörterbuch des Deutschen / erarbeitet von einem Autorenkollektiv des Zentralinstituts fur Sprachwissenschaft unter der Leitung von Wolfgang Pfeifer ; Autoren, Wilhelm Braun ... [et al.]*. Akademie-Verlag, Berlin, 1989.

[PCS+04] Shankar Pal, Istvan Cseri, Oliver Seeliger, Gideon Schaller, Leo Giakoumakis, and Vasili Zolotov. Indexing xml data stored in a relational database. In *Proceedings of the Thirtieth international conference on Very large data bases - Volume 30*, VLDB '04, Seiten 1146–1157. VLDB Endowment, 2004. URL: http://dl.acm.org/citation.cfm?id=1316689.1316787.

[PNL02] Adam Pease, Ian Niles, and John Li. The suggested upper merged ontology: A large ontology for the semantic web and its applications. In *In Working Notes of the AAAI-2002 Workshop on Ontologies and the Semantic Web, Edmonton, Canada*. AAAI Press, 2002.

[PPG09] Daniel Peintner and Santiago Pericas-Geertsen. Efficient xml interchange (exi) primer, 2009. Letzter Zugriff: 26.11.2012. URL: http://www.w3.org/TR/2009/WD-exi-primer-20091208/.

[REG+11] Gunther Reinhard, Philipp Engelhardt, Emin Genc, Thomas Irrenhauser, Michael Niehues, Martin Ostgathe, and Kristen Reisen. Einsatz von RFID in der Wertschöpfungskette - Konsortium entwickelt RFID-basierte hybride Steuerungsarchitektur und Bewertungsmethode für Wertschöpfungsketten. *RFID im Blick - Sonderausgabe RFID in der Region München*, 3 2011.

[Ros97] O. Rosenbaum. *Das Barcode-Lexikon*. Edition advanced. BHV-Verlag, 1997.

[RSS09] Rss 2.0 specification, 2009. Letzter Zugriff: 26.11.2012. URL: http://www.rssboard.org/rss-specification.

[Sä08] Anselmi Sääksvuori, Antti ; Immonen. *Product lifecycle management : with ... 5 tables*. Springer Berlin / Heidelberg, Berlin, 3. ed. edition, 2008.

[SAH+11a] Daniel Schreiber, Syed Zahid Ali, Aristotelis Hadjakos, Mika Hillukkala, Ivan Delchev, Markus Miche, Andreas Budde, and Safdar Ali. D.6.3.2, d.6.4.2, d.6.5.2: Final smart products communication middleware, final sensor and actuator integration framework and final context and environment model

framework. *SmartProducts WP6 - Technical Framework, Integration and System Architecture*, 2011. URL: http://www.smartproducts-project.eu/mainpage/publications.

[SAH+11b] Daniel Schreiber, Syed Zahid Ali, Melanie Hartmann, Ivan Delchev, and Mika Hillukkala. D.6.2.2: Final architecture and specification of platform core services. *SmartProducts WP6 - Integrated Concepts fro Smart Products and Proactive Knowledge*, 2011. URL: http://www.smartproducts-project.eu/mainpage/publications.

[SAHS06] Rudi Studer, Anupriya Ankolekar, Pascal Hitzler, and York Sure. A semantic future for AI. *IEEE Intelligent Systems*, 21(4):8–9, 2006.

[Sch95] William N. Schilit. *A system architecture for context-aware mobile computing*. PhD thesis, Columbia University, New York, NY, USA, 1995.

[Sch02] Albrecht Schmidt. *Ubiquitous Computing - Computing in Context*. PhD thesis, Lancaster University, November 2002.

[Sch07] Michael Schneider. Towards a general object memory. In *Proceedings of the First International Workshop on Design and Integration Principles for Smart Objects (DIPSO 2007), Innsbruck, Austria*, Seiten 307 – 312. Springer Berlin / Heidelberg, 2007.

[Sch10] Michael Schneider. *Resource-Aware Plan Recognition in Instrumented Environments*. PhD thesis, Dept. Computer Science, Saarland University, 2010.

[Sch11] Martin Schäfer. LEO - Lightweight Editor for Ontologies. Bachelorthesis, Universität des Saarlandes, 2011. URL: http://aida.cs.uni-sb.de/thesis/show/52.

[Sev03] M. Sevcenko. Online presentation of an upper ontology. In *In Proceedings of Znalosti, Ostrava, Czech Republic*. VŠB, 2003.

[SFAB11] Christoph Stahl, Jochen Frey, Jan Alexandersson, and Boris Brandherm. Synchronized realities. *Journal of Ambient Intelligence and Smart Environments (JAISE)*, 3(1 / 2011):13–25, 1 2011.

[SHSS10] Marc Seissler, Ines Heck, Peter Stephan, and Jochen Schlick. Bridging the gap between digital object memory information and its binary representation. In Michael Schneider, Alexander Kröner, Peter Stephan, Thomas Plötz, Fahim Kawsar, and Gerd Kortuem, editors, *DOME-IoT: International Workshop on Digital Object Memories in the Internet of Things. Workshop on Digital Object Memories (DOME-2010), located at Ubicomp, September 26-29, Copenhagen, Denmark*, Seiten 25–30. ACM, 9 2010.

[Sim87] Peter M. Simons. *Parts: A Study in Ontology*. Oxford University Press, 1987.

[SK08] Michael Schneider and Alexander Kröner. The smart pizza packing: An application of object memories. In *Proceedings of the 4th International Conference on Intelligent Environnments. Intelligent Environments, in Conjunction with 4th International Conference on Intelligent Environments, July 21-22, Seattle„ WA, United States*, Seiten 1 – 8. The Institution of Engineering and Technology, 2008.

[SKjBe+98] Norbert A. Streitz, S. Konomi, H. j. Burkhardt (eds, Cooperative Buildings Integrating, Norbert A. Streitz, Jörg Geißler, and Torsten Holmer. Roomware for cooperative buildings: Integrated design of architectural spaces and information spaces. In *1 st Intl Workshop on Cooperative Buildings, Darmstadt, Germany*. Springer Berlin / Heidelberg, 1998.

[SKKM+05] Lea Skorin-Kapov, Hrvoje Komericki, Maja Matijasevic, Igor S. Pandzic, and Miran Mosmondor. Muva: a flexible visualization architecture for multiple client platforms. *Journal Mobile Multimedia*, 1(1):3–17, 2005.

[SKSM07] Nico Schlitter, Florian Kähne, Stiefen T. Schilz, and Holger Mattke. *Operations and Technology Management*, volume 4 of *Innovative Logistics Management*, chapter Potential and Problems of RFID-Based Cooperation in a Supply Chain, Seiten 147–164. Erich Schmidt Verlag, 2007.

[Sma12] Smart products webseite, 2012. Letzter Zugriff: 26.11.2012. URL: `http://www.smartproducts-project.eu/`.

[SMK+10] Peter Stephan, Gerrit Meixner, Holger Koessling, Florian Floerchinger, and Lisa Ollinger. Product-mediated communication through digital object memories in heterogeneous value chains. In *8th Annual IEEE International Conference on Pervasive Computing and Communications. IEEE International Conference on Pervasive Computing and Communications (PerCom-2010), March 29 - April 2, Mannheim, Germany*, Seiten 199–207. IEEE Computer Society, 2010.

[SRLRT11] Bernd Scholz-Reiter, Dennis Lappe, Carmen Ruthenbeck, and Christian Toonen. Introduction of a hybrid control approach for automotive logistics. In *Proceedings of the 11th WSEAS international conference on robotics, control and manufacturing technology, and 11th WSEAS international conference on Multimedia systems & signal processing*, ROCOM'11/MUSP'11, Seiten 69–73, Stevens Point, Wisconsin, USA, 2011. World Scientific and Engineering Academy and Society (WSEAS). URL: `http://dl.acm.org/citation.cfm?id=1965760.1965771`.

[SS02] York Sure and Rudi Studer. On-to-knowledge methodology. In F. van Harmelen (eds.) J. Davies, D. Fensel, editor, *On-To-Knowledge: Semantic Web enabled Knowledge Management*, Seiten 33–46. Wiley, 2002.

[SvH05] Heiner Stuckenschmidt and Frank van Harmelen. *Information Sharing on the Semantic Web*. Springer Berlin / Heidelberg, Berlin, 2005.

[SVH10] Michael Schneider, Michael Velten, and Jens Haupert. The objectrules framework - providing ad hoc context-dependent assistance in dynamic environments. In Vic Callaghan, Achilles Kameas, Simon Egerton, Ichiro Satoh, and Michael Weber, editors, *Proceedings of the Sixth International Conference on Intelligent Environments. International Conference on Intelligent Environments (IE-10), July 19-21, Kuala Lumpur, Malaysia*, Seiten 122–127. IEEE Computer Society, 2010.

[TAB+10] Ken Traub, Felice Armenio, Henri Barthel, Paul Dietrich, John Duker, Christian Floerkemeier, John Garret, Mark Harrison, Bernie Hogan, Jin Mitsugi, Josef Preishuber-Pfluegl, Oleg Ryaboy, Sanjay Sarma, KK Suen, and John Williams. The EPCglobal Architecture Framework Version 1.4, 2010. URL: `http://www.epcglobalinc.org`.

[TG00] Kei Nam Tsoi and Eduard Gröller. Adaptive visualization over the Internet (TR-186-2-00-21), November 2000.

[TTS09] Conrad Thiede, Christian Tominski, and Heidrun Schumann. Service-oriented information visualization for smart environments. In *IV '09: Proceedings of the 2009 13th International Conference Information Visualisation*, Seiten 227–234, Washington, DC, USA, 2009. IEEE Computer Society.

[UHM11] Dieter Uckelmann, Mark Harrison, and Florian Michahelles, editors. *Architecting the Internet of Things*. Springer Berlin / Heidelberg, Berlin, 2011.

[U.S00] U.S. Deparment of Commerce, National Institure of Standards and Technology. Digital signature standard (dss), 2000.

[Wah03] Wolfgang Wahlster. Towards symmetric multimodality: Fusion and fission of speech, gesture, and facial expression. In *in Proceedings of the 26th German Conference on Artificial Intelligence, September 2003*, Seiten 1–18. Springer Berlin / Heidelberg, 2003.

[Wah08a] Wolfgang Wahlster. SmartWeb — Ein multimodales Dialogsystem für das semantische Web (2004–2007). In *Informatikforschung in Deutschland*, Seiten 300–311. Springer Berlin / Heidelberg, 2008.

[Wah08b] Wolfgang Wahlster. Von Suchmaschinen zu Antwortmaschinen: Semantische Technologien und Benutzerpartizipation im Web 3.0. In *Wie arbeiten die Suchmaschinen von morgen?*, Seiten 59–75. Fraunhofer Irb, 2008.

[Wah13a] Wolfgang Wahlster. volume SemProM - Foundations of Semantic Product Memories for the Internet of Things. Springer Berlin / Heidelberg, 2013. im Erscheinen.

[Wah13b] Wolfgang Wahlster. *The Semantic Product Memory: An Interactive Black Box for Smart Objects*, volume SemProM - Foundations of Semantic Product Memories for the Internet of Things, Seiten 3 – 22. Springer Berlin / Heidelberg, 2013. im Erscheinen.

[Wei95] Mark Weiser. Human-computer interaction. chapter The computer for the 21st century, Seiten 933–940. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1995.

[Wei12] Luc Weiler. Mobile Unterstützung bei der Durchführung und Protokollierung komplexer Aufgaben. Bachelorthesis, Universität des Saarlandes, 2012. URL: `http://aida.cs.uni-sb.de/thesis/show/77`.

[WKSB08] Wolfgang Wahlster, Alexander Kröner, Michael Schneider, and Jörg Baus. Sharing memories of smart products and their consumers in instrumented environments. *it – Information Technology*, 50(1):45–50, 2008.

ANHANG A

# RDFa

**Quellcode A.1: RDFa-Darstellung von OMM-Gedächtnissen**

```
<!DOCTYPE html>
<html lang="en" xmlns:omm="http://www.w3.org/2005/Incubator/omm/elements/1/">
  <head>
    <meta charset="utf-8" />
    <title>OMM-Sample</title>
  </head>
  <body>
    <article typeof="omm:omm">
      <header>
        <span >
          <section typeof="omm:blockList">
            <p>
              OMM-Blocks:
            </p>
            <ul>
              <li rel="omm:hasBlock">
                <section typeof="omm:block">
                  OMM-Block Entry:
                  <ul>
                    <li>
                      ID: <span property="omm:blockID">12345</span>
                    </li>
                    <li>
                      Title: <span property="omm:title" xml:lang="en">sample
                          title</span>
                    </li>
                    <li>
                      Description: <span property="omm:description" xml:lang="
                          en">sample description</span>
                    </li>
                    <li>
                      Namespace: <span property="omm:namespace">http://www.w3.
                          org/2005/Incubator/omm/microdata/ns/sample</span>
                    </li>
```

```
32                    <li rel="omm:hasCreation">
33                      Creation:
34                      <ul typeof="omm:creation">
35                        <li>
36                          Creator: <span property="omm:creator">195505177</span
                                >
37                          of type: <span property="omm:type">duns</span>
38                        </li>
39                        <li>
40                          Date: <span property="omm:date" datatype="xsd:
                                dateTime">2011-01-30T18:30:00+02:00</span>
41                          of encoding: ISO8601
42                        </li>
43                      </ul>
44                    </li>
45                    <li rel="omm:hasContribution">
46                      Contribution:
47                      <ul>
48                        <li>
49                          Contributor:<br/>
50                          <ul typeof="omm:contribution">
51                            <li>
52                              Contributor: <span property="omm:contributor">
                                    user@dfki.de</span>
53                              of type: <span property="omm:type">email</span>
54                            </li>
55                            <li>
56                              Date: <span property="omm:date" datatype="xsd:
                                    dateTime">2011-01-31T08:12:50+02:00</span>
57                              of encoding: ISO8601
58                            </li>
59                          </ul>
60                        </li>
61                      </ul>
62                    </li>
63                    <li rel="omm:hasFormat">
64                      <span typeof="omm:format">
65                        Format: <span property="omm:format">application/xml</
                              span>of schema: <span property="omm:schema">http://
                              www.w3.org/2005/Incubator/omm/microdata/schema/
                              sample.xsd</span> with encryption: <span property="
                              omm:encryption">none</span>
66                      </span>
67                    </li>
68                    <li rel="omm:hasType">
69                      <span typeof="omm:type">
70                        Type:
71                        <span property="omm:type">
72                          http://purl.org/dc/dcmitype/Dataset
73                        </span>
74                      </span>
75                    </li>
```

```
76                      <li rel="omm:hasSubject">
77                        <ul typeof="omm:subject">
78                          <li rel="omm:hasTag">
79                            <span typeof="omm:ontologyValue">
80                              Value:
81                              <span property="omm:value">http://o.org/def.owl#
                                    ManufacturerData</span>
82                              of type ontology
83                            </span>
84                          </li>
85                          <li rel="omm:hasTag">
86                            <span typeof="omm:textValue">
87                              Value:
88                              <span property="omm:value">
89                                ingredients
90                              </span>
91                              of type text
92                            </span>
93                          </li>
94                          <li rel="omm:hasTag">
95                            <span typeof="omm:hierarchicalTextValue">
96                              Value:
97                              <span property="omm:value">norms</span>.<span rel="
                                    omm:hasTag">
98                                <span typeof="omm:hierarchicalTextValue">
99                                  <span property="omm:value">din</span>.<span rel
                                        ="omm:hasTag">
100                                   <span typeof="omm:textValue">
101                                     <span property="omm:value">e12</span>
102                                   </span>
103                                 </span>
104                               </span>
105                             </span>
106                           </span>
107                         </li>
108                       </ul>
109                     </li>
110                     <span rel="omm:hasPayload">
111                       <span property="omm:value">...</span>
112                       <span property="omm:encoding" content="base64"></span>
113                     </span>
114                   </ul>
115                 </section>
116               </li>
117             </ul>
118           </section>
119         </header>
120     </article>
121   </body>
122 </html>
```

ANHANG B

# Microdata

**Quellcode B.1: Microdata-Darstellung von OMM-Gedächtnissen**

```
<!DOCTYPE html>
<html lang="en" xmlns:omm="http://www.w3.org/2005/Incubator/omm/elements/1/">
  <head>
    <meta charset="utf-8" />
    <title>OMM-Sample</title>
  </head>
  <body>
    <article typeof="omm:omm">
      <header>
        <span >
          <section typeof="omm:blockList">
            <p>
              OMM-Blocks:
            </p>
            <ul>
              <li rel="omm:hasBlock">
                <section typeof="omm:block">
                  OMM-Block Entry:
                  <ul>
                    <li>
                      ID: <span property="omm:blockID">12345</span>
                    </li>
                    <li>
                      Title: <span property="omm:title" xml:lang="en">sample
                          title</span>
                    </li>
                    <li>
                      Description: <span property="omm:description" xml:lang="
                          en">sample description</span>
                    </li>
                    <li>
                      Namespace: <span property="omm:namespace">http://www.w3.
                          org/2005/Incubator/omm/microdata/ns/sample</span>
                    </li>
```

```
32                    <li rel="omm:hasCreation">
33                      Creation:
34                      <ul typeof="omm:creation">
35                        <li>
36                          Creator: <span property="omm:creator">195505177</span
                                >
37                          of type: <span property="omm:type">duns</span>
38                        </li>
39                        <li>
40                          Date: <span property="omm:date" datatype="xsd:
                                dateTime">2011-01-30T18:30:00+02:00</span>
41                          of encoding: ISO8601
42                        </li>
43                      </ul>
44                    </li>
45                    <li rel="omm:hasContribution">
46                      Contribution:
47                      <ul>
48                        <li>
49                          Contributor:<br/>
50                          <ul typeof="omm:contribution">
51                            <li>
52                              Contributor: <span property="omm:contributor">
                                    user@dfki.de</span>
53                              of type: <span property="omm:type">email</span>
54                            </li>
55                            <li>
56                              Date: <span property="omm:date" datatype="xsd:
                                    dateTime">2011-01-31T08:12:50+02:00</span>
57                              of encoding: ISO8601
58                            </li>
59                          </ul>
60                        </li>
61                      </ul>
62                    </li>
63                    <li rel="omm:hasFormat">
64                      <span typeof="omm:format">
65                        Format: <span property="omm:format">application/xml</
                              span>of schema: <span property="omm:schema">http://
                              www.w3.org/2005/Incubator/omm/microdata/schema/
                              sample.xsd</span> with encryption: <span property="
                              omm:encryption">none</span>
66                      </span>
67                    </li>
68                    <li rel="omm:hasType">
69                      <span typeof="omm:type">
70                        Type:
71                        <span property="omm:type">
72                          http://purl.org/dc/dcmitype/Dataset
73                        </span>
74                      </span>
75                    </li>
```

```
76                      <li rel="omm:hasSubject">
77                        <ul typeof="omm:subject">
78                          <li rel="omm:hasTag">
79                            <span typeof="omm:ontologyValue">
80                              Value:
81                              <span property="omm:value">http://o.org/def.owl#
                                    ManufacturerData</span>
82                              of type ontology
83                            </span>
84                          </li>
85                          <li rel="omm:hasTag">
86                            <span typeof="omm:textValue">
87                              Value:
88                              <span property="omm:value">
89                                ingredients
90                              </span>
91                              of type text
92                            </span>
93                          </li>
94                          <li rel="omm:hasTag">
95                            <span typeof="omm:hierarchicalTextValue">
96                              Value:
97                              <span property="omm:value">norms</span>.<span rel="
                                    omm:hasTag">
98                                <span typeof="omm:hierarchicalTextValue">
99                                  <span property="omm:value">din</span>.<span rel
                                        ="omm:hasTag">
100                                   <span typeof="omm:textValue">
101                                     <span property="omm:value">e12</span>
102                                   </span>
103                                 </span>
104                               </span>
105                             </span>
106                           </span>
107                         </li>
108                       </ul>
109                     </li>
110                     <span rel="omm:hasPayload">
111                       <span property="omm:value">...</span>
112                       <span property="omm:encoding" content="base64"></span>
113                     </span>
114                   </ul>
115                 </section>
116               </li>
117             </ul>
118           </section>
119         </header>
120     </article>
121   </body>
122 </html>
```